Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

КОМПЛЕКСНА БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Апаратно-програмний комплекс моніторингу стану оточуючого середовища на базі IoT засобів. Частина 3. Вебзастосунок для трансляції відеопотоку»

08-54. КБКР.040.00.000 ПЗ

Виконала: студентка 4 курсу, групи 2КІ-21б
спеціальності 123 — Комп'ютерна інженерія
освітня програма — «Комп'ютерна інженерія»
Вовковинська А. В.

Керівник: к.т. н., доц. кафедри ОТ
Войцеховська О.В.

Рецензент: к.фіз.-мат. н., доц. кафедри МБіС
Шиян А. А.

Допущено до захисту
Завідувач кафедри ОТ
д.т.н., проф. Азаров О.Д.
10.06 2025 року

Вінниця ВНТУ — 2025 рік

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Освітньо-кваліфікаційний рівень бакалавр
Галузь знань – 12 – Інформаційні технології
Спеціальність – 123 – Комп'ютерна інженерія
Освітньо-професійна програма – Комп'ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н., проф._____Азаров О.Д.
«21» березня 2025 року

З А В Д А Н Н Я
НА КОМПЛЕКСНУ БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ
СТУДЕНТУ
Вовковинській Аліні Вадимівні

1 Тема роботи: Апаратно-програмний комплекс моніторингу стану оточуючого середовища на базі IoT засобів. Частина 3. Вебзастосунок для трансляції відеопотоку, керівник роботи Войцеховська Олена Валеріївна, к.т.н., доцент кафедри ОТ, затверджено наказом вищого навчального закладу від «20» березня 2025 року №97.

2 Термін подання студенткою роботи 13.05.2025

3 Вихідні дані до роботи: технічна документація на модуль ESP32-CAM, базові схеми підключення пристрою, прошивка з відкритим кодом для мікроконтролера, середовище розробки — Visual Studio Code, технології для створення клієнтської частини.

4 Зміст розрахунково-пояснювальної записки: вступ; огляд та аналіз сучасних систем трансляції потокового відео, проєктування вебзастосунку із трансляцією відеопотоку; програмна реалізація вебзастосунку, тестування та аналіз результатів; висновки.

5 Перелік ілюстративного матеріалу: структурна схема взаємодії компонентів, дизайн-макети вебзастосунку у середовищі Figma, відображення головної сторінки на персональному комп'ютері у світлій темі, відображення

головної сторінки на персональному комп'ютері у темній темі, відображе
головної сторінки на мобільному пристрої у світлій темі, сторінка «Трансляц
при увімкненій трансляції в темній темі, сторінка «Трансляція» при увімкне
трансляції на мобільному пристрої у світлій та темній темі, сторінка «Показни
у темній темі, сторінка «Показники» на мобільному пристрої у світлій та тем
темі.

6 Консультанти розділів роботи приведено в таблиці 1.

Таблиця 1 — Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Дата		Підп
		видав	прийняв	
1-4	Войцеховська О. В., к.т.н., доцент каф. ОТ	21.03.25	13.05.25	
Нормоконтроль	ас. каф. ОТ Швець С. І.	14.05.25	30.05.25	

7 Дата видачі завдання 21.03.2025 р.

8 Календарний план виконання МКР приведений в таблиці 2.

Таблиця 2 — Календарний план

№ з/п	Назва етапів дипломної роботи	Термін виконання	Підпис
1	Аналіз завдання	21.03.25 — 30.05.25	вик.
2	Огляд сучасних систем відеоспостереження	21.03.25 — 30.05.25	вик.
3	Аналіз апаратно-програмних рішень для передачі відеопотоку	21.03.25 — 30.05.25	вик.
4	Обґрунтування вибору апаратного забезпечення та засобів розробки інтерфейсу вебзастосунку	31.03.25—13.04.25	вик.
6	Розробка структурної схеми вебзастосунку та мапи вебзастосунку	14.04.25—27.04.25	вик.
7	Реалізація прошивки ESP32-CAM та налаштування тунельного рішення	14.04.25—27.04.25	вик.
9	Розробка клієнтської частини вебзастосунку	14.04.25—27.04.25	вик.
10	Тестування системи, оцінка відео та адаптивності	14.04.25—27.04.25	вик.
11	Оформлення пояснювальної записки та ілюстративного матеріалу	28.04.25—11.05.25	вик.
12	Перевірка якості виконання бакалаврської кваліфікаційної роботи та усунення недоліків	13.05.25	вик.

Студентка Аліна ВОВКОВИНСЬКА

Керівник роботи к.т.н., доц. каф. ОТ Олена ВОЙЦЕХОВСЬКА

АНОТАЦІЯ

УДК 004.77:004.738.5

Вовковинська А. В. Вебзастосунок для трансляції відеопотоку. Бакалаврська кваліфікаційна дипломна робота зі спеціальності 123 «Комп'ютерна інженерія», освітня програма «Комп'ютерна інженерія». Вінниця: ВНТУ, 2025. 79 с.

На укр.мові. Бібліогр.: 25 назв, рис.: 21, табл.: 6.

У комплексній бакалаврській кваліфікаційній роботі реалізовано вебзастосунок для трансляції відеопотоку в межах апаратно-програмного комплексу моніторингу стану навколишнього середовища.

В роботі спроєктовано структуру вебзастосунку для перегляду і запису відеопотоку в режимі реального часу та відображення показників вимірювань датчиків з пристроїв IoT, що входять до складу комплексу моніторингу. Розроблено клієнтську частину вебзастосунку з налаштованим HTTPS-тунелем для передачі відеопотоку, яка забезпечує масштабованість, адаптивність інтерфейсу та можливість подальшого розширення.

У роботі описано особливості взаємодії програмних компонентів із мікроконтролером ESP32-CAM, що виконує функції збору відеоінформації, а також представлено результати тестування функціональних можливостей розробленого вебінтерфейсу.

Ключові слова: моніторинг середовища, IoT, трансляція відеопотоку, мікроконтролер ESP32-CAM, вебзастосунок, Angular, TypeScript.

ABSTRACT

Vovkovynska A. V. Web application for broadcasting video stream. Bachelor's qualification thesis in specialty 123 “Computer Engineering”, educational program “Computer Engineering”. Vinnytsia: VNTU, 2025. 79 p.

In Ukrainian. Bibliogr.: 25 titles, figures: 21, tables: 6.

The web application for broadcasting a video stream within the hardware and software complex of environmental monitoring was realized in the complex bachelor's qualification work.

The structure of a web application for viewing and recording a video stream in real time and displaying sensor readings from IoT devices that are part of the monitoring complex is designed in this work. The client part of the web application with a configured HTTPS tunnel for video stream transmission has been developed, which provides scalability, adaptability of the interface and the possibility of further expansion.

The bachelor's qualification work describes the peculiarities of interaction between the software components and the ESP32-CAM microcontroller, which performs the functions of collecting video information, and also presents the results of testing the functionality of the developed web interface.

Keywords: environment monitoring, IoT, video streaming, ESP32-CAM microcontroller, web application, Angular, TypeScript.

ЗМІСТ

ВСТУП

Системи моніторингу відіграють важливу роль у сучасному світі, забезпечуючи безперервний контроль за різноманітними об'єктами, процесами або довкіллям. Їх актуальність зростає у зв’язку з розвитком цифрових технологій, що дозволяють автоматизувати збір, обробку та аналіз даних. Сучасні системи моніторингу все частіше базуються на розподілених обчисленнях, хмарних сервісах та бездротовому зв’язку, що дає можливість контролювати віддалені об’єкти в реальному часі. На сьогодні особлива увага приділяється моніторингу стану оточуючого середовища. Важливим елементом таких систем моніторингу є програмне забезпечення, яке визначає логіку роботи системи та забезпечує взаємодію з користувачем.

Актуальність роботи полягає в тому, що при створенні апаратно-програмного комплексу моніторингу стану оточуючого середовища на базі IoT засобів, виникла необхідність реалізації вебзастосунку для прямої взаємодії з користувачем, а також надання користувачеві зручних засобів для перегляду відеотрансляції, показників стану навколишнього середовища.

Метою роботи є розробка вебзастосунку, який забезпечує трансляцію відео з мікроконтролера ESP32 в складі апаратно-програмного комплексу моніторингу стану оточуючого середовища на базі IoT-технологій, організовує взаємодію користувача з системою через інтуїтивний інтерфейс, а також підтримує зберігання та обробку отриманих даних.

Для досягнення поставленої мети необхідно виконати такі **задачі**:

— проаналізувати існуючі рішення у сфері моніторингу стану оточуючого середовища та вебзастосунків для IoT моніторингу;

— обґрунтувати вибір апаратного забезпечення для реалізації частини комплексу моніторингу стану оточуючого середовища;

— визначити архітектуру вебдодатку та обрати відповідний стек технологій для його реалізації;

— розробити вебінтерфейс для перегляду відеопотоку з мікроконтролера ESP32;

— забезпечити можливість віддаленого доступу відеопотоку;

— інтегрувати модуль ESP32 з вебзастосунком та забезпечити стабільну трансляцію;

— реалізувати можливість запису та збереження отриманого відеопотоку;

— провести тестування працездатності роботи, зручність користування інтерфейсом та оцінити якість відео, що передається для трансляції.

Об’єктом дослідження є процес перетворення відеосигналу із зовнішнього пристрою в цифровий формат.

Предметом дослідження є вебзастосунок для трансляції відеопотоку з мікропроцесора ESP32.

Методи дослідження включають архітектурний підхід до побудови клієнт-серверних додатків, принципи компонентного програмування, а також використання хмарних технологій.

Апробація результатів бакалаврської роботи: зроблено доповідь на LIV науково-технічну конференцію підрозділів ВНТУ [1].

Публікації за темою роботи: Можливості фреймворку Angular для розробки вебдодатку потокової трансляції відео / Вовковинська А. В., Войцеховська О. В. //Матеріали LIV наукової-технічної конференції підрозділів ВНТУ, 2025 р. Режим доступу: https://d.conf.vntu.edu.ua/index.php/all-fitki/all-fitki-2025/paper/view/23785/19632

1 ОГЛЯД ТА АНАЛІЗ СУЧАСНИХ СИСТЕМ ТРАНСЛЯЦІЇ ПОТОКОВОГО ВІДЕО

Сучасні системи відеоспостереження та трансляції відео в реальному часі значно еволюціонували від простих аналогових камер до складних інтелектуальних рішень, що поєднують апаратне і програмне забезпечення, штучний інтелект (ШІ) та хмарні технології. Ці системи не лише фіксують відео, але й активно аналізують події в реальному часі, забезпечуючи високий рівень безпеки та ефективності.

1.1 Огляд та загальна характеристика систем відеоспостереження

Системи відеоспостереження є невід'ємною складовою сучасних технологій забезпечення безпеки та моніторингу. Вони активно використовуються в різних сферах від охорони громадського порядку та промислових об'єктів до контролю умов навколишнього середовища. Основна функція таких систем полягає у візуальному контролі подій у реальному часі та фіксації відеозаписів для подальшого аналізу.

Історично розвиток відеоспостереження розпочинався з аналогових систем, які передавали сигнал за допомогою коаксіального кабелю та зберігали зображення на магнітні носії. Хоча такі рішення були технологічним проривом у свій час, вони мали суттєві обмеження щодо якості відео, можливості масштабування та автоматизації обробки даних.

Розвиток цифрових технологій у поєднанні з удосконаленням засобів передачі інформації сприяв переходу до IP-систем відеоспостереження, що відзначаються високою роздільною здатністю, гнучкістю конфігурації, підтримкою віддаленого доступу та централізованого керування. Сучасні камери часто підтримують функції відеоаналітики, які дозволяють не лише фіксувати зображення, а й автоматично виявляти рух, розпізнавати обличчя, номери автомобілів, залишені об'єкти тощо.

Суттєвим етапом еволюції відеоспостереження стало впровадження технологій Інтернету речей (англ. IoT — Internet of Things), що дозволило використовувати компактні та енергоефективні пристрої, такі як ESP32, для

створення розподілених систем із можливістю потокової передачі відео через Інтернет. Завдяки цьому відеоспостереження стало доступним навіть для невеликих об'єктів, освітніх установ і дослідницьких проєктів.

Ще одним важливим напрямом розвитку є інтеграція систем відеоспостереження з хмарними сервісами, такими як Google Firebase, AWS чи Azure, які дозволяють зберігати відео, керувати пристроями, забезпечувати безпеку даних і масштабувати рішення відповідно до потреб. У поєднанні з вебтехнологіями, такими як Angular, це відкриває нові можливості для створення зручних та функціональних вебінтерфейсів для керування та перегляду відео в реальному часі.

Таким чином, сучасні системи відеоспостереження — це складні апаратно-програмні комплекси, які поєднують мережеві технології, інтелектуальний аналіз, хмарні сервіси та інтерфейси для кінцевих користувачів. Їхній розвиток продовжує рух у напрямку автоматизації, гнучкості, мобільності та зниження витрат на розгортання й обслуговування. Але важливою складовою такої системи є вебдодаток, який надає користувачу зрозумілий інтерфейс з можливостю перегляду трансляції відео та, за потреби, підтримує додаткові функції керування та взаємодії.

1.2 Аналіз сучасних апаратно-програмних комплексів на базі IoT

Інтернет речей (англ. Internet of Things, IoT) відкрив нові можливості для побудови апаратно-програмних комплексів (АПК), що забезпечують збір, передавання та обробку даних у реальному часі. Сучасні IoT-системи широко застосовуються у сфері моніторингу навколишнього середовища, розумного дому, аграрного сектору, медицини, безпеки, транспорту та інших галузях. Їхня поширеність зумовлена доступністю компонентів, енергоефективністю пристроїв, розширеною функціональністю та можливістю інтеграції з хмарними сервісами.

На рисунку 1.1 показано трирівневу архітектуру Інтернету речей (IoT)**.** Вона складається з трьох основних шарів, кожен із яких виконує певні функції у процесі збору, передавання та обробки інформації в системах IoT [2]:

— апаратного рівня, тобто фізичних пристроїв, таких як мікроконтролери, сенсори, камери, які здійснюють збір даних;

— мережного рівня у вигляді модулів передачі даних (Wi-Fi, Bluetooth, GSM, LoRaWAN), що забезпечують зв'язок між пристроями та центральними серверами або хмарою;

— програмного рівня, тобто інтерфейсів, баз даних, серверних логік і вебзастосунків, які аналізують, зберігають та візуалізують зібрану інформацію.

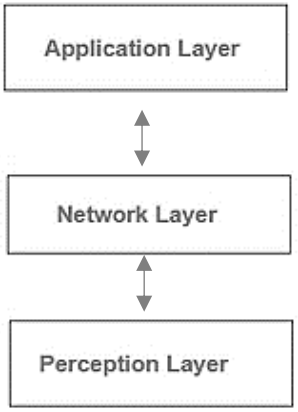

Рисунок 1.1 — Трирівнева архітектура Інтернету речей (IoT)

Одним з найбільш популярних і доступних рішень для розробки IoT-систем є мікроконтролер ESP32 виробництва компанії Espressif, який зображений на рисунку 1.2. Це енергоефективний модуль з вбудованим Wi-Fi та Bluetooth, який підтримує роботу з сенсорами, камерами та іншими периферійними пристроями. У поєднанні з компактністю та низькою вартістю ESP32 став базовим компонентом для багатьох освітніх, дослідницьких та промислових проєктів [3].

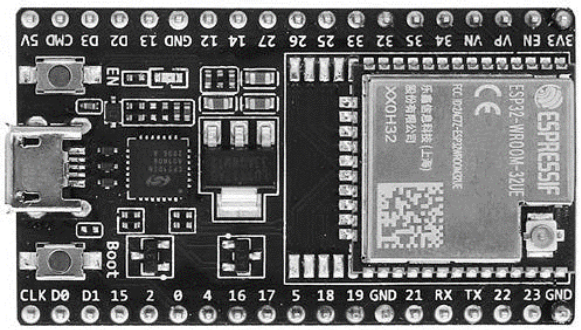

Рисунок 1.2 — Вигляд мікроконтролера ESP32 виробництва компанії Espressif

У контексті побудови систем відеоспостереження ESP32, зокрема його модифікація ESP32-CAM, який зображений на рисунку 1.3, дозволяє реалізувати функцію відеострімінгу в реальному часі з використанням веб-серверів, HTTP або RTSP протоколів. Такий пристрій може бути налаштований для автономної роботи, зберігання даних на SD-карті або передавання відео на зовнішній сервер чи в хмарне сховище [4].

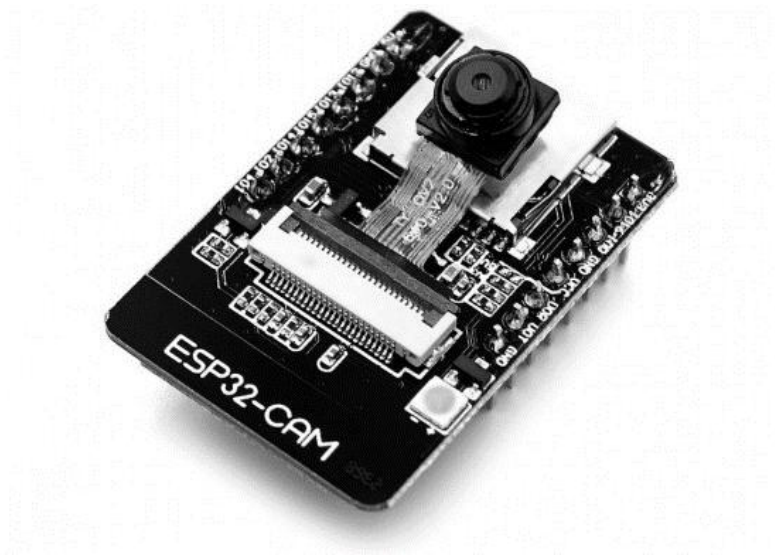

Рисунок 1.3 — Вигляд модуля ESP32-CAM виробництва компанії Espressif

На програмному рівні IoT-системи часто інтегруються з платформами, такими як, Firebase, AWS IoT, ThingsBoard або Blynk, які надають інструменти для зберігання даних, автентифікації користувачів, надсилання сповіщень та візуалізації результатів.

Також до програмного рівня відноситься інтерфейс для взаємодії між користувачем і пристроєм. Для створення сучасного, адаптивного, гнучкого до масштабування та функціонального додатку існує великий вибір технологій.

Таким чином, сучасні апаратно-програмні комплекси на базі IoT — це високоефективні системи з розподіленою архітектурою, які поєднують доступність, масштабованість та гнучкість у розгортанні. Їхня перевага полягає в можливості швидкого розгортання, легкості модернізації та мінімальних витратах на обслуговування, що робить їх особливо привабливими для систем моніторингу в освітніх, екологічних та соціальних проєктах.

1.3 Огляд технологій для відеотрансляцій в реальному часі

Передача відео в реальному часі є однією з ключових функцій у багатьох сучасних системах моніторингу, зокрема й у проєктах на базі IoT. Технології відеотрансляцій дозволяють не лише спостерігати за подіями онлайн, а й оперативно реагувати на зміну ситуації. Залежно від потреб системи (затримки, якості, типу пристрою чи доступності ресурсів — обираються відповідні технічні рішення.

Однією з найпоширеніших технологій є HTTP MJPEG (Motion JPEG) — проста у реалізації форма трансляції, відповідно до якої відео подається як послідовність зображень JPEG через HTTP-з'єднання. Такий підхід часто використовується в бюджетних рішеннях, зокрема на ESP32-CAM, завдяки невибагливості до обчислювальних ресурсів. Проте MJPEG не підтримує аудіо та не є ефективним за пропускною здатністю мережі [5].

Продуктивною є технологія RTSP (Real-Time Streaming Protocol) — протокол потокової передачі мультимедійних даних, що працює поверх TCP/UDP. RTSP забезпечує низьку затримку, можливість передавання аудіо та відео одночасно, що робить його популярним у професійних системах відеоспостереження. Однак, RTSP практично не підтримується безпосередньо веббраузерами або мобільними ОС. Ані Android, ані iOS не мають вбудованих RTSP-програвачів у стандартних засобах (і браузери також не можуть відтворювати RTSP без додаткових плагінів чи перетворень). Тому RTSP сьогодні майже не використовується для доставки відео кінцевому користувачу через браузер. Натомість, RTSP залишається стандартом для передачі від камери до сервера або шлюзу, у системах відеоспостереження камера передає RTSP-потік на мережевий відеореєстратор чи медіасервер, який вже перетворює його в інший формат для користувача. Наприклад, IoT-камера може відправляти RTSP до хмарного сервісу, де потік перекодується в HLS або WebRTC для вебпереглядача [5].

Іншою сучасною альтернативою є використання WebRTC (Web Real-Time Communication) — технології, яка дозволяє здійснювати передачу аудіо, відео та даних безпосередньо між браузерами без потреби в додатковому програмному

забезпеченні. WebRTC підтримується більшістю сучасних браузерів і є ефективною для створення низьколатентних відеочатів або трансляцій. Для IoT-пристроїв таких як ESP32 реалізація WebRTC є складнішою, але можлива при використанні зовнішнього сигнального сервера. Завдяки використанню UDP і розширених механізмів передачі (SRTP, DTLS) WebRTC досягає наднизької латентності і може передавати потокові дані прямо між пристроями без проміжних серверів (за допомогою технологій NAT-traversal). Браузери Chrome, Firefox, Safari, Edge підтримують WebRTC без плагінів, у тому числі на мобільних платформах. Це означає, що відео з IoT-датчика можна безпосередньо показати у вебзастосунку в реальному часі [5].

Для IoT-моніторингу WebRTC підходить, коли потрібен прямий перегляд з мінімальною затримкою, наприклад, дистанційне керування роботом з камерою або перегляд камери спостереження в режимі реального часу. При цьому WebRTC включає в себе вбудовані засоби адаптації до мережі (контроль втрат пакетів, зміна бітрейту) і підтримує сучасні кодеки (обов'язково H.264 і VP8, опційно VP9, AV1 тощо). Також весь трафік WebRTC шифрується автоматично (DTLS/SRTP), що є плюсом з точки зору безпеки.

Основною проблемою цієї технології є масштабування. WebRTC був спроєктований для моделі "кожен-з-кожним" (наприклад, відеодзвінки), тому для трансляції від одної камери на дуже велику кількість глядачів необхідно використовувати спеціалізовані медіасервери (SFU/MCU) або транслювати через CDN з підтримкою WebRTC. Без цього навантаження на камеру або вузол зростає (потрібно надсилати окремий стрім кожному клієнту). Попри ці складнощі, вже з'являються рішення для масштабного WebRTC-стрімінгу, наприклад розподілена SFU-інфраструктура, що дозволяють охопити навіть мільйон глядачів із суб-секундною затримкою. В цілому, WebRTC – ідеальний вибір для інтерактивних або критичних до затримки застосувань IoT (відеоконференції, телеметрія з камер, дистанційний перегляд у режимі on-line).

Для спрощеної реалізації у фронтенді можна також застосовувати HLS (HTTP Live Streaming) – це протокол потокового мовлення від Apple, який став де-

факто стандартом для масового стрімінгу відео через Інтернет. Його основною перевагою є підтримка відкладеної трансляції та висока сумісність із вебтехнологіями. HLS працює зовсім інакше, ніж RTSP/RTMP/WebRTC. Він розбиває відеопотік на дрібні HTTP-сегменти (фрагменти кілька секунд завдовжки) і віддає їх клієнтам через звичайний веб-сервер. Поряд з сегментами генерується плейлист (файл .m3u8), який містить перелік URL сегментів і оновлюється в режимі реального часу. Клієнт (програвач) завантажує плейлист і почергово завантажує сегменти відео, тому через затримку (10–30 секунд) HLS не підходить для реального часу. В IoT-сценаріях HLS підходить для задач, де потрібно транслювати відео великій кількості користувачів (наприклад, громадський стрім з міської екологічної камери) і де допустима певна затримка. Він менш доцільний для інтерактивного керування чи моментального реагування, але працює для оглядового моніторингу, публічних трансляцій, а також для запису архіву (оскільки сегменти можна скласти в файл). Багато систем здійснюють запис відео саме у форматі HLS/DASH, зберігаючи сегменти на диску для подальшого доступу.

Таким чином, вибір технології трансляції залежить від характеристик проєкту, адже MJPEG є простим і швидким у впровадженні, RTSP — більш професійним, WebRTC — оптимальним для вебінтерфейсів з мінімальною затримкою, а HLS — зручним для великої аудиторії спостерігачів. У випадку застосування ESP32-CAM найпоширенішим рішенням є MJPEG-трансляція, однак із використанням проміжного проксі-сервера можна реалізовувати також трансляцію у форматах WebRTC або HLS.

1.4 Протоколи передачі даних для трансляції потокового відео

Передача даних та IoT є критичним елементом, що визначає швидкість, стабільність і якість взаємодії між пристроями. Залежно від архітектури системи, обсягу переданої інформації, енергоспоживання та типу середовища, обираються відповідні протоколи зв’язку. Найпоширенішими серед них є Wi-Fi, Bluetooth та RTSP.

Wi-Fi (Wireless Fidelity) — це один із найпоширеніших та використовуваних

протоколів бездротової передачі даних у локальних мережах. Його головними перевагами є висока швидкість (до сотень Мбіт/с), підтримка відеострімінгу, широка зона покриття (до 100 м у відкритому просторі) та сумісність із більшістю сучасних пристроїв. У системах на базі ESP32 з камерою (ESP32-CAM) саме Wi-Fi використовується для передавання відео на вебінтерфейс або хмарний сервер. З огляду на енерговитрати, Wi-Fi є прийнятним рішенням для стаціонарних пристроїв з постійним живленням.

Bluetooth, порівняно з Wi-Fi, має менший радіус дії, однак вирізняється низьким енергоспоживанням. У системах відеоспостереження він може використовуватися як допоміжний засіб, зокрема для початкового налаштування або підключення до мобільного пристрою.

RTSP (Real-Time Streaming Protocol) — спеціалізований мережний протокол, що використовується для передавання потокового відео. Він дозволяє клієнтам управляти відеопотоком: запускати, призупиняти, перемотувати та зупиняти трансляцію. На відміну від Wi-Fi та Bluetooth, RTSP — це не фізичний канал, а протокол прикладного рівня, який зазвичай працює поверх TCP або UDP. RTSP ідеально підходить для систем з високими вимогами до швидкодії та якості відео, але вимагає відповідної підтримки на серверній і клієнтській стороні. У проєктах з ESP32 його реалізація можлива, однак вимагає додаткових ресурсів або зовнішніх серверів.

Слід зазначити, що в реальних системах ці протоколи часто використовуються разом. Наприклад, ESP32 може передавати відео через RTSP або MJPEG по Wi-Fi, при цьому Bluetooth використовується для налаштування або резервного з’єднання. При розробці вебзастосунків особливу увагу слід приділяти вибору формату потоку, оскільки підтримка RTSP у браузерах обмежена, що потребує транскодування або застосування проксі-сервера.

Таким чином, правильний вибір протоколів передачі даних напряму впливає на ефективність роботи системи відеоспостереження. Wi-Fi забезпечує стабільне бездротове з’єднання, Bluetooth — низькоенергетичну підтримку, а RTSP — контрольований потік високоякісного відео. У поєднанні вони формують гнучке та

надійне рішення для сучасних IoT-комплексів.

1.5 Варіативний аналіз аналогів

У процесі проєктування апаратно-програмного комплексу для відеомоніторингу на базі IoT-технологій доцільно провести аналіз існуючих аналогічних рішень, для яких реалізується можливість трансляції відеопотоку. Це дозволяє оцінити сучасні підходи до реалізації систем відеоспостереження, виявити їхні сильні та слабкі сторони, а також визначити напрями для вдосконалення власної розробки.

Одним із найпоширеніших прикладів є розумні камери відеоспостереження, зображені на рисунку 1.7, типу Xiaomi, EZVIZ або TP-Link Tapo, які мають вбудований модуль Wi-Fi, підтримують хмарне зберігання відео, нічне бачення, детекцію руху та управління через мобільний застосунок. Такі системи характеризуються високою якістю зображення, простотою налаштування та зручним інтерфейсом. Однак, їхній код є закритим, що унеможливлює модифікацію або адаптацію під нестандартні завдання. Крім того, вони потребують стабільного інтернет-з’єднання та часто мають платну підписку на хмарні сервіси.

Рисунок 1.7 — Сучасні камери відеонагляду

Інший приклад — системи на базі Raspberry Pi з камерними модулями, що дають змогу реалізувати кастомізовані рішення з гнучкою архітектурою. Raspberry Pi підтримує трансляцію відео через RTSP, HLS, WebRTC та зберігання записів на локальному або мережному сховищі. Основна перевага таких систем — повний контроль над апаратною та програмною частинами. Проте їх недоліком є висока

вартість у порівнянні з мікроконтролерами, більші енергозатрати та необхідність у зовнішніх компонентах (карта пам’яті, джерело живлення, мережевий адаптер).

Більш економічною альтернативою є рішення на основі ESP32-CAM — модуля з інтегрованою камерою та Wi-Fi. Такі пристрої можуть працювати автономно, реалізовуючи стрімінг відео у форматі MJPEG або через простий HTTP-сервер. Їхні переваги — низька вартість, компактність, відкрите середовище розробки та мінімальні вимоги до енергоспоживання. З іншого боку, ESP32 має обмежену обчислювальну потужність, що ускладнює реалізацію складної відеоаналітики або шифрування даних без оптимізації.

Також існують промислові IoT-платформи такі як Bosch IoT Suite або Siemens MindSphere, які інтегрують сенсори, відеоаналітику та управління у рамках єдиної екосистеми. Такі рішення орієнтовані на великий бізнес, мають високу вартість, але забезпечують масштабованість, SLA-підтримку (Service Level Agreement) та глибоку інтеграцію зі SCADA (Supervisory Control and Data Acquisition) або ERP-системами (Enterprise Resource Planning). Вони масштабні для локального чи освітнього застосування, однак можуть слугувати орієнтиром з точки зору архітектури.

У результаті аналізу можна зробити висновок, що найбільш оптимальним за співвідношенням функціональність, гнучкість, вартість для розробки навчального або прикладного IoT-рішення є використання ESP32-CAM із можливістю інтеграції у вебзастосунок через Wi-Fi і хмарні сервіси на зразок Firebase. Це дозволяє створити ефективну систему відеоспостереження з мінімальними витратами, відкритим кодом та підтримкою подальшої адаптації під специфіку конкретного застосування.

На рисунку 1.8 зображено приклад мобільного інтерфейсу приватної системи відеоспостереження, яка використовується для охорони житлових комплексів, під’їздів або територій біля будинку.

У верхній частині зображення видно трансляцію з камери, з можливістю збільшення масштабу (2.1x), а також доступ до мікрофона, керування звуком і сітки відображення кількох камер одночасно. Середовище має набір інтерактивних

кнопок для перегляду записів, зйомки скріншотів, запису відео вручну, зміни якості потоку (HD/SD), а також відтворення з архіву. Ключовим елементом є стрічка часу із зазначенням часу активності, який дозволяє переміщуватися по часовій шкалі та вибирати потрібний момент перегляду, що значно полегшує пошук подій.

На відміну від публічних систем, цей мобільний додаток працює з архівом відеозаписів, які автоматично зберігаються в хмарному або локальному сховищі. Система фіксує ключові моменти (наприклад, виявлення руху) й позначає їх на таймлайні, що дає змогу швидко віднайти події без потреби переглядати весь потік вручну. Користувач має можливість здійснювати перегляд відео з точністю до секунд, завантажувати його, робити скріншоти або зберігати відеофрагменти для подальшого аналізу або доказів.

Рисунок 1.8 — Інтерфейс додатку трансляції відео

Попри широкі функціональні можливості, такі мобільні додатки можуть також мати обмеження. Наприклад, у деяких випадках відсутнє розпізнавання облич або аналітика на основі штучного інтелекту (зокрема класифікація подій), і не всі моделі камер підтримують керування кутом огляду чи масштабом

зображення. Крім того, зберігання архівів обмежене за часом або об’ємом пам’яті в залежності від тарифного плану чи типу підключення.

На рисунку 1.9 зображено розділ «Вебкамери» вебресурсу bukovel.com, який ілюструє приклад інформаційного публічного сервісу відеоспостереження, призначеного для туристів та відвідувачів курорту [6]. Його основна функція полягає в наданні доступу до перегляду трансляцій у режимі реального часу з камер, розміщених на ключових ділянках курорту — біля озера, підйомників, туристичних маршрутів. Інтерфейс оформлено у мінімалістичному стилі з акцентом на відеопотік: головна камера займає центральну частину сторінки, праворуч представлено мініатюри з інших камер. Користувач може перемикатись між локаціями вручну, переглядаючи поточну картинку з об’єкта. На зображенні з камери також виводиться інформація про температуру, що корисно для туристів у плануванні відпочинку.

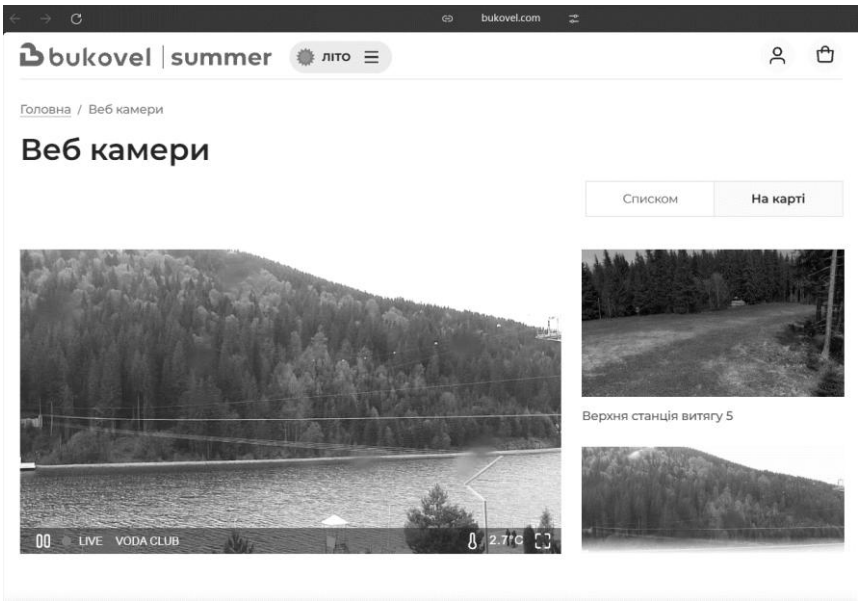

Рисунок 1.9 — Приклад публічного вебінтерфейсу для перегляду відео з камер спостереження

Однак можливості цієї системи досить обмежені. Відео доступне лише в на даний момент часу — переглянути попередні записи неможливо. Відсутній архів відео або інші форми записів, як-от фотофіксація, немає детекції подій (наприклад, великого скупчення людей або зміни погодних умов), і також відсутня можливість

зробити знімок чи завантажити фрагмент потоку. У користувача немає можливості змінювати якість трансляції або управляти камерою, а навігація між камерами здійснюється через перезавантаження сторінки. Також не передбачено особистого кабінету або системи авторизації, через що функціональні можливості не можна персоналізувати. Це робить сервіс переважно оглядовим і статичним — він не дозволяє взаємодіяти з матеріалами або зберігати їх для подальшого використання.

Натомість вебдодаток snih.info, що зображений на рисунку 1.10, забезпечує користувачам доступ до великої кількості камер, розміщених на популярних гірськолижних курортах України: Буковель, Драгобрат, Пилипець, Плай, Тростян, Рахів тощо [7].

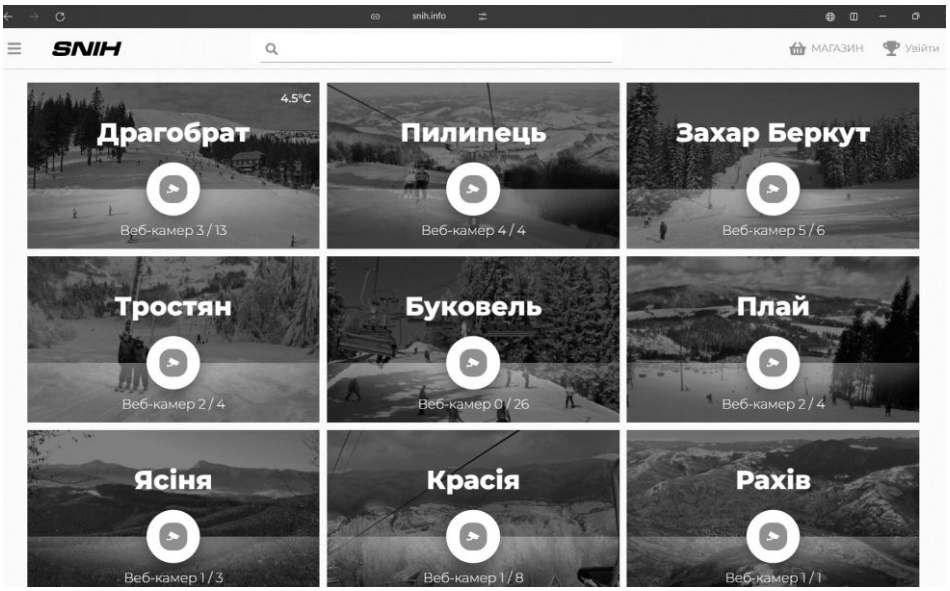

Рисунок 1.10 — Бібліотека відеотрансляцій в реальному часі

Інтерфейс розроблено зручним і візуально привабливим — головна сторінка містить інтерактивні плитки з фотографіями камер, назвою курорту та кількістю доступних трансляцій. Після вибору локації користувач переходить на сторінку з конкретною камерою, що зображено на рисунку 1.11, де доступне актуальне зображення з часовою позначкою та архів фотографій, які автоматично створюються і зберігаються системою з певною періодичністю.

Ці фото є результатом роботи автоматизованого програмного забезпечення, яке робить знімки у встановлений час (наприклад, кожні кілька годин) і зберігає їх

в архів із можливістю перегляду користувачем. Таким чином, система дозволяє відслідковувати погодні умови, зміну ландшафту або сезонну динаміку. Проте тут також відсутнє потокове відео в прямому ефірі, що значно обмежує оперативність реагування на події. Користувач не може переглянути повну відеоісторію, змінити кут огляду, керувати камерою або отримати сповіщення про зміну ситуації. Відсутність гнучких інструментів навігації, зміни масштабу або синхронізації з іншими службами (наприклад, погодними API) також є обмеженням. Як і у випадку з bukovel.com, функціональні можливості не підтримують реєстрацію чи налаштування персонального облікового запису.

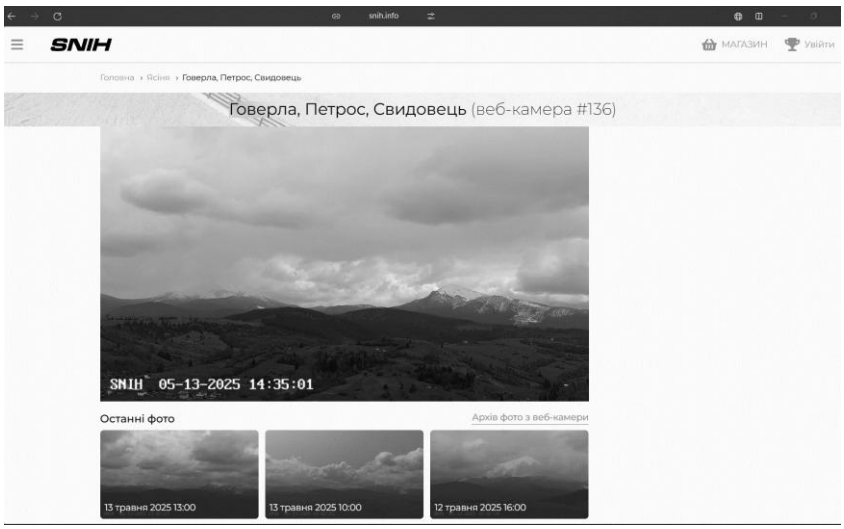

Рисунок 1.11 — Вебінтерфейс перегляду трансляції з камери в реальному часі

Загалом, обидва сервіси демонструють приклади публічного доступу до систем відеоспостереження з акцентом на інформативність та відкритість. Проте обидві системи мають переважно пасивну архітектуру, адже користувач може лише переглядати контент, сформований автоматично або у реальному часі, але не має можливості активно взаємодіяти з системою, налаштовувати параметри, здійснювати пошук по архіву або отримувати аналітику. Це свідчить про обмеженість функціональних можливостей і вказує на потенційні напрями для подальшого розвитку — впровадження відеоархівів, аналітичних модулів, персоналізованого доступу, а також підтримки інтеграції з мобільними додатками або сервісами штучного інтелекту.

1.6 Постановка завдання на розробку вебзастосунку

На основі проведеного аналізу сучасних систем відеоспостереження, технологій передачі відео в реальному часі, апаратно-програмних рішень на базі IoT та вивчення аналогів, можна сформулювати вимоги до розробки вебзастосунку, що є складовою апаратно-програмного комплексу моніторингу навколишнього середовища.

Метою розробки є створення вебзастосунку, який забезпечує зручний і стабільний перегляд відеопотоку з пристрою ESP32-CAM у реальному часі з будь-якого сучасного браузера, а також інтеграцію з хмарною інфраструктурою для обробки та зберігання даних. Під час реалізації системи особлива увага приділяється простоті інтерфейсу, низькій затримці трансляції та можливості масштабування в майбутньому.

У межах поставленого завдання необхідно реалізувати такі функції:

— інтеграція ESP32-CAM з вебінтерфейсом через Wi-Fi;

— створення вебінтерфейсу з вікном відеотрансляції, базовими елементами керування (відображення потоку, запис трансляції, збереження фрагментів трансляції тощо);

— адаптація інтерфейсу під мобільні пристрої та персональні комп’ютери;

— забезпечення безперервної трансляції з мінімальною затримкою.

Розроблений вебзастосунок повинен забезпечити користувачам доступ до відеопотоку в реальному часі без потреби у встановленні додаткового програмного забезпечення. Його архітектура має бути легко масштабованою, з можливістю додавання нових функцій.

Таким чином, постановка завдання охоплює як технічні вимоги до функціонування системи, так і вимоги до зручності її використання кінцевим користувачем, що є необхідною умовою для успішної реалізації апаратно-програмного комплексу моніторингу на базі IoT.

2 ПРОЄКТУВАННЯ ВЕБЗАСТОСУНКУ ІЗ ТРАНСЛЯЦІЄЮ ВІДЕОПОТОКУ

Проєктування вебзастосунку є важливим етапом при створенні апаратно-програмного комплексу для моніторингу стану навколишнього середовища. Основною метою є розробка надійної, адаптивної та масштабованої архітектури вебзастосунку, яка забезпечує збирання, передавання, обробку та візуалізацію відеопотоку в режимі реального часу з можливістю збереження архіву.

2.1 Вибір та обґрунтування апаратного забезпечення

У процесі розробки апаратно-програмного комплексу моніторингу стану навколишнього середовища однією з першочергових задач було обрати надійне, компактне, енергоефективне та достатньо функціональне апаратне забезпечення, яке відповідатиме вимогам до відеотрансляції в реальному часі, зберігання відеозаписів, віддаленого доступу та можливості масштабування.

2.1.1 Аналіз мікроконтролерів для IoT-системи з трансляцією відео

Для побудови IoT-системи з трансляцією відео можна виділити чотири найпоширеніших мікроконтролери — ESP8266, ESP32, WINC1500, а також ATmega32u4.

Мікроконтролер ESP8266 є одним із найпопулярніших бюджетних рішень для бездротових IoT-систем. Він обладнаний однокристальним процесором Tensilica L106 із тактовою частотою до 160 МГц та приблизно 80 КБ оперативної пам’яті. Контролер підтримує Wi-Fi 2,4 ГГц, однак не має вбудованого Bluetooth і USB-порту — для програмування необхідне підключення через зовнішній перетворювач USB-UART. Його ключовою перевагою є низька вартість і простота використання, проте обмежений обсяг пам’яті та відсутність периферійних інтерфейсів роблять його непридатним для ресурсомістких задач, такі як передавання відео в реальному часі.

Мікроконтролер ESP32, на відміну від ESP8266, має двоядерний процесор Tensilica LX6 із тактовою частотою до 240 МГц, значно більший обсяг оперативної

пам’яті — 520 КБ, а також можливість підключення додаткової PSRAM (до 4 МБ), що є критично важливим для відеопотоків. Контролер підтримує Wi-Fi та Bluetooth (BLE), що розширює можливості бездротової взаємодії. Попри те, що USB-порт також відсутній, існують модифікації плат із microUSB. ESP32 є оптимальним вибором для задач відеоспостереження завдяки продуктивності, гнучкості та великій кількості доступних бібліотек і прикладів.

WINC1500 — це не самостійний мікроконтролер, а Wi-Fi-модуль, який найчастіше поєднується з іншими MCU, наприклад, SAM D21 (на базі ARM Cortex-M0+). Його використання дозволяє додати бездротовий зв’язок до традиційних Arduino-плат. Модуль має обмежену продуктивність (до 48 МГц, до 32 КБ ОЗП), не підтримує Bluetooth, але підтримує USB-комунікацію через додаткові чипи. Він підходить для простих задач передавання даних, але через низьку обчислювальну потужність та залежність від зовнішнього мікроконтролера не є оптимальним для відеоаналізу.

32u4 FONA — це поєднання класичного мікроконтролера ATmega32u4 з GSM-модулем FONA (від Adafruit), який дозволяє реалізувати мобільний зв’язок через SIM-карту, включно з надсиланням SMS, здійсненням дзвінків або GPRS-з’єднанням. Контролер має мінімальний обсяг пам’яті (2,5 КБ SRAM) та низьку тактову частоту (16 МГц), однак підтримує USB-з’єднання без потреби в зовнішніх адаптерах. Завдяки GSM-модулю цей варіант доцільний у місцях, де відсутнє Wi-Fi-покриття, але через відсутність підтримки відео, обмежену пам’ять і високу ціну він не підходить для систем трансляції відео.

В таблиці 2.1 наведено результати порівняння характеристик та можливостей кожного мікроконтролера. Серед розглянутих варіантів ESP32 є найоптимальнішим вибором для побудови IoT-системи відеоспостереження. Він забезпечує поєднання продуктивності, пам’яті, функціональності бездротових інтерфейсів і низької вартості. Його переваги роблять можливим захоплення, передачу та обробку відеопотоку в реальному часі, що критично важливо для ефективної роботи системи.

Таблиця 2.1 — Порівняльна характеристика мікроконтролерів для побудови IoT-системи відеоспостереження

Характеристика	ESP8266	ESP32	WINC1500	32u4 FONA
Процесор	Tensilica L106 (1 ядро)	Tensilica LX6 (2 ядра)	ARM Cortex-M0+	ATmega32u4
Тактова частота	80–160 МГц	до 240 МГц	до 48 МГц	16 МГц
ОЗП / SRAM	Близько 80 КБ	520 КБ SRAM + PSRAM (опц.)	до 32 КБ	2,5 КБ SRAM
Wi-Fi	Так (2,4 ГГц)	Так (2,4 ГГц)	Так (через модуль)	Ні
Bluetooth	Ні	Так (BLE)	Ні	Ні
Наявність USB	Ні (треба адаптер)	Ні (треба адаптер або microUSB)	Так (через USB чип)	Так (вбудований USB)
Живлення	3,3 В	3,3 В	3,3 В	5 В
Ціна (орієнтовно)	Близько 2–4 USD	Близько 4–6 USD	Близько 10–15 USD	Близько 20–25 USD
Особливості	Низька вартість, мінімальний набір функцій, простота	Два ядра, підтримка Bluetooth, PSRAM, розширена периферія	Wi-Fi модуль, сумісність з Arduino, використовуєтьс я з окремим MCU	GSM модуль FONA для мобільного зв’язку (SMS, дзвінки, GPRS)

2.1.2 Аналіз та вибір адаптерів USB-UART (FTDI/CH340) для живлення та прошивки

Для налаштування, прошивки та налагодження плати ESP32-CAM необхідно використовувати перетворювач, що дозволяє перетворити USB-сигнал у послідовний UART-інтерфейс. Серед найбільш поширених варіантів – перетворювач на базі мікросхем FTDI FT232RL, CH340G та CP2102, кожен з яких має свої переваги та недоліки, що наведено в таблиці 2.2.

FTDI FT232RL є лідером серед подібних перетворювачів. Він забезпечує високу швидкість передачі даних (до 3 Мбіт/с), сумісний з усіма сучасними операційними системами, має офіційні драйвери, що автоматично встановлюються, і відзначається стабільною роботою. Додатковою перевагою є підтримка перемикання напруги між 3,3 В та 5 В, що дає гнучкість у використанні з різними мікроконтролерами. Основним недоліком є вища вартість у порівнянні з іншими рішеннями (орієнтовно 5–7 USD).

Таблиця 2.2 — Порівняльна характеристика перетворювачів для побудови IoT-системи відеоспостереження

Характеристика	FTDI FT232RL	CH340G	CP2102
Мікросхема	FTDI	WCH	Silicon Labs
Сумісність із ОС	Windows, Linux, macOS	Windows, Linux, macOS	Windows, Linux, macOS
Максимальна швидкість передачі	3 Мбіт/с	до 2 Мбіт/с	1 Мбіт/с
Наявність драйверів	Офіційні драйвери, легко встановлюються	Потрібно встановити вручну, але доступні	Офіційні драйвери, стабільні
Стабільність з’єднання	Висока	Середня	Висока
Підтримка 3,3В/5В	Так (перемикач або джампер)	Так (зазвичай 3,3В за замовчуванням)	Так (іноді автоматично)
Ціна	Орієнтовно 5–7 USD	Орієнтовно 1–2 USD	Орієнтовно 3–5 USD
Особливості	Стабільність, надійність, висока якість	Дешевий, масово використовується в китайських модулях	Компактний, зручний, хороша альтернатива FTDI

CH340G – бюджетна альтернатива від китайського виробника WCH. Він має хорошу базову функціональність, але дещо нижчу стабільність і потребує самостійного встановлення програмного забезпечення (драйверів) на деяких системах, особливо Windows. Незважаючи на це, CH340G залишається надзвичайно популярним завдяки своїй низькій ціні (близько 1–2 USD) і доступності у готових модулях.

CP2102 від Silicon Labs — це надійний адаптер із хорошою підтримкою операційних систем, компактним розміром і стабільними офіційними драйверами. Його максимальна швидкість передачі дещо нижча, ніж у FTDI (орієнтовно 1 Мбіт/с), проте вона є цілком достатньою для прошивки ESP32-CAM. CP2102 має оптимальне співвідношення між ціною, надійністю та зручністю.

Для стабільної та безпечної роботи з ESP32-CAM на етапі розробки оптимальним варіантом для застосування є CP2102, як перетворювач з підтвердженою якістю та офіційною підтримкою.

2.2 Обгрунтування вибору програмного рішення для прийому, обробки та трансляції відеопотоку

Одним із основних рішень, що використовується для побудови вебзастосунку з трансляцією відео є серверна інфраструктура, яка виконує функції прийому відеопотоку з пристрою ESP32-CAM, його обробки, збереження архіву та роздачі даних кінцевим користувачам через вебінтерфейс або ж не серверне рішення, яке буде забезпечувати передавання відеопотку через тунельне рішення. Розглянемо чотири основні варіанти реалізації данної частини, такі як, Firebase Functions в поєднанні з Storage, Node.js на VPS, Docker-сервер на власному VPS та тунельне рішення ngrok. У таблиці 2.3 наведено порівняння чотирьох різних варіантів реалізації, які можуть бути використані у вебзастосунку з передаванням відео в реальному часі.

Firebase Functions у поєднанні з Firebase Storage є безсерверним хмарним рішенням, що не вимагає управління інфраструктурою та ідеально підходить для швидкого розгортання MVP чи невеликих проєктів. Його перевагами є простота налаштування, автоматичне масштабування та низькі витрати в межах безкоштовних планів користування. Однак система має обмежену підтримку WebRTC, що ускладнює реалізацію потокової трансляції в реальному часі, і може вимагати додаткові сервіси для забезпечення повноцінної взаємодії з відеопотоком.

Node.js, розгорнутий на VPS-сервері (наприклад, через платформи Render або Fly.io), надає гнучкість. Такий підхід дозволяє реалізувати повністю індивідуально налаштовано систему логіки обробки відео, інтегрувати бібліотеки для WebRTC, налаштувати авторизацію та базу даних відповідно до потреб проєкту. Вартість такого рішення є вищою за Firebase, однак вона залишається в межах прийнятного для невеликих або середніх систем. Налаштування серверу вимагає базових знань з адміністрування, однак у результаті забезпечує значно більший контроль над процесами.

Таблиця 2.3 — Порівняльна характеристика серверних рішень для побудови IoT-системи відеоспостереження

Критерій	Firebase Functions + Storage	Node.js на VPS	Docker-сервер на власному VPS	ESP32-CAM + ngrok (безсерверне)
Тип розгортання	Хмарне (безсерверне)	Хмарне або локальне	Локальне / хмарне (контейнеризоване)	Безпосередній тунель (ngrok)
Продуктив-ність	Середня	Висока	Висока / дуже висока	Залежить від ESP32 (середня)
Масштабо-ваність	Висока (автоматично)	Середня (залежить від плану)	Висока (через оркестратори)	Обмежена (один потік)
Складність налашту-вання	Низька (спрощене налаштування)	Середня (потрібно конфігурувати вручну)	Висока (вимагає досвіду DevOps)	Низька
Можливість зберігання архіву	Так (Storage)	Так (через диск або S3)	Так (локально або в S3)	Частково (через frontend)
Вартість	Низька / безкоштовна в межах квот	Середня (близько 5–10 USD/міс)	Вища (VPS + адміністрування)	Безкоштовно / низька (ngrok)
Інтеграція з WebRTC	Обмежена (потрібні додаткові сервіси)	Так (через бібліотеки WebRTC)	Так (можливість власної реалізації)	Ні (MJPEG, можлива реалізація вручну)
Підтримка багатьох клієнтів	Середня (через CDN та Storage)	Висока (можна масштабувати вручну)	Висока (контроль над усім стеком)	Обмежена (1–2 користувачі стабільно)
Потреба в адмініструванні	Мінімальна	Потрібна базова підтримка	Висока (потрібен постійний супровід)	Мінімальна
Оптимальне використання	Швидкий старт, MVP, прототипи	Гнучка система, невеликі проєкти	Корпоративні рішення, великі системи	Навчальні проєкти, демо, MVP

Ще одним варіантом є розгортання інфраструктури за допомогою Docker-контейнерів на власному VPS. Такий підхід забезпечує повну автономність, масштабованість (завдяки оркестраторам типу Kubernetes) та найвищу продуктивність. Його доцільно використовувати для розгортання великих або корпоративних систем з багатьма потоками, високими вимогами до безпеки, доступності та обробки даних у реальному часі. Недоліками такого підходу є складність налаштування, висока потреба в супроводі та загальна вартість підтримки такого рішення.

Передавання відеопотоку через тунельне рішення є ефективним методом забезпечення доступу до потокових даних із пристрою, розміщеного у локальній мережі, з глобального Інтернету без необхідності у наявності публічної IP-адреси чи ручного конфігурування мережевого обладнання.

Тунель — це спеціалізований сервіс, що створює захищене з'єднання між локальним сервером (на якому розгорнуто потокову трансляцію з ESP32-CAM) і публічно доступним хостом. Таким чином, дані з пристрою можуть бути отримані кінцевим користувачем у браузері через автоматично згенероване публічне посилання, яке переспрямовується на локальний порт.

Використання мікроконтролера ESP32-CAM у поєднанні з ngrok є простим і доступним способом забезпечення потокової трансляції відео з пристрою без необхідності налаштування складної серверної інфраструктури. Завдяки сервісу ngrok, який створює безпечний тунель до локального вебінтерфейсу ESP32-CAM, можна швидко отримати публічне посилання для перегляду стріму з будь-якої точки світу. Такий підхід особливо зручний для прототипів, демонстрацій або індивідуальних проєктів. До переваг належать легкість налаштування, відсутність потреби в відкритті портів чи використанні фіксованої IP-адреси, а також можливість обходу CORS-конфліктів через проксі. Проте у безкоштовному плані ngrok діють обмеження на тривалість сесій та кількість підключень, що може впливати на стабільність тривалих трансляцій.

На основі проведеного порівняння можна зробити висновок, що використання публічного тунелю ngrok для прямого доступу до ESP32-CAM ззовні є найоптимальнішим рішенням для реалізації трансляції відеопотоку в межах роботи. На відміну від варіантів, що передбачають повноцінну серверну обробку відео, наприклад, через Node.js або Firebase Functions, обраний підхід зосереджується на простоті, мінімальній затримці та відсутності потреби в окремому сервері для обробки відеопотоку. Замість маршрутизації відео через сервер, трансляція здійснюється безпосередньо з ESP32-CAM за допомогою MJPEG-потоку, який транслюється через публічну адресу, надану ngrok.

Для реалізації розробки було обрано сучасний стек технологій, що забезпечує ефективну взаємодію між апаратною та програмною частинами системи. Мікроконтролер ESP32-CAM з вбудованою камерою та підтримкою Wi-Fi використаємо як модуль збору відеоданих, що дозволяє здійснювати бездротову трансляцію відео. Для забезпечення доступу до локального відеопотоку ESP32-CAM із будь-якої точки світу використовується сервіс ngrok, який створює безпечний тунель до локального вебінтерфейсу пристрою. Це дозволяє легко транслювати відео в режимі реального часу без потреби в складних налаштуваннях мережі чи відкритті портів. Відеопотік передається у браузер користувача за допомогою протоколів MJPEG, що забезпечує мінімальну затримку та не потребує встановлення додаткового програмного забезпечення.

2.3 Обгрунтування вибору стеку технологій для розробки вебзастосунку

У процесі проєктування апаратно-програмного комплексу з трансляцією відеопотоку в реальному часі ключову роль відіграє вибір технологій, які забезпечують стабільну роботу системи, її масштабованість, інтерактивність та простоту в підтримці. Архітектура системи базується на IoT-підході, де пристрій збору даних (ESP32-CAM) взаємодіє з вебінтерфейсом через мережу за допомогою сучасних вебтехнологій.

З боку мікроконтролера було обрано платформу Arduino, яка забезпечує зручне середовище розробки для ESP32-CAM. Arduino IDE дозволяє швидко реалізувати базову прошивку з підтримкою HTTP, WebSocket або WebRTC-серверів, передавання відеопотоку у форматі MJPEG, а також роботу з картою microSD. Arduino має відкриті бібліотеки, детальну документацію та підтримку модуля OV2640, що робить її оптимальним рішенням для швидкої розробки.

У процесі проєктування клієнтського інтерфейсу для вебзастосунку з трансляцією відеопотоку було розглянуто чотири основні підходи — чистий HTML/CSS/TypeScript, React, Vue та Angular. Кожен із варіантів має свої переваги, недоліки та специфіку використання, які суттєво впливають на стабільність, масштабованість та ефективність розробки.

Базовим підходом є реалізція вебзастосунку на основі технологій, таких як, HTML/CSS/TS, що передбачає створення інтерфейсу без застосування фреймворків. Такий варіант надає прямий контроль над структурою DOM, мінімальне споживання ресурсів та високу продуктивність, однак при цьому розробник самостійно реалізує керування станом, маршрутизацію, повторне використання компонентів та обробку взаємодії з WebRTC. Такий підхід практичний для невеликих прототипів, однак він суттєво обмежує масштабованість проєкту.

Однією з найпотужніших мов, що активно використовується у створенні клієнтських частин, є TypeScript — строго типізована мова програмування, яка базується на JavaScript. Мова TypeScript забезпечує можливість створення масштабованих, структурованих та безпечних у підтримці застосунків, особливо у випадку з багатофункціональними вебінтерфейсами. Вона забезпечує статичну перевірку типів, автодоповнення, інтеграцію зі складними API, наприклад, MediaRecorder, WebRTC, Canvas, і значно знижує ризики помилок на етапі виконання [8].

Високорівневий фреймворк Angular використовується для розробки клієнтських частин вебзастосунків, створений та підтримується компанією Google. Його архітектура побудована за архітектурним шаблоном Model–View–Controller (MVC) із чітко регламентованим розподілом функціональних модулів, що сприяє масштабованості та підтримуваності проєктів [9]. Angular повністю реалізований на мові TypeScript, що забезпечує типізованість, статичну перевірку коду та інтеграцію з сучасними засобами розробки. До складу фреймворку входять вбудовані модулі для маршрутизації, реактивного управління формами, здійснення HTTP-запитів, обробки подій та використання шаблонів залежностей (Dependency Injection). На рисунку 1.4 наведено приклад структури додатку з використання фреймворку Angular. Схема ілюструє ключові архітектурні елементи Angular-застосунку та взаємодію між ними. Основу складають модулі (Module), які об’єднують компоненти, сервіси й зовнішні залежності в логічні блоки. Компоненти (Component) є центральною одиницею користувацького інтерфейсу та

поєднують шаблон (Template), логіку і метадані, що описують поведінку елемента. Шаблони включають HTML-розмітку з директивами Angular, які динамічно керують відображенням вмісту. Зв’язок між компонентом і шаблоном здійснюється через механізм прив’язки властивостей (Property Binding) для передачі даних та прив’язки подій (Event Binding) для реакції на дії користувача. Сервіси (Service) використовуються для спільного доступу до логіки або даних і підключаються до компонентів через інжектор залежностей (Injector). Цей механізм реалізує шаблон Dependency Injection, забезпечуючи низький рівень зв’язаності та гнучке повторне використання коду. Крім того, у структурі можуть використовуватись директиви (Directive), які впливають на поведінку елементів DOM [10].

Крім того, Angular активно застосовує бібліотеку RxJS для реалізації реактивного програмування, що дозволяє ефективно обробляти асинхронні потоки даних, зокрема в задачах потокової трансляції відео у реальному часі. У вебзастосунках для трансляції відеопотоку Angular дозволяє структурувати інтерфейс як набір окремих модулів: стрімінг, архів, нотифікації, налаштування, аналітика, і підтримує високий рівень масштабованості та розмежування коду [10].

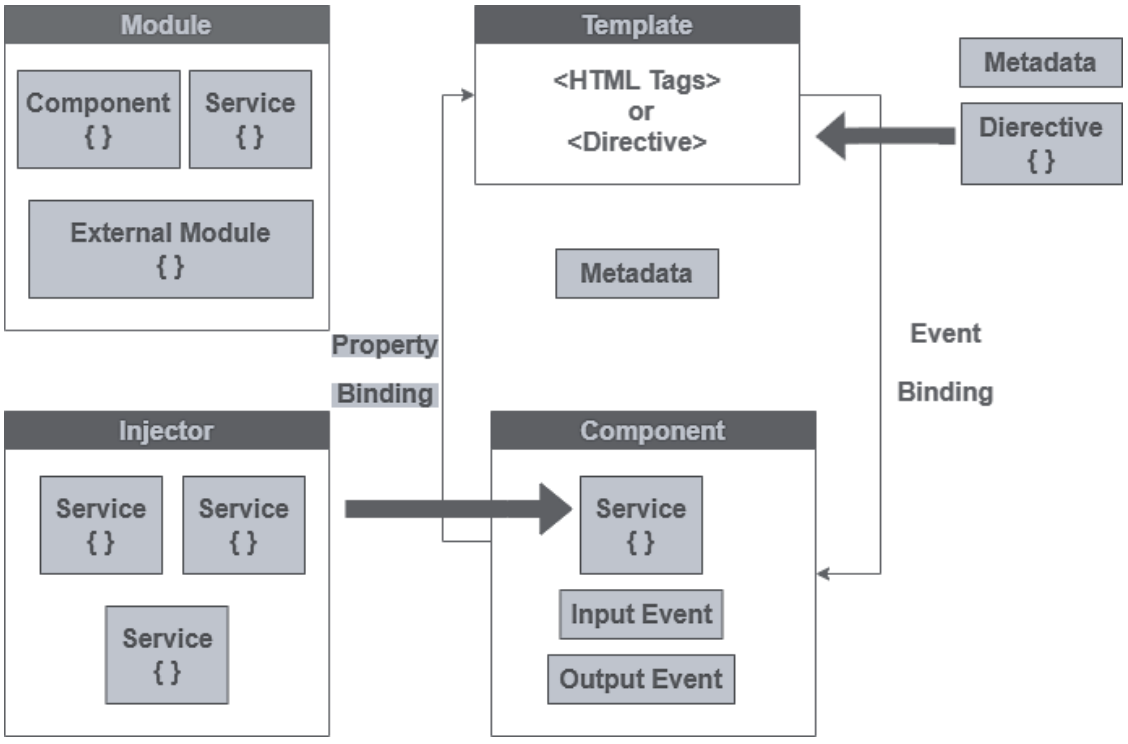

Рисунок 2.1 — Структура вебдодатку на Angular

Бібліотека React, розроблена компанією Meta (Facebook), застосовується для побудови інтерфейсів користувача, яка фокусується на компонентно-орієнтованому підході та декларативному описі інтерфейсів. React реалізує односпрямований потік даних (one-way data binding), що сприяє передбачуваності станів застосунку та полегшує налагодження. Основною відмінністю є використання JSX синтаксичного розширення, яке дозволяє поєднувати HTML-подібні розмітки з логікою на JavaScript в межах одного компонента. На рисунку 2.2 наведено приклад структури додатку написаного з використання фреймворку React, де можна зазначити, що обсяг файлів значно менший за рахунок поєднання HTML розмітки з логікою на JavaScript. Зображена структура проєкту демонструє організацію коду в типовому React застосунку, що орієнтований на модульність, масштабованість та зручність підтримки. Уся логіка міститься в кореневій директорії src/, яка є стандартним входом до вихідного коду [10].

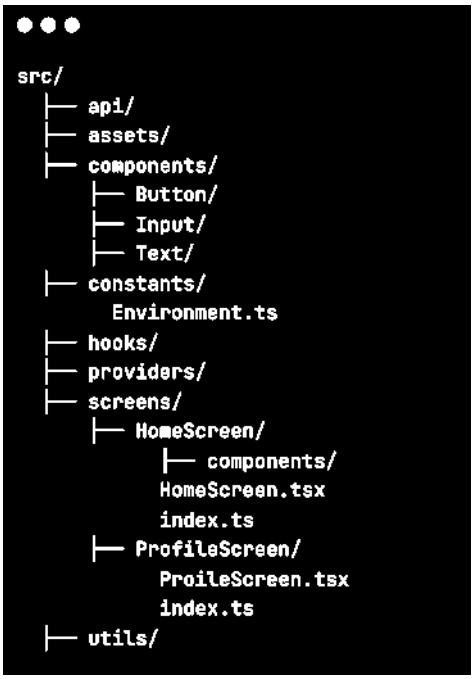

Рисунок 2.2 — Структура вебдодатку на React

Окрема директорія api/ призначена для зберігання модулів, що відповідають за взаємодію з API — сюди зазвичай поміщають функції для HTTP-запитів або конфігурації клієнтів. Папка assets/ містить усі статичні ресурси — зображення, шрифти, іконки, тобто те, що використовується в інтерфейсі, але не є логікою

програми.

Папка components/ організована за типами багаторазових інтерфейсних елементів — кнопок (Button), полів вводу (Input), текстових блоків (Text). Це базові елементи, які можна перевикористовувати у різних частинах застосунку. Директорія constants/, де зберігаються незмінні значення на рівні проєкту. Наприклад, файл Environment.ts часто містить змінні середовища, API-ключі, базові шляхи тощо.

У hooks/ зберігаються власні React-хуки, які реалізують повторювану логіку, наприклад, авторизацію або тему оформлення. У providers/ містяться контексти (Context API), що охоплюють додаток і надають доступ до спільних даних на глобальному рівні (тематика, мова, аутентифікація).

У папці screens/, у якій зберігаються окремі сторінки або екрани застосунку. Кожна сторінка, наприклад HomeScreen або ProfileScreen, має власну папку, всередині якої є файл з основною логікою (HomeScreen.tsx, ProfileScreen.tsx), вкладені компоненти та index.ts для зручного експорту. Такий підхід ізоляції сторінок дозволяє уникати надмірної зв’язаності між модулями.

Також, utils/ містить утилітарні функції, які не залежать від основної логіки, але часто використовуються в різних частинах програми — форматування дат, перевірка email, обробка рядків.

Хоча React сам по собі є легковаговим інструментом, для побудови повноцінного клієнтського застосунку необхідне використання додаткових бібліотек, таких як React Router (маршрутизація), Axios (HTTP-запити), Redux або Zustand (керування станом). Підтримка TypeScript у React є опційною, однак вона широко використовується у проєктах середнього та великого масштабу для підвищення надійності коду.

Vue.js є прогресивним JavaScript-фреймворком, що забезпечує можливість поступової інтеграції у застосунок, починаючи з окремих компонентів до повноцінного SPA-рішення (Single Page Application). Фреймворк поєднує концепції реактивного програмування та декларативного опису інтерфейсу, що забезпечує високу ефективність при обробці динамічних даних. На рисунку 2.3 наведено

приклад структури додатку написаного з використання фреймворку Vue. Усі основні файли та директорії розміщено в кореневій папці, зокрема директорія src/, яка містить вихідний код застосунку. Усередині неї директорія assets/ відповідає за зберігання ресурсів, таких як зображення, стилі CSS, шрифти тощо. Папка components/ включає багаторазові компоненти — це одиничні .vue файли, які можуть бути використані у різних частинах застосунку. Директорія router/ містить конфігурації маршрутів, які визначають логіку переходів між сторінками (наприклад, через Vue Router) [9].

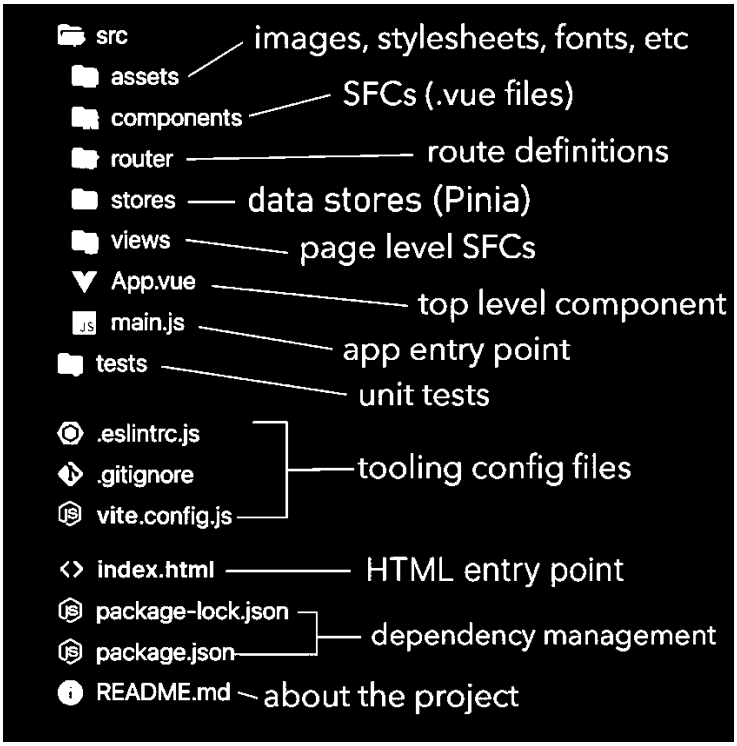

Рисунок 2.3 — Структура вебдодатку на Vue

Папка stores/, яка використовується для централізованого управління станом застосунку за допомогою Pinia — офіційної state management бібліотеки для Vue. Далі, папка views/ містить компоненти, які відповідають цілим сторінкам або екранам у SPA (Single Page Application), тобто відображають контент відповідно до маршруту.

Файл App.vue виступає як кореневий компонент, навколо якого будується вся структура застосунку. main.js є точкою входу — саме з нього ініціалізується Vue-застосунок та підключаються маршрути, стани, глобальні компоненти. Для

забезпечення якості коду використовується папка tests/, де зберігаються модулі юніт-тестування.

Позасистемні файли включають конфігурації для інструментів розробки: .eslintrc.js для аналіз коду на предмет помилок, порушень стилю та потенційних багів, .gitignore для визначення файлів, які не потрапляють до репозиторію, та vite.config.js, який конфігурує Vite, сучасний збирач модулів для Vue.

Файл index.html — це HTML-шаблон, у який буде інтегровано застосунок. Файли package.json та package-lock.json управляють залежностями проєкту та описують конфігурацію застосунку, а README.md містить загальну інформацію, опис і інструкції з використання або розгортання.

Синтаксис Vue максимально наближений до класичного HTML, JS, CSS, що полегшує навчання та швидке прототипування. Починаючи з версії Vue 3, забезпечено повноцінну та стабільну інтеграцію з TypeScript, а також впроваджено Composition API, який полегшує повторне використання логіки між компонентами. Офіційна екосистема включає Vue Router (маршрутизація) та Vuex (сховище стану), що дозволяє з мінімальними залежностями створювати гнучкі та продуктивні вебінтерфейси, придатні для інтеграції в IoT-рішення, зокрема для візуалізації потокових відеоданих.

У таблиці 2.4 зображено короткий порівняльний аналіз технологій для побудови клієнтської частини вебзастосунку. Усі ці фреймворки підтримують побудову структури інтерфейсу користувача, маршрутизацію, роботу з WebRTC, WebSocket, REST API та мають документацію. З огляду на вбудовану архітектуру MVC, типізацію за допомогою TypeScript, модульність і високий рівень безпеки завдяки шаблонізованому HTML та строгому контролю залежностей, Angular обрано як найдоцільнищий варіант для побудови вебзастосунку. Angular також забезпечує ефективну роботу зі станом додатку, реалізацію адаптивного інтерфейсу, можливість масштабування, що є важливим для подальшого розширення функціоналу, наприклад, підтримки архіву записів, повідомлень, керування декількома камерами.

Таблиця 2.4 — Порівняльна характеристика технологій для побудови клієнтської частини IoT-системи відеоспостереження

Критерій	HTML/CSS/TS (Vanilla)	React	Vue	Angular
Підхід до структури	Без фреймворку (ручне керування станом)	Компонентний підхід	Компонентний підхід	MVC + компоненти
Мова програмування	HTML, CSS, TypeScript	TypeScript	TypeScript (або JavaScript)	TypeScript
Типізація	Обмежена (ручна через TS)	Так (через TS)	Так (опціонально)	Так (вбудована)
Підтримка SPA (Single Page Application)	Обмежена (потрібно реалізовувати вручну)	Так	Так	Так (повна підтримка)
Інтеграція з WebRTC	Так (низькорівневий доступ)	Так	Так	Так
Крива навчання	Низька	Середня	Низька	Висока (але структуровано)
Ком’юніті та підтримка	Обмежене, переважно індивідуальні рішення	Велика спільнота, активна підтримка	Середня, зростає	Дуже велика, офіційна підтримка Google
Продуктив-ність	Висока (без накладних витрат)	Висока	Висока	Висока
Масштабо-ваність	Низька	Середня	Середня	Висока
Застосування у промислових системах	Рідко використовується, прототипи	Часто використовуєтьс я у вебзастосунках	Підходить для середніх систем	Широко використовуєтьс я у промислових та корпоративних рішеннях

Як хмарну платформу для розміщення вебдодатку було обрано Firebase. Вона дозволяє хостити клієнтську частину (Firebase Hosting), використовувати безсерверні функції (Firebase Functions), зберігати відеоархіви або метадані (Firebase Storage та Firestore), реалізовувати автентифікацію (Firebase Auth), що робить її комплексним рішенням для систем, де важливі простота розгортання, масштабування та відсутність потреби у підтримці фізичного сервера [11].

Отже, стек технологій Arduino, ngrok, Angular, Firebase було обрано як найбільш збалансоване рішення, яке забезпечує ефективну взаємодію між апаратною частиною та користувацьким інтерфейсом, мінімізує затримки передачі

відео, підтримує сучасні стандарти безпеки та дозволяє легко масштабувати систему.

2.4 Проєктування структурної схеми вебдодатку для трансляції відеопотоку

Структурне проєктування вебдодатку є одним із ключових етапів створення надійної системи відеоспостереження, оскільки саме від правильного розподілу ролей між компонентами залежить стабільність роботи, масштабованість та зручність використання [12]. У межах розробленого рішення вебзастосунок виконує роль клієнтської точки доступу до потокового відео та інформації з сенсорів, забезпечуючи інтерфейс перегляду MJPEG-трансляції, доступу до екологічних параметрів, оновлення даних у реальному часі та взаємодії з кінцевим користувачем.

На структурній схемі, поданій на рисунку В.1, зображено структурну схему вебзастосунку, призначену для організації трансляції відеопотоку з пристрою ESP32-CAM до вебінтерфейсу кінцевого користувача.

Основу IoT-системи становить мікроконтролер ESP32-CAM, який генерує відеопотік у форматі MJPEG. Відеодані транслюються через вбудований HTTP-сервер мікроконтролера. Оскільки ESP32-CAM зазвичай працює в межах локальної мережі та не має публічної IP-адреси, для забезпечення доступу до потоку ззовні використовується тунельне рішення ngrok, що виконує функцію захищеного тунелювання локального ресурсу у глобальну мережу.

Ngrok динамічно створює публічний HTTPS-URL, який є проксі-адресою до локального MJPEG-потоку. Таким чином, клієнтська частина вебзастосунку отримує можливість доступу до відеоданих незалежно від мережного розташування користувача.

Клієнтська частина, що представлена у вигляді односторінкового вебінтерфейсу, здійснює відображення відеопотоку в реальному часі, а також ініціює запити до REST API для отримання екологічних показників, таких як, температура, тиск, освітленість та рівень CO_2. Вебінтерфейс адаптовано для роботи з різними типами пристроїв, включаючи персональні комп’ютери та мобільні

пристрої.

Використання Firebase Hosting забезпечує централізоване розгортання клієнтського інтерфейсу з підтримкою HTTPS, адаптивною маршрутизацією та високою доступністю. Уся логіка обробки відеопотоку виконується на стороні клієнта, а Firebase виступає надійною платформою для доставки інтерфейсу користувачу.

Таким чином, описана архітектура поєднує переваги вбудованих IoT-пристроїв, хмарних інструментів і технологій клієнтської розробки для забезпечення стабільної, масштабованої та доступної візуалізації відеоданих у режимі реального часу.

Таким чином, структурна схема охоплює три ключові елементи:

— інтерфейс вебзастосунку (Angular), що відповідає за взаємодію з користувачем, трансляцію відеопотку і виведення показників;

— ESP32-CAM, що слугує джерелом MJPEG-потоку та API-сенсорних даних;

— Ngrok, який здійснює тунелювання локального API і стріму в публічний інтернет;

— Firebase Hosting для хостингу клієнтської частини.

Така структура дозволяє у майбутньому безперешкодно інтегрувати нові модулі, наприклад, архів відео, аналітику, авторизацію, без зміни основної архітектури.

2.5 Розробка мапи вебзастосунку для трансляції відеопотоку

Мапа вебзастосунку, що зображена на рисунку 2.4, надає візуальну логічну структуру навігації між основними функціональними сторінками та компонентами користувацького інтерфейсу. Застосунок реалізовано у вигляді односторінкового вебінтерфейсу (SPA – Single Page Application), де навігація відбувається без повного перезавантаження сторінки, що забезпечує кращу продуктивність та зручність використання [13]. Основу інтерфейсу становлять три головні розділи: «Головна сторінка», «Трансляція» та «Показники». Головна сторінка виконує функцію презентаційного блоку, у якому користувач отримує загальну інформацію

про призначення системи та має змогу здійснити перехід до розділів «Трансляція» та «Показники».

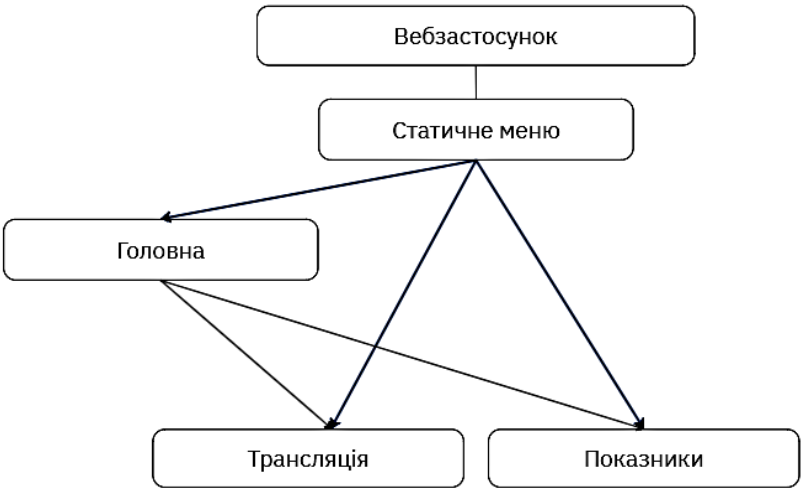

Рисунок 2.4 — Мапа вебзастосунку для трансляції відеопотоку

Розділ трансляції забезпечує перегляд відеопотоку в режимі реального часу, надає елементи керування записом, а також має вбудований блок для перегляду екологічних показників середовища, що отримуються з сенсорів IoT пристроїв, що входять до складу апаратно-програмного комплексу, та оновлюються у реальному часі.

Сторінка показників середовища виводить окремо усі поточні дані з сенсорів, із текстовими поясненнями щодо кожного параметра (температура, тиск, освітленість, CO_2), та індикатором поточного стану навколишнього середовища.

Наявність статичного навігаційного меню дозволяє здійснювати швидкий доступ до всіх сторінок незалежно від поточного стану інтерфейсу.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ

Програмна реалізація є ключовим етапом створення апаратно-програмного комплексу для моніторингу оточуючого середовища на базі мікроконтролера ESP32-CAM. Основна мета цього етапу — забезпечити ефективну та зручну для користувача взаємодію з IoT-системою у режимі реального часу через вебінтерфейс. Розроблений вебзастосунок повинен виконувати функції отримання відеопотоку, виводу трансляції на клієнтський пристрій, управління записами та виводу дані показників із сенсорів апаратно-програмного комплексу.

Застосунок повинен враховувати сучасні вимоги до безпеки, адаптивності, масштабованості та продуктивності. Клієнтську частину вебзастосунку реалізовано з використанням Angular фреймворку для створення інтерфейсів, що побудований на мові програмування TypeScript. Завдяки цьому забезпечено модульну структуру проєкту, зручну маршрутизацію між сторінками, підтримку реактивних форм і високий рівень типізації, що сприяє покращенню якості та підтримуваності коду. Інтерфейс вебзастосунку повинен забезпечити дотримання принципів UX/UI-дизайну, що дозволить досягти інтуїтивності, візуальної простоти та зручності використання як на персональних комп’ютерах, так і на мобільних пристроях.

3.1. Реалізація прошивки ESP32-CAM

Для забезпечення передачі відеопотоку в режимі реального часу з модуля ESP32-CAM було використано спеціалізовану прошивку, яка дозволяє транслювати відео у форматі MJPEG через протокол HTTP. Такий підхід є сумісним із веббраузерами та дозволяє безпосередньо переглядати відео з камери, не використовуючи додаткові плагіни чи конвертери.

Прошивка взята з електронного ресурсу з вільним доступом і дозволяє :

— здійснювати відеотрансляцію у браузер через MJPEG-потік;

— записувати відео у форматі AVI на SD-карту;

— запускати RTSP-сервер для передачі відео в зовнішні клієнти;

— підтримувати керування пристроєм через MQTT, HTTP API, WebSocket;

— інтегрувати функціональність руху, серво-механізмів, мікрофону, датчиків та інше [14] .

Перед завантаженням прошивки у модуль, ESP32-CAM необхідно фізично під’єднати до комп’ютера за допомогою перетворювача USB-UART [15]. Для цього виконується з’єднання контактів, як зображено на рисунку 3.1, GND, 5V, TX, RX, а також перемичка IO0-GND, що переводить пристрій у режим завантаження коду.

У середовищі Arduino IDE після вибору відповідного порту послідовного з’єднання ініціюється процес компіляції, після чого починається запис прошивки безпосередньо у флеш-пам’ять пристрою.

У вікні монітору, яке зображено на рисунку 3.2, послідовного порту відображається покадрова інформація про хід запису, зокрема адреси, на які здійснюється запис, відсотковий прогрес і швидкість передавання.

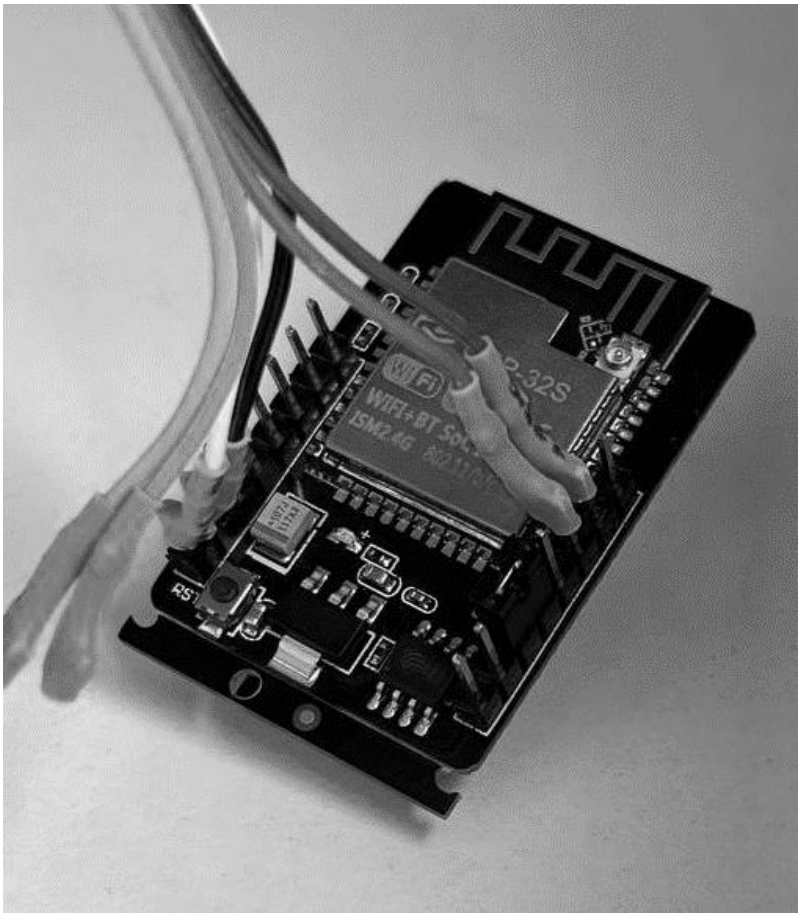

Рисунок 3.1 — Вигляд мікроконтролера ESP32-CAM під час прошивки

Рисунок 3.2 — Вікно середовища Arduino IDE під час прошивки мікроконтролера

Для зазначеної прошивки загальний обсяг даних складає 977 757 байт, що було записано за 23,3 секунди із середньою швидкістю 559,6 кбіт/с.

3.2. Налаштування тунельного рішення для передачі відеопотоку та інформації про показники

Для коректної роботи вебзастосунку з відеопотоком, що транслюється з ESP32-CAM, потрібно врахувати обмеження браузера, пов’язані з політикою CORS (Cross-Origin Resource Sharing). Через те, що ESP32-CAM не підтримує встановлення заголовків Access-Control-Allow-Origin, клієнтська частина вебзастосунку не може напряму отримати MJPEG-потік без помилки безпеки. Відтак виникає потреба в проміжному сервері, який виступатиме посередником між ESP32-CAM та клієнтською частиною.

У межах роботи передавання MJPEG-потоку здійснюється через сервіс Ngrok, який створює HTTPS-тунель до локального HTTP-сервера ESP32-CAM. Таким чином, відеопотік доступний за публічною HTTPS-адресою, що дозволяє обійти обмеження браузера без потреби в додатковому серверному проксі або ручному додаванні CORS-заголовків.

Ngrok забезпечує:

— безпечну передачу MJPEG-потоку з ESP32-CAM;

— доступ до потоку у браузері без помилок CORS;

— публічну URL-адресу для спостереження з будь-якого пристрою;

— швидку інтеграцію у фронтенд без складної конфігурації [16].

Відеопотік у форматі MJPEG транслюється безпосередньо у вебінтерфейс за допомогою HTML-елемента <img [src]="streamUrl">, де streamUrl — це також ngrok-адреса, що веде на /stream. Такий підхід дозволяє реалізувати відео в реальному часі без затримок і складних протоколів.

Таким чином, клієнтська частина Angular-застосунку може безпосередньо відображати відео, отримане з ESP32-CAM через HTTPS, сприймаючи його як безпечне джерело. Це рішення є простим, стабільним і дозволяє у подальшому масштабувати систему (наприклад, для кількох камер або збереження записів).

Комунікація з апаратною частиною, а саме модулем ESP32-CAM відбувається за допомогою HTTP-запитів до відкритого API-інтерфейсу, тунельованого через сервіс ngrok. Це дозволяє отримувати поточні показники від сенсорів, що входять до складу апаратно-програмного комплексу (температура, тиск, освітленість, рівень CO_2), без необхідності ручного налаштування маршрутизатора або відкриття портів. Дані надходять у JSON-форматі за маршрутом https://esp32environment.ngrok.app/api/data [17].

3.3. Програмна реалізація клієнтської частини вебзастосунку для трансляції відеопотоку

3.3.1 Розробка макету інтерфейсу вебзастосунку для трансляції відеопотоку

Макет інтерфейсу вебзастосунку, зображений на рисунку В.2, спроєктовано

у програмі для проєктування інтерфейсів користувача, Figma та відповідає принципам мінімалістичного, функціонального дизайну. Основні кольори було підібрано згідно з рекомендаціями UI-дизайну — темна та світла теми, контрастні кольори для індикаторів стану, проста та інтїтивно зрозуміла навігація на мобільних пристроях та персональних комп'ютерах. Кожна сторінка має чітко виражені зони — панель заголовка та меню, основний контент, який змінюється відносно активної сторінки.

3.3.2 Фізична схема вебзастосунку для трансляції відеопотоку

Вся логіка розроблюваного вебзастосунку організована всередині кореневої директорії src/app/, яку зображено на рисунку 3.3, що містить основні частини застосунку, розподілені за функціональними блоками. Компоненти, сторінки, сервіси, моделі та конфігурації структуруються таким чином, щоб спростити навігацію у проєкті та забезпечити зручність подальшої підтримки [18].

Рисунок 3.3 — Фізична схема проєкту, реалізованого з використанням Angular

Папка assets/ використовується для зберігання статичних ресурсів, таких як шрифти (fonts/) та зображення (images/), які застосовуються в інтерфейсі користувача.

Сторінки застосунку згруповано в директорії pages/, де кожна вкладена папка (наприклад, home/, indicators/, stream/) відповідає окремому маршруту програми. Кожна сторінка містить свої компоненти та шаблони і відображає окрему функціональну частину інтерфейсу: головну сторінку, сторінку прямої трансляції, сторінку відображення показників.

Кожен компонент, який може використовуватись повторно у кількох місцях застосунку, розміщується в директорії components. Таке розділення дозволяє інкапсулювати логіку відображення і стилі всередині компонентів, не дублюючи код у шаблонах сторінок.

Окремі сторінки застосунку такі як «Головна», «Трансляція», «Показники» винесені до директорії pages, де кожна сторінка являє собою окремий компонент верхнього рівня. Вони об’єднують у собі вкладені компоненти з components/, формуючи повноцінну структуру для користувача. Наприклад, сторінка трансляції містить компоненти показників записів.

Взаємодія із зовнішніми джерелами даних реалізована у вигляді сервісів, що зберігаються в папці services. Сервіси передаються у компоненти через механізм залежностей Angular і забезпечують однонапрямковий потік даних, який синхронізується за допомогою RxJS. Наприклад, Esp32Service, лістинг сервісу наведений у додатку Д, відповідає за взаємодію з мікроконтролером ESP32-CAM. Його основною функцією є отримання даних з датчиків, підключених до ESP32, через HTTP-запити до визначеного API. Цей сервіс інкапсулює логіку обміну з сервером, централізуючи доступ до екологічних показників (температура, тиск, освітленість, рівень CO_2) та обробку помилок.

Файл app-routing.module.ts виконує роль конфігуратора навігації у застосунку. У лістингу 3.1 наведено процес налаштування маршрутизації, отже app-routing.module.ts містить усі маршрути між сторінками, відповідальні за завантаження окремих модулів, а також правила доступу на основі guard-логіки. Головний модуль app.module.ts реєструє всі компоненти, сервіси, а також підключає сторонні бібліотеки, зокрема Firebase, FormsModule, HttpClientModule та інші, що забезпечують роботу з мережею, формами та асинхронними даними.

Лістинг 3.1 — Налаштування маршрутизації у файлі app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { HomeComponent } from './pages/home/home.component';
import { IndicatorsComponent } from './pages/indicators/indicators.component';
const routes: Routes = [{
    path: 'stream',
    loadComponent: () =>
      import('./pages/stream/stream.component').then(m => m.StreamComponent)
  },
  { path: 'home', component: HomeComponent },
  { path: 'indicators', component: IndicatorsComponent },
  { path: '', redirectTo: '/home', pathMatch: 'full' },
  { path: '**', redirectTo: '' }
];
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule] })
export class AppRoutingModule { }
```

3.3.3 Реалізація інтерфейсу користувача та функціональних сторінкок вебзастосунку

Уся структура застосунку розроблена з підтримкою можливості масштабування — нові сторінки або функціональність можуть додаватися із збереженням модульної архітектури без значного втручання в існуючі частини.

Клієнтська частина вебзастосунку реалізує повноцінний набір функцій, необхідних для взаємодії користувача з системою відеоспостереження в реальному часі. Центральною частиною є сторінка трансляції, яка відповідає за виведення відеопотоку безпосередньо з модуля ESP32-CAM. MJPEG-потік, переданий через HTTP, встановлене за допомогою тунельного сервера. Інтерфейс трансляції включає елементи керування, зокрема кнопки для перезапуску потоку у разі втрати з’єднання, виведення показників та початку, зупинки запису. Для забезпечення зручності взаємодії усі дії супроводжуються візуальним індикатором стану.

У структурі клієнтської частини застосунку виділено як спільні компоненти інтерфейсу, так і функціональні сторінки, кожна з яких виконує окрему роль у взаємодії користувача з системою:

Компонент верхньої панелі «header», який містить логотип, назву застосунку, перемикач теми (світла/темна), навігаційні елементи.

Нижня панель «footer», яка виконує роль інформаційного блоку. Містить посилання контактну інформацію.

Головна сторінка «home», що зображена на рисунках В.3 та В.4, призначена для ознайомлення користувача з функціональністю системи, базового опису можливостей та навігації до інших розділів.

Основна сторінка «stream», що зображена на рисунку В.6, передбачена для перегляду відеопотоку в реальному часі з ESP32-CAM. Вона включає виведення MJPEG-потоку, елементи керування, а також візуальні індикатори статусу.

Сторінка візуалізації екологічних даних «indicators», що зображена на рисунку В.8, отриманих від датчиків, підключених до ESP32. Показники температури, тиску, освітленості та рівню CO_2 відображаються у вигляді динамічних блоків з кольоровим статусом (норма, попередження, тривога) та текстовими підказками.

На рівні реалізації логіки клієнтська частина активно використовує можливості реактивного підходу. Дані, які надходять, обробляються за допомогою RxJS Observables, що дозволяє легко керувати асинхронними потоками, об’єднувати події, фільтрувати їх і трансформувати у потрібному вигляді. Весь механізм маршрутизації, у свою чергу, керується через Angular Router.

Таким чином, реалізована функціональність вебзастосунку охоплює як базові сценарії взаємодії з відеопотоком, так і передбачені можливі розширені функції керування, архівування та адміністрування. Система побудована з урахуванням вимог до швидкодії, масштабування та розширення, що дозволяє легко доповнювати її новими модулями в майбутньому.

3.3.4 Розгортання клієнтської частини вебзастосунку на Firebase Hosting

Клієнтська частина вебзастосунку була розгорнута за допомогою платформи Firebase Hosting, що забезпечило стабільну, захищену та масштабовану інфраструктуру для публікації програми. Вибір саме цього хостингу зумовлений

кількома ключовими перевагами. По-перше, завдяки вбудованій підтримці HTTPS усі запити до клієнтського інтерфейсу виконуються через захищене з’єднання без необхідності вручну генерувати SSL-сертифікати або конфігурувати проксі-сервери. По-друге, автоматичне кешування й глобальна CDN-мережа Firebase забезпечують миттєве завантаження сторінок незалежно від географічного розташування користувача [11]. Це особливо важливо для систем реального часу, де затримки навіть у межах кількох сотень мілісекунд можуть впливати на якість взаємодії з відеопотоком.

DevOps — це сучасний підхід до розробки програмного забезпечення, що поєднує процеси розробки (Development) та експлуатації (Operations). Його мета полягає в автоматизації, прискоренні та підвищенні надійності всіх етапів життєвого циклу застосунку — від написання коду до його доставки кінцевому користувачу. Одним із ключових інструментів цього підходу є CI/CD — безперервна інтеграція (Continuous Integration) та безперервне розгортання (Continuous Deployment), що передбачають автоматичне виконання серії дій після кожної зміни у репозиторії проєкту [20].

У межах розробленої системи відеоспостереження процес розгортання клієнтської частини реалізовано саме за принципами CI/CD, з використанням інструментарію Firebase CLI. Це дозволяє автоматизувати збірку Angular-застосунку та його розміщення на хостингу без ручного втручання. Після внесення змін у вихідний код, наприклад, у разі оновлення інтерфейсу, виправлення помилок або додавання нових функціональних можливостей, ініціюється процес компіляції застосунку за допомогою команди npm run build, що створює оптимізовану статичну збірку (HTML, CSS, JavaScript).

Наступним етапом є автоматизоване публікування цієї збірки на платформі Firebase Hosting — це здійснюється однією командою firebase deploy, яка передає вміст каталогу з готовим застосунком на вибраний хмарний хостинг Firebase [21]. У результаті, оновлення стає доступним для користувачів у режимі майже реального часу, що значно підвищує швидкість завантаження нових версій

застосунку та мінімізує ризик людських помилок під час ручного копіювання файлів.

Таким чином, реалізований процес розгортання є прикладом сучасної DevOps-практики, яка підвищує ефективність розробки, забезпечує гнучкість підтримки проєкту та створює передумови для масштабування вебзастосунку в майбутньому.

Клієнтська частина взаємодіє із серверною інфраструктурою переважно через HTTP-протоколи. Основний відеопотік, що надходить від ESP32-CAM, передається у форматі MJPEG через HTTP. Для забезпечення доступу з будь-якої точки світу використовується тунелювання через сервіс Ngrok, що дозволяє обійти обмеження браузера, пов’язані з політикою CORS, без потреби у додатковому проксі-сервері.

Запити, пов’язані з отриманням конфігурації, налаштувань або взаємодією з екологічними датчиками. Angular-застосунок виконує асинхронні HTTP-запити до відповідних маршрутів ESP32 або зовнішнього API, а відповіді обробляються у форматі JSON, що дозволяє гнучко керувати даними та відображати їх в інтерфейсі.

Завдяки інтеграції всіх компонентів у межах одного хостингового середовища, застосунок демонструє високу швидкодію, надійність та можливість масштабування. Інфраструктура легко піддається подальшому розширенню, наприклад, додаванням окремих функціональних сервісів, не порушуючи загальної архітектури застосунку.

3.3.5 Реалізація функції запису відеопотоку та його збереження на локальний пристрій

Окрім відображення відеопотоку в режимі реального часу, користувачеві необхідно надати можливість фіксації окремих трансляцій шляхом запису потоку у вигляді локального відеофайлу. Для реалізації цього функціоналу у клієнтській частині застосунку використано можливості елементів <canvas> і вбудованого інтерфейсу MediaRecorder, що дозволяє створювати локальний запис на основі графічного потоку.

Суть реалізації полягає у тому, що відеопотік, отриманий через MJPEG у вигляді HTML-зображення (<img>), зчитується за допомогою API CanvasRenderingContext2D, де кожен кадр промальовується з певною частотою (зазвичай 10 кадрів за секунду). Після накопичення достатньої кількості кадрів активується інтерфейс MediaRecorder, який створює потік на основі методу canvas.captureStream().

Запуск або зупинка запису здійснюється за допомогою методу toggleRecording(), наведений у лістингу 3.2.

Лістинг 3.2 — Функція для перемикання стану запису відеопотоку

```
toggleRecording(): void {
  if (!this.isRecordingAvailable) {
    console.warn('Зображення ще не готове');
    return;  }
  this.isRecording ? this.stopRecording() : this.startRecording(); }
```

Процес ініціалізації запису реалізовано у методі startRecording(), який наведений у лістингу 3.3. У ньому задаються параметри полотна (canvas), виконується промальовування кадрів та створення потоку MediaStream. Початок запису ініціюється після третього кадру, що дозволяє уникнути порожніх або пошкоджених відеофрагментів.

Лістинг 3.3 — Запуск запису з полотна Canvas

```
startRecording(): void {
    const canvas = this.canvasRef.nativeElement;
    const img = this.streamImageRef.nativeElement;
    canvas.width = img.width;
    canvas.height = img.height;
    const ctx = canvas.getContext('2d');
    if (!ctx) {
      console.error('Canvas context not available');
      return;    }
    const fps = 10;
    let frameCount = 0;
    let recordingStarted = false;
const drawLoop = () => {
if (!this.isRecording) return;
    try { ctx.drawImage(img, 0, 0, canvas.width, canvas.height);
```

```
    frameCount++;
    if (frameCount === 3 && !recordingStarted) {
      this.beginMediaRecorder(canvas.captureStream(fps));
      recordingStarted = true; }
  } catch (e) {
  console.warn('drawImage error:', e);}
  setTimeout(drawLoop, 1000 / fps);};
```

Метод beginMediaRecorder(), ініціалізує об’єкт MediaRecorder, який відповідає за запуск процесу запису відеопотоку, що надходить до HTML-елемента <canvas>. Потік створюється з кадрів трансляції (MJPEG або іншого формату), які копіюються на графічне полотно в реальному часі.

Фрагмент методу, що наведений у лістингу 3.4 ініціалізує об’єкт MediaRecorder, перевіряючи попередньо, чи підтримується обраний формат у браузері. Формат video/webm з кодеком vp8 є одним із найпоширеніших форматів, які підтримуються у сучасних браузерах. Якщо підтримка підтверджується, то створюється об'єкт MediaRecorder, а також буфер recordedChunks, у який згодом записуватимуться частини відео.

Лістинг 3.4 — Ініціалізація та перевірка підтримки формату запису

```
  private beginMediaRecorder(stream: MediaStream): void {
const mime = 'video/webm; codecs=vp8';
if (!MediaRecorder.isTypeSupported(mime)) {
  console.error('MIME type not supported:', mime);
  return; }
this.mediaRecorder = new MediaRecorder(stream, { mimeType: mime });
this.recordedChunks = [];
```

У лістингу 3.5 наведено фрагмент коду, який реалізує обробку подій запису, а саме кожного разу, коли браузер згенерує новий блок відеоданих, подія ondataavailable активується. Отримані фрагменти відеопотку перевіряються на наявність даних і, у разі позитивного результату, додаються до масиву recordedChunks. Це дозволяє зберігати весь запис частинами, які потім будуть об’єднані.

Лістинг 3.5 — Обробка подій запису ondataavailable

```
this.mediaRecorder.ondataavailable = (event) => {
  if (event.data.size > 0) {
```

```
this.recordedChunks.push(event.data);}};
```

У лістингу 3.6 наведено фрагмент коду, який виконує першу частину обробки завершення запису, об’єднує всі накопичені частини відео в єдиний об'єкт типу Blob, створює тимчасове посилання для перегляду або завантаження, формує унікальну назву файлу та додає запис до масиву recordings. Це забезпечує збереження інформації про записані файли.

Лістинг 3.6 — Обробка події onstop та формування Blob-об'єкта

```
private beginMediaRecorder(stream: MediaStream): void {
   const mime = 'video/webm; codecs=vp8';
   if (!MediaRecorder.isTypeSupported(mime)) {
     console.error('MIME type not supported:', mime);
     return; }
   this.mediaRecorder = new MediaRecorder(stream, { mimeType: mime });
   this.recordedChunks = [];
this.mediaRecorder.ondataavailable = (event) => {
if (event.data.size > 0) {
this.recordedChunks.push(event.data);}};
this.mediaRecorder.onstop = () => {
const blob = new Blob(this.recordedChunks, { type: 'video/webm' });
const url = URL.createObjectURL(blob);
const filename = `record_${Date.now()}.webm`;
const timestamp = new Date().toLocaleString('uk-UA');
this.recordings.unshift({ url, timestamp, filename });
```

У фрагменті коду, наведеному у лістингу 3.7, реалізовано механізм автоматичного завантаження сформованого відеофайлу. Створюється прихований елемент посилання <a>, який отримує згенерований URL і назву файлу. Далі відеофайл завантажується автоматично, без участі користувача, завдяки програмному натисканню на посилання. Це спрощує збереження записів у локальну файлову систему.

Лістинг 3.7 — Автоматичне завантаження відеофайлу через елемент <a>

```
 const a = document.createElement('a');
 a.href = url;
 a.download = filename;
 a.style.display = 'none';
 document.body.appendChild(a);
 a.click();
```

```
  document.body.removeChild(a);
  console.log('Запис збережено автоматично:', filename); };
```

У лістингу 3.8 наведено фрагмент коду, який забезпечує обробку можливих помилок під час роботи MediaRecorder. У разі виникнення помилки вона виводиться в консоль, що дозволяє розробнику або користувачу швидко діагностувати проблему.

Лістинг 3.8 — Обробка помилок під час запису

```
this.mediaRecorder.onerror = (e) => {
 console.error('MediaRecorder error:', e); };
```

Запуск запису за допомогою методу start(), що наведено у лістингу 3.9. Після цього об’єкт MediaRecorder починає записувати відео в режимі реального часу, викликаючи події ondataavailable і onstop відповідно до ходу роботи.

Лістинг 3.9 — Запуск процесу запису

```
this.mediaRecorder.start();
console.log('MediaRecorder started');
```

Функція stopRecording(), наведена у лістингу 3.10, завершує запис, перевіряючи поточний стан MediaRecorder.

Лістинг 3.10 — Функція stopRecording() для зупинки процесу запису

```
stopRecording(): void {
  if (this.mediaRecorder && this.mediaRecorder.state !== 'inactive') {
    this.mediaRecorder.stop();}
  this.isRecording = false;}
```

Таким чином, реалізована функціональність дає можливість користувачам зберігати відео з трансляції безпосередньо на локальний комп’ютер у форматі .webm. Цей підхід не потребує серверного зберігання чи зовнішніх інструментів, забезпечуючи автономність роботи клієнтської частини вебзастосунку

3.3.6 Реалізація адаптивності вебзастосунку для трансляції відеопотоку

Адаптивність сайту — це здатність інтерфейсу автоматично підлаштовуватись під розміри екрана пристрою, з якого користувач заходить на

сайт. Такий підхід забезпечує зручне та коректне відображення вмісту незалежно від того, чи це смартфон, планшет або комп’ютер. Адаптивний дизайн дозволяє сайту змінювати структуру та розміри елементів відповідно до параметрів вікна браузера, що створює комфортні умови користування на будь-якому пристрої.

Адаптивність інтерфейсу реалізована шляхом використання гнучкої сітки (CSS Flex/Grid), медіазапитів і Angular-сервісу, який динамічно визначає ширину вікна браузера і змінює компонування елементів залежно від поточного пристрою. Було задано точки перелому (breakpoints), зокрема ширини 1440px, 1024px, 768px та 375px, реалізація наведена в лістингу 3.11. Усі основні компоненти, включно з відеоплеєром, панеллю керування, архівом і повідомленнями, коректно масштабуються й змінюють розташування без зміщення або виходу за межі вікна. На мобільних пристроях передбачено згорнуте меню з плаваючою кнопкою доступу до архіву, що дозволяє зберігати фокус користувача на головному — відеотрансляції.

Лістинг 3.11 — Реалізація медіа запитів для забезпечення адаптивності сторінок на різних пристроїв, а саме ширина яких починається від 768px та 920px

```
  @media (min-width: 768px) { padding: 32px;
    .intro-section { margin-bottom: 48px;
      .intro-title { font-size: 32px;}
      .intro-text { font-size: 18px; margin-bottom: 24px;}
      .action-buttons { flex-direction: row; justify-content: center;
        .animated-btn {  font-size: 16px; }  } }
@media (max-width: 920px) { .stream-wrapper { padding: 16px;
    h2 { font-size: 24px; margin-bottom: 20px;  text-align: center; }
    .content-layout { flex-direction: column; gap: 24px; align-items: center; }
    .left-side, .right-side { width: 100%; }
    .video-container { width: 100%; aspect-ratio: 16 / 9;
                              padding: 8px; margin-bottom: 20px;
      img, canvas { top: 8px; left: 8px; width: calc(100% - 16px);
                          height: calc(100% - 16px); }
      .placeholder-text { font-size: 1rem; text-align: center; } }
    .controls { flex-direction: column;
      gap: 12px;
…}
```

3.3.7 Можливості для подальшого розвитку клієнтської частини вебзастосунку

Завдяки компонентній структурі та модульному підходу до розробки клієнтська частина вебзастосунку зберігає високу гнучкість і масштабованість, що відкриває широкі перспективи для подальшого функціонального розвитку. Одним із найбільш очікуваних напрямів розширення є підтримка множинних камер на базі ESP32-CAM. Це дозволить створити розподілену систему моніторингу навколишнього середовища, в якій кілька потоків будуть паралельно транслюватися на клієнтську частину з можливістю перемикання між ними або одночасного перегляду. Така функціональність вимагатиме як масштабування сигнальної логіки WebRTC, так і вдосконалення UI — зокрема, додавання вкладок або плиткового перегляду трансляцій.

Іншим важливим кроком є впровадження розширеної системи авторизації з підтримкою ролей та розділенням функціональних можливостей вебзастосунку на користувачів та адміністраторів. У межах цієї функції адміністратори зможуть контролювати дії користувачів, фіксувати зміну налаштувань або доступ до архівів. Уся інформація може зберігатись у Firestore або сторонній базі даних із фільтрацією за діями, часом і користувачем.

Модульна архітектура застосунку дозволяє додати сценарії автоматизації на основі обробки відео — розпізнавання обличчя, вмикання світла у разі виявлення руху або активація сирени. Такі сценарії можуть працювати на базі WebSocket-подій і бути розширені за допомогою Home Assistant чи зовнішніх API, що зробить систему не лише спостережною, а й інтерактивною.

Також можливо реалізувати механізм push-сповіщень у браузері, які інформуватимуть користувача про критичні події в реальному часі навіть у фоновому режимі.

На рівні інтерфейсу потенціальним покращенням є реалізація перегляду архівів у формі календаря або інтерактивної шкали подій, що дозволить візуально оцінювати активність за період та швидко переходити до потрібних фрагментів. Це

особливо актуально для великих обсягів даних, коли традиційний список стає неефективним.

Усі перелічені напрямки можуть бути впроваджені поетапно без втручання у реалізованф функціональні можливості застосунку, завдяки використанню Angular як фреймворку з підтримкою поступового завантаження, розділенням логіки за модулями. Таким чином, архітектура проєкту дозволяє не лише адаптуватися до поточних завдань, а й масштабуватись під нові виклики в майбутньому.

4 ТЕСТУВАННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

Після реалізації всіх компонентів апаратно-програмного комплексу було проведено тестування функціональності системи з метою перевірки її працездатності, надійності та відповідності вимогам, визначеним на етапі проєктування [22]. Тестування охоплювало як апаратну, так і програмну частину — модуль ESP32-CAM, серверну інфраструктуру та клієнтський вебзастосунок.

У межах цього етапу було розроблено методику перевірки, яка включала як функціональні сценарії, так і нефункціональні аспекти, зокрема затримку передачі відео, адаптивність інтерфейсу, UX-досвід, стабільність роботи в умовах нестабільного підключення та навантаження.

4.1 Розробка алгоритму тестування системи

Для перевірки функціональних можливостей та стабільності роботи розробленого вебзастосунку для трансляції відеопотоку було розроблено алгоритм для тестування, блок-схема алгоритму зобажена на рисунку 4.1, який охоплює усі основні компоненти комплексу: мікроконтролер ESP32-CAM, серверну частину з обробкою відеопотоку, а також клієнтський вебзастосунок.

Тестування системи проводилось як у локальному середовищі, так і в хмарній інфраструктурі. Було використано стандартне обладнання — персональний комп’ютер з Windows 11, Wi-Fi роутер із підтримкою частоти 2.4 GHz, а також модуль ESP32-CAM із камерою OV2640, яка забезпечує відеопотік у форматі MJPEG (через HTTP).

Для оцінки параметрів роботи системи, зокрема затримки передавання зображення, використовувалися тестові фрагменти коду з вбудованими часовими мітками в кадрах та вбудовані засоби діагностики браузера. Це дозволило здійснити аналіз потоку в реальному часі, контролювати стабільність з’єднання, формат відео та швидкість відновлення при втраті сигналу.

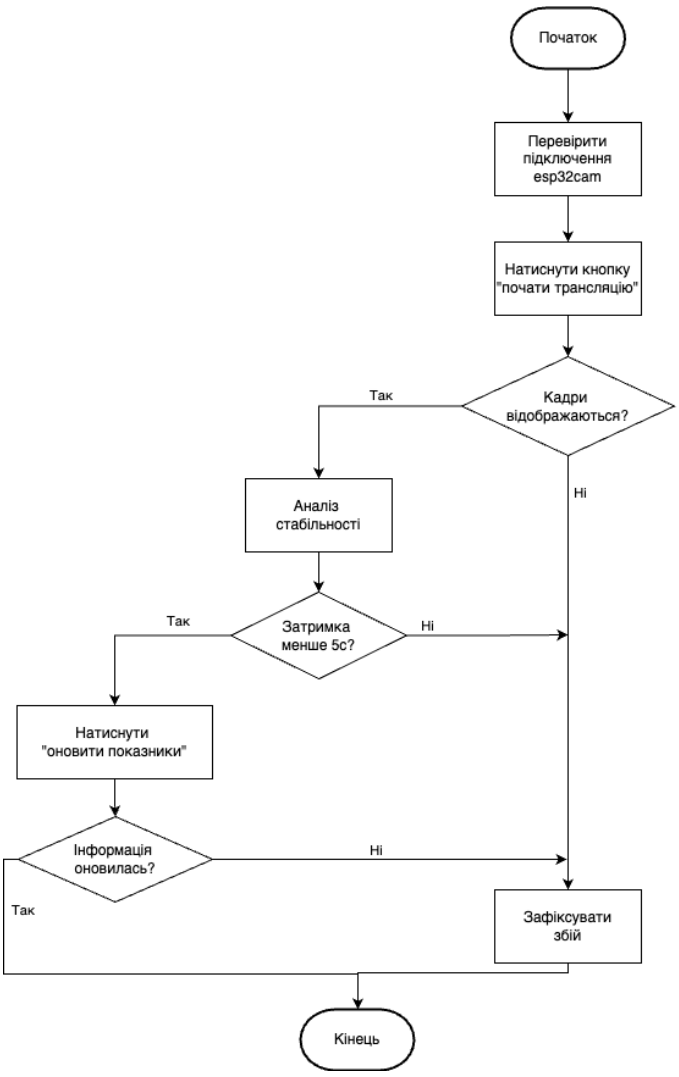

Рисунок 4.1 — Блок-схема алгоритму тестування вебзастосунку для трансляції відеопотоку

Значна частина тестування була зосереджена на стрес-сценаріях, зокрема на перевірці роботи системи при кількох одночасних з'єднаннях з камерою, а також її поведінки у випадках втрати Wi-Fi-з’єднання, оновлення сторінки чи повторного запуску трансляції.

Загалом тестування здійснювалося за принципом "чорного ящика", коли перевірка орієнтується на кінцеву поведінку системи без занурення у внутрішню реалізацію [23]. Основними критеріями оцінювання були стабільність з’єднання, відсутність критичних помилок, коректність маршрутизації між сторінками та відповідність кожного компонента очікуваному функціоналу.

4.2 Оцінювання якості відео

Оцінювання якості відеопотоку є критичним аспектом для системи відеоспостереження, оскільки від цього залежить можливість своєчасного реагування на події, точність ідентифікації об’єктів та зручність взаємодії з системою. У межах тестування було проаналізовано три основні параметри — затримку трансляції, чіткість зображення та стабільність передавання відео.

Аби визначити затримку кадру використовується значення millis() як маркер моменту захоплення кадру камерою ESP32-CAM. Воно передається у форматі JSON до клієнтської частини застосунку, де зіставляється з поточним локальним часом користувача. Завдяки цьому обчислюється затримка між моментом захоплення кадру та його фактичним відображенням у браузері. Щоб привести обидва часові вимірювання до єдиної системи координат, додатково враховується поправка у вигляді timeOffset, яка визначається один раз на початку роботи і дозволяє оцінити реальний час появи кадру в контексті локального годинника користувача. Це забезпечує більш точну, але все ж не конкретну затримку в системі відеоспостереження.

Під час трансляції відео у форматі MJPEG з модуля ESP32-CAM через пряме HTTP-з’єднання середнє значення затримки зазвичай становить від 700 до 1000 мс. Це є типовим показником для необробленого HTTP-потоку без спеціальної оптимізації або буферизації.

Як показано на рисунку 4.2, фактична затримка є більшою. Це пояснюється тим, що відеопотік і супутні дані передаються через різні тунельні канали, що створює додаткове навантаження та асинхронність доставки.

Крім того, у процесі вимірювання було враховано затримку взаємодії користувача, яка у конкретному випадку перевищила 3,8 секунди. З урахуванням цього, повна затримка була програмно обмежена до 5000 мс, щоб уникнути викривлення результатів через надмірно великі значення.

Слід зазначити, що при відсутності даної перевірки та видалення опрацювання часу в прошивці та додаткового тунелю, затримка значно зменшується, це оцінювалось візуально.

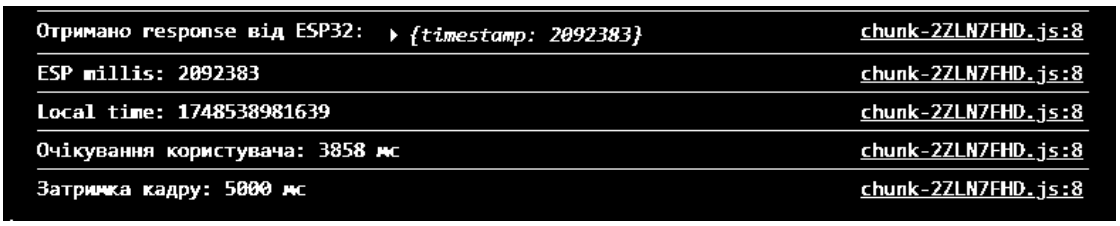

Рисунок 4.2 — Обчислення затримки кадрів

Щодо чіткості відео, зображення з камери OV2640 в стандартному режимі забезпечує роздільну здатність 640×480 або 800×600, що є достатнім для візуального моніторингу руху, проте обмеженим для розпізнавання дрібних об’єктів або обличчя. Тож для майбутньої реалізації можна перемикатись на більш чіткішу роздільну здатність, наприклад, 1600×1200 або 800×600 та безпосередньо покращувати апаратне забезпечення у разі необхідності.

Параметр стабільності потоку оцінювався шляхом тривалого перегляду трансляції в середньому по 30 хвилин за різних умов мережі. При використанні тунельного рішення і Wi-Fi-зв’язку затримки в потоці виникали лише в момент перепідключення або втрати сигналу, після чого система відновлювала трансляцію автоматично. Виявлено, що при зниженні швидкості мережі до рівня нижче 1 Мбіт/с зростає кількість обірваних кадрів, однак застосунок не реагує на це, тож варто додати повідомлення з інформацією про помилку й пропозицією перезапуску. За 10 спостережуваних сеансів не було виявлено критичних збоїв або повного розриву потоку, що свідчить про стійкість системи до коливань мережі.

Загалом, отримані результати підтвердили, що якість відеопотоку є прийнятною для основних цілей моніторингу. У разі потреби точнішого аналізу або інтеграції алгоритмів розпізнавання облич, можлива подальша оптимізація за рахунок підключення апаратного декодера чи переходу до зовнішньої відеосерверної інфраструктури.

4.3 Тестування адаптивності та оцінка досвіду користувача

Інтерфейс вебдодатку було спроєктовано з урахуванням адаптивності, що дозволяє коректно відображати його на екранах різних розмірів — від великих

моніторів до екранів планшетів і смартфонів. Завдяки гнучкому компонуванню основні блоки, такі як вікно трансляції, архів записів, сповіщення та кнопки керування, динамічно підлаштовуються під ширину екрана. У разі зменшення простору інтерфейс автоматично змінює розташування елементів, наприклад, розміщуючи блоки один під одним, щоб зберегти зручність користування.

Відеоплеєр реалізовано на основі стандартів HTML5, що забезпечує сумісність із усіма популярними браузерами та підтримку форматів потокового мовлення, таких як WebRTC або MJPEG. Це дозволяє користувачеві переглядати трансляцію в реальному часі, незалежно від того, з якого пристрою він підключається. Такий підхід до кросплатформності дозволяє інтегрувати інтерфейс у складніші системи, зокрема промислові рішення, або використовувати його як частину розподіленого середовища для моніторингу та керування.

Адаптивність вебзастосунку — це здатність інтерфейсу коректно відображатися та забезпечувати повноцінну функціональність на пристроях із різними розмірами екранів і роздільною здатністю, а саме персональних комп’ютерах, ноутбуках, планшетах та смартфонах [24]. Було перевірено, наскільки інтерфейс застосунку зберігає читабельність, доступність елементів керування та логіку взаємодії на різних пристроях.

Тестування проводилося вручну шляхом відкриття застосунку на різних екранах і емуляції пристроїв через інструменти розробника в браузерах Google Chrome та Arc. Зокрема, тестувалися режими Responsive, iPhone X, Pixel 5, iPad, а також широкоформатні дисплеї з роздільною здатністю понад Full HD (1920×1080). Також фіксувалась поведінка таких компонентів, як сторінка трансляції, блоки показників, навігаційне меню, кнопки керування потоком та індикатори стану, приклад тестування однієї із сторінок зображено на рисунку 4.3.

Під час перевірки було виявлено, що усі основні елементи змінюють розташування відповідно до доступної ширини екрана, наприклад, кнопки керування переходять у вертикальне розташування, текстові блоки автоматично переносяться, а сітка з показниками адаптується до 1 або 2 стовпців замість 4. Також було протестовано перемикання між темною та світлою темами, що впливає

на контрастність елементів. У обох варіантах інтерфейс зберігає достатню читабельність, а кольори статусів, таких як, норма, попередження, небезпека, залишаються контрастними.

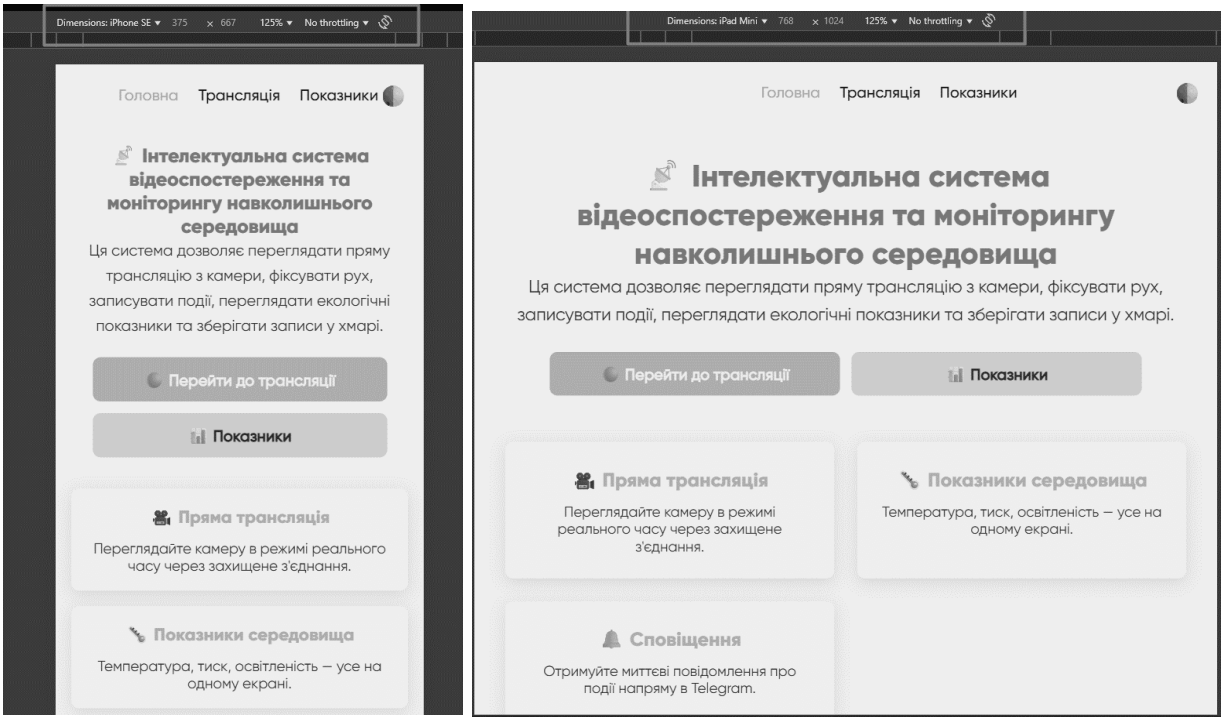

Рисунок 4.3 — Вигляд вікна браузера під час тестування адаптивності головної сторінки на мобільниому пристрої та планшеті

Також проводилось тестування на мобільних пристроях, на рисунках В.5 та В.7 зображено вебзастосунок у браузері Safari на мобільному пристрої IPhone, тож функціональні можливості були протестовані. Кнопки мають достатню зону натискання, не виходять за межі екрану, а сенсорна подія кліку коректно обробляється. На рисунку В.9 зображено, як на мобільних пристроях блоки з представленими значеннями показників змінюють своє розташування з горизонтального на вертикальне, цим самим забезпечуючи коректне їх відображення на пристрої без обрізання необхідної інформації.

Загалом, результати підтвердили повну адаптивність розробленого інтерфейсу, що гарантує зручну взаємодію як на великих екранах, так і на мобільних пристроях.

Досвід користувача (UX) — це характеристика якості взаємодії користувача

із системою, яка визначається зручністю використання, інтуїтивністю інтерфейсу, візуальним оформленням, швидкістю відгуку, а також емоційним комфортом користувача під час взаємодії з інтерфейсом [25].

Тестування UX охоплювало перевірку логіки навігації, сприйняття структури сторінок, часу виконання базових сценаріїв і реакції на дії користувача. При моделюванні типових сценаріїв, наприклад, відкриття трансляції, перегляд архіву, зміна налаштувань або авторизація користувач витрачав у середньому не більше 3–5 секунд на завершення кожної дії. Інтерфейс не містив зайвих елементів, інтуїтивно реагував на дотик, кліки та прокрутку. Контрастність тексту та кнопок відповідала рекомендаціям настанови WCAG 2.1. Застосунок використовує великі, добре читабельні шрифти й контрастні кольори, що полегшує візуальне сприйняття як на великих екранах, так і на мобільних пристроях. Усі інтерактивні елементи, зокрема кнопки та іконки, мають достатню зону натискання, що робить керування зручним навіть на сенсорних екранах. Реалізовано підтримку доступності — користувачі можуть перемикатися між темною та світлою темою, про що свідчить наявність відповідної іконки у верхньому правому куті інтерфейсу. Це не лише підвищує комфорт при довготривалому користуванні, а й дозволяє адаптувати інтерфейс до умов освітлення або особистих уподобань користувача.

Було також протестовано поведінку інтерфейсу при втраті мережевого з’єднання, перезавантаженні сторінки або відсутності даних, в таких випадках варто додати, аби застосунок відображав відповідні повідомлення, не допускаючи зависання або помилок. Усі переходи між сторінками здійснюються без перезавантаження завдяки внутрішній маршрутизації Angular, що забезпечує швидкий та плавний перехід.

Загалом результати тестування підтвердили високий рівень адаптивності інтерфейсу та позитивну взаємодію з користувачем. Незалежно від типу пристрою, застосунок зберігає повну функціональність і логічність структури, що робить його зручним як для технічних фахівців, так і для пересічних користувачів без спеціальної підготовки.

ВИСНОВКИ

У комплексній бакалаврській кваліфікаційній роботі було спроєктовано та реалізовано клієнтську частину вебзастосунку, призначеного для роботи у складі апаратно-програмного комплексу моніторингу навколишнього середовища на базі ESP32-CAM, який би забезпечував інтерактивну взаємодію з системою в реальному часі, з можливістю перегляду трансляцій, обробки подій та роботи з архівами відео.

Було проаналізовано існуючі рішення у сфері моніторингу стану оточуючого середовища та вебзастосунків для комплексів моніторингу на базі IoT засобів, що дозволило виявити актуальні підходи та обмеження сучасних систем.

Обґрунтовано вибір апаратного забезпечення для реалізації частини комплексу моніторингу стану оточуючого середовища, зокрема модуля ESP32-CAM, який забезпечує відеотрансляцію.

Проведено аналітичний огляд сучасних фреймворків і технологій, придатних для реалізації клієнтських вебінтерфейсів. Обґрунтовано вибір Angular як основи для розробки, з огляду на його модульну архітектуру, розвинену систему маршрутизації, підтримку реактивних форм, інтеграцію з Firebase та високу продуктивність в односторінкових застосунках (SPA).

Реалізовано клієнтську частину вебзастосунку, в якому наявні функції перегляду відеопотоку через MJPEG, відображення інформації про показники, можливість оновлення показників в реальному часі та запису з автоматичним збереження відеопотоку на локальний пристрій. Розроблено адаптивний інтерфейс, оптимізований для мобільних пристроїв і персональних комп’ютерів. Вебінтерфейс було розгорнуто на хостингу Firebase Hosting, що забезпечило доступність, безпечне HTTPS-з'єднання.

Під час розробки вебзастосунку були створені такі компоненти інтерфейсу, як відеоплеєр для відображення потокового відео з камери ESP32-CAM, панель керування, що містить елементи взаємодії з відеопотоком (відтворення, припинення та запис), навігаційне меню для перемикання між основними розділами застосунку, перемикач теми інтерфейсу, що забезпечує підтримку

світлого та темного режимів відображення, а також інформаційний блок сенсорних показників, який відображає актуальні дані навколишнього середовища в режимі реального часу.

Застосунок побудовано на основі реактивної архітектури з використанням RxJS, що забезпечило стійку роботу з асинхронними потоками даних. Адаптивність інтерфейсу реалізовано таким чином, що застосунок коректно працює на пристроях із різними розмірами екранів та автоматично підлаштовується під доступний простір.

Проведене тестування клієнтської частини вебзастосунку підтвердило працездатність усіх реалізованих функцій, задовільну якість відеотрансляції, надійність передачі даних і швидкий відгук інтерфейсу. Було протестовано інтерактивні сценарії, включно з переглядом архіву, виявленням руху, генерацією сповіщень, налаштуванням параметрів трансляції та поведінкою при втраті з’єднання.

Таким чином, поставлені цілі комплексної бакалаврської кваліфікаційної роботи досягнуто, а результати розробки можуть бути використані як основа для подальших досліджень та впроваджень у галузі систем відеоспостереження, розумного дому або навчальних IoT-проєктів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Можливості фреймворку Angular для розробки вебдодаткупотокової трансляції відео // Войцеховська О. В., Вовковинська А. В. Матеріали LIV Всеукраїнської науково-технічної конференції підрозділів Вінницького національного технічного університету (Вінниця, 2025 р.) [Електронний ресурс]. Режим доступу: https://d.conf.vntu.edu.ua/index.php/all-fitki/all-fitki-2025/paper/view/23785

2. 3 layer IoT architecture : вебсайт. URL: https://www.geeksforgeeks.org/3-layer-iot-architecture/ (дата звернення: 21.03.2025)

3. ESP32 Series Datasheet : вебсайт. URL: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf (дата звернення: 22.03.2025)

4. ESP32-CAM WiFi+Bluetooth+Camera Module : вебсайт. URL: https://media.digikey.com/pdf/Data%20Sheets/DFRobot%20PDFs/DFR0602_Web.pdf (дата звернення: 22.03.2025)

5. From Traffic Cameras to Web Applications: A Streaming Architecture Overview : вебсайт. URL: https://medium.com/@gold_olar/from-traffic-cameras-to-web-applications-a-streaming-architecture-overview-218759743d86 (дата звернення: 23.03.2025)

6. Вебресурс для моніторнгу навколишнього середовища у Буковелі : вебсайт. URL: https://bukovel.com/cams (дата звернення: 23.03.2025)

7. Broadcasting 39 ski resorts of Ukraine : вебсайт. URL: https://snih.info/en (дата звернення: 23.03.2025)

8. The TypeScript Handbook : вебсайт. URL: https://www.typescriptlang.org/docs/handbook/intro.html (дата звернення: 31.03.2025)

9. React, Vue, or Angular: Making the Right Choice for Your Project in 2025 : вебсайт. URL: https://medium.com/@wutamy77/react-vue-or-angular-making-the-right-choice-for-your-project-in-2025-d6939751e575 (дата звернення: 31.03.2025)

10. Angular vs Vue.js: що обрати : вебсайт. URL: https://dou.ua/lenta/columns/angular-vs-vuejs/ (дата звернення: 31.03.2025)

11. Firebase developer documentation : вебсайт. URL: https://firebase.google.com/docs (дата звернення: 01.04.2025)

12. Web Application Architecture: The Basics : вебсайт. URL: https://www.intellectsoft.net/blog/web-application-architecture/ (дата звернення: 01.04.2025)

13. SPA (Single-page application) : вебсайт. URL: https://developer.mozilla.org/en-US/docs/Glossary/SPA (дата звернення: 01.04.2025)

14. ESP32-CAM_MJPEG2SD: RTSP/MJPEG прошивка для ESP32-CAM з підтримкою запису на SD-карту та трансляції у браузер. : вебсайт. URL: https://github.com/s60sc/ESP32-CAM_MJPEG2SD (дата звернення: 14.04.2025)

15. Getting Started With ESP32-CAM : вебсайт. URL: https://lastminuteengineers.com/getting-started-with-esp32-cam/ (дата звернення: 14.04.2025)

16. Тунельна передача відеопотоку : вебсайт. URL: https://ngrok.com/ (дата звернення: 15.04.2025)

17. HTTP request methods : вебсайт. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Methods (дата звернення: 14.04.2025)

18. Вебзастосунок у системі контролю версій GitHub : вебсайт. URL: https://github.com/alinavovkov/stream-app (дата звернення: 14.04.2025)

19. DevOps Model Defined : вебсайт. URL: https://aws.amazon.com/devops/what-is-devops/ (дата звернення: 17.04.2025)

20. What is CI/CD? : вебсайт. URL: https://about.gitlab.com/topics/ci-cd/ (дата звернення: 18.04.2025)

21. Firebase Hosting : вебсайт. URL: https://firebase.google.com/docs/hosting (дата звернення: 18.04.2025)

22. A Complete Guide to Web Testing: How to Test a Website : вебсайт. URL: https://sandra-parker.medium.com/a-complete-guide-to-web-testing-how-to-test-a-website-a245afc2041f (дата звернення: 24.04.2025)

23. Black Box Testing : вебсайт. URL: https://www.imperva.com/learn/application-security/black-box-testing/ (дата звернення: 24.04.2025)

24. What is Adaptive Web Design? : вебсайт. URL: https://www.geeksforgeeks.org/adaptive-web-design/ (дата звернення: 24.04.2025)

25. User Experience (UX) Design : вебсайт. URL: https://www.interaction-design.org/literature/topics/ux-design?srsltid=AfmBOoriUkRgZ_R4Zt4CY9ytyQguGF7ODTolSnFzG_dtFaHvcv0frj-9 (дата звернення: 25.04.2025)

ДОДАТОК А

Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н., проф. Азаров О.Д.
«21» березня 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ
на виконання бакалаврської кваліфікаційної роботи
«Апаратно-програмний комплекс моніторингу стану оточуючого середовища на базі iot засобів. Частина 3. Вебзастосунок для трансляції відеопотоку»
08-54. КБКР.040.00.000 ТЗ

Науковий керівник: к.т.н., доц. каф. ОТ
_____Войцеховська О. В.

Студентка групи 2КІ-21б
_____Вовковинська А. В.

м. Вінниця — 2025

1 Підстава для використання бакалаврської кваліфікаційної роботи (КБКР)

1.1 У сучасному світі, де цифрові технології відіграють ключову роль у розбудові систем моніторингу, особливої актуальності набуває використання IoT-засобів для отримання, передачі та обробки даних у реальному часі. Однією з таких задач є створення ефективної системи трансляції відеопотоку для апаратно-програмного комплексу моніторингу стану оточуючого середовища.

1.2 Наказ про затвердження теми кваліфікаційної роботи.

2 Мета КБКР і призначення розробки

2.1 Мета роботи – розробка вебзастосунку, який забезпечує трансляцію відеопотоку в реальному часі з пристрою ESP32-CAM, реалізованого як частина апаратно-програмного комплексу моніторингу стану навколишнього середовища.

2.2 Призначення розробки — створення клієнтської частини вебінтерфейсу для доступу до трансляції відео та перегляду показників, що надходять із сенсорів середовища через IoT-пристрій ESP32-CAM.

3 Вихідні дані для виконання КБКР

3.1 Проведення аналізу сучасних систем відеоспостереження, методів трансляції відеопотоку та підключення IoT-пристроїв;

3.2 Розробка структури, логіки взаємодії компонентів та макетів інтерфейсу вебзастосунку;

3.3 Реалізація прошивки для ESP32-CAM та налаштування протоколу передачі відео;

3.4 Розробка клієнтської частини вебзастосунку з використанням сучасних фреймворків (HTML, SCSS, Angular);

3.5 Тестування стабільності трансляції відео, адаптивності інтерфейсу та швидкодії застосунку;

3.6 Впровадження вебінтерфейсу для перегляду відеопотоку та супровідних показників навколишнього середовища.

4 Вимоги до виконання КБКР

Головна вимога – створити адаптивний, функціональний та стабільний вебзастосунок, який дозволяє здійснювати перегляд відеопотоку з ESP32-CAM у режимі реального часу та обробку супровідної інформації.

5 Етапи КБКР та очікувані результати

Етапи роботи та очікувані результати приведено в Таблиці А.1.

№ етапу	Назва етапу	Термін виконання	Очікувані результати
1	Аналіз сучасних систем монітиронгу оточуючого середовища	21.03.25 – 30.03.25	Аналітичний огляд
2	Проєктування вебзастосунку	31.03.25 – 13.04.25	Розділ 2
3	Реалізація клієнтської частини та інтеграція з ESP32-CAM	14.04.25 – 27.04.25	Розділ 3
4	Тестування і аналіз результатів	14.04.25 – 27.04.25	Розділ 4
5	Оформлення пояснювальної записки, графічного матеріалу та презентації	28.04.25 – 11.05.25	Пояснювальна записка, графічний матеріал та презентація

6 Матеріали, що подаються до захисту КБКР

До захисту подаються: пояснювальна записка КБКР, графічні та ілюстративні матеріали, протокол попереднього захисту на кафедрі, відгук наукового керівника, відгук опонента, протоколи складання державних екзаменів, анотації до КБКР українською та іноземною мовами, довідка про відповідність оформлення вимогам.

7 Порядок контролю виконання та захисту КБКР

Контроль виконання етапів здійснюється науковим керівником згідно з встановленими термінами. Захист БКР проходить на засіданні екзаменаційної комісії, затвердженої наказом ректора.

8 Вимоги до оформлення та порядок виконання КБКР

8.1 При оформленні БКР використовуються:

— ДСТУ 3008 : 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302 : 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 123 «Комп'ютерна інженерія» (освітня програма «Комп'ютерна інженерія») — кафедра обчислювальної техніки, ВНТУ 2023.

8.2 Порядок виконання БКР викладено в «Положення про кваліфікаційні роботи на першому (бакалаврському) рівні вищої освіти СУЯ ВНТУ-03.02.02-П.001.01:21»

ДОДАТОК Б

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи:

Апаратно-програмний комплекс моніторингу стану оточуючого середовища на базі IoT засобів. Частина 3. Вебзастосунок для трансляції відеопотоку

Тип роботи: Бакалаврська кваліфікаційна робота

(БДР, МКР)

Підрозділ кафедра обчислювальної техніки

(кафедра, факультет)

Показники звіту подібності StrikePlagiarism

Оригінальність 99,5% Схожість 0,5%

Аналіз звіту подібності (відмітити потрібне):

- ✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.
- □ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.
- □ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Захарченко С. М.
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою StrikePlagiarism щодо роботи.

Автор роботи _____ Вовковинська А. В.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Войцеховська О. В.
(підпис) (прізвище, ініціали)

ДОДАТОК В

ІЛЮСТРАТИВНА ЧАСТИНА

АПАРАТНО-ПРОГРАМНИЙ КОМПЛЕКС МОНІТОРИНГУ СТАНУ ОТОЧУЮЧОГО СЕРЕДОВИЩА НА БАЗІ ІОТ ЗАСОБІВ. ЧАСТИНА 3. ВЕБЗАСТОСУНОК ДЛЯ ТРАНСЛЯЦІЇ ВІДЕОПОТОКУ

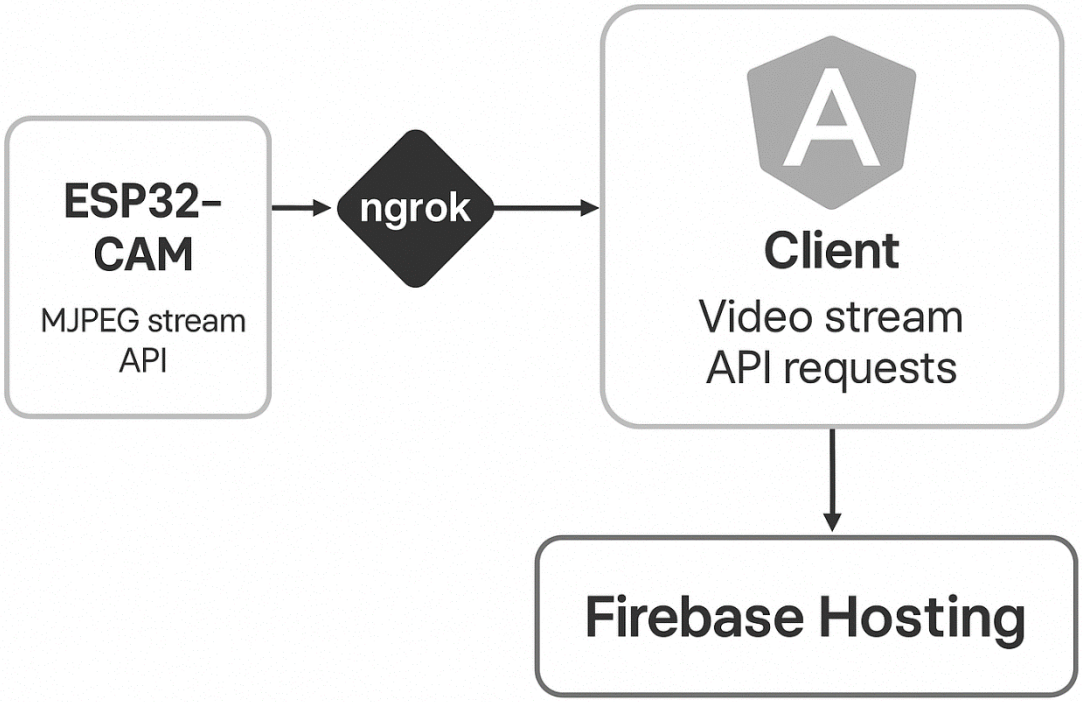

Рисунок В.1 — Структурна схема взаємодії компонентів

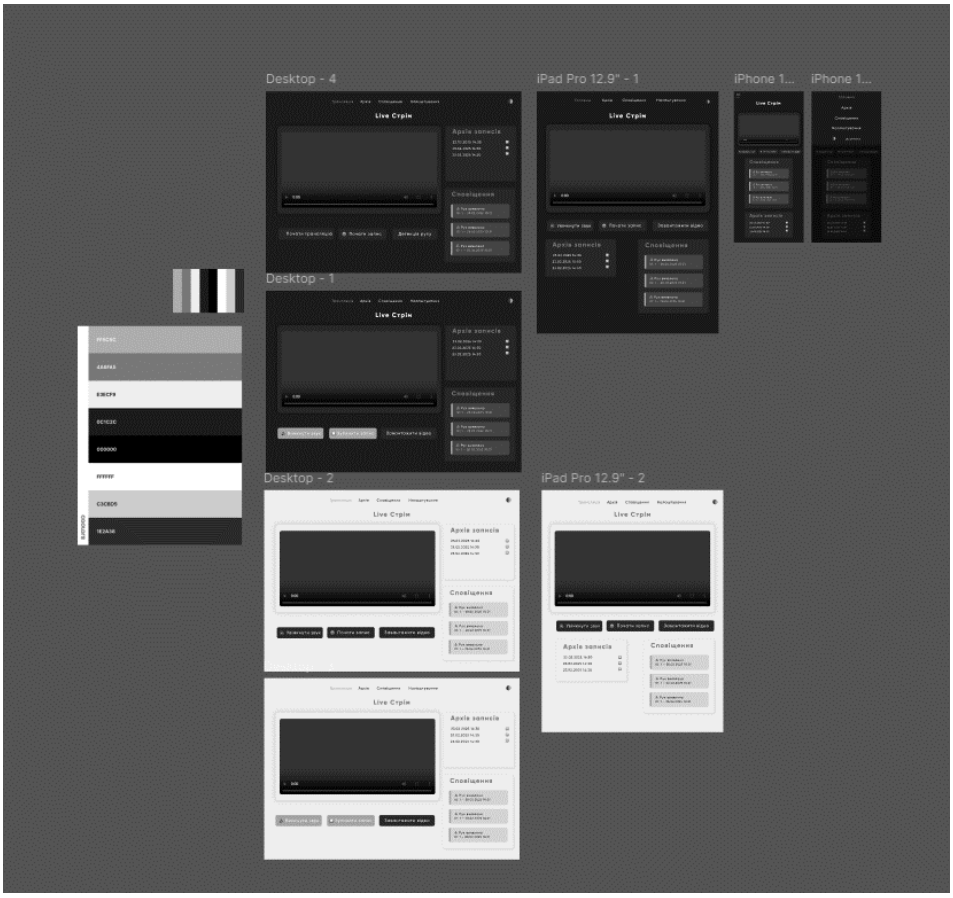

Рисунок В.2 — Дизайн-макети вебзастосунку у середовищі Figma

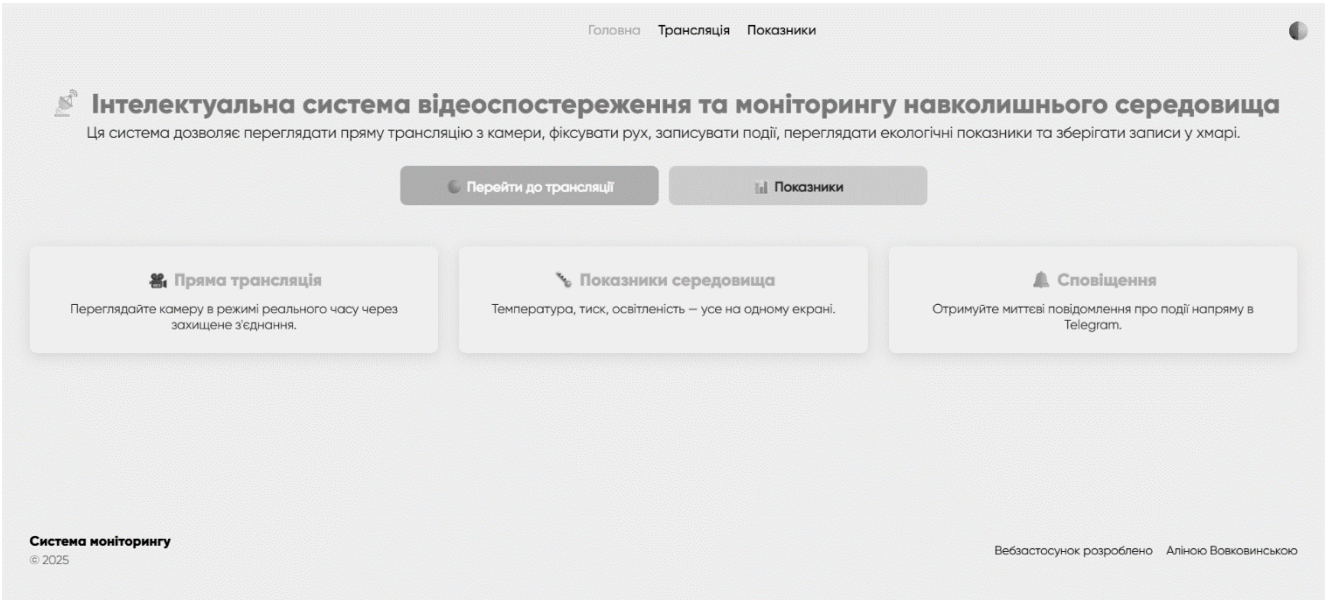

Рисунок В.3 — Відображення головної сторінки на персональному комп’ютері у світлій темі

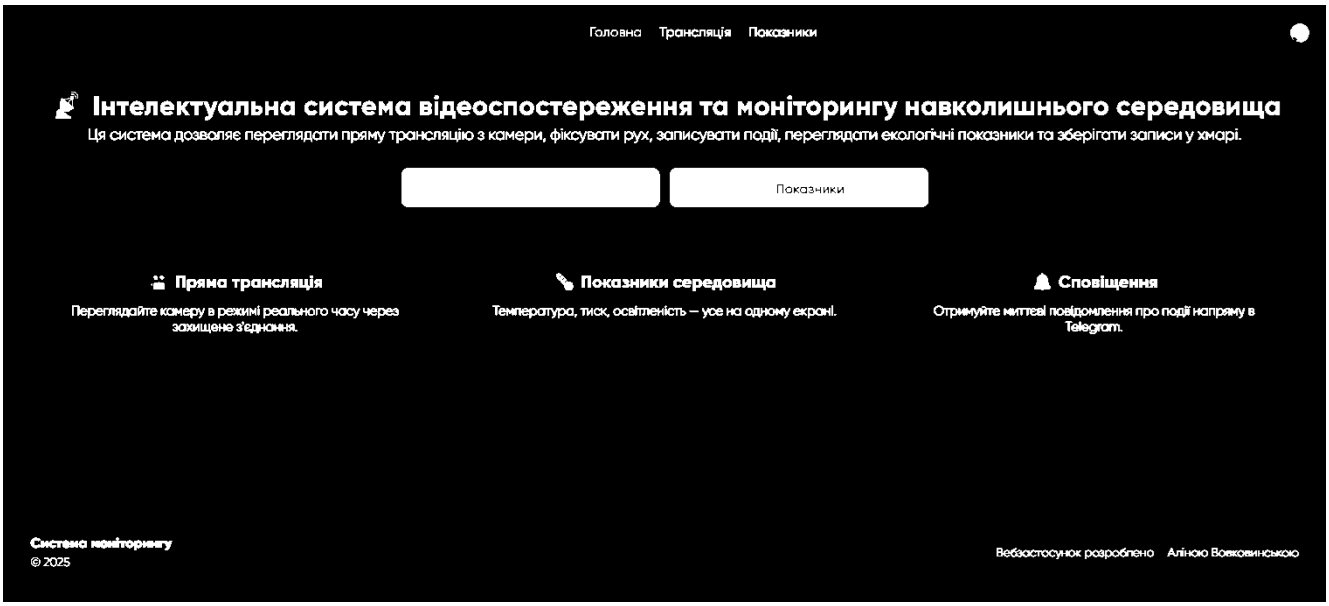

Рисунок В.4 — Відображення головної сторінки на персональному комп’ютері у темній темі

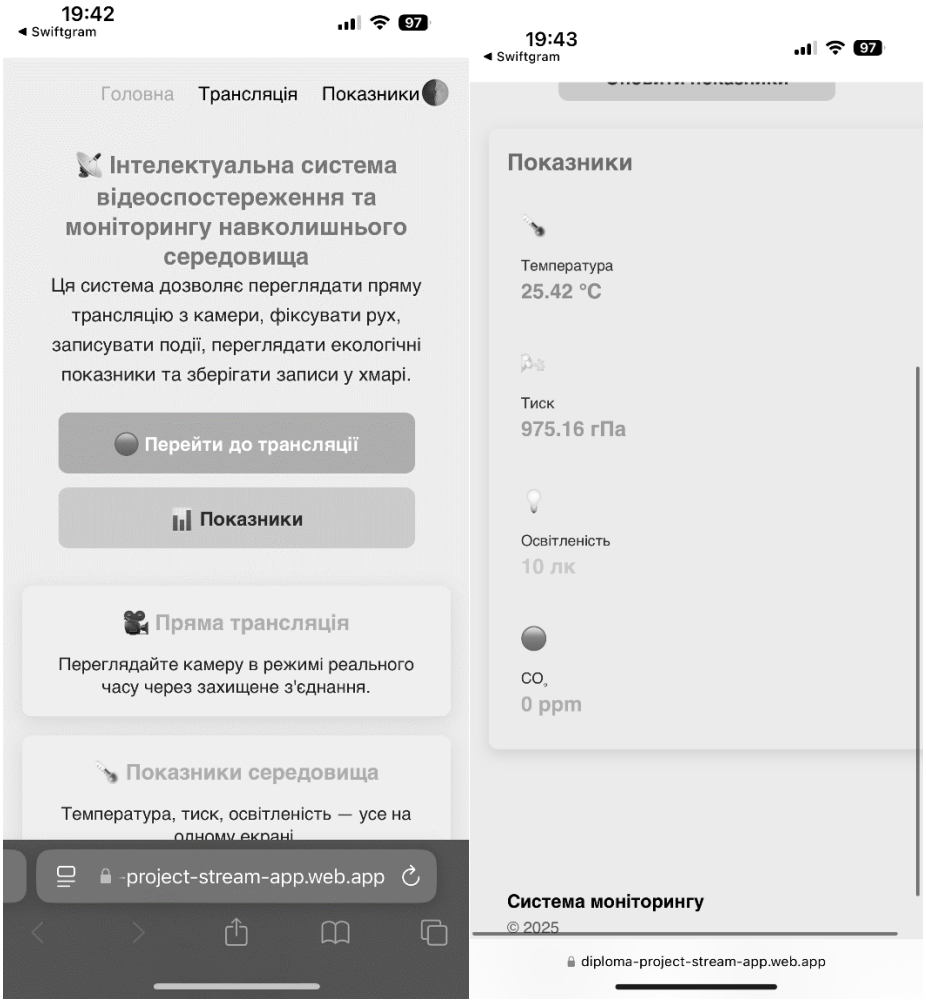

Рисунок В.5 — Відображення головної сторінки на мобільному пристрої у світлій темі

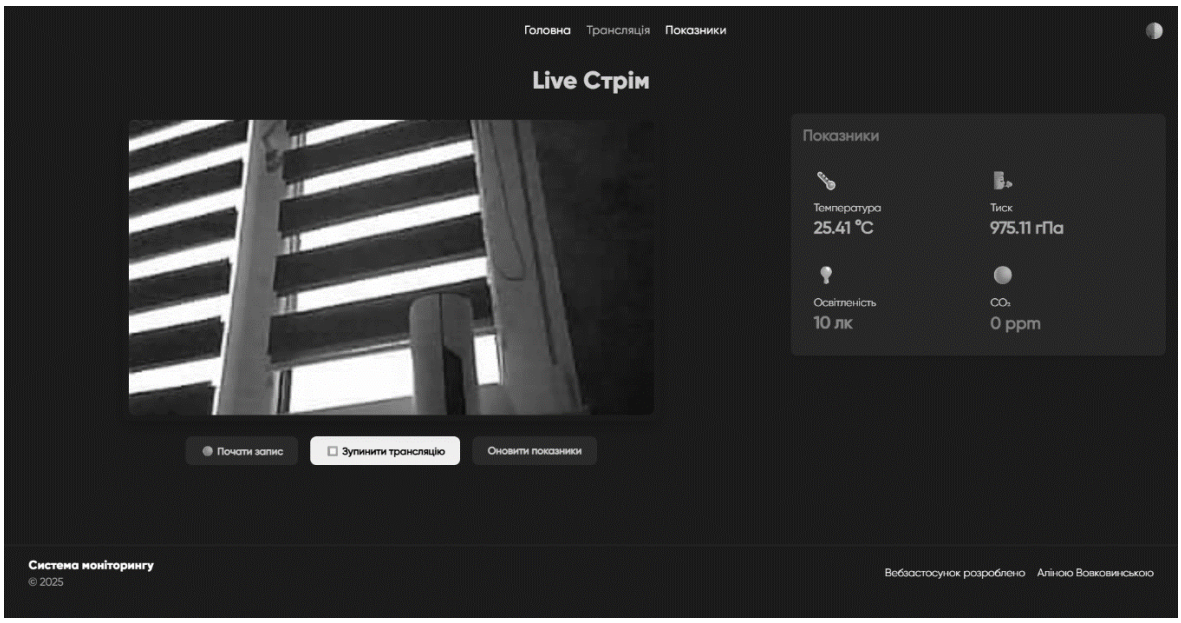

Рисунок В.6 — Сторінка «Трансляція» при увімкненій трансляції в темній темі

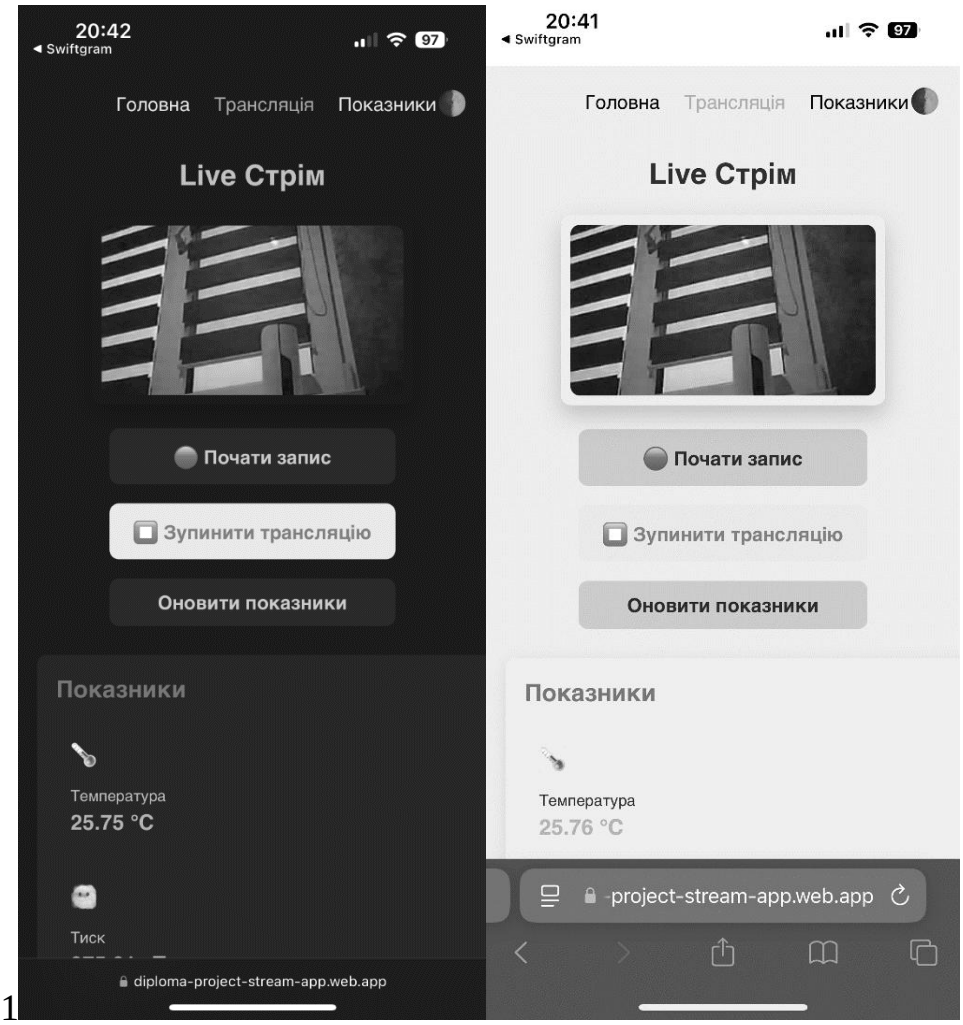

Рисунок В.7 — Сторінка «Трансляція» при увімкненій трансляції на мобільному пристрої у світлій та темній темі

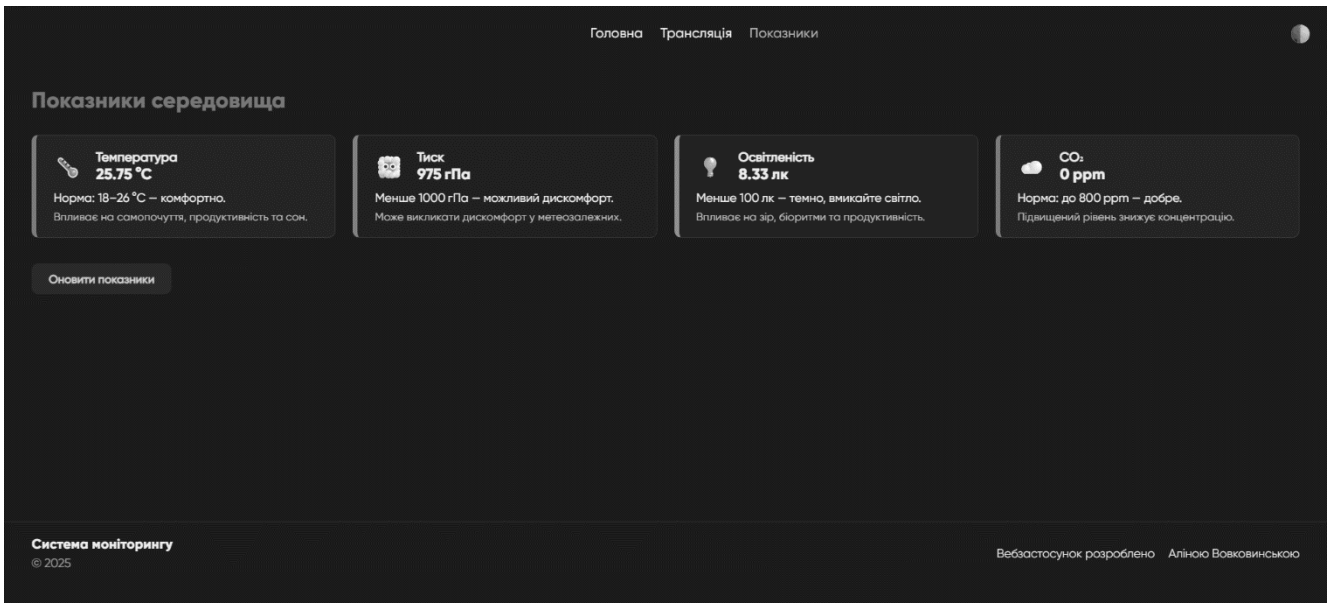

Рисунок В.8 — Сторінка «Показники» у темній темі

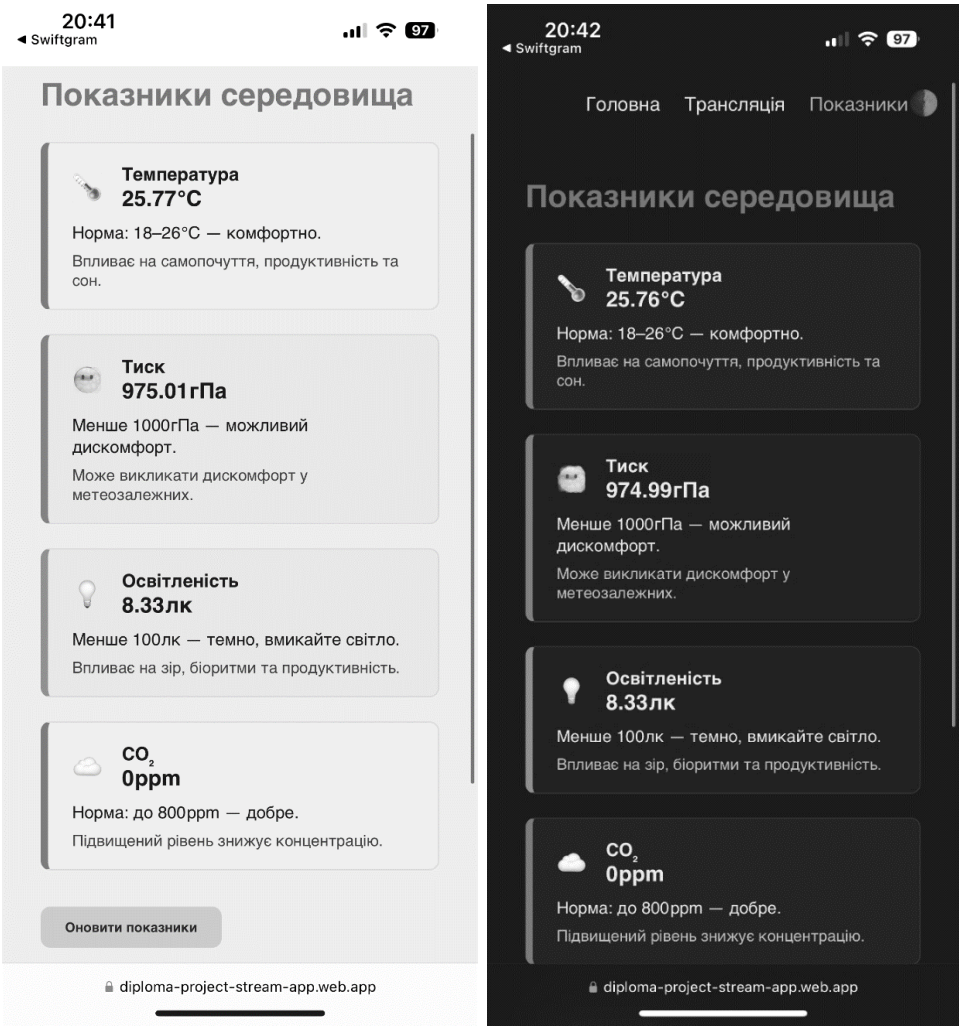

Рисунок В.12 — Сторінка «Показники» на мобільному пристрої у світлій та темній темі

ДОДАТОК Г

Лістинг програмного коду файлу esp32.service.ts

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import { throwError } from 'rxjs';
import { catchError, map } from 'rxjs/operators';
export interface SensorData {
  temperature: number;
  pressure: number;
  light: number;
  co2?: number;
}
@Injectable({
  providedIn: 'root'
})
export class Esp32Service {
  private baseUrl = 'https://esp32environment.ngrok.app/api/data';
  constructor(private http: HttpClient) {}
  getSensorData(): Observable<SensorData> {
    return this.http.get<SensorData>(this.baseUrl).pipe(
      catchError(err => {
        console.error('Помилка при запиті до ESP32', err);
        return throwError(() => new Error('Не вдалося отримати дані з ESP32'));
      })
    );
  }
  getEnvironmentData(): Observable<SensorData> {
    return this.http.get<SensorData>(this.baseUrl).pipe(
      map(data => ({
        temperature: data.temperature ?? 0,
        pressure: data.pressure ?? 0,
        light: data.light ?? 0,
        co2: data.co2 ?? 0
      })),
      catchError(err => {
        console.error('Помилка при отриманні даних з ESP32:', err);
        return throwError(() => new Error('Не вдалося отримати дані з ESP32'));
      })
    );
}
```