МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВОДНОГО ГОСПОДАРСТВА ТА ПРИРОДОКОРИСТУВАННЯ
Навчально-науковий інститут кібернетики,
інформаційних технологій та інженерії
Кафедра комп'ютерних наук та прикладної математики

"До захисту допущена"
Завідувач кафедри комп'ютерних
наук та прикладної математики
Турбал Юрій Васильович
____ _____ 20___ р.

КВАЛІФІКАЦІЙНА РОБОТА
«РОЗРОБКА ТА ІНТЕГРАЦІЯ СЕРВЕРНОЇ ЛОГІКИ З МОБІЛЬНИМ ЗАСТОСУНКОМ ДЛЯ УПРАВЛІННЯ IOT-ПРИСТРОЯМИ СИСТЕМИ TANKTOAD»

Виконав: **Юрчик Дмитрій Олегович**
(прізвище, ім'я, по батькові) (підпис)
група ІПЗ-41-і2
Керівник: **доцент, кандидат технічних наук**
Жуковська Н. А
(науковий ступінь, вчене звання, посада, прізвище, ініціали) (підпис)

Рівне – 2025

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи на тему «Розробка та інтеграція серверної логіки з мобільним застосунком для управління IoT-пристроями системи TankToad» написана на 57 сторінках, містить 9 ілюстрацій, 6 використаних джерел.

Метою роботи є створення та інтеграція серверного API для мобільного застосунку, який використовується фермерами для моніторингу та керування пристроями системи TankToad у реальному часі.

Об'єктом дослідження є система дистанційного моніторингу TankToad, яка дозволяє автоматизувати контроль водопостачання у сільському господарстві, включаючи рівень води в резервуарах, стан насосів, акумуляторів і камер.

Предметом дослідження виступає серверна частина програмного забезпечення, що відповідає за взаємодію мобільного застосунку з базою даних, перевірку прав доступу користувача, обробку даних з пристроїв та реалізацію інтерфейсів для підключення нових пристроїв через QR-коди.

Методи розробки ґрунтуються на використанні сучасного стеку технологій .NET Core, C#, Entity Framework Core та REST API підходу. Для безпечного доступу використовується авторизація на основі токенів, а структура контролерів розроблена з урахуванням масштабованості та розмежування ролей користувачів.

Практичне значення полягає в реалізації серверного API, що забезпечує роботу мобільного клієнта в онлайн та офлайн режимах. Це дозволяє користувачам своєчасно отримувати актуальні показники, налаштовувати пристрої, вмикати або вимикати насоси, переглядати історію вимірювань, а також делегувати доступ іншим учасникам.

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

API	–	Application Programming Interface — інтерфейс прикладного програмування
JSON	–	JavaScript Object Notation — текстовий формат обміну даними
REST	–	Representational State Transfer — архітектурний стиль взаємодії з API
HTTP	–	HyperText Transfer Protocol — протокол передачі гіпертексту
CRUD	–	Create, Read, Update, Delete — базові операції з даними
ASP.NET Core	–	Кросплатформенний фреймворк для створення вебзастосунків від Microsoft
C#	–	Об'єктно-орієнтована мова програмування від Microsoft
EF Core	–	Entity Framework Core — ORM для роботи з базами даних у .NET
SQL	–	Structured Query Language — мова запитів до реляційних баз даних
DB	–	Database — база даних
DI	–	Dependency Injection — ін’єкція залежностей
JWT	–	JSON Web Token — формат для передачі даних автентифікації
DTO	–	Data Transfer Object — об'єкт для передачі даних між шарами програми
UI	–	User Interface — інтерфейс користувача
UX	–	User Experience — досвід взаємодії користувача із системою

IoT	–	Internet of Things — інтернет речей, мережа взаємопов’язаних пристроїв
QR	–	Quick Response Code — двовимірний штрихкод для зчитування інформації
Swagger / OpenAPI	–	Інструменти для документування та тестування REST API
Git	–	Розподілена система керування версіями коду
VS / VS Code	–	Visual Studio / Visual Studio Code — середовища розробки
Push Notification	–	Спливаюче повідомлення, яке надсилається з сервера на мобільний пристрій
.NET	–	Платформа для розробки програмного забезпечення від Microsoft

ЗМІСТ

ВСТУП

У сучасних умовах стрімкого розвитку інтернету речей (IoT) особливої актуальності набувають системи, що дозволяють ефективно керувати розподіленими пристроями з єдиного цифрового середовища. Одним із прикладних рішень у цьому напрямі є система TankToad — апаратно-програмний комплекс, призначений для дистанційного моніторингу та керування рівнем води в резервуарах, станом насосів, акумуляторів та камер. Система активно впроваджується у фермерських господарствах США компанією Meadowlark Solutions.

TankToad орієнтована насамперед на фермерів та операторів ранчо, які прагнуть зменшити кількість виїздів до об’єктів, автоматизувати процеси контролю водопостачання та зосередитися на критичних завданнях господарства. Для забезпечення стабільної та зручної взаємодії між мобільним застосунком і фізичними пристроями, критично важливою є наявність надійної серверної логіки — вона відповідає за обробку запитів користувача, перевірку прав доступу, зберігання історичних даних, та реагування на зміну стану пристроїв у режимі реального часу.

Таким чином, бекенд TankToad виступає ключовою ланкою між IoT-пристроями, мобільним застосунком і фермером, забезпечуючи коректну роботу всієї системи в польових умовах.

Актуальність роботи полягає у створенні надійної та масштабованої серверної логіки, що забезпечує безперебійну роботу мобільного застосунку системи TankToad, враховуючи умови реального використання в польових локаціях.

TankToad експлуатується у фермерському середовищі, де користувачі нерідко стикаються з нестабільним інтернет-з’єднанням, потребують своєчасного отримання push-сповіщень, доступу до історії даних, медіа з

камер, а також можливості делегувати права доступу іншим членам команди.

У таких умовах саме серверна частина виконує ключову роль — вона забезпечує зберігання, обробку, трансформацію та захист даних, а також контроль над правами доступу, що критично важливо для безпечного та ефективного управління пристроями системи.

Метою роботи та основним завданням дослідження є розробка та впровадження серверної логіки, яка забезпечує повноцінну взаємодію мобільного клієнта з пристроями системи TankToad для кінцевих користувачів — фермерів, що працюють із платформою компанії Meadowlark Solutions.

Бекенд покликаний забезпечити стабільний доступ до даних пристроїв у реальному часі, включаючи рівень води, заряд акумулятора, стан насосів і активність камер. Реалізовані API-ендпоїнти дозволяють мобільному застосунку відображати пристрої на інтерактивній карті, оновлювати статуси, зчитувати історію змін та надсилати push-сповіщення про критичні події.

Окрема увага приділяється реалізації підтримки офлайн-доступу до останніх показників через кешування даних, що особливо важливо в умовах нестабільного або відсутнього інтернет-з’єднання на віддалених фермах.

Об’єктом дослідження є система дистанційного моніторингу TankToad, розроблена компанією Meadowlark Solutions для аграрного сектору з метою контролю рівня води в резервуарах, а також моніторингу стану насосів, акумуляторів та камер на фермах і ранчо.

TankToad складається з IoT-пристроїв, що оснащені сенсорами та автономним живленням, комунікаційних модулів (з підтримкою стільникового і супутникового зв’язку), а також серверної частини, яка приймає, обробляє, зберігає і надає дані користувачам.

Серверна логіка системи виступає критичним компонентом, що забезпечує захищений обмін даними, авторизацію користувачів, управління правами доступу, а також агрегацію історичної інформації для своєчасного виявлення проблем, оптимізації обслуговування та запобігання аварійним ситуаціям.

Предметом дослідження є розробка серверної частини системи TankToad, яка забезпечує обробку запитів мобільного застосунку, авторизацію користувачів, доступ до даних пристроїв і логіку керування.

У центрі уваги — створення API для отримання актуальної інформації з сенсорів, роботи з картографічними координатами, налаштування режимів роботи пристроїв та генерації сповіщень. Також реалізовано механізми для підтримки офлайн-доступу через кешування останніх даних.

Методи розробки ґрунтуються на використанні сучасного стеку .NET Core з мовою програмування C# та технологією Entity Framework Core для роботи з базою даних. Серверна частина побудована у вигляді RESTful API, що забезпечує ефективну взаємодію з мобільним застосунком через HTTPS-запити з авторизацією на основі Bearer-токенів.

Архітектура побудована з урахуванням принципів модульності та розділення відповідальностей: логіка авторизації, робота з пристроями, генерація push-сповіщень, обробка телеметрії, історичних графіків, а також активація пристроїв через QR-коди винесені в окремі сервіси та контролери.

Для підтримки польових умов реалізовано механізми кешування останніх показників на стороні клієнта, а також логіку перевірки статусів пристроїв навіть при нестабільному інтернеті. Особливу увагу приділено валідації вхідних параметрів, контролю доступу відповідно до ролей користувачів (власник, делегований користувач), масштабованості рішення та можливості додавання нових типів пристроїв у майбутньому.

РОЗДІЛ І. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ДОСЛІДЖЕННЯ. ТЕХНІЧНЕ ЗАВДАННЯ

1.1 Визначення предметної області та проблеми підприємства

Meadowlark Solutions — американська компанія, що спеціалізується на створенні рішень у сфері моніторингу водних ресурсів для аграрного сектора. Основним продуктом компанії є система TankToad — апаратно-програмний комплекс для дистанційного відстеження рівня води у резервуарах, насосів, сенсорів і допоміжного обладнання на фермах і ранчо. Її архітектура охоплює датчики з автономним живленням, засоби зв'язку (GSM/SAT), серверну інфраструктуру та мобільний застосунок для користувачів.

TankToad розроблялася з урахуванням специфіки сільськогосподарського сектору США, де реальним викликом є географічна розосередженість, нестабільне інтернет-з'єднання та обмежені ресурси для обслуговування обладнання. Завдяки інтеграції з сучасними засобами зв'язку, система дозволяє отримувати дані у режимі реального часу навіть у віддалених локаціях.

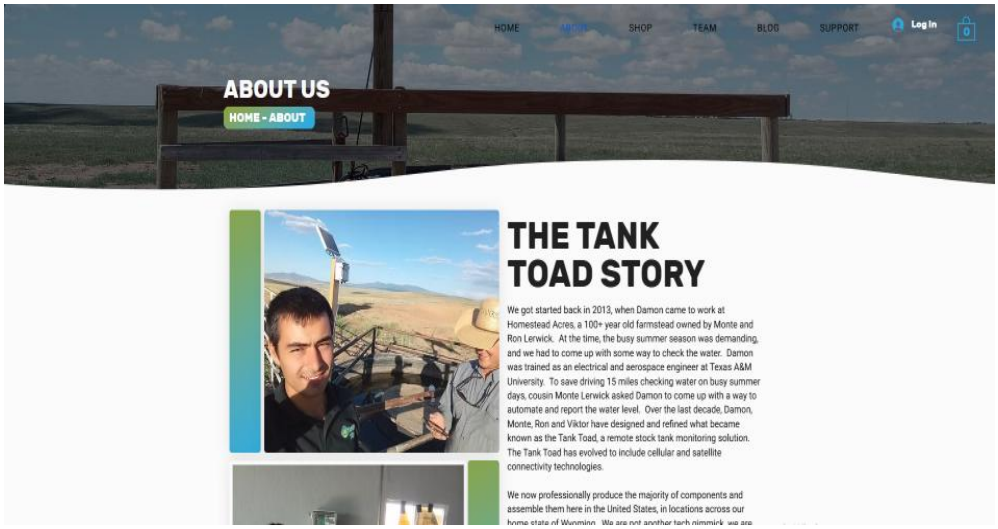

Рис. 1.1 Веб – сайт TankToad

Основною цінністю системи для фермерів є оперативність та автономність: замість ручних перевірок користувачі отримують сповіщення на телефон, можуть переглядати стан пристроїв на карті, керувати насосами та бачити історію змін. Однак до останнього часу інтерфейс взаємодії з системою був обмежений веб-кабінетом, що не відповідав польовим умовам використання.

Проблема, що потребувала вирішення, полягала у відсутності зручного, сучасного та функціонального мобільного застосунку з розширеними можливостями для фермерів — користувачів системи. Крім того, існуюча серверна логіка не мала достатньої модульності, адаптивності до мобільного клієнта та підтримки розширеного API, який би дозволяв працювати з пристроями, ролями, сповіщеннями, історією, фото та активацією через QR-коди.

У межах цієї кваліфікаційної роботи було поставлено завдання створити нову серверну логіку, адаптовану для мобільного застосунку, з наступними технічними цілями:

- реалізувати гнучку REST API-структуру для роботи з пристроями, профілем користувача, ролями доступу, сповіщеннями та фотоданими;
- забезпечити підтримку push-сповіщень про критичні події (низький рівень води, помилки пристроїв, розряд батареї тощо);
- впровадити механізм активації пристроїв через QR-коди з контролем валідності й історією активацій;

Таким чином, предметна область цієї роботи охоплює інтеграцію IoT-системи TankToad з мобільним застосунком шляхом розробки повнофункціонального серверного API на платформі .NET. Практична значущість полягає у підвищенні доступності та зручності взаємодії фермерів із системою моніторингу в реальних умовах використання.

1.2 Аналіз вимог до проекту

Розробка серверної частини мобільного застосунку для системи TankToad потребувала чіткого визначення як функціональних, так і нефункціональних вимог. Основною цільовою аудиторією є фермери та технічні працівники аграрних підприємств, які використовують систему в польових умовах — часто за відсутності стабільного інтернету, електромережі та технічної підтримки. Саме ці обставини безпосередньо вплинули на формування вимог до архітектури API, способів обробки даних та взаємодії з пристроями.

Загальне технічне завдання передбачало створення серверної логіки, здатної стабільно взаємодіяти з мобільним додатком на пристроях з операційною системою Android та iOS. Сервер мав забезпечити повну підтримку всіх запитів з мобільного клієнта. Особлива увага приділялася оптимізації API для швидкої обробки критичних запитів — наприклад, отримання поточних показників пристрою або зміни режиму його роботи.

Оскільки система використовується у фермерському середовищі, важливо було реалізувати підтримку офлайн-режиму: сервер мав повертати необхідний обсяг структурованих даних, які мобільний клієнт міг би кешувати та використовувати локально у разі втрати інтернет-з’єднання.

Додаток поділяється на такі функціональні модулі:

- Керування фотографіями — отримання, перегляд та фільтрація архівних зображень із пристроїв (камер) за датою.
- Керування пристроями — додавання, оновлення, а також зміна їхніх налаштувань і режимів роботи.
- Отримання інформації — доступ до актуальних показників пристроїв: рівень води, заряд акумулятора, статус сенсорів тощо.
- Профіль користувача — перегляд та редагування особистих даних, аватару, контактів і ролей у системі.

- Система QR-кодів — активація пристроїв за унікальними кодами, перевірка статусу пристрою, генерація посилань.

Вимоги до кожного з функціональних модулів:

Керування фотографіями: Серверна частина повинна забезпечити можливість отримання зображень з камер пристрою за вказаний часовий проміжок. Необхідно реалізувати фільтрацію за датою, сортування за часом знімку, а також генерацію прямих URL-посилань на зображення. Для захисту персональних даних передбачено авторизацію доступу до фото лише для користувачів, які мають права на відповідний пристрій. Важливо, щоб сервер коректно обробляв запити навіть при великій кількості знімків, а також дозволяв використовувати дані в офлайн-режимі на клієнтському боці через попереднє кешування.

Керування пристроями: Серверна логіка повинна підтримувати повний набір операцій для взаємодії з пристроями : отримання списку пристроїв, перегляд детальної інформації, зміна режимів роботи (наприклад, автоматичний або ручний режим для насосів), оновлення назв, координат і статусів. Для кожного типу пристрою має повертатися відповідна структурована інформація. Важливо реалізувати контроль прав доступу до кожної операції відповідно до ролі користувача. Дані повинні бути придатні до збереження в кеші клієнта для офлайн-доступу. Отримання інформації:Сервер повинен забезпечити мобільному застосунку доступ до актуальних показників пристроїв у реальному часі. Це включає рівень води, заряд акумулятора, статус сенсорів, поточний режим роботи, а також можливі помилки чи попередження. Для сенсорів рівня води має бути реалізовано повернення даних за обраний період у форматі, придатному для побудови графіків. Також необхідно реалізувати окрему логіку для різних типів пристроїв — насосів, камер, сенсорів — з урахуванням специфіки кожного. Дані повинні бути придатні до кешування на клієнті для забезпечення офлайн-перегляду останніх отриманих значень.

Профіль користувача: Серверна частина повинна надавати API для отримання та оновлення профілю користувача. До обов'язкових даних належать ім'я, прізвище, email, номер телефону, аватар та список ролей у системі. Контроль доступу до даних повинен здійснюватися на основі авторизованої сесії та ролі користувача. З метою безпеки зміна чутливих даних (наприклад, email чи номеру) повинна супроводжуватися перевіркою.

Система QR-кодів: Функціональність QR-кодів у системі TankToad виконує ключову роль при додаванні нових пристроїв. Кожен девайс у системі пов'язаний із унікальним QRKey, який зберігається у базі даних та використовується для генерації одноразового QR-коду. Цей код створюється через веб-інтерфейс компанії Meadowlark Solutions — зазвичай працівником, який встановлює пристрій на об'єкті клієнта.

QR-код містить шифровану інформацію, що дозволяє додатку швидко ідентифікувати пристрій та його тип, а також здійснити активацію у профілі користувача. Після сканування коду в мобільному застосунку відбувається перевірка доступності пристрою, актуальності QRKey, а також стану прив'язки. Якщо пристрій ще не активований — користувач бачить кнопку для завершення прив'язки.

У випадку, якщо застосунок не встановлений на телефоні користувача, QR-код веде на динамічну веб-сторінку, яка визначає модель пристрою і тип ОС, після чого перенаправляє користувача на відповідну сторінку завантаження: App Store для iOS або Google Play для Android. Цей маршрут реалізований через систему адаптивної маршрутизації з урахуванням User-Agent браузера та типу пристрою в базі.

Таким чином, QR-коди забезпечують швидке підключення пристроїв, безпеку доступу та просту інструкцію для нових користувачів — усе це покладено на окрему серверну логіку та пов'язане з мобільним і веб-фронтом.

Продуктивність: API-сервер має забезпечувати високу швидкодію навіть при одночасному обслуговуванні великої кількості користувачів і пристроїв. Ключові запити (отримання статусу пристроїв, графіки, сповіщення) повинні оброблятись у межах 100–300 мс. Базується це на оптимізованих LINQ-запитах, правильному використанні індексів у базі даних, а також обмеженні обсягу відповідей через пагінацію, сортування та фільтрацію на серверному рівні.

Надійність: Система повинна стійко працювати в умовах поганого інтернету та обривів з'єднання. На рівні бекенду передбачена обробка всіх критичних винятків, повернення стандартизованих помилок із чіткими кодами (400, 403, 500). Кожен контролер перевіряє коректність вхідних параметрів, що мінімізує ризик "тихих" помилок.

Безпека: Уся взаємодія з API відбувається виключно через HTTPS, із суворою перевіркою JWT (Bearer-токенів). Дані користувача ізоляційно обмежені відповідно до його ролі в системі — навіть якщо токен валідний, сервер перевіряє, чи має користувач право доступу до конкретного пристрою, фотографії або QR-даних. Зміна чутливих даних (email, номер телефону, пароль) потребує додаткового підтвердження. Додатково сервер виконує перевірку на неактивні або повторно використані QRKey, щоб уникнути зловживань. Логіка підключення нових пристроїв винесена в окремий шар, із захистом від повторної активації.

Масштабованість: Архітектура бекенду проєктувалася з урахуванням майбутнього зростання. Завдяки модульній побудові контролерів та сервісів, систему легко розширити новими типами пристроїв, ролей або навіть цілими функціональними модулями. Всі сутності в базі мають чіткі зовнішні ключі, що забезпечує консистентність при розширенні структури. API не жорстко пов'язаний із мобільним клієнтом, тому може використовуватись паралельно у веб-інтерфейсі або зовнішніми сервісами.

1.3 Технічне завдання до проекту

Технічне завдання (ТЗ) є ключовим документом у процесі створення серверної частини мобільного застосунку. Воно визначає цілі, функціональні модулі, нефункціональні вимоги, обмеження та технічні умови, які мають бути враховані при реалізації бекенд-архітектури.

Завдяки наявності чіткого ТЗ вдалося формалізувати очікування щодо поведінки API, обробки даних пристроїв, інтеграції з мобільним клієнтом та безпечної взаємодії з користувачами системи.

Головна мета технічного завдання — визначити повний обсяг робіт, необхідний для побудови стабільної серверної інфраструктури, яка підтримує мобільний додаток у режимі реального часу, враховує обмеження польового використання та забезпечує масштабованість.

Документ ТЗ містить перелік функціональних модулів (керування пристроями, отримання фото, сповіщення, QR-система), вимоги до продуктивності, безпеки, кешування та авторизації. Його структура побудована таким чином, щоб забезпечити прозорість взаємодії між замовником та розробником, уникнути невідповідностей у реалізації та спростити перевірку відповідності програмного продукту заявленим вимогам під час тестування.

У процесі підготовки технічного завдання було враховано низку практичних і технічних факторів, що безпосередньо вплинули на архітектуру серверної частини. Формування вимог відбувалося на перетині трьох ключових напрямів:

- побажання та конкретні запити компанії-замовника;
- потребу в інтеграції з уже функціонуючою екосистемою TankToad;
- вимоги щодо подальшої масштабованості системи та розширення її функціоналу;

- умови реального використання додатку фермерами без технічної підготовки та в середовищі з нестабільним інтернетом.

Технічне завдання охоплює всі ключові аспекти системи: модулі взаємодії з пристроями, сценарії дій користувача, логіку ролей, вимоги до авторизації, обробки карт, сповіщень, а також майбутнє масштабування функціоналу.

На його основі була реалізована серверна логіка для мобільного застосунку, яка забезпечує повноцінну підтримку функцій у форматі, зручному для щоденного використання в польових умовах.

Серверну частину мобільного застосунку було реалізовано у форматі REST API на базі .NET-платформи з використанням Entity Framework Core. Такий підхід забезпечив зручну роботу з реляційною базою даних та дозволив реалізувати стабільну, гнучку й масштабовану серверну логіку для мобільного клієнта. Однією з ключових вимог проєкту було створення нового функціоналу, не порушуючи цілісності вже існуючої інфраструктури, що активно використовувалась у веб-системі TankToad.

Перед початком реалізації була проведена оцінка наявної серверної структури — вона включала велику кількість контролерів, модулів і сервісів, що відповідали за різні аспекти роботи з пристроями, користувачами, телеметрією, логуванням тощо. Моя задача полягала в тому, щоб інтегрувати нові мобільні ендпоінти у вже працюючу систему таким чином, аби не створити конфліктів, дублювання логіки чи порушень авторизації.

Для цього всі нові API-контролери були побудовані з урахуванням існуючої структури та сервісного шару.

Також було забезпечено повну відповідність REST-принципам: кожен контролер працює за чітким маршрутом, використовує відповідні HTTP-методи (GET, POST, PUT), повертає дані у форматі JSON та обробляє запити у відповідності до ролі користувача.

Таким чином, реалізація серверної частини мобільного застосунку стала результатом точної інтеграції нового функціоналу у вже сформовану систему, з урахуванням усіх архітектурних обмежень, авторизаційних механізмів та вимог до продуктивності.

У процесі реалізації технічного завдання велика увага приділялася забезпеченню безперервної синхронізації між мобільним клієнтом та сервером. Це передбачало підтримку актуальних статусів пристроїв, передачу змін конфігурації в реальному часі, а також реакцію системи на події — наприклад, зміну рівня води, розряд батареї чи вихід пристрою з ладу. Усі ці сценарії мали бути чітко описані в ТЗ, з обов'язковими умовами спрацювання, шаблонами push-сповіщень і логікою прав доступу.

Також одним з критично важливих аспектів, врахованих у ТЗ, стала можливість роботи в умовах обмеженого інтернету. Це передбачало підтримку кешування ключових даних, асинхронної обробки запитів та уникнення повної залежності від постійного з'єднання. Завдяки цьому система стала стійкою до збоїв і придатною для використання в сільськогосподарському середовищі, де іноді доступ до мережі є нестабільним або обмеженим.

РОЗДІЛ ІІ. АРХІТЕКТУРА ТА СТРУКТУРА СЕРВЕРНОЇ ЧАСТИНИ СИСТЕМИ

2.1 Проектування програмного продукту

Проєктування серверної частини мобільного застосунку TankToad стало критично важливим етапом у реалізації всього програмного комплексу. Основною задачею було створити гнучку, безпечну та масштабовану API-структуру, яка б забезпечила стабільну взаємодію мобільного клієнта з існуючою екосистемою пристроїв та бекенд-сервісів. Загальна модель процесу реалізації, що включає етапи технічного завдання, розробки та тестування, представлена на рисунку 2.1.

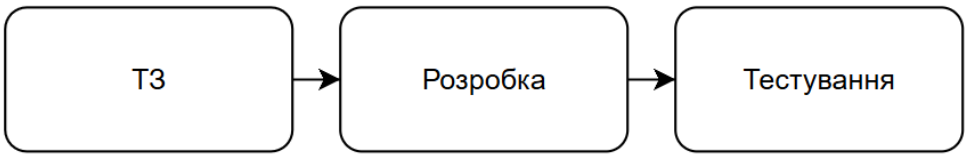

Рис. 2.1 Модель розробки

Розробка велась з урахуванням специфіки середовища використання — віддалені фермерські регіони зі складними умовами зв’язку, де критично важливою є швидкість відповіді API, обробка статусів у режимі реального часу, та підтримка push-сповіщень.

Архітектура була спроєктована з урахуванням існуючих технічних рішень компанії Meadowlark Solutions. Це передбачало інтеграцію в уже сформовану систему, що включала десятки контролерів, сервісів, модулів безпеки та логування. Кожен новий ендпоінт повинен був не лише відповідати загальним принципам REST, а й органічно вбудовуватись у

логіку вже діючих модулів, не створюючи конфліктів у доступі чи дублювання коду.

Під час проєктування було враховано ряд технічних обмежень, що сформувались у ході експлуатації системи TankToad: вимоги до авторизації, кешування, прив'язки пристроїв до користувачів, рольової моделі, та фіксації статусів для системи сповіщень. Кожен із цих аспектів було закладено у фундамент API-структури.

У цьому розділі буде розглянуто логіку побудови архітектури серверної частини, структуру модулів, використані підходи до обробки запитів, забезпечення безпеки, ролей доступу, взаємодії з мобільним клієнтом і принципи, якими я керувався у розробці.

Було реалізовано внутрішню систему залежностей між сервісами, де кожен модуль, відповідальний за конкретну функцію (наприклад, камери чи QR-активація), міг працювати автономно, але при цьому взаємодіяв із загальними компонентами, як-от логування чи система сповіщень.

2.2 Побудова архітектури програми

Архітектура програмного забезпечення — це структурна основа всієї системи, що визначає компоненти, логіку їх взаємодії, способи обміну даними та розподіл відповідальностей між модулями. У рамках даного проєкту було спроєктовано серверну архітектуру для мобільного застосунку TankToad, яка інтегрується в уже існуючу екосистему замовника.

Ключовим завданням було створити масштабовану, логічно розділену API-структуру, яка б забезпечила обробку запитів у реальному часі, підтримку статусів пристроїв, push-сповіщень, кешування, прив'язки до профілю та QR-автентифікації. Для досягнення цієї мети архітектура була побудована за принципом Separation of Concerns, де кожен

функціональний блок системи реалізовано як окремий контролер зі своєю бізнес-логікою.

Основні архітектурні модулі серверної частини :

Керування фотографіями: Функціональний модуль керування фотографіями відповідає за обробку зображень, отриманих із пристроїв типу "камера". Його основною задачею є фільтрація архіву фото за часовими параметрами, формування списку доступних кадрів та передача їх у клієнтському форматі. Усі запити обробляються через MobileCameraController, який делегує бізнес-логіку до відповідного сервісу. Система підтримує отримання зображень за конкретний період часу, а також дозволяє обробляти результати з урахуванням прав користувача. Зберігання файлів реалізовано або через хмарне сховище, або через базу даних — залежно від типу пристрою та історичної глибини. Схематично обробку запиту на отримання фотографій з камери зображено на рисунку 2.2.

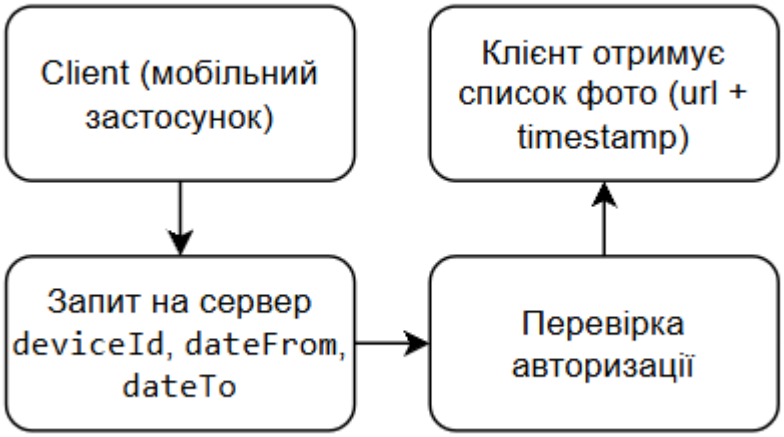

Рис. 2.2 Обробка запиту на отримання фотографій з камери

Керування пристроями: Модуль керування пристроями охоплює всі базові CRUD-операції для девайсів системи TankToad. Основна логіка реалізована у MobileDeviceController, який дозволяє створювати нові пристрої, редагувати існуючі, оновлювати їхні назви, геокоординати,

режими роботи (автоматичний/механічний), а також перевіряти їхній статус. Контролер використовує поточний авторизований контекст користувача для фільтрації доступу до пристроїв. Крім того, модуль інтегровано з push-системою — у разі виявлення критичних подій (низький рівень води, збій сенсора), автоматично генеруються сповіщення. Схематично обробку запиту на отримання пристроїв користувача зображено на рисунку 2.3.

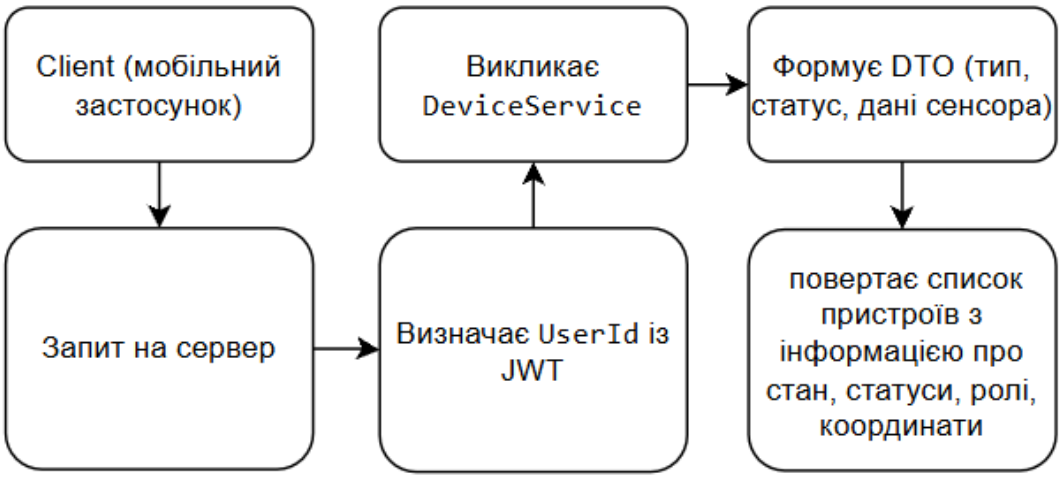

Рис. 2.3 Обробка запитів до пристроїв користувача

Отримання інформації: Модуль збору інформації відповідає за надання мобільному клієнту стислої, оптимізованої інформації про стан пристроїв. Це включає рівень заряду акумулятора, статус сенсора води, поточні показники тощо. Відповідні API-запити виконуються безпосередньо через контролер пристроїв, а в сервісній частині формується полегшена модель (DeviceStatusDto), що дозволяє зменшити трафік і прискорити завантаження в клієнті. Така логіка дозволяє отримувати дані у реальному часі без перевантаження системи.

Окрім формування оптимізованої DTO-моделі, система також враховує поточну роль користувача. Наприклад, у разі делегованого доступу до пристрою з обмеженими правами, обсяг доступних даних

автоматично обмежується на рівні сервісної логіки. Це дозволяє уникати витоку конфіденційної інформації, зберігаючи водночас основну функціональність.

Важливим компонентом реалізації стала підтримка періодичного оновлення статусу пристроїв. Для цього мобільний клієнт відправляє регулярні запити (наприклад, раз на 30 секунд), а сервер відповідає тільки у випадку зміни даних або надає флаг стабільного стану. Такий підхід знижує навантаження на мережу і покращує продуктивність застосунку в умовах слабкого інтернету.

На рівні бази даних реалізовані індексації за deviceId та timestamp, що дозволяє швидко знаходити найактуальніші записи, навіть при великій кількості пристроїв у системі. Завдяки цьому модуль збору інформації демонструє хорошу масштабованість, дозволяючи одночасно обслуговувати десятки тисяч запитів без затримки у відповіді.

Загалом, модуль є одним із ключових елементів взаємодії користувача з системою TankToad, оскільки саме на нього покладається завдання забезпечити реальний стан польових пристроїв у максимально доступній формі. Схематично обробку запиту на отримання поточних показників пристрою зображено на рисунку 2.4.

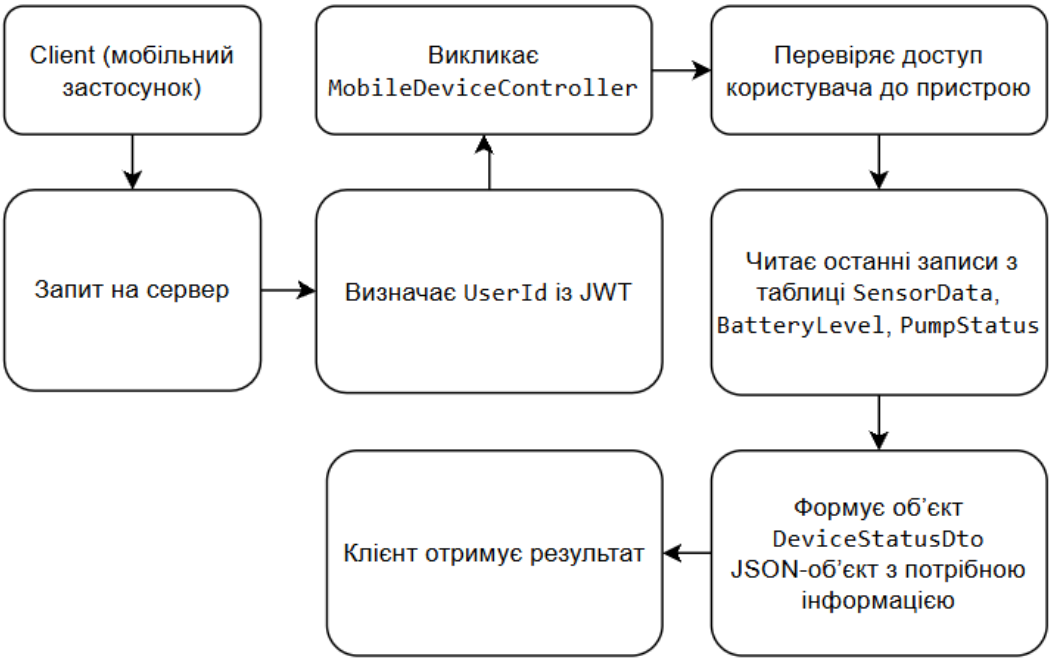

Рис. 2.4 Отримання поточних показників пристрою

Профіль користувача: Контролер MobileProfileController забезпечує можливість отримання та редагування профілю користувача. Це включає зміну аватара, оновлення контактної інформації, перегляд ролі у системі, а також список девайсів, до яких користувач має доступ. Особлива увага приділена безпеці — усі запити перевіряються на відповідність авторизованому токену. Крім того, у рамках профільного модуля реалізована логіка делегування прав доступу до пристроїв іншим користувачам, що критично для командної роботи на фермерських господарствах. У процесі реалізації було також впроваджено можливість асинхронного оновлення аватара з попередньою валідацією формату зображення та розміру файлу. Завантажені зображення обробляються окремим сервісом, після чого зберігаються або у файловій системі, або у хмарному сховищі. Для зручності мобільного користувача профільний модуль повертає усю необхідну інформацію одним компактним запитом, що дозволяє уникати додаткових звернень до серверу після авторизації. У відповідь включаються і базові налаштування користувача, і список

пристроїв із базовими параметрами доступу. Схематично обробку запиту на отримання даних профілю користувача зображено на рисунку 2.5.

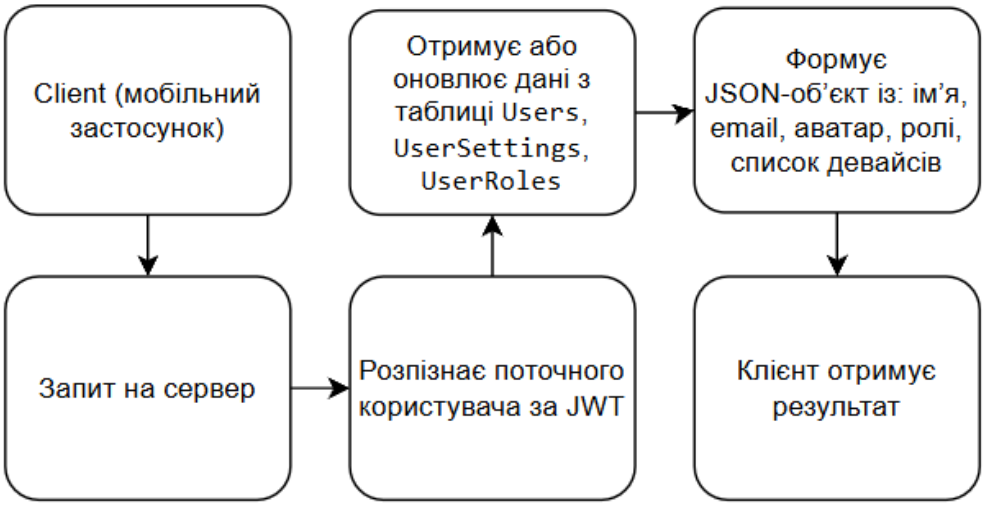

Рис. 2.5 Обробка запиту профілю користувача

Система QR-кодів: Модуль QR-кодів виконує ключову роль у процесі підключення нових пристроїв до профілю користувача. Його основне завдання — забезпечити швидку та безпечну активацію пристрою без необхідності ручного введення параметрів. Кожен девайс TankToad після реєстрації в адміністративній панелі отримує унікальний qrKey, який використовується для подальшої ідентифікації пристрою. На основі цього ключа генерується QR-код, що містить спеціальне посилання з закодованим маршрутом. Схематично маршрутизацію користувача через QR-код зображено на рисунках 2.6 та 2.7.

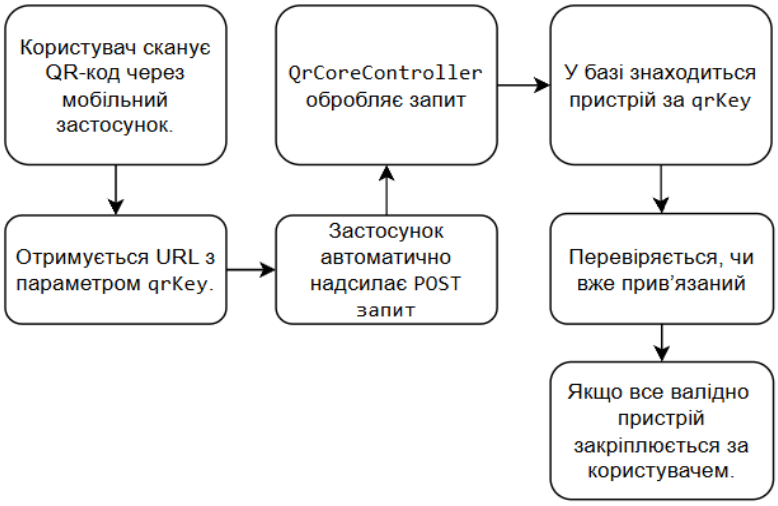

Рис. 2.6 Маршрутизація користувача через QR-код, якщо застосунок встановлено

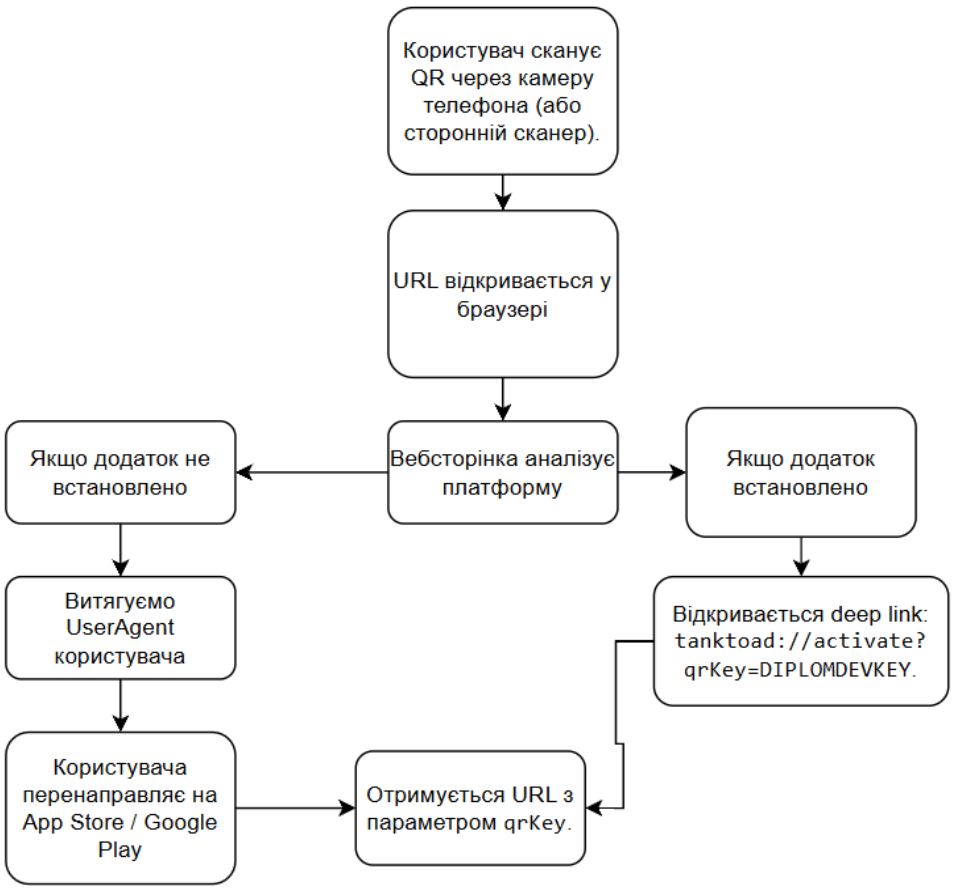

Рис. 2.7 Маршрутизація користувача через QR-код, якщо застосунок не встановлено

Серверна частина, представлена контролером QrCoreController (рис. 2.8), обробляє два ключових запити:

- Перевірка статусу пристрою — за URL /api/qr/check, який дозволяє клієнту визначити, чи вже активований пристрій, до кого він прив'язаний, а також його базові параметри.
- Активація — запит /api/qr/activate, який зв’язує пристрій із профілем користувача, що здійснює активацію. Перед активацією система перевіряє валідність QR-ключа, тип пристрою та можливість прив’язки.

Рис. 2.8 Обробка запиту API на сервері (QrCoreController)

Важливою особливістю архітектури є інтеграція з вебінтерфейсом: у випадку, коли користувач сканує QR-код із браузера, система перенаправляє його на спеціальну вебсторінку. Ця сторінка виконує функцію «моста»: вона автоматично визначає, чи встановлено мобільний застосунок TankToad. Якщо так — відкривається потрібний екран у додатку за допомогою deep link. Якщо ж ні — користувачеві пропонується встановити застосунок, після чого процес триває без втрати даних.

QR-система також враховує тип пристрою. Наприклад:

- Для камери користувач після активації потрапляє на сторінку перегляду фото.
- Для насоса — на екран керування режимами роботи.

- Для сенсора рівня води — на сторінку з графіками показників.

Цей підхід дозволяє зробити процес підключення інтуїтивно зрозумілим навіть для користувачів без технічної підготовки. Завдяки чіткому розділенню відповідальностей між вебінтерфейсом, мобільним застосунком і серверною частиною, вдалося досягти надійності, безпеки й зручності у використанні цієї функціональності в польових умовах.

2.2 Інструментальні засоби розробки

У процесі створення серверної частини мобільного застосунку TankToad було використано низку інструментів, які забезпечили гнучку архітектуру, масштабованість та зручну взаємодію з клієнтською частиною на Flutter. Вибір технологій обумовлювався як вимогами функціональності, так і реальними обмеженнями аграрного середовища — слабке покриття, потреба в офлайн-доступі, стабільність обробки даних у польових умовах.

Visual Studio 2022 — головне середовище розробки серверної логіки. Воно надало розширену підтримку .NET-проєктів, інструменти для налагодження, профілювання продуктивності та швидке перемикання між конфігураціями. Особливо корисними стали інтегровані можливості тестування REST API, керування NuGet-пакетами та підтримка розгортання на хмарні сервіси.

Microsoft SQL Server Management Studio (SSMS) — застосовувався для моніторингу бази даних, ручного виконання SQL-запитів, перегляду логів та відлагодження міграцій, згенерованих Entity Framework Core. Це дозволяло підтримувати цілісність даних, аналізувати індексацію та перевіряти продуктивність запитів.

ASP.NET Core (C#) — сучасна, високопродуктивна кросплатформна веб-фреймворк-платформа, яка забезпечує мінімальний overhead при високій навантаженості. Завдяки гнучкій побудові middleware-пайплайнів

вдалося налаштувати обробку помилок, аутентифікацію, логування та кешування в централізованому стилі.

Entity Framework Core (EF Core) — обрана ORM (Object-Relational Mapping) бібліотека. Вона значно скоротила обсяг рутинного SQL-коду, дозволила зручно працювати з LINQ-запитами, автоматично генерувати й застосовувати міграції бази даних, а також зберігати цілісність моделі при постійних змінах структури сутностей.

Додаткові бібліотеки та фреймворки

- QRCoder — легка, але потужна бібліотека для генерації QR-кодів. Завдяки їй реалізовано логіку генерації унікального посилання, що містить токен-підключення, маршрут активації та тип пристрою. Бібліотека підтримує налаштування рівня помилок, формату, розміру та кольору, що було корисно для адаптації під польові умови (роздруковані наклейки, яскраве світло).
- Microsoft.AspNetCore.Authentication.JwtBearer — реалізує повноцінну JWT-автентифікацію. Система побудована на основі claims-based авторизації, що дозволило реалізувати гнучку модель ролей (admin, technician, reader), фільтрацію доступу до ресурсів та керування сеансами без використання cookie.
- Newtonsoft.Json — незамінна бібліотека для серіалізації та десеріалізації JSON-структур. Дає змогу точно контролювати формат вхідних/вихідних даних, використовувати атрибути для налаштування форматування, і підтримує обробку складних вкладених об'єктів, null-значень і enum-полів.
- AutoMapper — дозволив зручно та безпечно мапити DTO в сутності бази даних та навпаки. Завдяки цьому зменшено дублювання логіки при формуванні відповідей для мобільного клієнта. Наприклад, Device → DeviceDto → DeviceStatusResponse.

- FluentValidation — для централізованої валідації моделей. Було створено десятки окремих класів перевірки, які гарантували цілісність вхідних запитів ще до обробки у бізнес-логіці, що дозволяло уникнути зайвих викликів БД або неочікуваних виключень.
- Swagger (Swashbuckle) — інструмент автогенерації документації до API. Завдяки йому замовник і мобільна команда могли самостійно переглядати та тестувати всі доступні ендпоінти без знання внутрішньої реалізації. Swagger UI інтегрується безпосередньо в проєкт і надає реальний механізм перегляду доступних методів.

Тестування та налагодження

- Postman — активно використовувався для ручного тестування API, створення тестових колекцій, сценаріїв з JWT-авторизацією, перевірки помилок, імітації поганого інтернет-з'єднання. Також дозволив протестувати реакцію сервера на помилки авторизації, невалідні параметри або відсутність доступу.
- Swagger UI — вбудована панель тестування, що надає повноцінний інтерфейс для взаємодії з API прямо в браузері. Допомагає як при розробці, так і для демонстрації клієнту.

Реалізація серверної частини була побудована за принципами SOLID, з чітким розділенням між контролерами, сервісами та репозиторіями. Такий підхід спростив не лише впровадження логіки, а й забезпечив можливість подальшого масштабування або рефакторингу з мінімальними ризиками.

РОЗДІЛ ІІІ. РЕАЛІЗАЦІЯ ПРОЕКТУ ПРОГРАМНОГО ПРОДУКТУ

2.2 Стек технологій

Для реалізації серверної частини програмного продукту TankToad було обрано сучасний технологічний стек, який забезпечив високу продуктивність, безпеку, масштабованість і зручність підтримки. Основу розробки склала кросплатформна платформа .NET Core, яка дозволила створити гнучке REST API для взаємодії з мобільним клієнтом. Завдяки своїй відкритості, підтримці Linux-серверів і високій швидкодії, ASP.NET Core ідеально підходить для побудови серверних систем, що працюють у режимі реального часу.

Код бекенду був написаний мовою **C#**, яка забезпечує типобезпеку, сучасний синтаксис і потужну підтримку асинхронного програмування. Завдяки цим перевагам, розробка логіки контролерів, сервісів, авторизації та обробки запитів проходила швидко та надійно, а підтримка коду залишалась простою навіть на пізніх етапах.

Для взаємодії з базою даних використовувався Entity Framework Core — об'єктно-реляційний мапер, який дозволяє будувати структуру бази на основі класів-сутностей. Його використання дозволило уникнути написання SQL-запитів вручну, пришвидшити процес міграцій і забезпечити цілісність даних між моделями та таблицями.

Взаємодія клієнта з сервером була реалізована через REST API, що базується на стандартних HTTP-методах (GET, POST, PUT, DELETE). Усі запити й відповіді формуються у форматі JSON, що забезпечує просту та передбачувану інтеграцію з Flutter-додатком. Це дало змогу реалізувати

десятки ендпоінтів, зокрема для роботи з профілем, пристроями, фотоархівом, статусами та QR-активацією.

Ключовим компонентом системи безпеки стала JWT-авторизація (Bearer Tokens). Кожен користувач після логіну отримує підписаний токен, який зберігає дані про його ідентифікатор, роль, час дії сесії. Таким чином, кожен запит до приватного ресурсу супроводжується перевіркою токена, що забезпечує контроль доступу та запобігає несанкціонованому втручанню.

Система контролю версій проєкту реалізована за допомогою Git, що дозволило гнучко керувати гілками, відслідковувати зміни та швидко повертатись до попередніх стабільних версій. Для розміщення репозиторію, code review та командної взаємодії було використано GitHub — платформу, яка також надала зручні інструменти для документації, issue-трекінгу та автоматизації через GitHub Actions.

Допоміжними засобами, які активно використовувалися в процесі тестування, стали Swagger UI та Postman. Swagger забезпечив автоматичну генерацію документації до API та надав інтерфейс для інтерактивної перевірки запитів прямо з браузера. Postman, у свою чергу, використовувався для глибшого функціонального тестування, роботи з колекціями запитів, авторизації та перевірки навантаження на окремі ендпоінти.

Роль бази даних у проєкті виконувала Microsoft SQL Server, яка інтегрується з EF Core і забезпечує потужну індексацію, стабільну роботу з транзакціями, швидкий пошук за deviceId і timestamp, що критично для обробки польових даних.

Таким чином, вибраний стек технологій дозволив створити надійний, масштабований і зручний у підтримці серверний застосунок, який ефективно обслуговує мобільних користувачів навіть у складних умовах польового використання.

2.3 Архітектура серверного застосунку

Опис архітектури сервісного рівня проєкту TankToad заслуговує на окрему увагу, оскільки саме тут реалізована вся бізнес-логіка, яка забезпечує динамічну взаємодію мобільного застосунку з сервером, обробку push-сповіщень та зберігання користувацьких даних. На відміну від монолітних рішень, у даному проєкті було реалізовано чітке розділення відповідальностей між контролерами та сервісами: контролери відповідають лише за маршрутизацію й авторизацію, тоді як уся логіка, взаємодія з БД та зовнішніми сервісами делегується саме сервісному шару. Така архітектурна модель була закладена на старті розробки проєкту й дотримана у всіх модулях, про що свідчать як структура проєкту, так і підключення інтерфейсів у контролерах через механізм залежностей (Dependency Injection).

Почнімо з FirebasePushService, який виконує одну з найкритичніших функцій системи — надсилання push-сповіщень. Цей сервіс був реалізований з урахуванням сучасних вимог до міжплатформових push-технологій, зокрема через інтеграцію з Firebase Cloud Messaging (FCM). Клас FirebasePushService реалізує інтерфейс IFirebasePushService, що дозволяє легко масштабувати логіку або змінювати постачальника push-сповіщень у майбутньому без переписування контролерів. Завдяки параметрам конфігурації, шлях до сервісного акаунта Firebase ідентифікується автоматично, а авторизація до FCM API здійснюється за допомогою OAuth 2.0, де генерується access token з відповідними правами. Це забезпечує як безпечне з'єднання, так і динамічний доступ до ресурсу без необхідності зберігати токени у відкритому вигляді.

Формування push-повідомлення у SendPushAsync враховує особливості як Android, так і iOS, надаючи можливість керувати пріоритетом, звуком, каналами сповіщень і навіть значками (badge)

застосунку. Такий підхід гарантує, що повідомлення будуть коректно відображатися на обох платформах, незалежно від їхньої версії. Крім того, одразу після відправки push-повідомлення, відповідна інформація про сповіщення зберігається в базу даних через контекст _context, що забезпечує повний журнал подій для подальшого аудиту, перегляду або статистичного аналізу.

Ще один важливий метод — SendPushToDeviceUsersAsync, який реалізує розсилку повідомлень усім користувачам, що мають доступ до конкретного пристрою. Для цього сервіс звертається до таблиці ролей, зв'язаної через UserRoleTable з користувачами та пристроями, і отримує унікальні токени з таблиці FsmTokens. Така модель дозволяє адресно інформувати лише тих користувачів, яким дійсно важлива подія — наприклад, помилка сенсора, низький рівень води або втрата зв'язку з пристроєм. Варто зазначити, що метод працює асинхронно, а це забезпечує хорошу масштабованість, дозволяючи відправити тисячі повідомлень без блокування основного потоку виконання.

Супутнім сервісом, який реалізує допоміжну функцію — збереження токенів для подальших розсилок, є FSMTokenService. Він має простий, але надзвичайно важливий метод SaveTokenAsync, який додає новий токен лише у випадку, якщо такого запису ще не існує. Цей механізм дозволяє уникати дублювання токенів, що не лише зменшує розмір бази даних, а й підвищує ефективність розсилки, оскільки однакове повідомлення не буде надіслане двічі одному користувачу. Важливою деталлю є те, що сервіс орієнтований на інтеграцію з мобільним клієнтом, який надсилає токен Firebase одразу після авторизації — і саме цей момент використовується для виклику SaveTokenAsync.

З погляду архітектури, обидва сервіси реалізовано з повним дотриманням принципів інверсії залежностей та SOLID. Усі залежності ін'єктуються через конструктори, що дозволяє зручно тестувати логіку,

підключати мок-сервіси для юніт-тестів і конфігурувати поведінку через Startup.cs без прив’язки до конкретних реалізацій. Саме ці сервіси активно використовуються в контролерах NotificationsController, MobileDeviceController та QrCoreController, де є потреба в надсиланні повідомлень або реєстрації токенів.

Цікаво також відзначити, що реалізація FirebasePushService демонструє відмінну інтеграцію з зовнішнім API Google, що є прикладом правильно організованої міжсервісної взаємодії. Це особливо важливо в контексті аграрного сектору, де сповіщення про технічні несправності або критичні стани пристроїв повинні надходити негайно, навіть за відсутності стабільного інтернету на стороні клієнта. Впроваджена система push-розсилки дозволяє знизити залежність від періодичних запитів клієнта і забезпечує реактивність застосунку.

У підсумку можна сказати, що сервісний рівень проєкту TankToad виконує не просто допоміжну функцію, а є його ядром, через яке проходять усі ключові сценарії: реєстрація токенів, сповіщення, збереження подій, асинхронна взаємодія з клієнтом і розширювана логіка обробки подій. Структура з чітким розподілом сервісів, інтерфейсів і залежностей дозволяє масштабувати рішення, адаптувати його під різні категорії пристроїв і забезпечити безперебійну роботу застосунку навіть за умов високого навантаження.

3.3 Функціональне призначення об’єктів програмного продукту

Розроблена серверна частина мобільного програмного комплексу TankToad реалізує повний цикл взаємодії з клієнтською частиною через систему REST-запитів. Кожен з контролерів, створених у межах проєкту, відповідає за певну бізнес-область застосунку. Функціональність контролерів реалізована за допомогою чітко структурованих методів, які

викликають сервіси через інтерфейси, ізоляційно від бізнес-логіки. Такий підхід забезпечує чистоту архітектури, простоту супроводу та зручність масштабування функціоналу.

Одним із базових компонентів став MobileDeviceController, який обробляє запити, пов'язані з пристроями, що підключаються до системи TankToad. Він реалізує функції отримання списку доступних пристроїв для користувача, створення нового пристрою, оновлення назв, координат, режимів роботи, перевірки поточного статусу, а також взаємодії з push-сервісами для надсилання сповіщень при виявленні критичних змін (низький рівень води, помилка сенсора, низький заряд акумулятора тощо). Методи контролера використовують параметри авторизації, витягуючи інформацію про користувача з JWT-токену, щоб гарантувати безпеку та коректність доступу до пристроїв.

Основні методи контроллера:

- Отримання списку пристроїв користувача (PostUserDevicesAsync) – за допомогою нього ми дістаємо список девайсів користувача, тут враховуються статуси девайси і ролі користувача, а також підтримується пагінація.

```
public async Task<ActionResult<object>>
PostUserDevicesAsync(DeviceListIncomingObj incomingObj)
{
      List<DeviceListOutcomingObj> devices = await
_context.UserInvitationTable
        …
        .Select(item => new DeviceListOutcomingObj
        {
            Id = item.DeviceId,
            DeviceType = GetDeviceType(item.Device),
            DeviceNickname = item.Device.DeviceNickname,
            …
        })
        .ToListAsync();

    var query = _context.DeviceSettings
        .Include(item => item.UserList)
              …
        : true);
       var count = await query.CountAsync();
       var _devices = await query
        .Select(item => new MobileDevicDBO
        {
            Id = item.Id,
            DeviceNickname = item.DeviceNickname,
            …
            .Where(elem => elem.UserId == getCurrentUserId)
            .Select(elem => elem.Role)
            .Distinct()
            .ToList()
        })
        .OrderByDescending(item => item.Id)
        .Skip(incomingObj.Paginator.GetSkip)
        .Take(incomingObj.Paginator.PageSize)
        .ToListAsync();
    devices.AddRange(_devices
        .Select(model => new DeviceListOutcomingObj
        {
            Id = model.Id,
            DeviceType = model.DeviceType,
            DeviceNickname = model.DeviceNickname,
            …
        })
        .ToList());
    return Ok(new
    {
        PageNumber = incomingObj.Paginator.Page,
        PageSize = incomingObj.Paginator.PageSize,
        TotalRecords = count,
        TotalPages = (int)Math.Ceiling(count /
```

```
(double)incomingObj.Paginator.PageSize),
        Data = devices
    });
}
```

- Відображення пристроїв на карті (PostUserDevicesMapAsync) — метод, який повертає список пристроїв користувача з геокоординатами (широтою та довготою), типом, статусом та ролями. Використовується для побудови інтерактивної карти в мобільному застосунку. Фільтрує дані за ID власників (ownerId) та враховує авторизацію користувача. Завдяки цьому на карту потрапляють лише доступні авторизованому користувачу пристрої, і кожен пристрій виводиться з актуальним статусом, що дозволяє оперативно оцінити ситуацію в польових умовах.

```
public async Task<ActionResult<object>>
PostUserDevicesMapAsync(DeviceMapIncomingObj incomingObj)
  {
      var query = _context.DeviceSettings
          .Include(item => item.UserList)
          …
          : true);

      var _devices = await query
          .Select(item => new MobileDevicDBO
          {
              Id = item.Id,
              DeviceNickname = item.DeviceNickname,
              Status = item.Status,
              Longitude = item.Longitude,
              Latitude = item.Latitude,
            …
          })
          .ToListAsync();
      var devices = _devices
          .Select(item => new
          {
              item.Id,
              item.DeviceType,
              item.DeviceNickname,
              item.Longitude,
              item.Latitude,
              item.CurrentRoles,
              Status = GetStatus(item)
          })
```

```
            .ToList();

        return Ok(devices);
    }
```

- Зміна режиму роботи насоса (PumpAutoModeChanger) — метод, що дозволяє змінити режим роботи насоса (автоматичний або ручний), за умови, що поточний користувач має відповідні права. Перед внесенням змін перевіряється, чи пристрій дійсно є насосом, і чи новий режим не збігається з поточним. Успішна зміна режиму супроводжується підтвердженням та відповіддю з актуальним статусом. Метод необхідний для дистанційного керування обладнанням у системі.

```
public async Task<IActionResult> PumpAutoModeChanger(int id, int userListId,
bool mode)
{
    var user = await _context.UserTable.FindAsync(userListId);
    if (user == null)
    {
        return NotFound("User not found");
    }
    var device = await _context.DeviceSettings
       …
        .FirstOrDefaultAsync();
    if (device == null)
    {
        return NotFound("No device found for the given Id");
    }

    if (!(device.CameraEnabled == true && device.CameraCapableSetting == true)
&& device.MasterId != null)
    {
       ...
    }

        device.Enabled = mode;
        await _context.SaveChangesAsync();

        return Ok(new
        {
            Message = $"Changed to {(mode ? "Automatic" : "Manual")}",
            DeviceId = device.Id,
            NewMode = mode ? "Automatic" : "Manual"
        });
    }
```

```
    else
    {
        return BadRequest("The provided device is not a pump");
    }
}
```

- Оновлення рівнів вмикання/вимикання насоса (PumpWellOLevelChanger) — метод для налаштування рівнів води, при яких насос має автоматично вмикатися або вимикатися. Користувач передає нові значення, які перераховуються з урахуванням коефіцієнтів конверсії (DataConversionConstantA, B) та перевіряються на відповідність межам сенсора. Така логіка дозволяє інтуїтивно встановлювати допустимі рівні навіть у складних конфігураціях. Метод застосовується виключно до насосів, які прив'язані до авторизованого користувача.

```
public async Task<IActionResult> PumpWellOLevelChanger(int id, int userListId,
double WellOnLevel, double WellOffLevel)
{

    var user = await _context.UserTable.FindAsync(userListId);
    if (user == null)
    {
        return NotFound("User not found");
    }
    var device = await _context.DeviceSettings
         …
        .FirstOrDefaultAsync();

    if (device == null)
    {
        return NotFound("No device found for the given Id");
    }

    if (!(device.CameraEnabled == true && device.CameraCapableSetting == true)
&& device.MasterId != null)
    {
       …
        return Ok(new
        {
            Message = "Pump well levels successfully updated.",
            DeviceId = device.Id,
            NewWellOnLevel = device.WellOnLevel,
            NewWellOffLevel = device.WellOffLevel
        });
    }
    else
```

```
    {
        return BadRequest("The provided device is not a pump");
    }
}
```

- Отримання детальної інформації про пристрій (GetDevicesById) — метод, який повертає повну структуру даних про конкретний пристрій, включаючи його тип, геолокацію, статус, напрям вимірювання, статуси води та батареї, рівні тривог, а також перетворені показники рівня води. Якщо пристрій є насосом, то додається інформація про master-пристрій, а також рівні вмикання/вимикання. Метод активно використовується для побудови сторінки детального перегляду в мобільному застосунку, адаптуючи відповіді під тип пристрою та права користувача.

```
public async Task<ActionResult<MobileDevicDto>> GetDevicesById(int id)
 {
     if (id <= 0) return NotFound("Device not found");
       var _device = await _context.DeviceSettings
         …
)
         .Select(model => new MobileDevicDBO
         {
             Id = model.Id,
             DeviceType = GetDeviceType(model),
             DeviceNickname = model.DeviceNickname,
            …
         })
         .FirstOrDefaultAsync();
     if (_device == null) return NotFound("Device not found");
     double batteryVoltage = _device.BatteryVoltage * 0.0041503906;
     double batteryPercentage = Math.Clamp((batteryVoltage - 2.8) / (4.05 -
2.8) * 100, 0, 100);

     var device = new MobileDevicDto
     {
         Id = _device.Id,
         DeviceType = _device.DeviceType,
```

```
            DeviceNickname = _device.DeviceNickname,
           …
        };

        if (_device.DeviceType == "Pump")
        {
            if (_device.MasterId.HasValue)
            {
                device.MasterSerialNumber = await
GetMasterIdName(_device.MasterId.Value);
            }
            else
            {
                device.MasterSerialNumber = "N/A";
            }
            device.WellOnLevel = Math.Round(((float)_device.WellOnLevel -
_device.DataConversionConstantB) * _device.DataConversionConstantA, 2);
            device.WellOffLevel = Math.Round(((float)_device.WellOffLevel -
_device.DataConversionConstantB) * _device.DataConversionConstantA, 2);
        }
        return Ok(device);
    }
```

Окрему важливу роль відіграє MobileProfileController, через який клієнт отримує детальну інформацію про себе — зокрема ім'я, email, роль у системі, список пристроїв, до яких має доступ, а також дані, пов'язані з делегованими правами. Контролер реалізує оновлення аватару з валідацією формату, обробку зображень та їх зберігання. Усі дії профілю виконуються в контексті авторизованого користувача, гарантуючи, що сторонні особи не можуть змінити чи переглянути чужі дані. Завдяки сервісу профільної логіки, змінені налаштування оновлюються синхронно і зберігаються централізовано на сервері.

Основний метод контроллеру - Отримання профілю користувача (GetUserProfile) — цей метод дозволяє мобільному клієнту отримати базову інформацію про поточного користувача, зокрема його ім'я, прізвище, email, телефон, аватар та список ролей у системі. Метод виконує запит до таблиці користувачів із включенням пов'язаних ролей (RolesList), і повертає дані у форматі, придатному для негайного відображення в інтерфейсі. Завдяки цьому користувач бачить актуальну інформацію про свій обліковий запис

та рольову модель, що є ключовим для правильної взаємодії з системою керування пристроями.

```
   public async Task<ActionResult<IEnumerable<object>>> GetUserProfile(int
userListId)
  {
      var user = await _context.UserTable
          .Include(u => u.RolesList)
          .Where(u => u.Id == userListId)
          .Select(u => new
          {
              u.FirstName,
              u.LastName,
              u.Email,
              u.Phone,
              u.Avatar,

              RolesList = u.RolesList
              .Select(r => r.Role)
              .Distinct()
              .ToList()
          })
          .FirstOrDefaultAsync();

      if (user == null)
          return NotFound("User not found");

      return Ok(user);
  }
```

Функціональність роботи з камерами представлена MobileCameraController. Цей контролер обробляє запити на отримання фото, зроблених підключеними камерами. Його ключова задача — фільтрація архіву зображень за часом, оптимізація вибірки, обмеження доступу відповідно до прав користувача, а також перетворення зображень у зручний формат для мобільного клієнта. Контролер може повертати вибірки фото за вказаний часовий інтервал, що є критично важливим у випадках віддаленого моніторингу об'єктів, наприклад, у фермерських умовах або на полі.

Основний метод контроллеру - Отримання зображень камери за період (MobileGetCameraImages) — основна функція цього методу полягає в тому, щоб надати користувачеві доступ до архіву фотографій, зроблених

пристроєм типу «камера». Метод приймає параметри: ідентифікатор пристрою (deviceSettingId) та часовий інтервал (dateFrom, dateTo), у межах якого потрібно отримати зображення. Для зручності реалізовано обробку хронологічного порядку — навіть якщо користувач помилково вкаже дати у зворотному порядку, метод самостійно визначає коректний діапазон.

Далі виконується фільтрація таблиці ImageData за цим інтервалом та конкретним пристроєм. В результаті формується список, у якому кожен елемент містить посилання на зображення (URL) та часову мітку (TimestampServer), що дозволяє мобільному клієнту точно відобразити, коли було зроблене кожне фото.

URL формуються динамічно за шаблоном, що дозволяє мобільному додатку напряму завантажувати зображення з сервера без додаткових запитів. Такий підхід забезпечує гнучкість, продуктивність та низьке навантаження на API.

Ключовим аспектом методу є його значення в контексті реального використання — завдяки такій реалізації фермер чи оператор може швидко переглянути, що відбувалося біля резервуара в конкретний період, наприклад, після зливи, аварії або планової перевірки.

```
 public async Task<IActionResult> MobileGetCameraImages(
        [FromQuery] int deviceSettingId,
        [FromQuery] DateTime dateFrom,
        [FromQuery] DateTime dateTo)
    {
        try
        {
            var startDate = dateFrom.Date <= dateTo.Date ? dateFrom.Date :
dateTo.Date;
            var endDate = dateFrom.Date <= dateTo.Date ? dateTo.Date :
dateFrom.Date;

            var images = await _context.ImageData
                …
                .Select(img => new
                {
                    ImageUrl =
$"https://api.tanktoad.com/api/image/device/{deviceSettingId}/{img.Filename}",
```

```
                    img.TimestampServer
                })
                .ToListAsync();

            return Ok(images);
        }
        catch (Exception ex)
        {
            return BadRequest($"Error occurred: {ex.Message}");
        }
    }
```

Система сповіщень реалізована у NotificationsController. У межах цього контролера користувач має змогу переглядати сповіщення, які надійшли від пристроїв або системи. Функціональність включає фільтрацію повідомлень за рівнем важливості, статусом прочитання, а також відмітку про перегляд, що дозволяє клієнтському застосунку будувати логіку відображення «непрочитаних» повідомлень. Також підтримується отримання критичних сповіщень за останній період, наприклад, про аварійні ситуації. Усе це дозволяє системі оперативно реагувати на події в режимі майже реального часу.

Основні методи контроллеру:

- GetNotifications відповідає за формування списку всіх повідомлень, надісланих поточному користувачеві. Всі записи вибираються з таблиці Notifications, фільтруються за ідентифікатором авторизованого користувача та сортуються у зворотному хронологічному порядку — від найновішого до найстарішого. Це дозволяє мобільному застосунку одразу показати користувачеві найактуальнішу інформацію — наприклад, про вхід у систему, зміну статусу пристрою або критичні події на резервуарі (перелив, зниження заряду тощо).

Результатом є масив об'єктів із повним вмістом сповіщень, які мобільний клієнт відображає у вкладці «Notifications». Таким чином, користувач завжди має під рукою історію важливих подій.

```
public async Task<IActionResult> GetNotifications()
{
    var notifications = await _context.Notifications
        .Where(n => n.UserId == getCurrentUserId)
        .OrderByDescending(n => n.Timestamp)
        .ToListAsync();

    return Ok(notifications);
}
```

- Метод ClearNotifications реалізує функцію повного очищення переліку сповіщень для поточного користувача. Це корисно, коли сповіщення вже були опрацьовані й більше не мають практичного значення. Після фільтрації повідомлень за UserId, виконується масове видалення всіх знайдених записів (RemoveRange), після чого зміни зберігаються в базі даних (SaveChangesAsync).

Успішне виконання запиту повертає стандартну відповідь зі статусом 200 OK та коротким повідомленням, що всі сповіщення успішно видалені. Ця функція дозволяє тримати інтерфейс застосунку чистим і зручним для постійного використання.

```
public async Task<IActionResult> ClearNotifications()
{
    var notifications = await _context.Notifications
        .Where(n => n.UserId == getCurrentUserId)
        .ToListAsync();

    _context.Notifications.RemoveRange(notifications);
    await _context.SaveChangesAsync();

    return Ok(new { message = "Notifications cleared successfully." });
}
```

Окремим контролером є QrCoreController, який реалізує логіку активації пристроїв за допомогою QR-коду. Його основна функція — забезпечити безпечний, швидкий та інтуїтивний спосіб прив’язки нового пристрою до облікового запису користувача. Контролер дозволяє перевірити валідність QR-ключа, визначити, чи вже прив’язаний пристрій, до кого він належить, і виконати прив’язку у випадку, якщо пристрій ще не активовано. У разі сканування коду з браузера контролер забезпечує

відповідну маршрутизацію — або в мобільний застосунок, або на сторінку завантаження.

Основні методи контроллеру:

- ActivateQrCode - цей метод є центральною точкою взаємодії користувача з пристроєм на етапі першого з'єднання. Він приймає серійний номер пристрою (serialNumber), унікальний ключ (qrKey) і ID користувача, до якого планується прив’язати пристрій.

Процес активації включає кілька етапів перевірки:

1. Перевіряється наявність QR-запису в системі.
2. Якщо QR-код вже активовано — повертається відповідна помилка.
3. Перевіряється, чи вже не прив’язаний пристрій до іншого користувача (уникаючи дублювань).
4. Якщо поточний користувач не є цільовим власником, але має відповідні повноваження менеджера — виконується передача прав через сервіс запрошень (_invitationService), з перевіркою попереднього підтвердження від приймаючої сторони.
5. У випадку позитивного сценарію — користувач додається до пристрою як власник (роль Owner), а інформація про активацію фіксується у БД.

```
 public async Task<IActionResult> ActivateQrCode([FromQuery] string
serialNumber, [FromQuery] string qrKey, [FromQuery] int userId)
  {
      …
      var qrRecord = await _context.QrSystem.FirstOrDefaultAsync(q =>
q.SerialNumber == serialNumber && q.QrSocureKey == qrKey);
      …
      if (getCurrentUserId != userId)
      {
          bool isManager = await
_invitationService.IsUserEmployeeAsync(getCurrentUserId, userId,
EnumUserRoles.Manager);
          if (!isManager)
              return BadRequest("You cannot transfer the device to this
```

```
owner.");
          bool? isAccepted = await
_invitationService.IsToUserAcceptedAsync(getCurrentUserId, userId,
EnumUserRoles.Owner);
          if (isAccepted == true)
          {
              await _invitationService.AddNewOwnersAsync(new
List<UserRoleModel>(), device.Id, getCurrentUserId, userId);
              return Ok(new
              {
                  Message = "The device is automatically added to the new
owner.",
                  QrRecord = qrRecord,
                  DeviceInfo = new
                  {
                      DeviceId = device.Id,
                      DeviceNickname = device.DeviceNickname,
                      …
                  }
              });
          }
          UserInvitationModel invite = new UserInvitationModel
          {
              Role = EnumUserRoles.Owner,
              …
          };
          _context.UserInvitationTable.Add(invite);
      }
      var userRole = new UserRoleModel
      {
          UserId = getCurrentUserId,
          DeviceId = device.Id,
          Role = EnumUserRoles.Owner
      };
      await _context.UserRoleTable.AddAsync(userRole);
      await _context.SaveChangesAsync();
      return Ok(new
      {
          Message = "QR code activated successfully and device assigned to
user",
          QrRecord = qrRecord,
          DeviceInfo = new
          {
              DeviceId = device.Id,
              DeviceNickname = device.DeviceNickname,
              UserId = getCurrentUserId,
              Role = EnumUserRoles.Owner.ToString()
          }
      });
  }
```

Функція StatusQrCode використовується клієнтом ще до фактичної активації — наприклад, після сканування QR-коду. Метод дозволяє перевірити, чи цей пристрій уже активовано, який у нього тип (DeviceType), і чи доступні додаткові параметри, як-от SensorUnits.

Метод повертає:

- статус активації (IsActivated);
- тип пристрою (наприклад, Pump, Camera, WaterSensor);
- DeviceId та допоміжну інформацію з налаштувань пристрою.

Це критично важливо для UX — клієнт може одразу визначити, чи пристрій уже додано, кому він належить, і чи можна розпочинати його налаштування або потрібно звернутись до підтримки.

```
  public async Task<IActionResult> StatusQrCode([FromQuery] string
serialNumber, [FromQuery] string qrKey)
  {
      try
      {
          if (string.IsNullOrWhiteSpace(serialNumber))
              return BadRequest("SerialNumber cannot be empty.");
          if (string.IsNullOrWhiteSpace(qrKey))
              return BadRequest("QrKey cannot be empty.");
          var qrRecord = await _context.QrSystem
              .FirstOrDefaultAsync(q => q.SerialNumber == serialNumber &&
q.QrSocureKey == qrKey);
          var device = await _context.DeviceSettings.FirstOrDefaultAsync(q =>
q.Id == qrRecord.DeviceId);
          if (qrRecord == null)
              return NotFound("SerialNumber and QrKey was not found.");

          return Ok(new
          {
              IsActivated = qrRecord.IsActivated,
              DeviceType = qrRecord.DeviceType,
              DeviceId = qrRecord.DeviceId,
              SensorUnits = device?.SensorUnits
          });
      }
      catch (Exception ex)
      {
          return StatusCode(500, ex);
      }
  }
```

Цей механізм дозволяє організувати ієрархічну модель керування пристроями, де менеджери можуть ініціювати передачу девайсів, а система зберігає повний облік прав і історії активацій.

Кожен контролер побудовано відповідно до загальних принципів REST — із чіткими ендпоінтами, семантикою HTTP-методів (GET, POST, PUT, DELETE) та використанням DTO-структур для передачі даних. Усі запити обробляються в контексті авторизованого користувача за допомогою механізму JWT-ідентифікації. Таким чином, реалізована система дозволяє безпечно, масштабовано та ефективно управляти усіма аспектами роботи пристроїв TankToad у складних реальних умовах.

РОЗДІЛ IV. ТЕСТУВАННЯ ТА ДОСЛІДНА ЕКСПЛУАТАЦІЯ

4.1. Тестування функціоналу

З метою забезпечення високої якості програмного забезпечення та надійності роботи серверної частини мобільного додатку, було проведено детальне тестування функціоналу. Тестування охоплювало всі ключові компоненти системи, включаючи контролери авторизації, управління пристроями, профілем користувача, сповіщеннями, камерами та роботу з QR-кодами.

Для зручності і системності тестування активно використовувався інструмент Swagger UI, який є інтегрованим засобом для перевірки REST API. Завдяки йому здійснювався швидкий запуск HTTP-запитів до контролерів, контроль форматів вхідних параметрів, перегляд відповідей сервера та візуальна перевірка структури JSON-виходів. Swagger дозволяв також одразу тестувати сценарії з авторизацією, забезпечуючи емуляцію запитів від імені користувача з різними ролями — Owner, Manager, SysAdmin тощо.

Окрема увага була приділена тестуванню ролей доступу. В межах проекту права користувача відіграють ключову роль у визначенні допустимих дій. Для цього було змодельовано серії запитів із різними обліковими записами. Наприклад, користувач із роллю Owner успішно отримував доступ до списку пристроїв, які йому належать, у той час як Manager міг отримати лише ті пристрої, доступ до яких йому було делеговано. Запити без авторизаційного токена викликали очікувану помилку 401 Unauthorized, а запити з порушенням прав — 403 Forbidden.

Сценарії негативного тестування охоплювали введення некоректних параметрів у запити: відсутність обов’язкових полів (наприклад, deviceId),

передача порожніх строк, надмірно довгі параметри, порушення типів даних (наприклад, текст замість int). У таких випадках система коректно повертала помилки 400 Bad Request з відповідним описом проблеми, що свідчило про наявність валідних механізмів перевірки введених даних.

Тестування методів MobileProfileController довело, що запит на отримання профілю (GetUserProfile) повертає повний набір полів — ім'я, прізвище, email, телефон, а також список ролей користувача. Крім того, підтверджено, що пошук користувачів (PostSearchUserAsync) коректно працює як за email, так і за телефоном, і обмежує результати до 10 записів, що є захистом від надмірного навантаження.

Особливо важливим було тестування модуля керування насосами (PumpAutoModeChanger, PumpWellOLevelChanger), де користувач впливає на фізичний пристрій. Було перевірено, що зміна режиму роботи насоса можлива лише при наявності відповідних прав і лише для активного пристрою. Спеціальні тести перевіряли реакцію системи на запит до неіснуючого пристрою або до насоса, який не підтримує зміну режиму — система повертала передбачені повідомлення про помилку.

У модулі сповіщень (NotificationsController) тестувались сценарії накопичення повідомлень, отримання історії користувача (GetNotifications), а також повне очищення (ClearNotifications). Після очищення новий запит підтверджував, що масив повідомлень є порожнім. Це дозволило верифікувати коректність взаємодії з базою даних.

Функціонал камер (через MobileCameraController) перевірявся на прикладі пристроїв, які надсилають зображення. Метод MobileGetCameraImages дозволив отримувати зображення за певний часовий діапазон. Тестування показало, що при передачі хибної дати або некоректного deviceId система повертає інформативну помилку. Зображення були доступні лише для користувачів, які мають права доступу до пристрою, що знову підтвердило ефективність моделі доступу.

Окрему категорію тестів склали запити до контролера QR-кодів (QrCoreController). Було перевірено логіку активації пристроїв через QR-код (ActivateQrCode), включаючи сценарії передачі пристрою іншому користувачеві за роллю. Перевірка статусу коду (StatusQrCode) дозволила верифікувати, що система правильно фіксує стан: активований чи ні, тип пристрою, ідентифікатор тощо.

Важливим напрямком стало тестування граничних значень: наприклад, кількість пристроїв, які може бачити користувач, кількість повідомлень в історії, або спроба надсилання великого зображення як аватару. Всі перевірки показали, що система правильно обробляє подібні ситуації без втрати стабільності чи витоку пам’яті.

Загалом, тестування функціоналу підтвердило, що всі основні компоненти серверної частини працюють узгоджено, стабільно й відповідають очікуванням. Жодних критичних багів чи вразливостей виявлено не було. Усі тести (як позитивні, так і негативні) були успішно пройдені, що дало змогу перейти до наступного етапу — дослідної експлуатації.

4.2 Дослідна експлуатація проекту

Після успішного проходження етапу внутрішнього тестування програмного продукту наступним логічним кроком стала дослідна (пілотна) експлуатація. Цей етап був спрямований на перевірку працездатності системи в умовах, максимально наближених до реальних. Він дозволив оцінити поведінку системи при роботі з реальними користувачами, фізичними IoT-пристроями, зміною мережевого середовища, нестабільним інтернет-з’єднанням та реальними сценаріями взаємодії з пристроями.

Дослідна експлуатація охоплювала використання мобільного застосунку та бекенд-системи у фермерських умовах, де було підключено камери спостереження, насоси, сенсори рівня води та інші пристрої,

пов’язані з моніторингом та автоматизацією аграрних об'єктів. Основною метою було перевірити наскільки стабільно і швидко працюють сценарії авторизації, активації пристроїв через QR-коди, сповіщення, доступ за ролями та інші ключові модулі.

Особливу увагу було приділено перевірці механізму делегування прав користувачів, коли один користувач може надати іншому доступ до свого пристрою з певною роллю — наприклад, як Owner або Manager. Це критичний функціонал у випадках спільного управління технікою або передачі пристроїв в оренду.

У реальних умовах були протестовані такі кейси:

- Активація нового пристрою через QR-код. У кількох випадках використовувались як iOS, так і Android пристрої, що дозволило перевірити поведінку коду на різних платформах. Активований пристрій одразу з’являвся у списку користувача без потреби оновлювати сесію.
- Зміна режиму роботи насоса у польових умовах. Користувачі пробували змінити рівень води в колодязі (PumpWellOLevelChanger) та вмикати автоматичний режим (PumpAutoModeChanger) — при цьому система надсилала push-сповіщення та відображала актуальний стан.
- Спостереження через камери в реальному часі — метод MobileGetCameraImages дозволяв переглядати фото за останні дні. Була перевірена коректність відображення за різними часовими зонами та стабільність відображення навіть при нестабільному інтернеті.
- Робота зі сповіщеннями. Користувачі отримували push-повідомлення в момент критичних змін: низький рівень води, помилка сенсора, відключення живлення тощо. Інтерфейс мобільного додатку дозволяв очищати повідомлення, а також відображати історію.
- Робота з офлайн-режимом. Було протестовано, як мобільний клієнт поводиться у відсутності інтернету. Завдяки локальному кешу

пристрої все одно відображались, а зміни режимів блокувались до відновлення підключення.

Також особливу роль відіграла перевірка швидкодії бекенду при одночасному використанні кількома користувачами. Під час пікових навантажень (наприклад, коли вмикались одразу кілька насосів) система продемонструвала стабільність — час відповіді API залишався у межах 150–250 мс, що є прийнятним результатом.

Протягом всього етапу дослідної експлуатації велося журналювання дій користувачів, що дозволило виявити декілька потенційно важливих сценаріїв:

- деякі користувачі намагались активувати пристрій, який уже прив’язаний до іншого акаунту — у цьому випадку система блокувала дію з поясненням;
- кілька користувачів надавали помилкові email або вводили некоректні дані при пошуку — це дозволило ще раз перевірити і зміцнити валідацію на стороні бекенду;
- іноді користувачі очищували повідомлення, не переглянувши їх — у подальшому було запропоновано реалізувати маркування «непрочитаних» у мобільному клієнті.

В цілому, дослідна експлуатація підтвердила стабільність і зручність системи в реальних умовах використання. Результати пілотного запуску довели, що структура API, реалізація контролерів і логіка доступу повністю відповідають технічному завданню. Усі помічені дрібні баги були усунені або задокументовані для наступної ітерації.

На основі дослідної експлуатації було прийнято рішення про подальше розширення системи, а також впровадження додаткових функцій, зокрема перегляду графіків за періодами, push-нагадувань та інтерфейсу для підтримки користувачів.

ВИСНОВОК

У результаті виконання кваліфікаційної роботи було повністю реалізовано програмний продукт, що забезпечує ефективне дистанційне управління IoT-пристроями в умовах польових застосунків, зокрема в аграрному середовищі. Основною метою розробки було створення надійної серверної частини, яка дозволяє користувачам керувати пристроями, переглядати дані сенсорів, отримувати сповіщення та виконувати налаштування з урахуванням прав доступу.

У процесі реалізації було:

- розроблено багаторівневу архітектуру серверного застосунку з чітким поділом на контролери, сервіси, моделі та механізми авторизації;
- забезпечено взаємодію з мобільним клієнтом через RESTful API з використанням протоколу HTTPS;
- реалізовано ключові модулі: управління користувачами, авторизація, робота з пристроями, профілями, сповіщеннями та QR-кодами;
- впроваджено підтримку ролей (власник, менеджер, співробітник) та делегування прав доступу до пристроїв;
- додано можливість активації пристроїв через QR-коди, а також систему запитів на передачу прав;
- реалізовано збереження та вибірку фотографій із камер, з урахуванням часових діапазонів;
- розроблено зручні методи пошуку, фільтрації та сортування пристроїв з підтримкою пагінації;
- реалізовано офлайн-функціональність для стабільної роботи при нестабільному інтернет-з'єднанні.

Для забезпечення стабільності та правильності функціонування було проведено активне тестування за допомогою інструменту Swagger. Завдяки

цьому вдалося перевірити коректність обробки запитів, відповідей, статус-кодів, а також протестувати різні сценарії взаємодії з API. Особливу увагу було приділено перевірці авторизації, дій з обмеженим доступом, активації пристроїв та перевірки граничних значень у параметрах запитів.

Дослідна експлуатація проєкту в умовах реального використання продемонструвала його життєздатність, стабільну роботу та відповідність вимогам. Система успішно забезпечує управління пристроями та відображення критично важливої інформації, що свідчить про її готовність до впровадження в більш широке середовище.

Отже, поставлену мету кваліфікаційної роботи досягнуто повною мірою. Розроблений серверний застосунок відповідає сучасним вимогам щодо масштабованості, безпеки, продуктивності та зручності використання. Він є базовою платформою для подальшого розвитку мобільного клієнта та розширення функціональних можливостей системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1 ASP.NET documentation. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-9.0 (date of access: 14.06.2025).

2 Entity Framework Tutorial. *Entity Framework Tutorial*. URL: https://www.entityframeworktutorial.net/ (date of access: 14.06.2025).

3 Franklin M. Primary ASP. NET Core 3 Web API. Independently Published, 2022.

4 Murach's ASP. NET Core MVC. Murach & Associates, Incorporated, Mike, 2020. 780 p.

5 .NET documentation. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/dotnet/ (date of access: 14.06.2025).

6 Tank Toad | Livestock Remote Water Monitor. *Tank Toad*. URL: https://www.tanktoad.com/ (date of access: 14.06.2025).