

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ**

Факультет інформаційних технологій

ПОГОДЖЕНО
Декан факультету

_____ **Ігор БОЛБОТ**
(підпис)

«___» _____ 2026 р.

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ
Завідувач кафедри комп'ютерних наук

_____ **Белла ГОЛУБ**
(підпис)

«___» _____ 2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

**на тему: «Програмне забезпечення для системи обліку аптек першої
допомоги»**

Спеціальність «121 Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення»

Гарант освітньої програми
к.т.н., доцент

_____ **Ганна ВАЙГАНГ**
(підпис)

**Керівник бакалаврської
кваліфікаційної роботи**
старший викладач

_____ **Світлана ВАСИЛЮК-ЗАЙЦЕВА**
(підпис)

Виконав

_____ **Нікіта ВОЛКОВ**
(підпис)

Київ-2026

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ**

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук

к.т.н., доцент _____ Белла ГОЛУБ
(підпис)

«__» _____ 2025 р.

**ЗАВДАННЯ
ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
ЗДОБУВАЧУ**

Волкову Нікіті Олександровичу
(прізвище, ім'я, по батькові)

Спеціальність «121 Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи

«Програмне забезпечення для системи обліку аптечок першої допомоги»

затверджена наказом від «19» грудня 2025 р. № 3196 «С»

Термін подання завершеної роботи на кафедру «22» травня 2026 р.

Вихідні дані до бакалаврської кваліфікаційної роботи: технічне завдання на розробку системи для мобільного обліку та контролю термінів придатності медикаментів

Перелік питань, що підлягають дослідженню: аналіз предметної області, проектування бази даних, розробка серверної частини та створення мобільного інтерфейсу для ручного управління запасами аптечок.

Перелік графічних документів. діаграма варіантів використання, ER-діаграма бази даних, структурна схема архітектури системи, макети екранів мобільного додатку.

Дата видачі завдання «19» грудня 2025 р.

**Керівник бакалаврської
кваліфікаційної роботи**

(підпис)

Світлана ВАСИЛЮК-ЗАЙЦЕВА

**Завдання взяв до
виконання**

(підпис)

Нікіта ВОЛКОВ

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	4
ВСТУП.....	5
1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ	8
1.1 Опис предметної області	8
1.2 Аналіз вимог до програмної системи	11
1.3 Моделювання предметної області	15
1.4 Огляд інформаційних джерел та існуючих рішень	20
1.5 Постановка завдання	23
2 ПРОЄКТУВАННЯ ПРОГРАМНО-ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ.....	25
2.1 Логічна модель даних	25
2.2 Діаграма класів та кооперацій.....	28
2.3 Діаграма пакетів	31
2.4 Діаграма компонентів	34
3 РОЗРОБКА ПРОГРАМНО-ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ.....	36
3.1 Система управління інформаційною базою.....	36
3.2 Розробка інформаційної бази	38
3.3 Вибір інструментарію	39
3.4 Алгоритмізація та програмування програмних модулів	43
4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ.....	47
4.1 Тестування системи.....	47
4.2 Вимоги до апаратного та програмного забезпечення	54
4.3 Склад інсталяційного пакету.....	56
ВИСНОВКИ	60
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	62
ДОДАТОК А.	64
ДОДАТОК Б.....	69
ДОДАТОК В.	78

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

У цій пояснювальній записці використовуються такі скорочення та аббревіатури:

API – Application Programming Interface, програмний інтерфейс застосунку;

APK – Android Package Kit, інсталяційний пакет застосунку для Android;

ASP.NET – серверний фреймворк для побудови вебзастосунків;

CRUD – Create, Read, Update, Delete – базовий набір операцій над даними;

DTO – Data Transfer Object, об'єкт передачі даних між шарами застосунку;

EF Core – Entity Framework Core, ORM-фреймворк для платформи .NET;

FCM – Firebase Cloud Messaging, сервіс push-сповіщень компанії Google;

HTTP / HTTPS – протокол передачі даних;

IPA – iOS App Store Package, інсталяційний пакет застосунку для iOS;

JSON – JavaScript Object Notation, формат обміну даними;

JWT – JSON Web Token, стандарт токена доступу;

ORM – Object-Relational Mapping, об'єктно-реляційне відображення;

REST – Representational State Transfer, архітектурний стиль вебсервісів;

SaaS – модель надання програмного забезпечення як сервісу;

SDK – Software Development Kit, набір засобів розробки;

SQL – Structured Query Language, мова структурованих запитів;

TLS – Transport Layer Security, протокол захищеної передачі даних;

UML – Unified Modeling Language, уніфікована мова моделювання;

VPS – Virtual Private Server, віртуальний приватний сервер;

БД – база даних;

ЄСКД – Єдина система конструкторської документації;

ЄСПД – Єдина система програмної документації;

ОС – операційна система;

ПЗ – програмне забезпечення;

СУБД – система управління базами даних;

ВСТУП

Аптечка першої допомоги є обов'язковим елементом практично будь-якого закладу чи підприємства. Але на практиці нормального обліку ліків у ній майже ніде немає – або папір, або взагалі нічого. І коли реально потрібна допомога, виявляється що половина препаратів прострочена, а чогось потрібного немає.

Розроблена в рамках цієї роботи система FAIMS, тобто First Aid Inventory Management System, переводить процес обліку у цифровий формат. Це повноцінний продукт, що складається з кросплатформного мобільного застосунку для Android та iOS і серверної частини, розгорнутої на віртуальному сервері з публічним доменом. Система побудована як мультитенантне SaaS-рішення: кожна організація-клієнт працює зі своїми даними у повністю ізольованому середовищі, при цьому всі вони використовують одну й ту саму інсталяцію. Адміністратор організації веде реєстр аптечок, призначає відповідальних, формує звіти; відповідальна особа працює з конкретними закріпленими за нею аптечками, оновлює залишки, отримує сповіщення про наближення термінів придатності.

Метою роботи є розробка програмного забезпечення, яке автоматизує процеси обліку аптечок в організаціях, забезпечує попередження відповідальних осіб про наближення термінів придатності медикаментів і надає керівництву огляд стану аптечок у режимі реального часу. Актуальність задачі підтверджується тим, що на українському ринку відсутні рішення для цього. Представлені зараз продукти або є універсальними системами управління складом без урахування специфіки медикаментів, або є вузькими модулями всередині великих ERP-систем, які доступні лише корпоративним клієнтам.

Для досягнення поставленої мети у роботі вирішено такі завдання:

- проведено системний аналіз предметної області та сформовано вимоги до програмної системи;
- побудовано комплекс UML-моделей предметної області – діаграми прецедентів, послідовності, активності, класів, кооперацій;
- спроектовано логічну модель даних з обґрунтуванням обраної СУБД;
- розроблено архітектуру системи за принципами Clean Architecture з чітким розподілом на шари.
- реалізовано серверну частину на ASP.NET Core з підтримкою мультитенантності, JWT-автентифікації та push-сповіщень;
- реалізовано кросплатформний мобільний застосунок на Flutter з підтримкою двох ролей користувачів;
- налаштовано контейнеризацію через Docker та розгорнуто систему на віртуальному сервері з автоматичним отриманням TLS-сертифіката;
- проведено тестування системи та сформовано рекомендації щодо її впровадження.

Методи та технології розробки. Серверна частина побудована на ASP.NET Core 9.0 із застосуванням Entity Framework Core як об'єктно-реляційного відображення [4, 5]; як база даних використовується PostgreSQL 15 [7]. Безпека реалізована на основі JWT-токенів з ротацією refresh-токенів і хешування паролів через стандартний механізм ASP.NET Identity [9]. Push-сповіщення доставляються через сервіс Firebase Cloud Messaging [13]. Мобільний застосунок написано на Flutter з використанням Provider для управління станом і GoRouter для декларативної навігації [8]; локалізацію інтерфейсу реалізовано через стандартний механізм ARB-файлів. Контейнеризація реалізована через Docker Compose [11]; reverse-proxy і автоматичний HTTPS забезпечує Caddy [12]. Як середовище проєктування діаграм UML використано Draw.io, для проєктування моделей даних – CA ERwin Data Modeler.

Об’єкт дослідження – процеси обліку аптечок першої допомоги в організаціях: контроль термінів придатності медикаментів, підтримання комплектності аптечок та організація оповіщень відповідальних осіб.

Предмет дослідження – методи та засоби автоматизації процесів обліку аптечок за допомогою кроссплатформного мобільного застосунку із серверною частиною.

Практичне значення одержаних результатів. Розроблена система FAIMS може бути впроваджена в будь-якій організації - школі, заводі, офісі чи закладі харчування - для автоматизації обліку аптечок першої допомоги без капітальних витрат.

Апробація результатів. Основні положення роботи опубліковано у вигляді тез доповіді «Програмне забезпечення автоматизованої системи контролю за запасами та терміном придатності аптечок першої допомоги» на науково-практичній конференції студентів і аспірантів, що проводився факультетом інформаційних технологій НУБіП України. Розроблений продукт перебуває на етапі закритого тестування у Google Play з метою подальшого публічного релізу.

Структура записки. Пояснювальна записка складається зі вступу, чотирьох розділів, висновків, списку використаних джерел і додатків. У першому розділі проведено системний аналіз предметної області, побудовано моделі бізнес-процесів і розглянуто існуючі рішення-аналоги. У другому розділі описано проєктування інформаційного та програмного забезпечення: побудовано логічну модель даних, діаграми класів, кооперацій, пакетів і компонентів. Третій розділ присвячено розробці системи: обґрунтовано вибір СУБД, розглянуто реалізацію бази даних, описано вибраний інструментарій та наведено алгоритми ключових модулів. У четвертому розділі представлено рекомендації щодо тестування, апаратно-програмних вимог і складу інсталяційного пакету. Загальний обсяг основного тексту записки – близько 65 сторінок; список використаних джерел містить близько 30 найменувань.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Аптечка першої допомоги – це базовий елемент системи безпеки практично будь-якої організації, де перебувають люди. Школи, заводи, офіси, лікарні, навчальні корпуси, склади, спортивні зали – усі ці об’єкти за чинним законодавством зобов’язані тримати укомплектовані аптечки і слідкувати, щоб їхній вміст відповідав встановленим переліком та залишався придатним до застосування. На практиці ця вимога дуже часто виконується формально: аптечка стоїть, один раз на рік хтось її перевіряє, розписався в журналі – і вона знов простоює до наступної ревізії.

Проблема в тому, що ліки мають кінцевий термін придатності. Парацетамол, який пролежав три роки, у момент реальної потреби в кращому випадку не подіє, у гіршому – нашкодить. Той самий хлоргексидин після експірації втрачає антисептичні властивості. Бинт у запечатаному пакеті може бути придатним, але тільки якщо упаковка не була відкрита, не відсиріла, не зберігалася біля джерела тепла. Регулярний контроль термінів придатності та комплектності аптечки – це не бюрократія, а пряма умова того, що в момент травми чи приступу аптечка справді допоможе.

Нормативна база в Україні встановлює мінімальний склад аптечок різних типів. Для аптечок медичної допомоги загального призначення це Наказ МОЗ України, який визначає перелік ліків та виробів медичного призначення. Для виробничих об’єктів додатково діють вимоги Державної служби України з питань праці щодо облаштування робочих місць і доступності засобів першої допомоги.

У більшості організацій ці процеси досі ведуться вручну. Типовий сценарій виглядає так: за кожною аптечкою закріплена відповідальна особа – зазвичай це не лікар, а адміністратор, інженер з охорони праці, методист або просто співробітник, якому дали цей обов’язок. Раз на місяць, а буває, що раз на

квартал чи рідше, ця особа відкриває аптечку, перебирає вміст, виписує на папір те, що скінчилося або прострочилося, і передає список комусь, хто займається закупівлями. Закупівельник формує заявку до постачальника, отримує препарати, передає назад відповідальному, той кладе їх на місце і ставить позначку в журналі.

У цьому ланцюжку помилки виникають на кожному кроці. Відповідальний може забути перевірити аптечку вчасно. Може пропустити препарат, який лежить у глибині й термін його придатності збігає завтра. Може записати дані з помилкою. Список може загубитися дорогою до закупівельника. Закупівля може затриматися. Навіть коли все спрацювало, ніхто, крім самого відповідального, не бачить актуального стану аптечки в реальному часі – керівник служби охорони праці, наприклад, дізнається про проблему тільки під час позапланової перевірки.

Цю задачу і вирішує програмне забезпечення, яке є предметом дипломної роботи. Розроблена програма переводить весь цикл обліку в цифровий формат. Кожна аптечка реєструється в системі з прив'язкою до конкретної кімнати у складі підрозділу організації. За кожною аптечкою закріплюється відповідальна особа. До аптечки додаються медикаменти із зазначенням назви, кількості, одиниці виміру та терміну придатності. Система автоматично відстежує наближення термінів і завчасно надсилає push-сповіщення відповідальній особі: за тиждень до експірації, за місяць, відразу після спливу. Крім того, адміністратор, у будь-який момент бачить агрегований стан усіх аптек: скільки з них у нормі, у скількох є критичні позиції, які зараз потребують поповнення, які препарати слід утилізувати.

Окремий важливий аспект – це **мультитенантність**. Система одночасно працює багато організацій-клієнтів, але дані кожної повністю ізольовані [6]. Школа, що використовує систему, ніколи не побачить даних промислового підприємства, яке працює в тій самій системі.

Така архітектура одночасно вирішує два завдання: забезпечує конфіденційність даних кожного клієнта і дозволяє масштабувати систему без перепакування під кожну організацію окремо.

Розроблена система охоплює такі функціональні можливості:

- ведення реєстру аптек з прив'язкою до фізичного місця та відповідальної особи;
- облік медикаментів у складі аптечки з контролем кількості та термінів придатності;
- автоматичний моніторинг наближення термінів та push-сповіщення;
- списання використаних або прострочених препаратів з фіксацією причини;
- поповнення запасів;
- журналювання всіх дій для подальшого аудиту;
- формування звітів для внутрішніх потреб та для перевірок з боку контролюючих органів;
- розмежування доступу між адміністратором організації та відповідальними за окремі аптечки.

1.2 Аналіз вимог до програмної системи

З аналізу виходить список вимог. Функціональні – що система взагалі вміє робити. Нефункціональні – наскільки добре і надійно.

1.2.1 Функціональні вимоги

Реєстрація та автентифікація. Адміністратор повинен мати можливість зареєструвати свою організацію у системі, створивши при цьому власний обліковий запис із роллю адміністратора. Решта створюються вже всередині організації самим адміністратором. Вхід здійснюється за email і паролем; система видає токен доступу для подальшої роботи з API.

Управління організаційною структурою. Адміністратор може створювати, редагувати та видаляти підрозділи організації. Усередині підрозділу – кімнати. Підрозділ і кімната дають фізичну адресу, до якої прив'язується конкретна аптечка.

Управління аптечками. Адміністратор створює аптечки, задає для кожної унікальний номер у межах організації, закріплює за конкретною кімнатою і призначає відповідальну особу. За потреби – змінює призначеного відповідального або переносить аптечку в іншу кімнату. Передбачено можливість видалити аптечку, якщо вона списана.

Шаблони аптечок. Для прискорення створення типових аптечок реалізовано механізм шаблонів. Шаблон – це попередньо складений набір медикаментів із зазначенням найменування, одиниці виміру та рекомендованої кількості. При створенні нової аптечки адміністратор може обрати шаблон, і система автоматично згенерує початковий перелік позицій. У базі підтримуються як системні шаблони, побудовані за нормативними переліками, так і користувацькі шаблони, створені в межах конкретної організації.

Управління медикаментами в аптечці. Адміністратор має повний доступ до медикаментів у будь-якій аптечці своєї організації. Відповідальна особа – лише до тих аптечок, які за нею закріплені. Доступні операції:

- додавання нового медикаменту із зазначенням назви, кількості, одиниці виміру, терміну придатності та опціонально мінімального допустимого запасу;
- редагування параметрів існуючої позиції;
- списання частини кількості при використанні з обов'язковим зазначенням причини;
- утилізація з фіксацією причини, зазвичай це закінчення терміну придатності;
- поповнення запасу новою партією, яка може мати власний термін придатності, відмінний від попередньої.

Контроль термінів придатності. Кожному медикаменту система присвоює статус, який обчислюється автоматично за принципом «світлофора» на основі різниці між сьогоднішньою датою і датою спливу терміну: «нормальний» – більше місяця в запасі, «увага» – менше місяця, «критичний» – менше тижня, «прострочений» – термін уже минув. Статус відображається кольоровим маркером поряд із позицією і використовується для побудови звітів та сповіщень.

Push-сповіщення. Серверна частина щодня сканує всю базу медикаментів і формує сповіщення для відповідальних осіб про позиції, які перейшли в категорії «увага», «критичний» або «прострочений». Сповіщення доставляються через Firebase Cloud Messaging [13] на мобільні пристрої з установленим застосунком.

Журналювання дій. Кожна операція з медикаментом – додавання, списання, утилізація, поповнення – фіксується у журналі дій із зазначенням типу дії, кількості, причини, дати, виконавця та аптечки, до якої належить запис. Журнал є джерелом для звітів і для розслідування спірних ситуацій.

Звітність. Система формує такі PDF-звіти за період як список на утилізацію і список на закупівлю. Звіти потрібні як для внутрішнього контролю, так і для надання перевіряючим органам.

Аналітика. Адміністратору доступні агреговані графіки, які показують динаміку витрат і поповнень за обраний період. Це дозволяє планувати бюджет на закупівлі та виявляти аномалії, наприклад надмірне списання конкретного препарату.

1.2.2 Нефункціональні вимоги

Продуктивність. Час відгуку API на типові запити не повинен перевищувати трьох секунд за умови нормального навантаження. Система проєктується з розрахунку на одночасну роботу до 1000 аптечок та до 10 000 позицій медикаментів у межах однієї організації – це покриває потреби середнього підприємства або мережі шкіл.

Надійність. Усі операції зміни стану виконуються в межах транзакцій бази даних. Це гарантує, що частково завершена операція, наприклад при обриві з'єднання посеред списання, не залишить базу в неконсистентному стані. Резервне копіювання бази даних виконується автоматично кожні шість годин зі збереженням за останні три доби.

Безпека. Паролі користувачів зберігаються у вигляді хешів через стандартний механізм IPasswordHasher ASP.NET Identity [4]. Автентифікація реалізована через JWT-токени з обмеженим часом життя [9]; для безшовного оновлення сесії використовується refresh-токен з ротацією. Усі звернення до API виконуються через HTTPS із сертифікатом Let's Encrypt. Доступ до даних розмежований за ролями: адміністратор працює тільки з даними своєї організації, відповідальна особа – тільки з даними закріплених за нею аптечок. На рівні бази даних мультитенантність підкріплена глобальним фільтром EF Core [5], який автоматично додає умову відбору за ідентифікатором організації до кожного запиту.

Захист від зловживань. Запити до серверної частини обмежуються rate-лімітами: 5 запитів на хвилину для ендпоінтів автентифікації як захист від bruteforce-атак [10], і 150 запитів на хвилину для решти API.

Зручність використання. Інтерфейс мобільного застосунку повністю українізовано із заготовкою англійської локалізації для подальшого виходу на закордонні ринки. Шрифт основного інтерфейсу – Manrope, який добре читається на мобільних екранах різного розміру. Реалізовано миттєвий пошук по медикаментах за назвою. Партії одного й того самого препарату з різними термінами придатності автоматично групуються, щоб не захаращувати екран.

Сумісність. Мобільний застосунок підтримує Android починаючи з версії 6.0 та iOS починаючи з версії 12.0 – це покриває переважну більшість пристроїв, що активно використовуються.

Розгортання. Серверна частина розгортається у контейнерах Docker, що дозволяє підняти середовище на будь-якій сучасній Linux-машині однією командою. На продуктивному сервері використовуються чотири контейнери: PostgreSQL, серверний застосунок ASP.NET Core, реверс-проксі Caddy із автоматичним отриманням TLS-сертифіката, та окремий контейнер для періодичного резервного копіювання бази.

1.3 Моделювання предметної області

Щоб краще зрозуміти що і як потрібно автоматизувати – побудовано набір UML-моделей. Кожна дивиться на предметну область з іншого кута. Моделі описують процеси ширше, ніж те що реалізовано в системі – це зроблено навмисно, щоб бачити повну картину і свідомо вирішувати, що автоматизувати зараз, а що залишити на потім.

1.3.1 Діаграма прецедентів

Діаграма прецедентів описує функціональні сценарії, які мають місце в предметній області, а також акторів – ролі, які виконують або ініціюють ці сценарії. Для предметної області обліку аптек виділено чотирьох акторів. Працівник підприємства – звичайний співробітник організації, який звертається до аптеки в момент потреби, тобто є кінцевим споживачем послуги. Відповідальний за аптеку – особа, на яку покладено обов’язки з ведення обліку, регулярних перевірок, поповнення та утилізації. Менеджер із закупівель – особа, яка обробляє заявки на нові медикаменти і здійснює взаємодію з постачальниками. Постачальник – зовнішня сторона, яка постачає медикаменти на підставі укладених договорів.

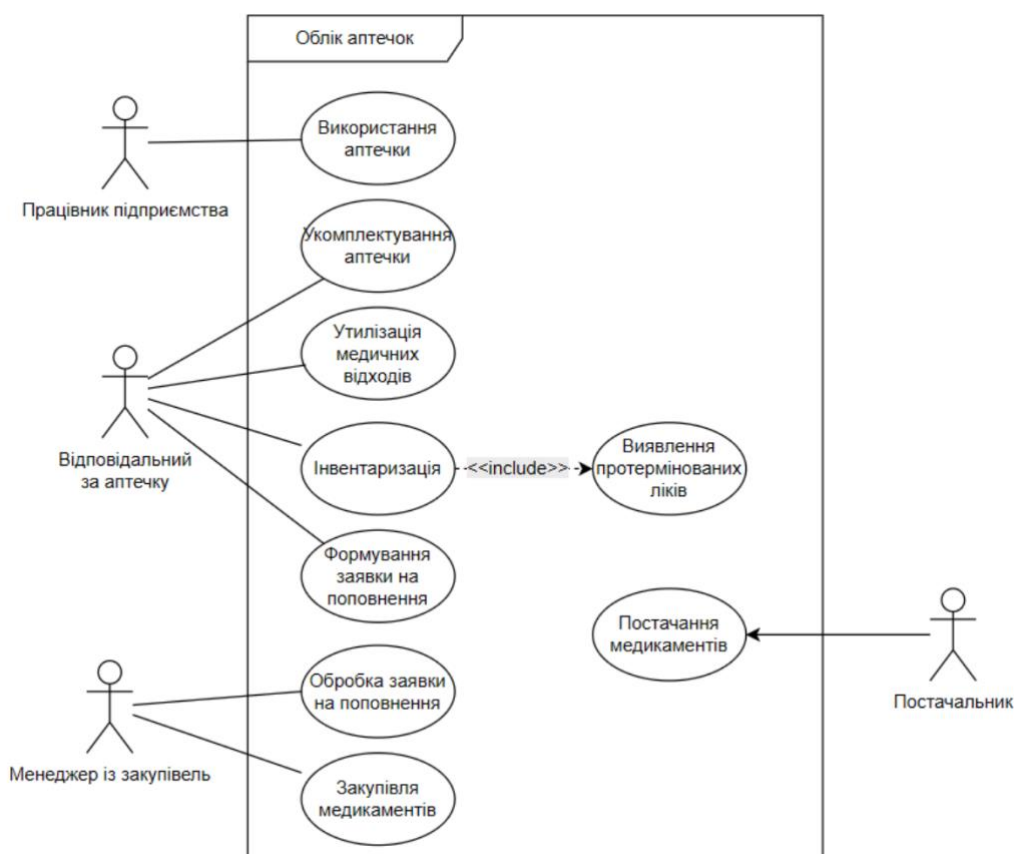


Рис. 1.1. Діаграма прецедентів предметної області

Серед сценаріїв, представлених на діаграмі, основними є використання аптечки, її укомплектування, інвентаризація з виявленням протермінованих ліків, утилізація медичних відходів, формування заявки на поповнення, обробка цієї заявки і безпосередньо закупівля медикаментів. Зв'язок «include» між «Інвентаризацією» та «Виявленням протермінованих ліків» підкреслює, що друге є невід'ємною частиною першого: інвентаризація без перевірки термінів придатності не має сенсу.

У розробленій системі реалізовано не всі представлені сценарії. Зокрема, повний цикл закупівлі – взаємодія з постачальниками, обробка рахунків, оплата – залишений за межами поточної версії продукту і може бути додатково реалізований у наступних ітераціях. У межах системи реалізовано всі сценарії, що пов'язані з самою аптечкою: інвентаризацію, контроль термінів, утилізацію, поповнення запасів. Це підкреслює тезу про те, що модель предметної області природно ширша за модель самого програмного продукту.

1.3.2 Діаграма послідовності

Діаграма послідовності уточнює діаграму прецедентів, показуючи у часі обмін повідомленнями між акторами та об'єктами в межах двох найважливіших сценаріїв: інциденту, тобто звернення працівника по допомогу, і поповнення запасів. У сценарії інциденту працівник підприємства звертається до відповідального з повідомленням про потребу; відповідальний шукає необхідний засіб у аптечці, надає допомогу і фіксує зменшення кількості. У сценарії поповнення відповідальний виявляє нестачу під час планової інвентаризації, формує і передає заявку менеджеру із закупівель, той розміщує замовлення в постачальника, оплачує рахунок, отримує медикаменти і передає їх назад відповідальному, який кладе їх на місце і оновлює журнал.

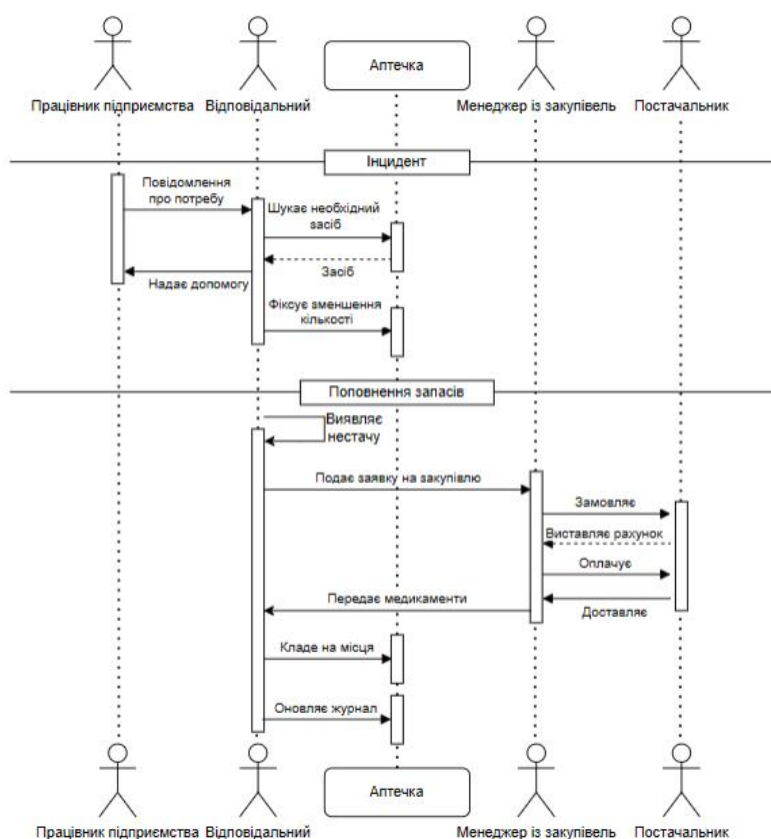


Рис. 1.2. Діаграма послідовності

1.3.3 Діаграма активності

Діаграма активності зосереджена на алгоритмічній стороні процесу: вона показує послідовність дій від моменту планової перевірки до моменту повного оновлення вмісту аптечки. Алгоритм містить точку прийняття рішення: якщо при перевірці виявлено протерміновані або відсутні препарати, активується цикл утилізації, складання списку нестачі, замовлення у постачальника, оплати, доставки та передачі відповідальному; якщо ж усе в нормі – процес одразу завершується. Розподіл активностей між трьома доріжками – відповідальний, менеджер із закупівель, постачальник – наочно показує, які саме дії виконуються кожною роллю.

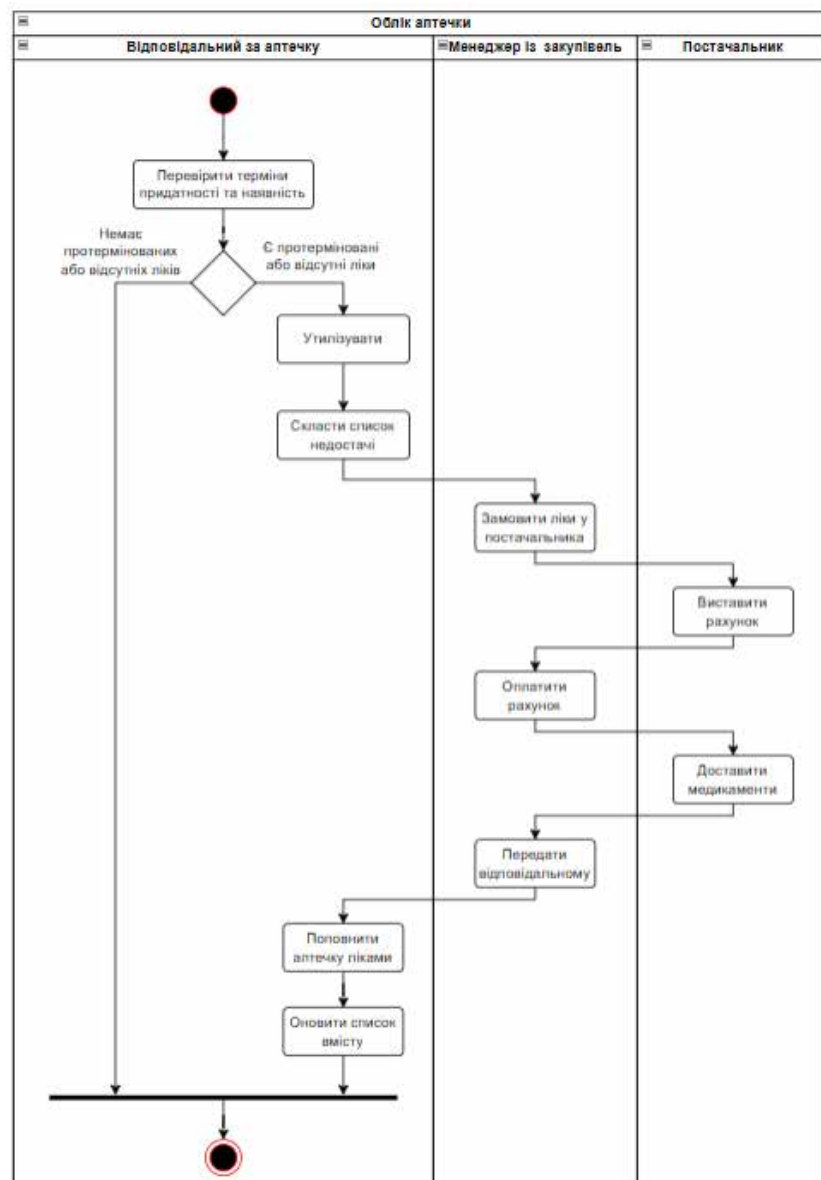


Рис. 1.3. Діаграма активності процесу обліку аптечки

1.3.4 Виділення ключових абстракцій

На основі аналізу діаграм виділено перелік ключових абстракцій предметної області. Кожна абстракція описана своїми властивостями і обов'язками. Цей крок є важливим для подальшого переходу до проектування: кожна виділена абстракція стане або сутністю в моделі даних, або класом доменного шару системи. Властивості перетворюються в атрибути, обов'язки – в методи.

Визначені абстракції сутностей

Постачальник Властивості: Назва компанії Контактні дані Номер договору Терміни доставки Обов'язки Прийняти замовлення Доставити товар Надання супровідних документів	Відповідальна особа Властивості: ПІБ Посада Контактні дані Закріплена аптечка Обов'язки Проводити огляд аптечки Перевіряти дати на упаковках Ініціювати заміну ліків Вести журнал обліку	Аптечка Властивості: Номер Місцезнаходження Поточний стан Дата останньої перевірки Обов'язки Зберігати медичні препарати Забезпечувати швидкий доступ до засобів Проходити регулярну інвентаризацію
Менеджер із закупівель Властивості: ПІБ Службовий номер Бюджет на закупівлі Обов'язки Отримати заявку від відповідального Сформувати замовлення для постачальника Оплатити рахунок Прийняти товар	Графік перевірок Властивості: Частота контролю Дата останньої ревізії Дата наступної ревізії Статус Обов'язки Розраховувати дату наступної перевірки Повідомляти про настання терміну Змінювати періодичність Фіксувати факт виконання	Медичний препарат Властивості: Назва Серійний номер Термін придатності Кількість Обов'язки Мати визначений термін придатності Бути використаним для надання допомоги Підлягати списанню при протермінованості

Рис. 1.4. Виділені абстракції предметної області

Серед абстракцій – Постачальник, Відповідальна особа, Аптечка, Менеджер із закупівель, Графік перевірок, Медичний препарат. Як і у випадку з діаграмою прецедентів, ця множина абстракцій ширша за модель реалізованого програмного забезпечення. Постачальник і Менеджер із закупівель не отримали свого втілення у фінальній структурі бази даних, оскільки модуль закупівель залишений поза межами системи. Абстракція Аптечка перетворилася на центральну сутність моделі даних, абстракція Відповідальна особа – на сутність User з відповідною роллю, абстракція Медичний препарат – на сутність Medication, а функції Графіка перевірок було прибрано.

1.4 Огляд інформаційних джерел та існуючих рішень

Перед розробкою власної системи проведено огляд існуючих рішень на ринку. Аналоги можна розбити на чотири категорії: паперові журнали, універсальні системи обліку складу, медичні модулі в ERP і спеціалізовані рішення для клінік.

1.4.1 Паперові журнали обліку

Найбільш поширений на сьогодні підхід – це повністю ручний облік на паперових бланках. Відповідальний веде окремий журнал для кожної аптечки, в якому фіксує дати інвентаризацій, виявлені проблеми та дії. Перевага цього підходу одна – нульова вартість впровадження. Недоліки очевидні: відсутність автоматизованого контролю термінів, висока трудомісткість при великій кількості аптечок, ризик втрати або зміну записів, відсутність будь-якої централізованої видимості стану аптечок керівництвом. Саме цей сценарій є стартовою точкою для більшості клієнтів, яким адресовано систему FAIMS.

1.4.2 Універсальні системи управління складом

До цієї категорії належать продукти типу EZOfficeInventory, Sortly, inFlow Inventory. Це досить зрілі рішення, які дозволяють вести облік будь-яких матеріальних активів. Перевагами є гнучкість моделі даних, готова мобільна частина, наявність штрихкодування. Однак для задачі обліку аптечок такі системи мають істотні недоліки. По-перше, у них немає вбудованого розуміння терміну придатності як ключового атрибута. По-друге, інтерфейс орієнтований на загальний склад, а не на роботу з аптечкою як єдиним об'єктом. По-третє, ці продукти орієнтовані переважно на англомовний ринок, не локалізовані українською і не враховують специфіки української нормативної бази.

1.4.3 Медичні модулі всередині ERP-систем

Великі компанії з власним відділом охорони праці іноді ведуть облік аптечок як частину корпоративної ERP-системи на базі SAP, Oracle або 1C. Перевагами є інтеграція з рештою бізнес-процесів – закупівлями, бюджетуванням, кадровим обліком. Недоліками – висока вартість впровадження і супроводу, складність налаштування, надлишкова функціональність для задачі такого масштабу, прив'язка до внутрішнього IT-відділу клієнта. Для малих і середніх організацій – шкіл, дитячих садків, невеликих підприємств – цей підхід є невиправданим.

1.4.4 Спеціалізовані медичні рішення

На міжнародному ринку існують продукти типу MedStock, які орієнтовані на облік медикаментів у лікарнях, аптеках і клініках. Вони мають хорошу підтримку специфіки медикаментів, серійних номерів, термінів придатності, нормативних вимог. Однак мають незручний інтерфейс, відсутня локалізація української мови і невідповідність законодавству України.

1.4.5 Висновок з огляду

Аналіз показує, що ніша спеціалізованої, простої у використанні, локалізованої українською мовою системи обліку аптек першої допомоги для масового сегменту малих і середніх організацій залишається фактично відкритою. Існуючі рішення або занадто загальні, або занадто складні, або занадто дорогі, або відсутні на українському ринку. Ця обставина є основним обґрунтуванням актуальності розробки власної системи, яка цілеспрямовано орієнтована саме на цей сегмент і саме на цей сценарій використання.

Для наочного порівняння розглянутих рішень складено зведену таблицю за ключовими критеріями вибору системи обліку аптек (табл. 1.1).

Порівняльний аналіз існуючих рішень та системи FAIMS

Критерій	Паперовий журнал	EZOfficeInventory / Sortly	SAP / 1C	MedStock	FAIMS
Push-сповіщення про терміни придатності	Ні	Обмежено	Ні	Так	Так
Мобільний додаток	Ні	Так	Частково	Так	Так
Мультитенантність	Ні	Так	Так	Так	Так
Ціна впровадження	Низька	Середня	Висока	Низька	Низька
Локалізація українською мовою	—	Ні	Частково	Ні	Так
Специфіка аптечки першої допомоги	Так	Ні	Ні	Так	Так

1.5 Постановка завдання

Після всього вищесказаного можна чітко сформулювати завдання.

Об’єкт дослідження – процеси обліку аптечок першої допомоги в організаціях.

Предмет дослідження – методи та засоби автоматизації цих процесів за допомогою кросплатформного мобільного застосунку із серверною частиною.

Завдання роботи – розробити мультитенантну SaaS-систему обліку аптечок першої допомоги, яка складається з мобільного клієнта на платформах Android та iOS і серверної частини з REST API.

Система має забезпечити такі ключові можливості:

- самостійну реєстрацію організації-клієнта та подальше управління її організаційною структурою – підрозділами, кімнатами, аптечками;
- розподіл прав доступу між адміністратором організації та відповідальними особами із суворою ізоляцією даних різних організацій;
- облік медикаментів у складі аптечок з контролем кількості та термінів придатності за принципом «світлофора»;
- автоматичні push-сповіщення про наближення термінів придатності;
- журналювання всіх операцій для подальшого аудиту;
- формування PDF-звітів та аналітичних графіків для адміністратора;
- безпечну автентифікацію через JWT з можливістю безшовного оновлення сесії;
- розгортання у контейнерах Docker з автоматичним отриманням TLS-сертифіката для публічного домену.

Поза межами поточної версії продукту залишено повний цикл взаємодії з постачальниками – формування заявок, обробку рахунків, ведення договорів. Ці процеси можуть бути додані в наступних ітераціях системи без принципової зміни архітектури.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Моделі цього розділу описують саме те, що реально реалізовано. Це відрізняє їх від моделей попереднього розділу, де розглядалася ширша предметна область. Так, з 10 абстракцій, виділених у пункті 1.3.4, у фінальну модель даних потрапили далеко не всі: «Постачальник», «Менеджер із закупівель», «Накладна» залишені поза межами MVP, оскільки модуль закупівель винесений за межі поточної версії продукту. Натомість додалися сутності, яких не було у початковій абстрактній моделі, але які виявилися необхідними при проєктуванні саме програмного рішення: BaseEntity як базовий клас спадкування, RefreshToken для механізму оновлення JWT-сесії, KitTemplate і KitTemplateItem для шаблонів аптечок.

2.1 Логічна модель даних

Логічна модель даних описує структуру інформаційної бази системи в термінах сутностей, їх атрибутів і зв'язків між ними. Модель побудована з розрахунку на реалізацію в реляційній СУБД, відповідає вимогам третьої нормальної форми і підтримує мультитенантність – ізоляцію даних різних організацій-клієнтів.

Модель містить 10 змістовних сутностей. Ще одна сутність – BaseEntity – є технічним базовим класом і не має власної таблиці у фізичній реалізації; її поля Id, OrganizationId, CreatedAt, UpdatedAt успадковуються кожною таблицею як власні стовпці. Перелік сутностей такий: Organization – організація-клієнт; Department – підрозділ організації; Room – кімната у складі підрозділу; FirstAidKit – аптечка; Medication – медикамент у складі аптечки; User – користувач системи; Journal – журнал дій користувачів; KitTemplate – шаблон

комплектації аптечки; KitTemplateItem – позиція шаблону; RefreshToken – refresh-токен для оновлення JWT-сесії.

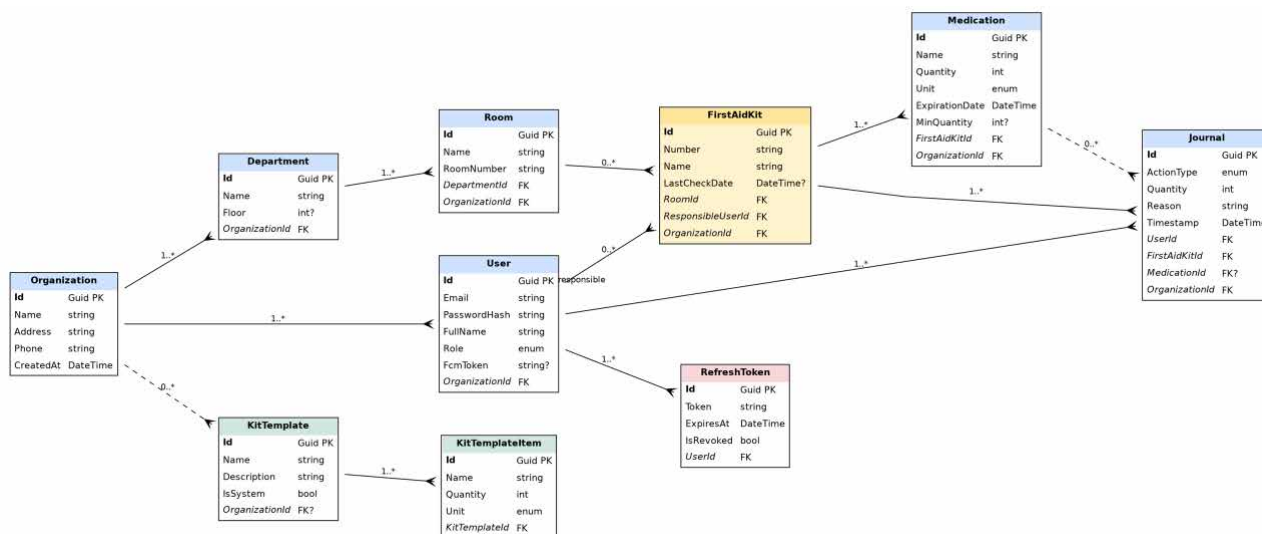


Рис. 2.1. ER-діаграма реалізованої системи FAIMS

Центральною сутністю моделі є FirstAidKit. Така структура відображає реальну ситуацію: аптечка не існує сама по собі, вона завжди прив'язана до конкретного фізичного місця, за нею завжди стоїть конкретна людина, і вона завжди містить деякий набір медикаментів.

Зв'язок між FirstAidKit і Medication є композиційним: видалення аптечки тягне за собою видалення всіх її медикаментів, а медикамент не може існувати поза якоюсь аптечкою. На рівні бази даних це реалізовано через зовнішній ключ FirstAidKitId з правилом ON DELETE CASCADE. Аналогічно – для пар Department–Room і Organization–Department.

Зв'язок User → FirstAidKit, що позначає закріплення відповідальної особи, є асоціативним і опціональним з боку аптечки: ResponsibleUserId є nullable-полем, оскільки технічно можлива ситуація, коли аптечка щойно створена і відповідального ще не призначено. З боку користувача цей зв'язок один-до-багатьох: один користувач може відповідати за декілька аптечок одночасно.

Сутність Journal зберігає історію всіх операцій з медикаментами і має три зовнішні ключі: UserId – хто виконав дію, FirstAidKitId – на якій аптечці,

MedicationId – над яким конкретно медикаментом. Поле ActionType є переліком, який включає такі значення: Add, Use, Replenish, Dispose.

Окремо стоять сутності KitTemplate і KitTemplateItem. Вони не містять інформації про конкретні медикаменти конкретних аптек – це лише шаблони, з яких можна створити нову аптечку, заповнивши її початковим набором позицій за натиском однієї кнопки. Шаблони бувають двох видів: системні, які створюються автоматично при першому запуску сервера на основі нормативних переліків, і користувацькі, які створюються адміністратором в межах своєї організації. Дворівнева структура «шаблон – позиція шаблону» дозволяє у майбутньому додавати функціональність на кшталт версіонування шаблонів або імпорту-експорту шаблонів між організаціями.

Сутність RefreshToken існує для реалізації механізму безшовного оновлення JWT-сесії. Кожному успішному логіну користувача відповідає створення одного RefreshToken з обмеженим часом життя. Поле IsRevoked дозволяє примусово відкликати токен, наприклад при логавті користувача з усіх пристроїв, не чекаючи природного завершення його дії.

Мультитенантність на рівні моделі даних. Поле OrganizationId присутнє у всіх сутностях, які належать клієнтським даним: Department, Room, FirstAidKit, Medication, User, Journal та користувацьких KitTemplate. Це поле пов’язує запис з конкретною організацією і використовується глобальним фільтром EF Core для автоматичної фільтрації запитів. Сутність RefreshToken не містить OrganizationId безпосередньо, оскільки звертатися до неї дозволено тільки за логікою автентифікації, де ідентифікатор організації встановлюється з самого токена. Системні KitTemplate також не мають OrganizationId – це загальнодоступні шаблони, спільні для всіх клієнтів системи.

2.2 Діаграма класів та кооперацій

На основі логічної моделі даних побудовано об'єктну модель системи. Кожна змістовна сутність бази даних представлена своїм доменним класом у шарі Domain. Усі класи успадковують абстрактний базовий клас BaseEntity, який містить технічні поля, спільні для всіх сутностей: Id, OrganizationId, CreatedAt, UpdatedAt. Це дозволяє писати загальні фільтри, репозиторії і базові класи, які працюють з будь-якою сутністю без знання її конкретного типу.

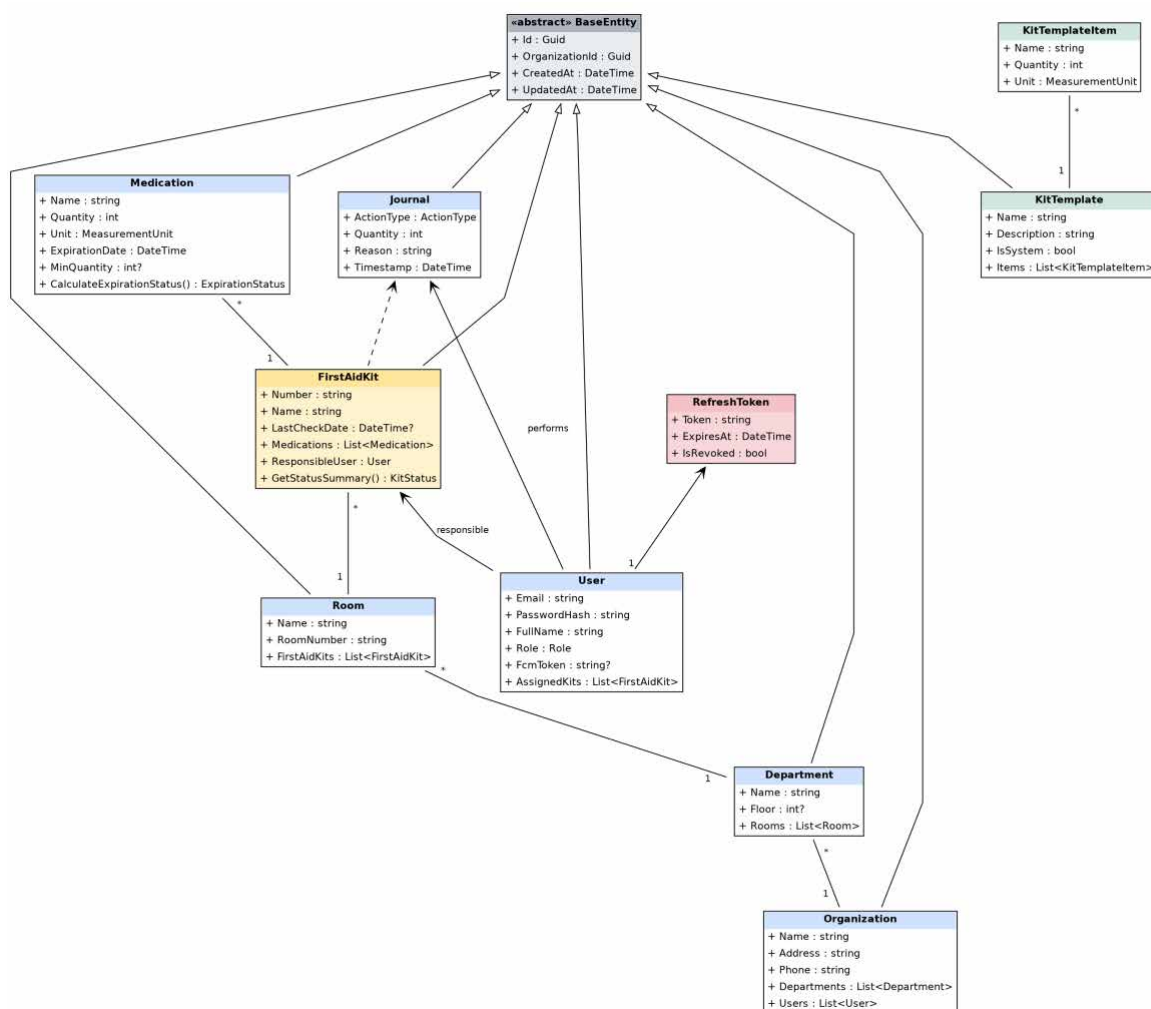


Рис. 2.2. Діаграма класів доменного шару FAIMS

На відміну від моделі даних, об'єктна модель додає до атрибутів також методи. Серед методів, які заслуговують особливої уваги: `CalculateExpirationStatus` у класі `Medication` – метод, що обчислює статус «світлофора» на основі поточної дати і дати спливу терміну придатності;

GetStatusSummary у класі FirstAidKit – метод, що агрегує статуси всіх медикаментів у аптечці і повертає загальний стан аптечки. Ці методи інкапсують бізнес-логіку всередині відповідних доменних класів і дозволяють використовувати її в різних місцях коду без дублювання.

Загальна діаграма класів великого розміру і складна для одночасного сприйняття. Тому при проектуванні було корисно розділити її на простіші кооперації – групи з кількох класів, що взаємодіють у межах одного бізнес-сценарію. Нижче представлені три ключові кооперації, кожна з яких відповідає одному з основних сценаріїв системи.

2.2.1 Кооперація «Інвентаризація та контроль аптечки»

Ця кооперація описує сценарій планової перевірки стану аптечки: відповідальна особа отримує сповіщення або переглядає аптечку самостійно, бачить статуси кожного медикаменту відповідно до правил «світлофора» і за результатами перевірки фіксує дату останньої перевірки в полі LastCheckDate.



Рис. 2.3. Кооперація «Інвентаризація та контроль аптечки»

2.2.2 Кооперація «Закупівля та поповнення запасів»

Друга кооперація описує процес поповнення запасів. У повній моделі предметної області в ньому беруть участь зовнішні актори – менеджер із закупівель і постачальник; у реалізації MVP поповнення зводиться до операції додавання нової партії медикаменту відповідальною особою через мобільний застосунок. Сутність Постачальник присутня на діаграмі для повноти моделі, але не реалізована в поточній версії продукту.

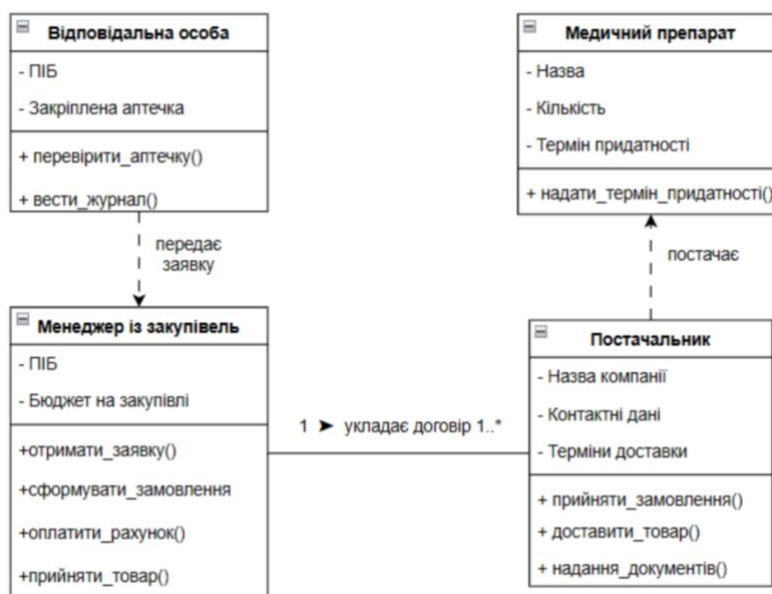


Рис. 2.4. Кооперація «Закупівля та поповнення запасів»

2.2.3 Кооперація «Використання аптечки»

Третя кооперація описує найбільш частотний сценарій – використання аптечки в момент потреби. Відповідальний за аптечку відкриває аптечку, шукає потрібний засіб, видає його і фіксує зменшення кількості. Сутність Працівник підприємства присутня на діаграмі як ініціатор сценарію, але в системі не моделюється явно: облік від конкретного «пацієнта» не ведеться, важливо тільки те, що медикамент списано з конкретної кількості і причиною списання було реальне використання.

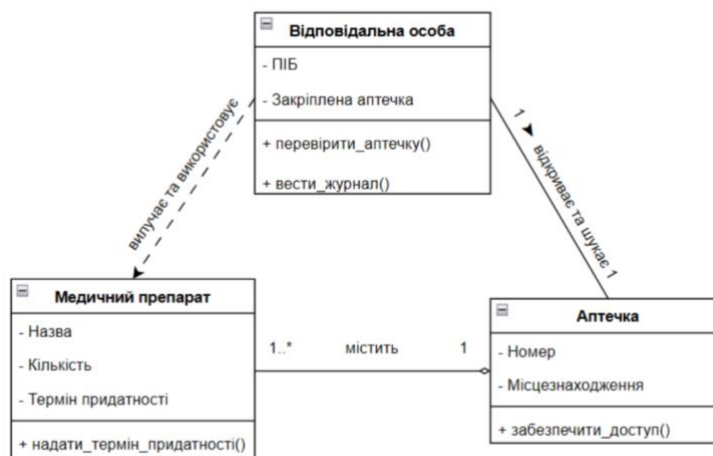


Рис. 2.5. Кооперація «Використання аптечки»

Декомпозиція на кооперації дозволяє зіставити кожну з них з окремим варіантом використання, що спрощує сприйняття діаграм. Крім того, така структура природно відповідає патернам Repository та Service, які і застосовані в реалізованій системі.

2.3 Діаграма пакетів

Якщо діаграма класів показує систему зсередини, то діаграма пакетів — об'єкти і зв'язки. Архітектура системи побудована за принципами Clean Architecture [4], які полягають у чіткому розділенні системи на шари з суворим однонапрямленим правилом залежностей: внутрішні шари не повинні знати про існування зовнішніх.

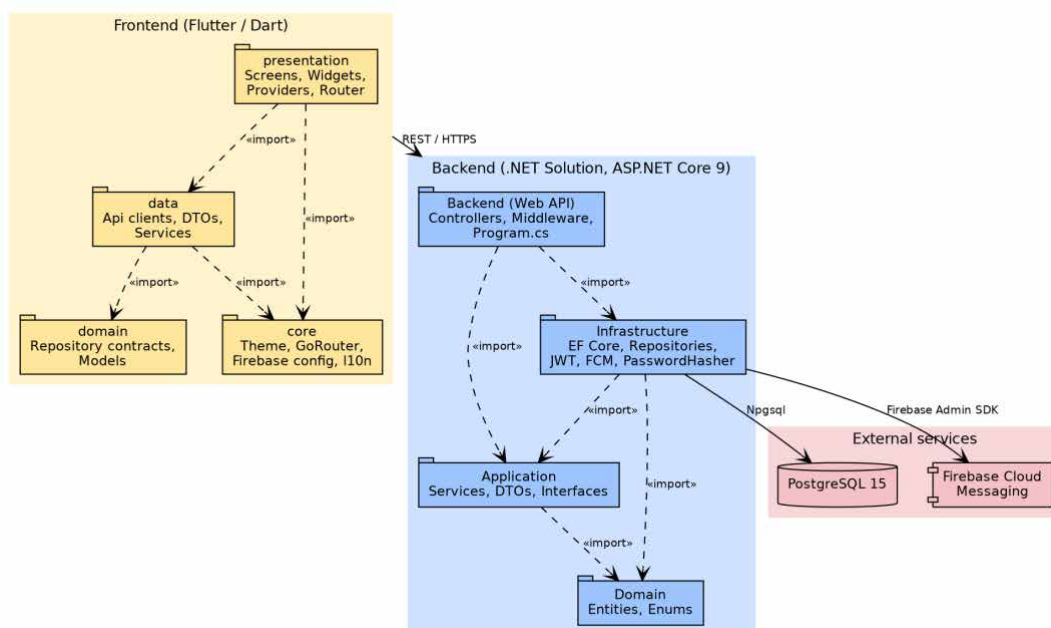


Рис. 2.6. Діаграма пакетів системи FAIMS

Серверна частина організована як .NET Solution, що містить чотири проєкти, які відповідають чотирьом шарам Clean Architecture.

Domain – ядро системи, найвнутрішніший шар. Містить класи сутностей, переліки, базовий клас BaseEntity, доменні виключення. Важливо, що цей пакет не залежить ні від чого: ні від ASP.NET Core, ні від Entity Framework, ні від будь-якої іншої зовнішньої бібліотеки. Це гарантує, що бізнес-правила можна тестувати ізольовано і переносити між проєктами без додаткових залежностей.

Application – шар бізнес-логіки. Містить інтерфейси сервісів і репозиторіїв, реалізації сервісів, DTO-класи для передачі даних між шарами, конфігурації AutoMapper. Application залежить тільки від Domain. У цьому шарі реалізовано основні алгоритми системи: обчислення статусу медикаменту, формування звітів, перевірка прав доступу.

Infrastructure – шар, що реалізує доступ до зовнішніх ресурсів: бази даних через EF Core, сервіс push-сповіщень через Firebase Admin SDK, генерацію JWT-токенів, хешування паролів. Цей шар реалізує контракти, оголошені в Application, але сама Application про конкретні реалізації не знає – вона працює тільки з абстракціями. Така інверсія залежностей є ключовою ідеєю Clean

Architecture: завдяки їй заміна постачальника бази даних або провайдера push-сповіщень не вимагає змін у бізнес-логіці.

Backend – зовнішній шар, точка входу веб-API. Містить REST-контролери, middleware для обробки помилок і автентифікації, фоновий сервіс перевірки термінів придатності `ExpirationCheckHostedService`, конфігурацію `Program.cs`. Цей шар залежить від `Application` та `Infrastructure` через DI-контейнер.

Мобільний клієнт організовано за схожим принципом, з адаптацією до особливостей Flutter. Усередині директорії `frontend/lib` знаходяться чотири пакети.

presentation – екрани користувацького інтерфейсу, віджети, провайдери стану на базі `Provider`, налаштування навігації через `GoRouter`.

data – клієнти REST API, DTO для серіалізації і десеріалізації JSON, сервіси-обгортки для виклику серверних ендпоінтів, локальне безпечне зберігання токенів через `flutter_secure_storage`.

domain – контракти репозиторіїв і моделі. На відміну від серверного `Domain`, тут немає багатої доменної логіки – мобільний клієнт працює переважно з даними, отриманими від API, і не дублює бізнес-правила сервера.

core – загальна інфраструктура застосунку: тема оформлення, конфігурація `Firebase`, визначення маршрутизатора, ARB-файли локалізації для української та англійської мов, утиліти для генерації PDF-звітів.

Між клієнтом і сервером встановлено єдиний канал зв'язку – REST API через HTTPS. Усі дані передаються у форматі JSON. Для автентифікації мобільний клієнт додає до кожного запиту заголовок `Authorization` з JWT-токеном, отриманим при логіні.

2.4 Діаграма компонентів

Діаграма пакетів описує організацію коду. Діаграма компонентів натомість показує, з яких артефактів складається готова до розгортання система: бінарних файлів, бібліотек, скриптів, конфігураційних файлів. Це останній рівень проєктування перед фактичним розгортанням системи.

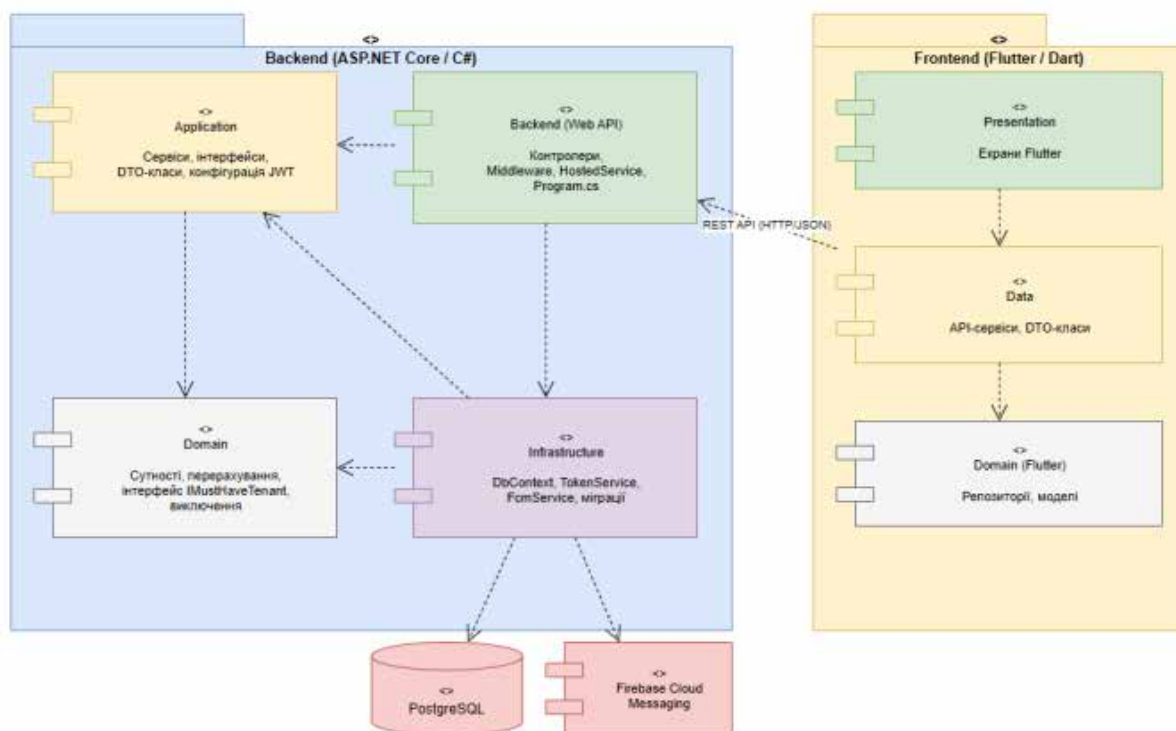


Рис. 2.7. Діаграма компонентів системи FAIMS

На рівні компонентів серверна частина перетворюється на чотири DLL-бібліотеки: `Domain.dll`, `Application.dll`, `Infrastructure.dll`, `Backend.dll`. Залежності між ними повторюють залежності між пакетами вихідного коду. Усі чотири компоненти збираються командою `dotnet build` і об'єднуються командою `dotnet publish` у фінальний артефакт – папку з `Backend.dll`, конфігураційними файлами `appsettings.json` та `appsettings.Production.json` і всіма необхідними залежностями. Цей артефакт надалі копіюється в Docker-образ і запускається на продуктивному сервері.

Мобільний клієнт компілюється за іншою схемою. Перед збіркою виконується автогенерація DTO-файлів і файлів локалізації командами `dart run`

`build_runner build` та `flutter gen-l10n`. Потім команда `flutter build apk` формує Android-пакет `FAIMS.apk`, а команда `flutter build ios` – iOS-архів `FAIMS.ipa`. Ці артефакти і є тим, що розповсюджується кінцевим користувачам через Google Play та App Store.

Окремо стоять зовнішні компоненти, які не входять у склад розповсюджуваних артефактів, але є необхідними для роботи системи: PostgreSQL як сервер бази даних і Firebase Cloud Messaging як зовнішній сервіс push-сповіщень. PostgreSQL у продуктивному середовищі запускається в окремому Docker-контейнері поряд з Backend, а Firebase Cloud Messaging – це хмарний сервіс Google, з яким Backend взаємодіє через Firebase Admin SDK по протоколу HTTPS.

3 РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Система управління інформаційною базою

При виборі системи управління інформаційною базою для проєкту FAIMS розглядалися такі варіанти: файлове зберігання у вигляді JSON або SQLite на стороні мобільного клієнта, реляційні СУБД серверного класу – Microsoft SQL Server, MySQL, PostgreSQL – та документоорієнтовані СУБД на кшталт MongoDB.

Файлове зберігання було відкинуто одразу. Воно технічно достатнє для застосунку з одним користувачем, але не може забезпечити мультитенантність, паралельний доступ кількох клієнтів і централізовану звітність. Документоорієнтована СУБД MongoDB також була невідповідною тому, що модель даних FAIMS виразно реляційна.

З трьох розглянутих реляційних СУБД обрано PostgreSQL версії 15 [7]. Вибір обґрунтовано такими аргументами.

По-перше, PostgreSQL є відкритою СУБД з ліцензією, що дозволяє безкоштовне комерційне використання. Для цього проєкту вартість ліцензій є критичним фактором. Мінімальна ліцензія Microsoft SQL Server Standard коштує близько кількох тисяч доларів на рік, що для невеликого продукту з повільною монетизацією є недоречно і невигідно.

По-друге, PostgreSQL має повноцінну підтримку Entity Framework Core через офіційний провайдер Npgsql [5]. Це означає, що всі основні можливості EF Core – міграції, LINQ-запити, автоматичне мапінг типів, глобальні фільтри запитів.

По-третє, PostgreSQL підтримує розширені типи даних, які виявилися корисними при моделюванні системи. Зокрема, тип UUID використовується як тип первинного ключа для всіх сутностей; це дозволяє генерувати ідентифікатори на стороні застосунку, не звертаючись до бази даних, і безпечно

об'єднувати дані з різних джерел без ризику колізій. Підтримка enum-типів дозволяє представляти такі поля, як ActionType у журналі або Role у користувача, як типобезпечні переліки замість «магічних» чисел або рядків.

По-четверте, PostgreSQL має зрілу підтримку транзакцій з рівнем ізоляції за замовчуванням Read Committed і реалізацією багатоверсійного контролю паралелізму MVCC. Це гарантує, що довгі звіти, які читають великі обсяги даних, не блокують короткі транзакції оновлення стану медикаментів і навпаки.

По-п'яте, PostgreSQL має широку екосистему хмарних провайдерів і можливість розгортання у будь-якому Linux-середовищі через Docker. Для FAIMS обрано саме контейнерне розгортання на віртуальному сервері DigitalOcean у регіоні Frankfurt, що дає прийнятну затримку для українських користувачів і відповідає вимогам європейського законодавства про захист персональних даних.

3.2 Розробка інформаційної бази

Перехід від логічної моделі до фізичної реалізації виконувався не вручну, а через механізм Code First, який підтримує Entity Framework Core [5]. Суть підходу полягає в тому, що структура бази даних описується не SQL-скриптами і не в спеціалізованому інструменті типу ERwin, а безпосередньо у вигляді C#-класів. EF Core аналізує ці класи разом з конфігураційним файлом DbContext і автоматично генерує SQL-скрипти створення таблиць, ключів, індексів та обмежень.

Перевагою такого підходу є те, що схема бази даних і доменна модель завжди залишаються синхронізованими. Інструмент дотнет-консолі `dotnet ef migrations add` створює міграційний скрипт за результатами порівняння поточного стану моделі з останньою застосованою міграцією; команда `dotnet ef database update` застосовує цей скрипт до фізичної бази. У продуктивному середовищі міграції виконуються автоматично при запуску контейнера Backend.

Лістинг опису сутності наведено в додатку A.1

Конфігурація бази даних і налаштування мультитенантного фільтра виконується в класі `ApplicationDbContext`. Глобальний фільтр додається в метод `OnModelCreating` через рефлексію – для кожного типу сутності, що реалізує `IMustHaveTenant`, автоматично додається умова відбору за поточним `OrganizationId`. Поточний ідентифікатор організації береться з `ITenantProvider` – сервісу, який витягує його з JWT-токена поточного запиту.

Лістинг глобального фільтра мультитенантності наведено в додатку A.2

Ключовою перевагою такого підходу є те, що жоден з прикладних сервісів не повинен пам'ятати про мультитенантність явно. Звичайний LINQ-запит на кшталт `_db.FirstAidKits.Where(k => k.Number == "001").ToListAsync()` автоматично перетворюється EF Core у SQL з додатковою умовою `WHERE OrganizationId = @currentTenant`. Помилитися і випадково отримати дані іншої організації фактично неможливо: для цього потрібно навмисно відключити

фільтр через метод `IgnoreQueryFilters`, що використовується тільки в спеціальних адміністративних сценаріях.

Для індексації найчастіше запитуваних полів додатково використовується `Fluent API EF Core`. Зокрема, на полі `Email` сутності `User` створено унікальний індекс у складі `OrganizationId` – це означає, що один email може використовуватися кількома різними організаціями, але всередині однієї організації email є унікальним. На полі `ExpirationDate` сутності `Medication` створено звичайний індекс, який пришвидшує щоденне сканування медикаментів фоновим сервісом.

3.3 Вибір інструментарію для створення прикладного програмного забезпечення

Після вибору СУБД потрібно було визначитися з інструментами для написання самого коду. Розглянув кілька варіантів для сервера і для мобільного клієнта.

3.3.1 Серверна частина

Як платформу для серверної частини обрано `ASP.NET Core 9.0` разом з мовою `C#` [4]. На користь цього вибору були такі аргументи. Платформа `.NET` є зрілою, добре документованою і має офіційну підтримку `Microsoft` з регулярними оновленнями безпеки. Мова `C#` – статично типізована, з потужною системою типів і підтримкою сучасних функціональних конструкцій, що зменшує ймовірність помилок ще на етапі компіляції. `ASP.NET Core` має вбудовану підтримку `Dependency Injection` через стандартний контейнер, що значно спрощує реалізацію `Clean Architecture`. Платформа крос-платформна: серверний застосунок збирається у `Linux`-контейнер без жодних змін у вихідному коді.

Альтернативами розглядалися Node.js з фреймворком Express або NestJS, Python з Django або FastAPI, Java з Spring Boot. Node.js був відкинутий через слабшу систему типів навіть у TypeScript-варіанті і відсутність порівнянної з EF Core ORM. Python розглядався, але був визнаний менш ефективним для довготривалих серверних процесів через GIL.

Перелік основних бібліотек серверної частини наведено нижче. Усі вони є офіційними або де-факто стандартними рішеннями для відповідних задач у середовищі ASP.NET Core.

- Microsoft.AspNetCore.App – базовий метапакет ASP.NET Core 9.0;
- Microsoft.EntityFrameworkCore – ORM, версія 9.x;
- Npgsql.EntityFrameworkCore.PostgreSQL – провайдер EF Core для PostgreSQL;
- Microsoft.AspNetCore.Authentication.JwtBearer – middleware для перевірки JWT;
- System.IdentityModel.Tokens.Jwt – генерація і валідація JWT;
- FirebaseAdmin – клієнт Firebase Admin SDK для FCM;
- Serilog.AspNetCore – структуроване логування;
- Swashbuckle.AspNetCore – генерація Swagger / OpenAPI документації;
- AutoMapper – мапінг між DTO і доменними сутностями;
- Bogus – генерація реалістичних тестових даних для seeder’а;

3.3.2 Мобільний клієнт

Як платформу мобільного клієнта обрано Flutter – фреймворк компанії Google для створення кросплатформних застосунків [8]. Flutter дозволяє з єдиної кодової бази на мові Dart збирати нативні застосунки для Android та iOS з ідентичним зовнішнім виглядом і поведінкою. Альтернативами були React Native, нативна розробка окремо під Android і iOS, а також веб-додаток.

Flutter рендерить інтерфейс власним графічним рушієм Skia, що гарантує ідентичний вигляд на обох платформах і дає повний контроль над піксельним відтворенням, що є критично для системи, де колірне кодування статусів медикаментів є одним з ключових елементів інтерфейсу. Flutter має зрілу екосистему пакетів, включно з пакетами для Firebase, генерації PDF, автентифікації, графіків. Hot reload суттєво пришвидшує цикл розробки. Нарешті, одна кодова база замість двох знижує загальні витрати на підтримку приблизно вдвічі.

Перелік основних пакетів мобільного клієнта:

- provider – управління станом за патерном Provider/InheritedWidget;
- go_router – декларативна навігація з підтримкою глибоких посилань;
- google_fonts – фірмова типографіка (шрифт Manrope);
- fl_chart – інтерактивні графіки для модулів аналітики;
- pdf, printing – генерація та експорт PDF-звітів на пристрої;
- firebase_messaging – отримання push-сповіщень;
- flutter_secure_storage – безпечне зберігання токенів автентифікації;
- json_serializable, build_runner – автогенерація серіалізації DTO;
- intl, flutter_localizations – локалізація через ARB-файли;

3.3.3 Інфраструктурні інструменти

Для контейнеризації використовується Docker з оркестрацією через docker-compose [11]. Вибір очевидний: Docker де-факто є стандартом для розгортання сучасних серверних застосунків. У файлі docker-compose.yml описано чотири сервіси: postgres_db – база даних, backend – серверний застосунок, caddy – reverse-проху, pg_backup – періодичне резервне копіювання. Кожен сервіс має обмеження пам'яті і політику перезапуску для забезпечення стабільності на сервері з обмеженими ресурсами.

Caddy обрано як reverse-проху через дві ключові переваги перед альтернативами Nginx або Apache: автоматичне отримання і поновлення TLS-сертифікатів від Let's Encrypt без жодних додаткових налаштувань, та надзвичайно простий синтаксис конфігураційного файлу. Конфігурація Caddyfile для FAIMS вміщується в кілька рядків – він приймає трафік на порту 443, проксує його на порт серверного застосунку, і в разі відсутності сертифіката автоматично запитує новий через ACME-протокол.

Як хмарний провайдер обрано DigitalOcean. На користь нього спрацювала прозора цінова політика, простота управління через вебінтерфейс і API, наявність регіону Frankfurt з низькою затримкою для українських користувачів, і досвід використання у попередніх проєктах. Альтернативами розглядалися AWS і Hetzner – перший складніший в управлінні і дорожчий для невеликого продукту, другий має більш строгу політику щодо контенту користувачів.

3.4 Алгоритмізація та програмування програмних модулів

Тут розглянуто три ключових алгоритми: обчислення статусу за принципом «світлофора», автентифікація через JWT з refresh-токенами і фоновий сервіс сповіщень.

3.4.1 Алгоритм обчислення статусу медикаменту

Алгоритм «світлофора» викликається постійно – при кожному запиті списку медикаментів, при формуванні звітів, при скануванні перед сповіщеннями. Він простий і це добре: детермінований, без зовнішніх запитів, виконується за константний час. Блок-схема алгоритму наведена на рис. 3.1.

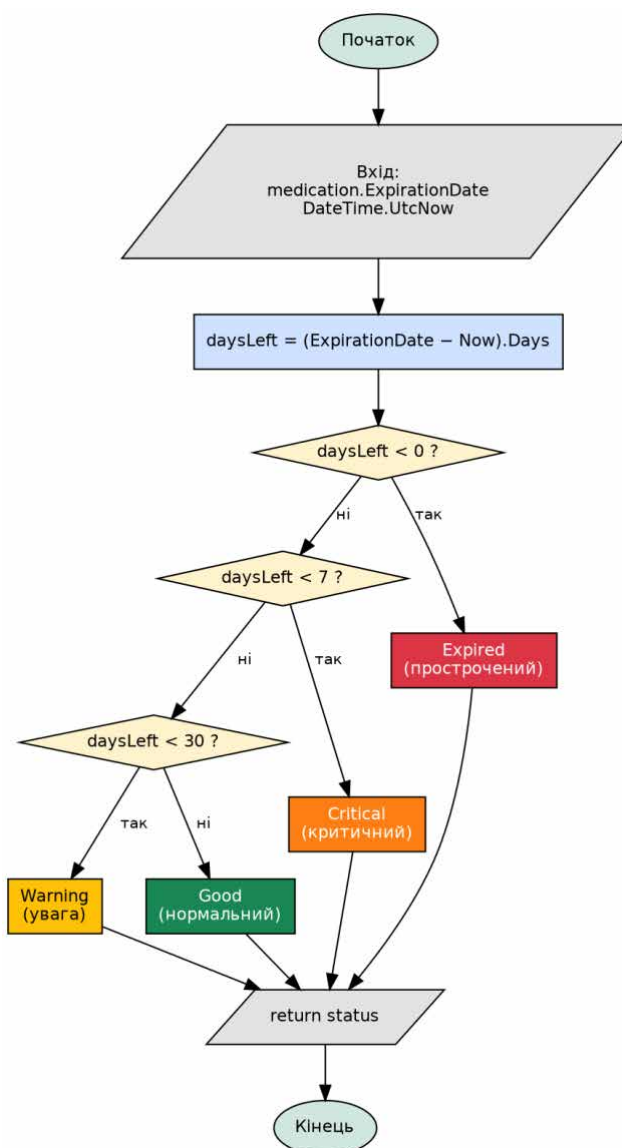


Рис. 3.1. Блок-схема алгоритму обчислення статусу медикаменту

Реалізація алгоритму на «світлофора» показана в додатку А.4.

Винесення поточної дати у параметр методу – навмисне: це дозволяє писати юніт-тести з фіксованою датою без потреби у мокуванні системного часу. Граничні випадки тестуються прямо: рівно 0 днів дає Critical, бо термін збігає сьогодні; -1 – Expired; рівно 30 – Warning; рівно 7 – Critical.

3.4.2 Алгоритм автентифікації через JWT

Автентифікація користувача побудована за стандартною схемою з двома типами токенів. Короткоживучий access token, тобто JWT, діє 15 хвилин і використовується для авторизації кожного запиту до API. Refresh token живе 30 днів, зберігається в безпечному сховищі мобільного пристрою і використовується тільки для отримання нового access token, коли поточний закінчується. Така схема дозволяє швидко обмежити доступ скомпрометованому пристрою.

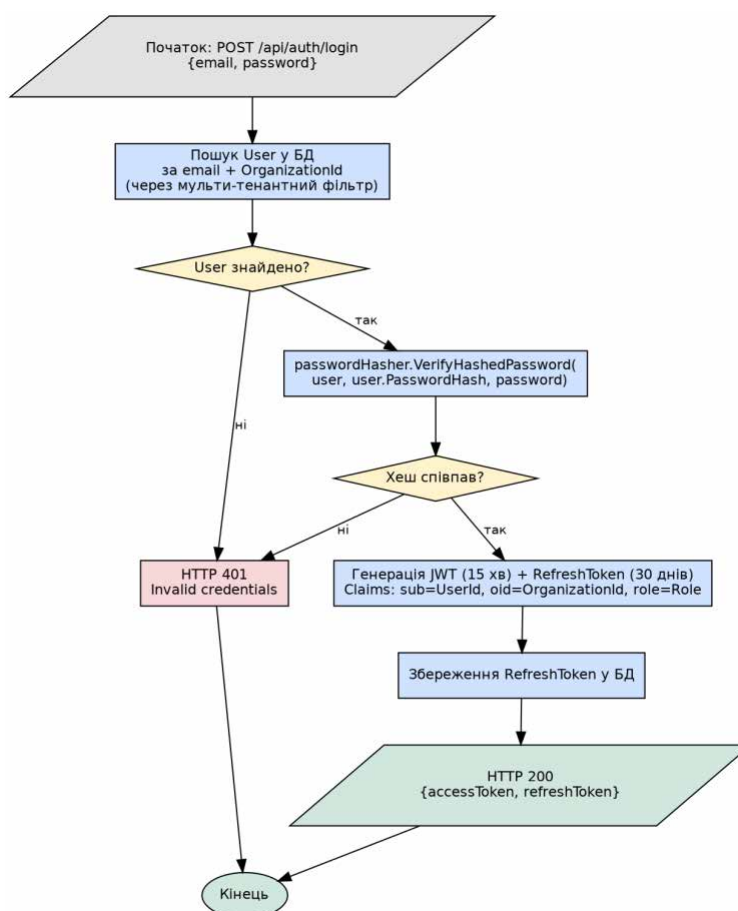


Рис. 3.2. Блок-схема алгоритму логіну

Ключові кроки логіну реалізовані в AuthService приблизно так:

```
public async Task<AuthResponseDto> LoginAsync(LoginDto dto)
{
    var user = await _db.Users
        .IgnoreQueryFilters() // глобальний
    логін
        .FirstOrDefaultAsync(u => u.Email == dto.Email);

    if (user is null)
        throw new UnauthorizedException("Invalid credentials");

    var verifyResult = _passwordHasher.VerifyHashedPassword(
        user, user.PasswordHash, dto.Password);

    if (verifyResult == PasswordVerificationResult.Failed)
        throw new UnauthorizedException("Invalid credentials");

    var accessToken = _jwtService.GenerateAccessToken(user);
    var refreshToken = _jwtService.GenerateRefreshToken(user);

    await _refreshTokens.AddAsync(refreshToken);
    await _db.SaveChangesAsync();

    return new AuthResponseDto(accessToken, refreshToken.Token);
}
```

Лістинг 3.5. Метод входу користувача в систему

Звернути увагу варто на виклик `IgnoreQueryFilters`: під час логіну ми ще не знаємо, до якої організації належить користувач, тому глобальний фільтр мультитенантності тимчасово вимикається. Після успішної автентифікації ідентифікатор організації витягується з користувача і кладеться в JWT-claim `oid`, з якого далі його забирає `ITenantProvider` при кожному наступному запиті.

Сам JWT генерується класом `JwtService` і наведений в додатку A.6.

3.4.3 Фоновий сервіс перевірки термінів придатності

Сервіс `ExpirationCheckHostedService` – це `BackgroundService` у термінології ASP.NET Core, тобто фоновий процес, який запускається при старті серверного застосунку і виконується паралельно з обробкою HTTP-запитів. Цей сервіс раз на добу сканує всі медикаменти у системі і для кожного медикаменту перевіряє, чи перейшов він у стан «увага», «критичний» або «прострочений» з моменту попередньої перевірки.

Якщо так – сервіс формує і відправляє push-сповіщення відповідальній особі через `Firebase Cloud Messaging`.

Фоновий сервіс перевірки термінів наведено в додатку A.5.

Усередині методу `CheckAndNotifyAsync` виконується сам обхід:

```
public async Task CheckAndNotifyAsync(CancellationTokent ct)
{
    var medications = await _db.Medications
        .IgnoreQueryFilters()
        .Where(m => m.Quantity > 0)
        .Include(m => m.FirstAidKit)
        .ThenInclude(k => k.ResponsibleUser)
        .ToListAsync(ct);

    foreach (var med in medications)
    {
        var status = ExpirationCalculator.Calculate(med.ExpirationDate);
        if (status == ExpirationStatus.Good) continue;

        var user = med.FirstAidKit.ResponsibleUser;
        if (user?.FcmToken is null) continue;

        await _fcm.SendAsync(user.FcmToken, BuildMessage(med, status), ct);
    }
}
```

Лістинг 3.8. Сканування медикаментів і відправка сповіщень

Сама відправка push-сповіщення через `Firebase Admin SDK` – це по суті обгортка над двома виключно простими методами `FirebaseMessaging.DefaultInstance.SendAsync`, з тим тільки нюансом, що `FirebaseAdmin` вимагає одноразової ініціалізації при старті застосунку з JSON-файлом сервісного облікового запису `Google Cloud`.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1 Тестування системи

Тестування йшло разом з розробкою. Мета – щоб кожна функція реально працювала як задумано, і всі знайдені баги фіксувалися до того, як хтось починає цим користуватися.

4.1.1 Види тестування

У проєкті використано три види тестування, кожен з яких відповідає за свій рівень коректності системи.

Модульне тестування – перевірка коректності окремих класів і методів у ізоляції від решти системи. Для серверної частини написано набір модульних тестів на платформі xUnit, які розташовані у проєкті Tests. Найбільше тестів охоплює ExpirationCalculator – клас обчислення статусу медикаменту, оскільки саме його коректність є критичною для всієї логіки сповіщень. Тести побудовані за принципом «дано-коли-тоді»: задається конкретна дата і термін придатності медикаменту, викликається метод обчислення, перевіряється, чи відповідає повернений статус очікуваному. Особлива увага приділена граничним значенням: рівно сьогоднішня дата, дата на одну добу в минулому, дати, що відстоять рівно на 7 і 30 днів від сьогодні.

Інтеграційне тестування – перевірка взаємодії кількох компонентів системи між собою. До цієї категорії належать сценарії, які охоплюють весь шлях запиту від REST-контролера через сервісний шар і репозиторій до бази даних і назад. Інтеграційне тестування дозволяє виявити проблеми, які неможливо побачити модульними тестами: неправильно налаштовані EF Core міграції, помилки в конфігурації автентифікації, некоректну роботу глобального фільтра мультитенантності.

Ручне функціональне тестування – проходження ключових сценаріїв системи людиною з мобільного пристрою. Цей вид тестування є незамінним для перевірки користувацького досвіду: чи зрозуміло, як виконати ту чи іншу дію, чи коректно відображаються кольори статусів, чи приходять push-сповіщення на реальний пристрій, чи відкривається PDF-звіт у мобільному застосунку для перегляду PDF. Ручне тестування проводилося як на емуляторі Android Studio, так і на реальних пристроях під керуванням Android та iOS.

4.1.2 Тестові сценарії

Нижче наведено перелік ключових сценаріїв, які перевірялися під час функціонального тестування. Кожен сценарій описує послідовність дій користувача і очікуваний результат.

1. Реєстрація нової організації. Незареєстрований користувач відкриває застосунок, переходить на екран реєстрації організації, заповнює форму з назвою організації, своїм ім'ям, email і паролем. Очікується: створено організацію, створено першого користувача з роллю адміністратора, виконано автоматичний вхід, відкрито головний екран адміністратора.

2. Створення підрозділу і кімнати. Адміністратор переходить у розділ управління підрозділами, створює новий підрозділ з назвою «Перший поверх», у ньому створює кімнату «101». Очікується: підрозділ і кімната відображаються у списку, доступні для прив'язки аптечок.

3. Створення аптечки з шаблону. Адміністратор створює нову аптечку, прив'язує її до раніше створеної кімнати, обирає системний шаблон «Базова аптечка», призначає відповідальну особу. Очікується: аптечка створена, в неї автоматично додано позиції з шаблону з нульовою кількістю.

4. Поповнення медикаменту. Відповідальна особа відкриває свою аптечку, обирає медикамент, натискає «Поповнити», вводить кількість і термін придатності. Очікується: запис у журналі дій з типом «Поповнення», оновлена кількість у списку.

5. Списання при використанні. Відповідальна особа списує частину медикаменту, вказуючи причину. Очікується: запис у журналі з типом «Використання», зменшена кількість.

6. Перевірка статусу «світлофора». Створюються медикаменти з різними термінами придатності: 0 день, +3 дні, +20 днів, +60 днів. Очікується: відображаються відповідно статуси Expired, Critical, Warning, Good з правильними кольорами.

7. Отримання push-сповіщення. Створюється медикамент з терміном придатності за 5 днів. Запускається фоновий сервіс. Очікується: на пристрій відповідальної особи приходить push-сповіщення з відповідним текстом.

8. Формування PDF-звіту. Адміністратор переходить у розділ звітів, обирає період і тип звіту «Критичні позиції», натискає «Сформувати». Очікується: відкривається PDF з переліком медикаментів зі статусом Critical і Expired.

9. Ізоляція даних організацій. Створюються дві різні організації під різними обліковими записами. Адміністратор однієї організації намагається через прямий API-запит з валідним токеном звернутися до даних іншої. Очікується: повертається порожній результат через дію глобального фільтра мультитенантності.

10. Безшовне оновлення сесії. Здійснюється логін, очікується спливання терміну дії access-токена, виконується запит до API. Очікується: клієнт автоматично використовує refresh-токен для отримання нового access-токена і повторює запит без участі користувача.

4.1.3 Підготовка тестового середовища

Для проведення ручного тестування передбачено вбудований у серверну частину механізм заповнення бази даних реалістичними тестовими даними. Він реалізований на бібліотеці Bogus, яка дозволяє генерувати правдоподібні імена, адреси, email-адреси, дати тощо. При запуску команди `dotnet run --seed` серверний застосунок повністю очищує всі таблиці і заповнює їх свіжим набором

даних: 5 тестових організацій, кілька десятків відділів і кімнат у кожній, по кілька аптек у середньому на організацію, по 10–20 медикаментів у кожній аптеці з реалістичними назвами препаратів та коректно прив’язаними одиницями виміру. Дати термінів придатності генеруються в широкому діапазоні, включаючи кілька навмисно прострочених позицій для перевірки роботи статусу Expired.

Після завершення seed у файл seed-credentials.txt записуються облікові дані всіх створених адміністраторів і відповідальних осіб. Це дозволяє швидко увійти в систему під будь-яким з тестових користувачів і спостерігати поведінку системи з різних точок зору.

4.1.3.1 Виявлені дефекти та способи їх усунення

У процесі тестування виявлено кілька нетривіальних проблем, які варто зафіксувати окремо.

Проблема з глобальним фільтром при логіні. Під час тестування сценарію 9 (ізоляція даних організацій) виявилось, що сервіс автентифікації під час входу повертав помилку 401 навіть для коректних облікових даних. Причина виявилась неочевидною: глобальний фільтр мультитенантності EF Core застосовувався і до запиту пошуку користувача під час логіну, але на той момент OrganizationId у ITenantProvider ще не було встановлено. Рішення: додати виклик IgnoreQueryFilters() саме для запиту пошуку користувача в AuthService. Цей момент добре видно у коді сервісу (Додаток А.4).

Проблема з Docker Compose і порядком запуску контейнерів. При першому розгортанні на VPS сервер ASP.NET Core стартував швидше за PostgreSQL і одразу падав з помилкою “connection refused”. Docker Compose за замовчуванням чекає лише на запуск контейнера, а не на готовність бази всередині нього. Проблема вирішено через healthcheck: postgres_db отримав перевірку pg_isready, а backend – залежність condition: service_healthy. Тепер сервер стартує лише після того, як PostgreSQL підтвердив готовність приймати з’єднання.

Проблема з push-сповіщеннями на iOS під час тестування через TestFlight. Під час перевірки сценарію 7 на реальному iPhone push-сповіщення не приходили, хоча на Android все працювало коректно. З'ясувалося, що для iOS потрібно окремо налаштувати APNs-ключ у консолі Firebase і правильно вказати bundle identifier застосунку. Після додавання ключа і перезбірки застосунку через TestFlight сповіщення почали надходити стабільно. Цей нюанс не очевидний з документації Firebase і зайняв кілька годин на налагодження.

Проблема з відображенням статусу на Flutter при переключенні між екранами. При поверненні з екрану деталей медикаменту на список аптечки кольори статусів іноді не оновлювалися – показувався старий стан. Причина: Provider не тригерував rebuild після оновлення даних, бо список повертався з кешу. Виправлено явним викликом notifyListeners() після кожної мутуючої операції та додаванням обробника didPopRoute у навігаційному шарі GoRouter для примусового оновлення стану при поверненні.

4.1.4 Скриншоти роботи системи

Нижче наведено скриншоти ключових екранів мобільного застосунку, отримані під час ручного тестування. Скриншоти ілюструють як інтерфейс адміністратора, так і інтерфейс відповідальної особи. Повний альбом скриншотів усіх екранів системи розміщено в Додатку Г.



Рис. 4.1. Головний екран адміністратора

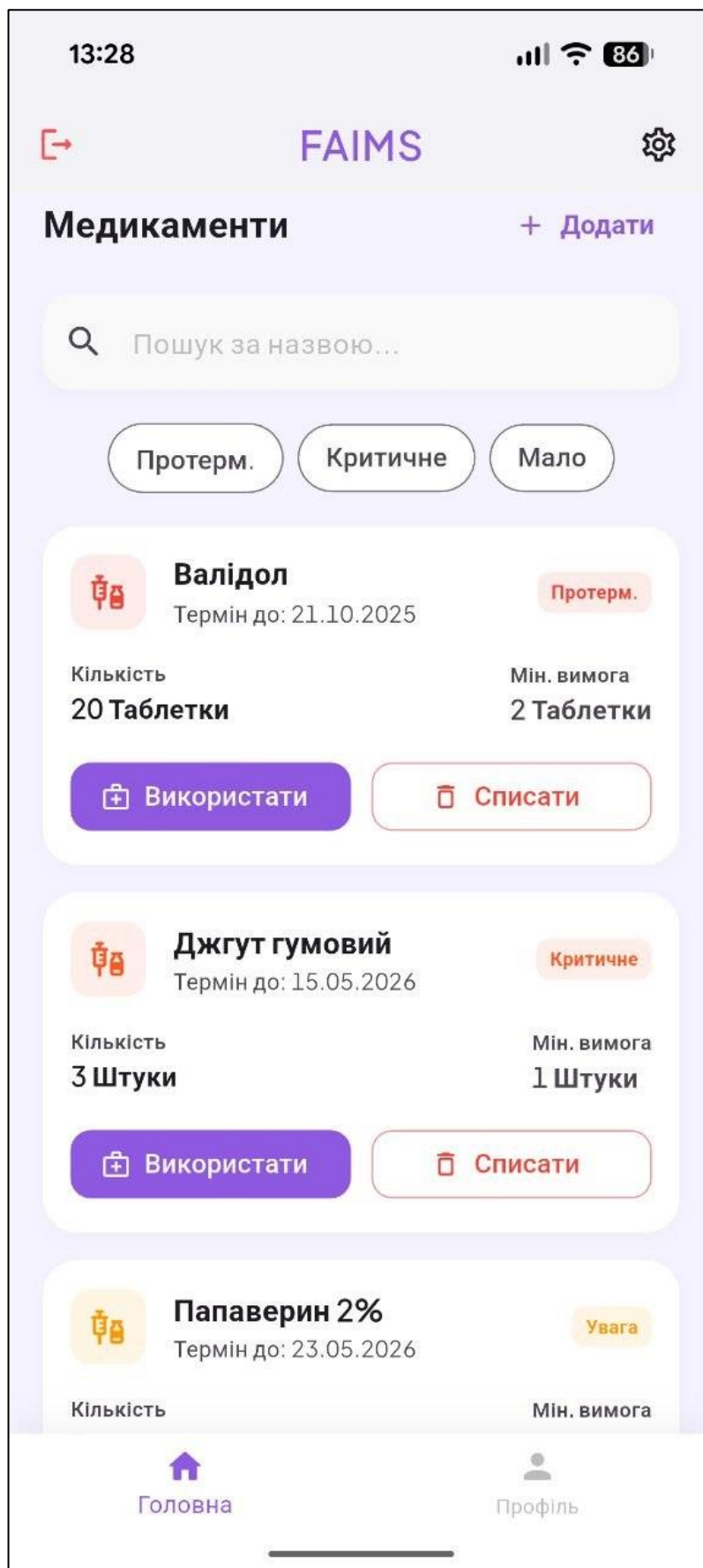


Рис. 4.2. Вміст аптечки з відображенням статусів медикаментів

4.2 Вимоги до апаратного та програмного забезпечення

Вимоги до обладнання і ПЗ – прямо з топології системи. Діаграма розгортання показує де що крутиться і як між собою пов'язано.

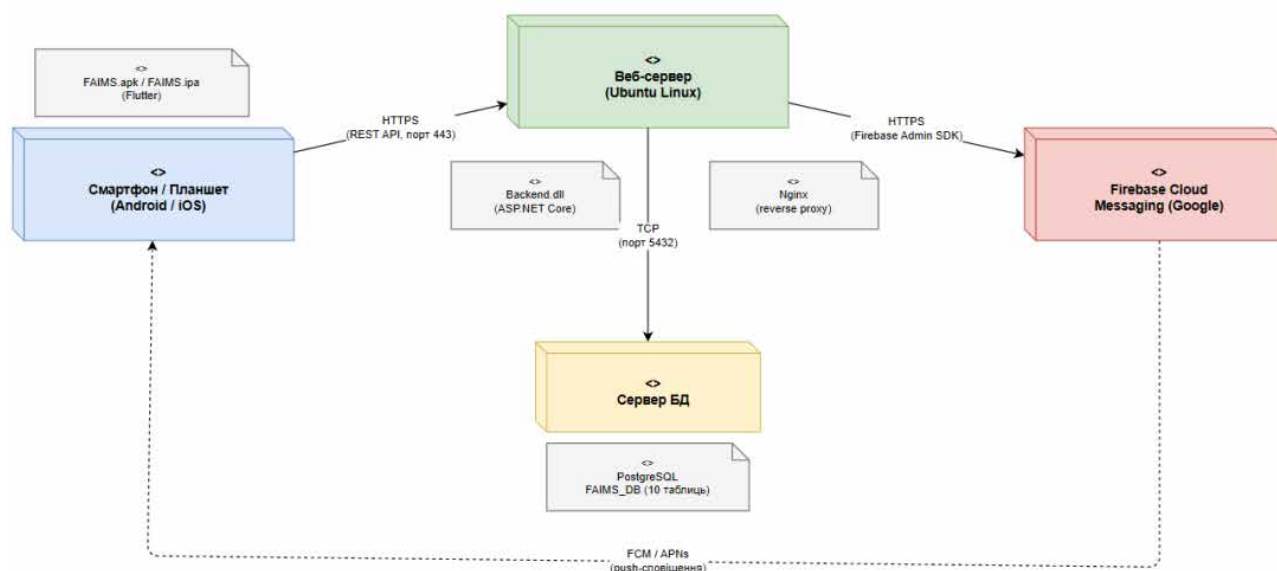


Рис. 4.6. Діаграма розгортання системи FAIMS

Система складається з чотирьох основних типів вузлів: мобільний пристрій кінцевого користувача, віртуальний сервер у хмарі, на якому розгорнуто серверний застосунок, контейнер бази даних, і зовнішній сервіс Firebase Cloud Messaging компанії Google.

4.2.1 Серверні вимоги

Серверна частина системи розгортається на віртуальному приватному сервері. У продуктивному середовищі використовується VPS DigitalOcean типу Basic Droplet з такими характеристиками: 1 ядро процесора, 1 ГБ оперативної пам'яті, 25 ГБ SSD-сховища, операційна система Ubuntu 24.04 LTS, регіон Frankfurt. Ці характеристики є мінімальними і покривають потреби середнього клієнта з кількома десятками організацій.

Вибір такої скромної конфігурації обґрунтований тим, що у файлі docker-compose.yml для кожного контейнера явно встановлено обмеження пам'яті: 256 МБ для PostgreSQL, 300 МБ для серверного застосунку, 64 МБ для Caddy і 64 МБ

для контейнера резервного копіювання. Загальне максимальне споживання – 684 МБ, що залишає запас для системних процесів. Для PostgreSQL додатково встановлено параметри `shared_buffers=64MB`, `work_mem=4MB`, `maintenance_work_mem=32MB`, `max_connections=50`, які адаптовані під обмежений обсяг доступної пам'яті.

Список програмного забезпечення, необхідного на сервері: Docker Engine версії 20.10 або новіше, Docker Compose версії 2.0 або новіше, відкриті порти 80 і 443 у фаєрволі для трафіку HTTP і HTTPS відповідно. Жодних інших залежностей не потрібно – все необхідне (PostgreSQL, .NET runtime, Caddy) поставляється у вигляді готових Docker-образів.

4.2.2 Вимоги до клієнтських пристроїв

Мобільний застосунок підтримує дві платформи: Android починаючи з версії 6.0 і iOS починаючи з версії 12.0. Цей вибір покриває близько 95% активних мобільних пристроїв станом на момент розробки. Особливих вимог до апаратних характеристик пристрою немає – застосунок працює навіть на бюджетних моделях. Об'єм встановленого APK становить близько 35 МБ, додаткові дані застосунку зростають у міру використання за рахунок локального кешу зображень і логів – у середньому до 50 МБ за кілька місяців активного використання.

Для роботи застосунку необхідне постійне інтернет-з'єднання, оскільки всі дані зберігаються на сервері і завантажуються «на льоту». Швидкості з'єднання у 1 Мбіт/с цілком достатньо. У разі тимчасової відсутності з'єднання застосунок продовжує працювати з останніми завантаженими даними, але операції зміни (поповнення, списання) тимчасово недоступні.

4.2.3 Зовнішні сервіси

Система використовує два зовнішні сервіси, надані сторонніми постачальниками. Перший – Firebase Cloud Messaging від Google. Він використовується виключно для доставки push-сповіщень з серверної частини на мобільні пристрої. Сервіс безкоштовний і не потребує платної підписки навіть

при значній кількості користувачів. Для його використання потрібен JSON-файл сервісного облікового запису, отриманий у консолі Firebase, який монтується у контейнер серверної частини як файл, доступний тільки для читання.

Другий – DuckDNS. Це безкоштовний сервіс динамічного DNS, який використовується для отримання публічного домену `faims.duckdns.org` для VPS. Він обраний тимчасово, на час розробки і первинного розгортання; при виведенні продукту на ринок його буде замінено власним доменом другого рівня.

4.3 Склад інсталяційного пакету

Інсталяційний пакет системи поділяється на дві частини: серверну, яка розгортається на VPS адміністратором клієнта або постачальника послуги, і клієнтську, яка встановлюється кінцевими користувачами на мобільні пристрої через офіційні магазини застосунків.

4.3.1 Серверний інсталяційний пакет

Серверний пакет постачається у вигляді git-репозиторію з відкритим вихідним кодом. Альтернативно – у вигляді tar-архіву, що містить ту саму структуру файлів. До складу пакету входять такі компоненти.

Файл `docker-compose.yml` – основний оркестраційний файл, який описує всі контейнери системи, їхні взаємозалежності, обмеження ресурсів і налаштування. Запуск всієї системи зводиться до однієї команди `docker compose up -d` у директорії з цим файлом.

Директорія `Backend` – вихідний код серверного застосунку разом з `Dockerfile`, який збирає Docker-образ. Збірка виконується автоматично при першому запуску команди `docker compose up`.

Файл `Caddyfile` – конфігурація reverse-проху. Містить опис маршрутизації трафіку з 443 порту на внутрішній порт серверного застосунку і налаштування автоматичного отримання TLS-сертифіката від Let's Encrypt.

Файл .env.example – шаблон файлу змінних оточення. Адміністратор копіює його у файл .env і заповнює конкретними значеннями: пароль для PostgreSQL, секретний ключ для підпису JWT-токенів, дозволені CORS-домени.

JSON-файл Firebase – сервісний обліковий запис Firebase, отриманий у консолі firebase.google.com. Цей файл є секретом і не повинен потрапляти у публічний git-репозиторій. У продуктивному розгортанні його кладуть на сервер вручну і монтують у контейнер як read-only volume.

Директорія scripts – допоміжні скрипти, серед яких run_flutter.sh для Linux і macOS та run_flutter.bat для Windows. Вони автоматично визначають IP-адресу хост-машини і запускають Flutter-застосунок з правильно налаштованою адресою API.

Файл README.md – документація з покроковими інструкціями зі встановлення, налаштування і запуску системи.

Структура файлу docker-compose.yml у спрощеному вигляді показана нижче. Він описує чотири сервіси, які запускаються в єдиній мережі Docker і взаємодіють між собою за внутрішніми іменами хостів.

```
services:
  postgres_db:
    image: postgres:15-alpine
    restart: always
    environment:
      POSTGRES_USER: ${POSTGRES_USER:-faims_admin}
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
      POSTGRES_DB: ${POSTGRES_DB:-faims_db}
    volumes:
      - pgdata:/var/lib/postgresql/data
    deploy:
      resources:
        limits:
          memory: 256M

  backend:
    build:
      context: .
      dockerfile: Backend/Dockerfile
    restart: always
    environment:
      ConnectionStrings__PostgreSqlConnection: "Host=postgres_db;..."
      JwtSettings__SecurityKey: "${JWT_SECRET_KEY}"
      ASPNETCORE_ENVIRONMENT: "Production"
    depends_on:
      postgres_db:
```

```

        condition: service_healthy
    deploy:
        resources:
            limits:
                memory: 300M

caddy:
    image: caddy:2-alpine
    restart: always
    ports:
        - "80:80"
        - "443:443"
    volumes:
        - ./Caddyfile:/etc/caddy/Caddyfile:ro
    depends_on:
        - backend

pg_backup:
    image: postgres:15-alpine
    entrypoint: >
    sh -c 'while true; do
        pg_dump | gzip > /backups/faims_$(date +%Y%m%d_%H%M%S).sql.gz;
        find /backups -mtime +3 -delete;
        sleep 21600;
    done'
```

Лістинг 4.1. Спрощена структура docker-compose.yml

Розгортання на чистому VPS зводиться до такої послідовності команд: встановити Docker за офіційною інструкцією, клонувати git-репозиторій у /opt/faims, скопіювати .env.example у .env і заповнити його, скопіювати JSON-файл Firebase у директорію Backend, виконати команду `docker compose up -d`. У середньому через дві-три хвилини після цього система готова приймати трафік на домені, прописаному у Caddyfile.

4.3.2 Клієнтський інсталяційний пакет

Клієнтський застосунок постачається у вигляді APK-файлу для Android і IPA-файлу для iOS, отриманих за результатом збірки командами `flutter build apk --release` та `flutter build ios --release` відповідно. У продуктивному розгортанні ці файли публікуються через офіційні магазини застосунків – Google Play та Apple App Store.

Перед збіркою виконуються дві допоміжні команди: `dart run build_runner build --delete-conflicting-outputs` для автогенерації коду серіалізації DTO і `flutter`

gen-110n для збірки файлів локалізації з ARB-джерел. Без цих команд проєкт не збереться через відсутність згенерованих файлів.

Для тестового розгортання, коли застосунок ще не опубліковано в магазинах, APK-файл можна встановити безпосередньо з пристрою. На Android для цього потрібно дозволити встановлення застосунків з невідомих джерел у налаштуваннях системи. На iOS для тестового розгортання використовується сервіс TestFlight, який є частиною екосистеми Apple і дозволяє розповсюджувати застосунок серед обмеженого кола тестувальників до публічного релізу в App Store.

4.3.3 Резервне копіювання і відновлення

У склад інсталяційного пакету також входить контейнер pg_backup, який запускається автоматично разом з рештою системи і виконує резервне копіювання бази даних кожні шість годин. Резервні копії складаються у директорію ./backups на хост-машині, де часова мітка відображає момент початку копіювання. Стиснення gzip дає типові співвідношення стиснення близько 10:1, що для бази обсягом у кілька десятків мегабайт означає файли копій у кілька мегабайт.

Старі копії автоматично видаляються через три доби, що обмежує загальний обсяг архіву на хост-машині. Період зберігання обраний як компроміс між надійністю – щоб мати запас на випадок виявлення проблеми не одразу – і обсягом дискового простору.

ВИСНОВКИ

У результаті виконання дипломної роботи розроблено програмне забезпечення для системи обліку аптечок першої допомоги в організаціях. Система отримала назву FAIMS – First Aid Inventory Management System – і реалізована як мультитенантне SaaS-рішення з кросплатформним мобільним клієнтом і серверною частиною.

Проведено системний аналіз предметної області, виявлено типові проблеми, які виникають при ручному веденні обліку аптечок: відсутність автоматизованого контролю термінів придатності, висока трудомісткість, неможливість оперативно отримувати агрегований звіт, відсутність централізованої видимості стану аптечок керівництвом. Огляд існуючих рішень показав, що ніша спеціалізованої простої у використанні системи обліку аптечок першої допомоги для масового сегменту малих і середніх організацій української мови залишається фактично незаповненою, що підтверджує актуальність розробки.

На етапі проєктування побудовано комплекс UML-моделей: діаграми прецедентів, послідовності, активності для опису предметної області, а також діаграми класів, кооперацій, пакетів і компонентів для опису самого програмного продукту. Спроектовано логічну модель бази даних з обґрунтуванням третьої нормальної форми і реалізацією мультитенантності на рівні моделі даних; на її основі засобами EF Core Code First реалізовано фізичну структуру бази даних.

На етапі розробки реалізовано серверну частину на платформі ASP.NET Core 9.0 з використанням Entity Framework Core і бази даних PostgreSQL 15. Архітектура серверної частини відповідає принципам Clean Architecture з чітким розподілом на шари домену, бізнес-логіки, інфраструктури та представлення. Безпека реалізована через JWT-автентифікацію з механізмом refresh-токенів, хешування паролів через стандартний механізм ASP.NET Identity, обмеження частоти запитів та глобальний фільтр мультитенантності, що автоматично ізолює

дані різних організацій-клієнтів. Push-сповіщення про наближення термінів придатності реалізовано через сервіс Firebase Cloud Messaging.

Мобільний клієнт реалізовано на платформі Flutter з єдиною кодовою базою для Android та iOS. Застосунок підтримує дві ролі користувачів – адміністратора організації та відповідальної особи – з відповідним розмежуванням доступу і функціональності. Реалізовано повний набір функцій: реєстрація організацій, управління підрозділами, кімнатами та аптечками, шаблони аптек, ведення складу медикаментів з контролем термінів придатності за принципом «світлофора», журналювання дій, формування PDF-звітів, аналітика з графіками.

Система розгорнута у продуктивному середовищі на віртуальному сервері DigitalOcean у регіоні Frankfurt з автоматичним отриманням TLS-сертифіката від Let's Encrypt і доступна за публічним доменом. Розгортання повністю автоматизоване через Docker Compose і складається з чотирьох контейнерів: PostgreSQL, серверний застосунок, reverse-proxy Caddy і сервіс резервного копіювання, який зберігає копії бази даних кожні шість годин. Мобільний застосунок перебуває на етапі закритого тестування у Google Play, що передуює публічному релізу.

Поставлена мета дипломної роботи досягнута повністю. Усі завдання, сформульовані у вступі, виконано. Розроблений продукт є робочою системою, придатною до реального використання, що підтверджується успішним розгортанням у продуктивному середовищі і проходженням етапу тестування у Google Play.

Перспективи подальшого розвитку системи включають реалізацію модуля закупівель з інтеграцією з постачальниками медикаментів, додавання вебверсії інтерфейсу для роботи з настільних комп'ютерів, розширення системи аналітики, інтеграцію з нормативною базою МОЗ для автоматичного оновлення системних шаблонів аптек, а також впровадження механізмів сканування QR-кодів медикаментів для прискорення інвентаризації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Закон України «Про охорону праці» від 14.10.1992 № 2694-ХІІ [Електронний ресурс]. – Режим доступу: <https://zakon.rada.gov.ua/laws/show/2694-12>
2. Наказ МОЗ України «Про затвердження Методичних рекомендацій ведення обліку лікарських засобів та медичних виробів у закладах охорони здоров'я» від 09.09.2014 № 635 [Електронний ресурс]. – Режим доступу: <https://zakon.rada.gov.ua/rada/show/v0635282-14>
3. Положення про бакалаврську кваліфікаційну роботу у Національному університеті біоресурсів і природокористування України : затверджено вченою радою НУБіП України 03.03.2021 р., протокол № 7 [Електронний ресурс]. – Режим доступу: <https://nubip.edu.ua>
4. ASP.NET Core documentation. Microsoft Learn [Електронний ресурс]. – Режим доступу: <https://learn.microsoft.com/en-us/aspnet/core/>
5. Entity Framework Core documentation. Microsoft Learn [Електронний ресурс]. – Режим доступу: <https://learn.microsoft.com/en-us/ef/core/>
6. Multi-tenant applications. Microsoft Azure Architecture Center [Електронний ресурс]. – Режим доступу: <https://learn.microsoft.com/en-us/azure/architecture/guide/multitenant/overview>
7. PostgreSQL 15 Documentation. The PostgreSQL Global Development Group [Електронний ресурс]. – Режим доступу: <https://www.postgresql.org/docs/15/>
8. Flutter documentation. Google [Електронний ресурс]. – Режим доступу: <https://docs.flutter.dev/>
9. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT) : RFC 7519. Internet Engineering Task Force, May 2015 [Електронний ресурс]. – Режим доступу: <https://datatracker.ietf.org/doc/html/rfc7519>

10. OWASP Top Ten Web Application Security Risks. Open Worldwide Application Security Project, 2021 [Електронний ресурс]. – Режим доступу: <https://owasp.org/www-project-top-ten/>
11. Docker documentation. Docker Inc. [Електронний ресурс]. – Режим доступу: <https://docs.docker.com/>
12. Caddy Web Server Documentation [Електронний ресурс]. – Режим доступу: <https://caddyserver.com/docs/>
13. Firebase Cloud Messaging Documentation. Google [Електронний ресурс]. – Режим доступу: <https://firebase.google.com/docs/cloud-messaging>
14. Волков Н. О., Василюк-Зайцева С. В. Програмне забезпечення автоматизованої системи контролю за запасами та терміном придатності аптечок першої допомоги. Теоретичні та прикладні аспекти розробки комп'ютерних систем : матеріали наук.-практ. конф. студентів і аспірантів, Київ, квітень 2026 р. [Електронний ресурс]. – Режим доступу: <http://econference.nubip.edu.ua/index.php/taacsd/2026/author/submission/4208>

Лістинг ключових програмних модулів

У цьому додатку наведено повний код ключових програмних модулів системи FAIMS, що демонструють реалізацію основних архітектурних рішень: базової сутності з підтримкою мультитенантності, глобального фільтра запитів Entity Framework Core, алгоритму обчислення статусу медикаменту за принципом «світлофора» та сервісу автентифікації через JWT.

А.1 Базова сутність BaseEntity

```
public abstract class BaseEntity : IMustHaveTenant
{
    public Guid Id { get; set; } = Guid.NewGuid();
    public Guid OrganizationId { get; set; }
    public DateTime CreatedAt { get; set; } = DateTime.UtcNow;
    public DateTime UpdatedAt { get; set; } = DateTime.UtcNow;
}

public interface IMustHaveTenant
{
    Guid OrganizationId { get; set; }
}
```

А.2 Глобальний фільтр мультитенантності

```
public class ApplicationDbContext : DbContext
{
    private readonly ITenantProvider _tenantProvider;

    public ApplicationDbContext(
        DbContextOptions<ApplicationDbContext> options,
        ITenantProvider tenantProvider) : base(options)
    {
        _tenantProvider = tenantProvider;
    }

    public DbSet<Organization> Organizations => Set<Organization>();
    public DbSet<Department> Departments => Set<Department>();
    public DbSet<Room> Rooms => Set<Room>();
    public DbSet<FirstAidKit> FirstAidKits => Set<FirstAidKit>();
    public DbSet<Medication> Medications => Set<Medication>();
}
```

```

    public DbSet<User> Users => Set<User>();
    public DbSet<Journal> JournalEntries => Set<Journal>();
    public DbSet<KitTemplate> KitTemplates => Set<KitTemplate>();
    public DbSet<KitTemplateItem> KitTemplateItems =>
Set<KitTemplateItem>();
    public DbSet<RefreshToken> RefreshTokens => Set<RefreshToken>();

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);

        foreach (var entityType in modelBuilder.Model.GetEntityTypes())
        {
            if
(typeof(IMustHaveTenant).IsAssignableFrom(entityType.ClrType))
            {
                var method = typeof(ApplicationDbContext)
                    .GetMethod(nameof(SetTenantQueryFilter),
                        BindingFlags.NonPublic |
BindingFlags.Instance)!
                    .MakeGenericMethod(entityType.ClrType);
                method.Invoke(this, new object[] { modelBuilder });
            }
        }
    }

    private void SetTenantQueryFilter<T>(ModelBuilder modelBuilder)
        where T : class, IMustHaveTenant
    {
        modelBuilder.Entity<T>()
            .HasQueryFilter(e => e.OrganizationId ==
_tenantProvider.OrganizationId);
    }
}

```

A.3 Алгоритм обчислення статусу медикаменту

```

public enum ExpirationStatus { Good, Warning, Critical, Expired }

public static class ExpirationCalculator
{
    public static ExpirationStatus Calculate(DateTime expirationDate)
        => Calculate(expirationDate, DateTime.UtcNow);

    public static ExpirationStatus Calculate(
        DateTime expirationDate, DateTime now)
    {
        var daysLeft = (expirationDate.Date - now.Date).Days;
        return daysLeft switch
        {
            < 0 => ExpirationStatus.Expired,
            < 7 => ExpirationStatus.Critical,

```

```

        < 30 => ExpirationStatus.Warning,
        _    => ExpirationStatus.Good,
    };
}
}

```

A.4 Сервіс автентифікації через JWT

```

public class AuthService : IAuthService
{
    private readonly ApplicationDbContext _db;
    private readonly IPasswordHasher<User> _passwordHasher;
    private readonly IJwtService _jwtService;

    public async Task<AuthResponseDto> LoginAsync(LoginDto dto)
    {
        var user = await _db.Users
            .IgnoreQueryFilters()
            .FirstOrDefaultAsync(u => u.Email == dto.Email);

        if (user is null)
            throw new UnauthorizedException("Invalid credentials");

        var verifyResult = _passwordHasher.VerifyHashedPassword(
            user, user.PasswordHash, dto.Password);

        if (verifyResult == PasswordVerificationResult.Failed)
            throw new UnauthorizedException("Invalid credentials");

        var accessToken = _jwtService.GenerateAccessToken(user);
        var refreshToken = _jwtService.GenerateRefreshToken(user);

        await _db.RefreshTokens.AddAsync(refreshToken);
        await _db.SaveChangesAsync();

        return new AuthResponseDto(accessToken, refreshToken.Token);
    }
}

public class JwtService : IJwtService
{
    public string GenerateAccessToken(User user)
    {
        var claims = new[]
        {
            new Claim(JwtRegisteredClaimNames.Sub, user.Id.ToString()),
            new Claim("oid", user.OrganizationId.ToString()),
            new Claim(ClaimTypes.Role, user.Role.ToString()),
            new Claim(JwtRegisteredClaimNames.Email, user.Email),
        };
    }
}

```

```

        var key = new SymmetricSecurityKey(
            Encoding.UTF8.GetBytes(_jwtOptions.Secret));
        var creds = new SigningCredentials(key,
            SecurityAlgorithms.HmacSha256);

        var token = new JwtSecurityToken(
            issuer: _jwtOptions.Issuer,
            audience: _jwtOptions.Audience,
            claims: claims,
            expires: DateTime.UtcNow.AddMinutes(60),
            signingCredentials: creds);

        return new JwtSecurityTokenHandler().WriteToken(token);
    }
}

```

A.5 Фоновий сервіс перевірки термінів придатності

```

public class ExpirationCheckHostedService : BackgroundService
{
    private readonly IServiceScopeFactory _scopeFactory;
    private readonly ILogger<ExpirationCheckHostedService> _logger;

    protected override async Task ExecuteAsync(CancellationToken ct)
    {
        var period = TimeSpan.FromHours(24);
        using var timer = new PeriodicTimer(period);

        do
        {
            try
            {
                using var scope = _scopeFactory.CreateScope();
                var checker = scope.ServiceProvider
                    .GetRequiredService<IExpirationChecker>();
                await checker.CheckAndNotifyAsync(ct);
            }
            catch (Exception ex)
            {
                _logger.LogError(ex,
                    "Error during expiration check cycle");
            }
        } while (await timer.WaitForNextTickAsync(ct));
    }
}

public class ExpirationChecker : IExpirationChecker
{
    public async Task CheckAndNotifyAsync(CancellationToken ct)
    {
        var medications = await _db.Medications
            .IgnoreQueryFilters()

```

```

        .Where(m => m.Quantity > 0)
        .Include(m => m.FirstAidKit)
            .ThenInclude(k => k.ResponsibleUser)
        .ToListAsync(ct);

    foreach (var med in medications)
    {
        var status = ExpirationCalculator.Calculate(med.ExpirationDate);
        if (status == ExpirationStatus.Good) continue;

        var user = med.FirstAidKit.ResponsibleUser;
        if (user?.FcmToken is null) continue;

        await _fcm.SendAsync(user.FcmToken, BuildMessage(med, status),
ct);
    }
}

```

A.6 Генерація JWT токена

```

public string GenerateAccessToken(User user)
{
    var claims = new[]
    {
        new Claim(JwtRegisteredClaimNames.Sub, user.Id.ToString()),
        new Claim("oid", user.OrganizationId.ToString()),
        new Claim(ClaimTypes.Role, user.Role.ToString()),
        new Claim(JwtRegisteredClaimNames.Email, user.Email),
    };

    var key = new SymmetricSecurityKey(
        Encoding.UTF8.GetBytes(_jwtOptions.Secret));
    var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

    var token = new JwtSecurityToken(
        issuer: _jwtOptions.Issuer,
        audience: _jwtOptions.Audience,
        claims: claims,
        expires: DateTime.UtcNow.AddMinutes(15),
        signingCredentials: creds);

    return new JwtSecurityTokenHandler().WriteToken(token);
}

```

Скриншоти інтерфейсу системи

У цьому додатку наведено скриншоти ключових екранів мобільного застосунку FAIMS, що ілюструють інтерфейс адміністратора та відповідальної особи у різних сценаріях використання системи.

Б.1 Інтерфейс адміністратора

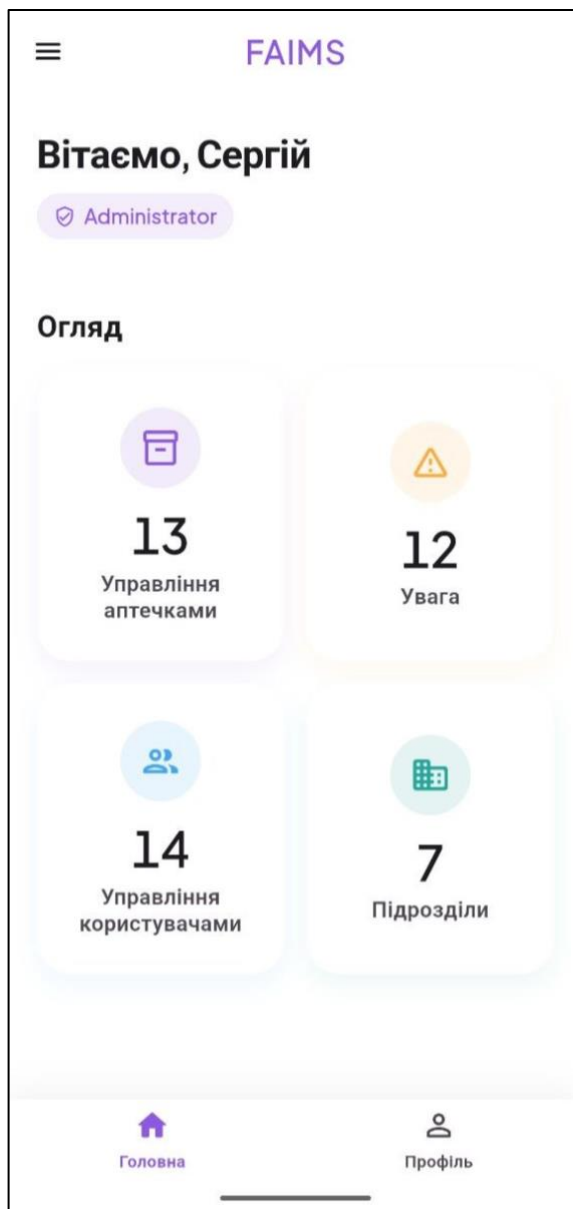


Рис. Б.1. Головний екран адміністратора

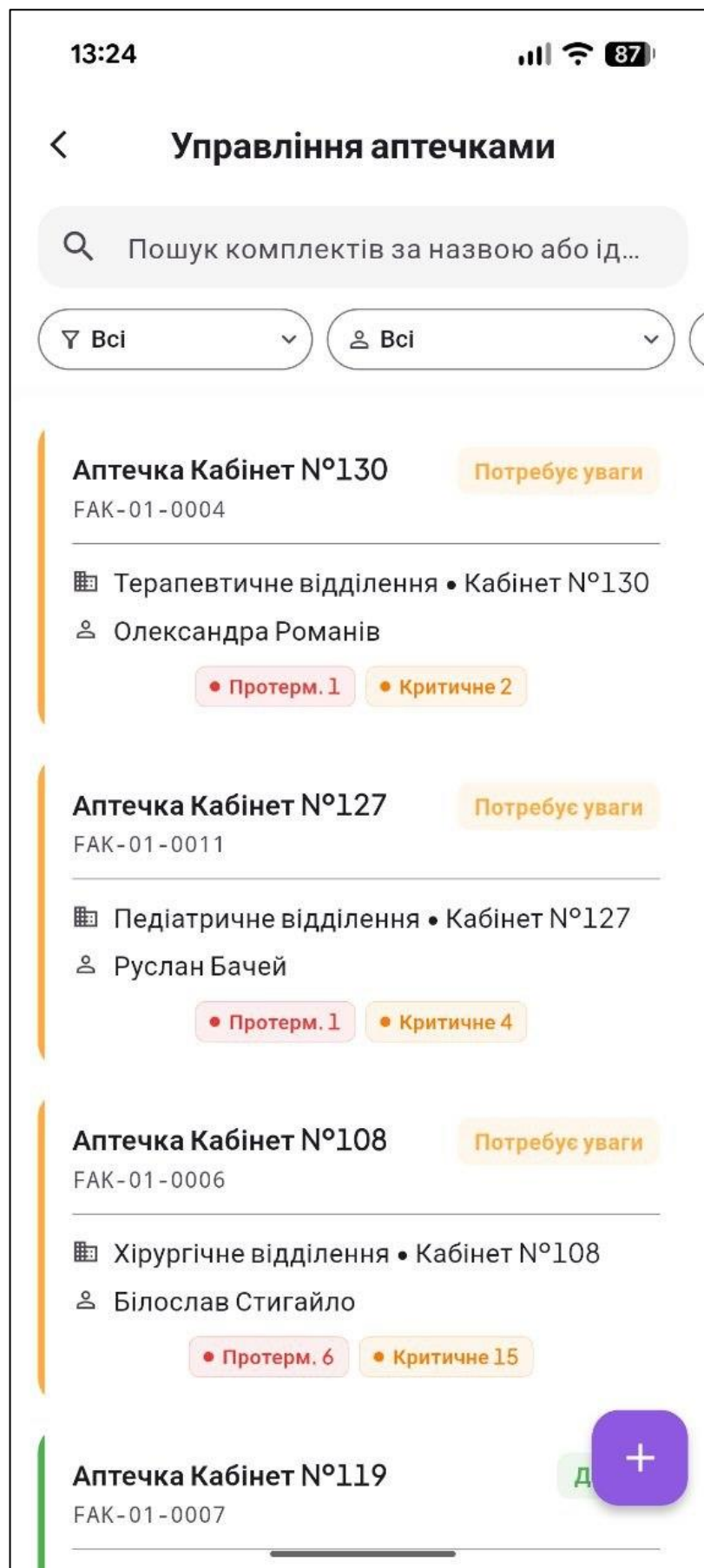


Рис. Б.2. Управління аптечками з кольоровими індикаторами стану

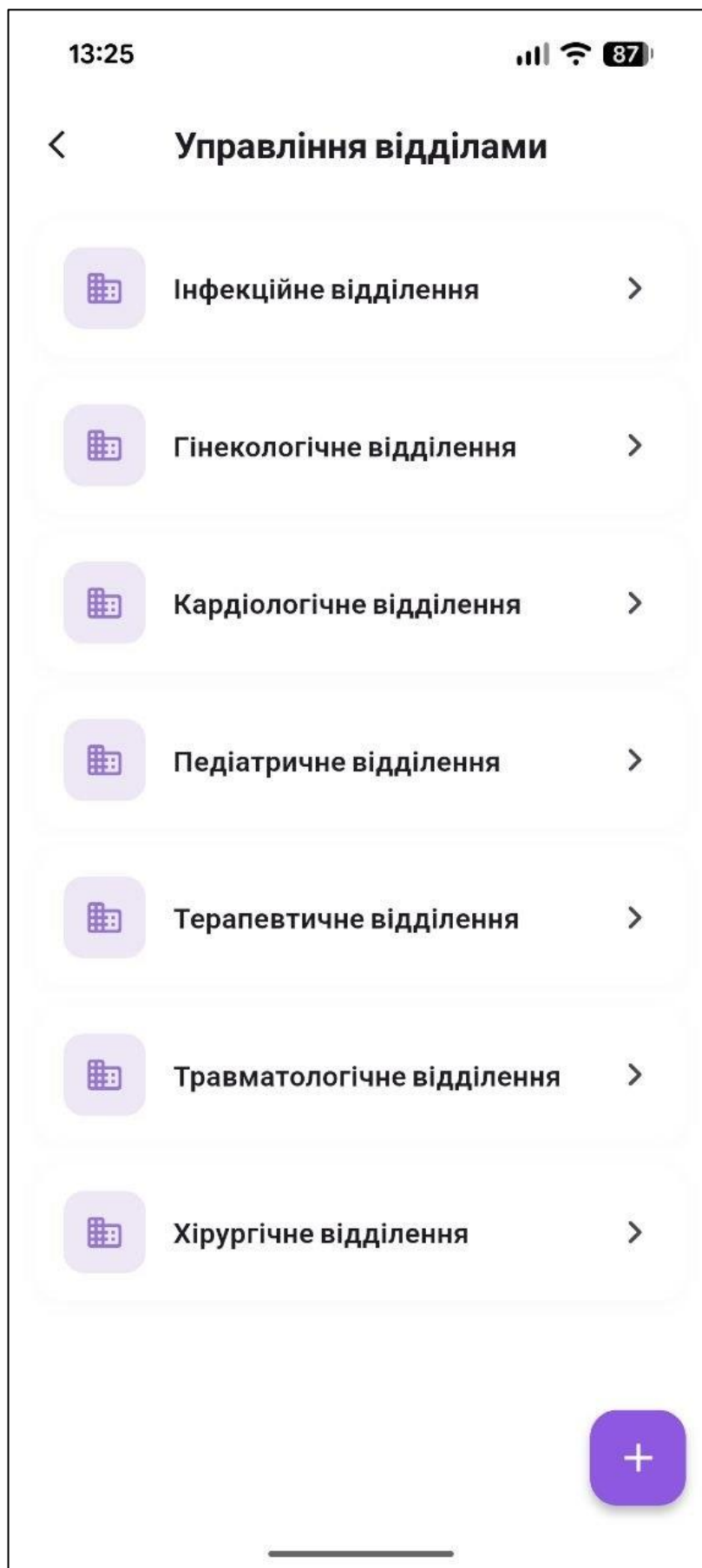


Рис. Б.3. Управління підрозділами та кімнатами

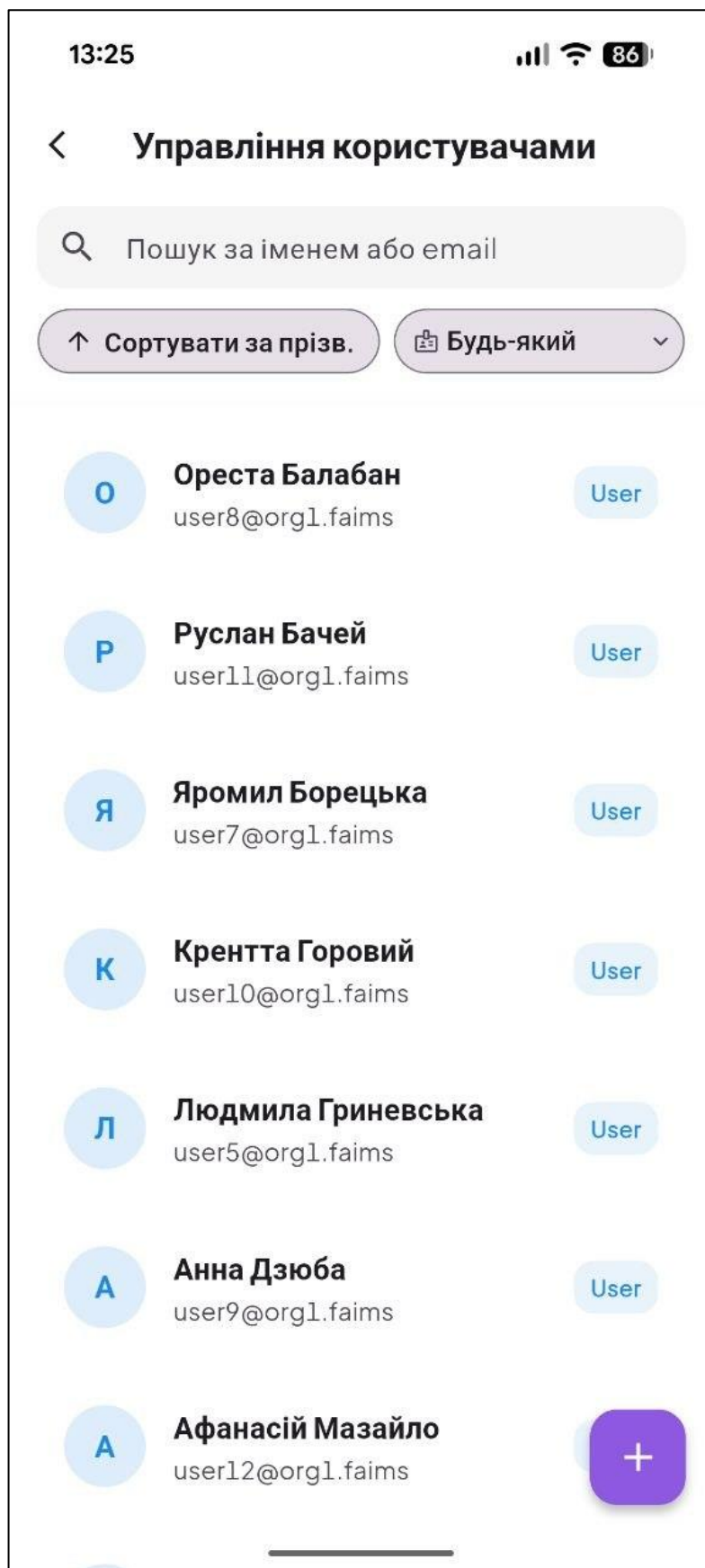


Рис. Б.4. Управління користувачами організації



Рис. Б.5. Аналітика - графіки витрат і поповнень

F FAIMS Звіт системи FAIMS На утилізацію			
ПЕРІОД ЗВІТУ 08.04.2026 – 08.05.2026		ВСЬОГО ЗАПИСІВ 2	
Назва	Кількість	Одиниці	Причина
Бинт стерильний 5м×10см	+2	Штуки	Протерміновано в аптечках
Джгут для зупинки кровотечі	+1	Штуки	Протерміновано в аптечках
Сформовано: 08.05.2026 10:59			

Рис. Б.6. Формування PDF-звіту

Б.2 Інтерфейс відповідальної особи

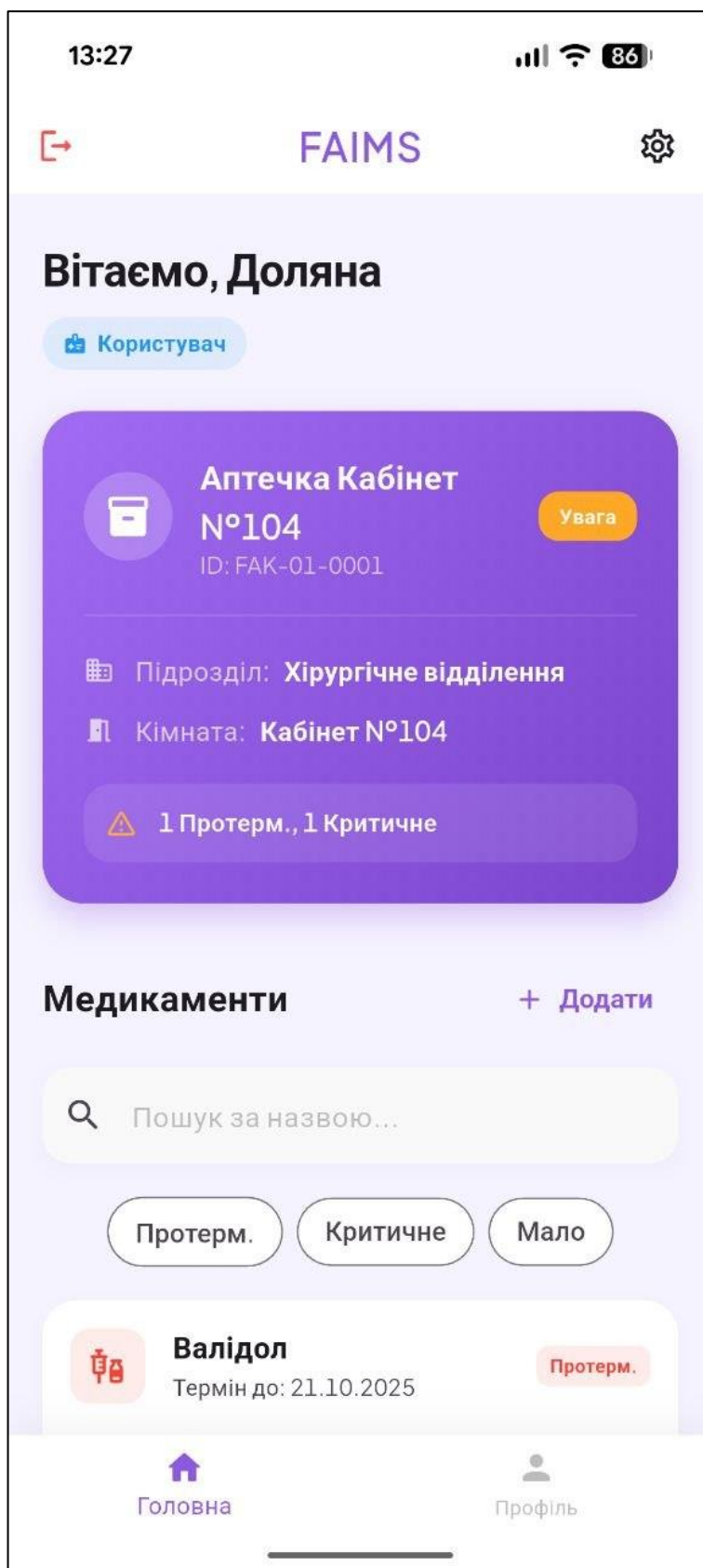


Рис. Б.7. Перелік закріплених аптечок

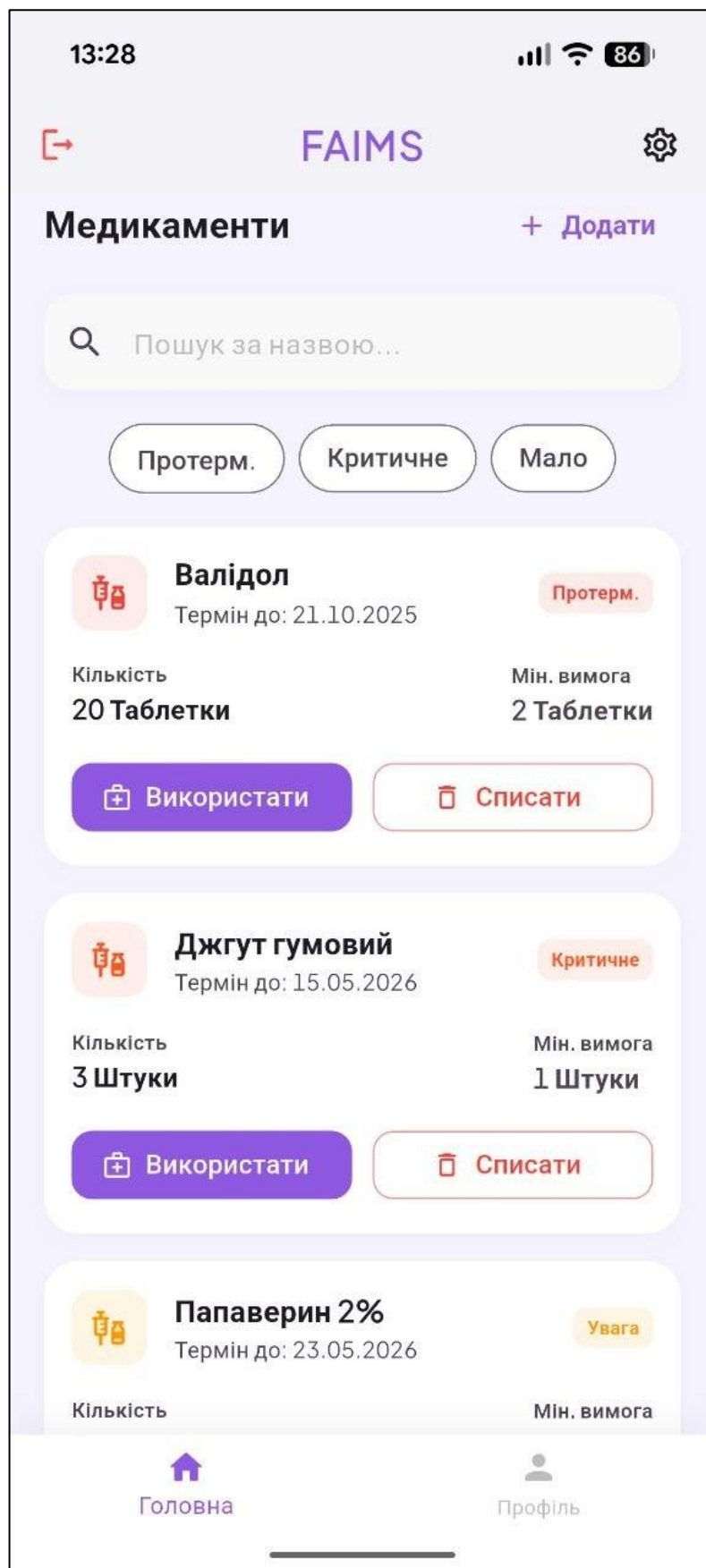


Рис. Б.8. Вміст аптечки з відображенням статусів медикаментів

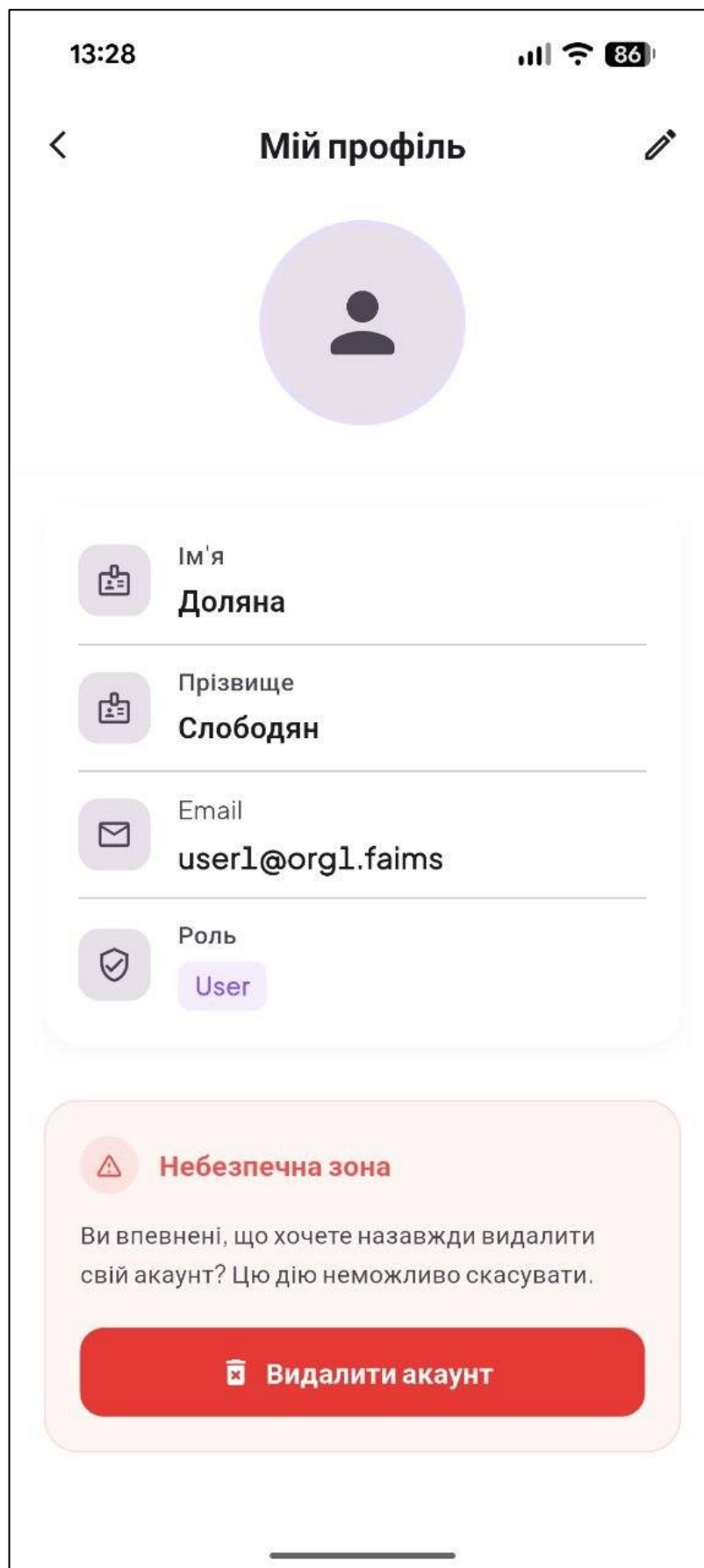


Рис. Б.9. Профіль користувача

ДОДАТОК В**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ****Факультет інформаційних технологій****Декларація про академічну доброчесність та використання ШІ****ДЕКЛАРАЦІЯ ЗДОБУВАЧА**

Я, Волков Нікіта Олександрович, здобувач НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота на тему «Програмне забезпечення для системи обліку аптечок першої допомоги» є результатом моєї особистої праці.
2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.
3. Щодо використання інструментів ШІ зазначаю, що:
 - ☐ інструменти ШІ Claude були використані для:
 - ☐ стилістичного редагування та перевірки граматики;
 - ☐ допомоги у написанні/відлагодженні програмного коду.

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

« ____ » _____ 2026 р. _____
(підпис)

Нікіта ВОЛКОВ