ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка програмного забезпечення для обліку домашньої аптечки»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Андрій МУЗИЧЕНКО
(підпис)

Виконав: здобувач(ка) вищої освіти групи ПД-42

Андрій МУЗИЧЕНКО

Керівник: Юрій ЗАДОНЦЕВ
доцент кафедри, кандидат технічних наук

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Музиченку Андрію Віталійовичу

1. Тема кваліфікаційної роботи: «Розробка програмного забезпечення для обліку домашньої аптечки»
керівник кваліфікаційної роботи доцент кафедри, кандидат технічних наук Юрій ЗАДОНЦЕВ,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56.

2. Строк подання кваліфікаційної роботи «02» червня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: Інформація про особливості організації обліку лікарських засобів у побуті, вимоги до користувацьких мобільних застосунків, технічна документація щодо засобів розробки на основі React Native, Expo, Node.js та методів локального зберігання даних.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз предметної області: проблеми обліку медикаментів у домашніх умовах.
2. Аналіз програмного забезпечення з подібним функціоналом.
3. Проєктування архітектури мобільного застосунку.
4. Реалізація функціоналу: додавання, редагування, пошук та нагадування.
5. Опис інтерфейсу користувача.
6. Оцінка функціональності та стабільності роботи мобільного застосунку.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Схема функціонування застосунку.
6. Діаграма класів або логічна структура компонентів.
7. Екранні форми.
8. Апробація розробленого рішення.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	25.02.-04.03.2025	
2	Аналіз та дослідження існуючих аналогів	05.03-13.03.2025	
3	Визначення вимог до мобільного застосунку для обліку домашньої аптеки	14.03-16.03.2025	
4	Проєктування архітектури та інтерфейсу мобільного застосунку	17.03-24.03.2025	
5	Реалізація основних функцій: додавання, редагування, пошук, нагадування	25.03-26.04.2025	
6	Оцінка функціональності та стабільності роботи мобільного застосунку	27.04-29.04.2025	
7	Оформлення роботи: вступ, висновки, реферат	30.04-11.05.2025	
8	Розробка демонстраційних матеріалів	12.05-18.05.2025	
9	Попередній захист роботи	19.05-30.05.2025	

Здобувач вищої освіти _____ Андрій МУЗИЧЕНКО

(підпис)

Керівник
кваліфікаційної роботи _____ Юрій ЗАДОНЦЕВ

(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 50 стор., 5 табл., 6 рис., 20 джерел.

Мета роботи: створення мобільного застосунку для організації обліку домашніх медикаментів з можливістю нагадувань, зручного керування даними та підтримки декількох аптечок.

Об'єкт дослідження: процес ведення обліку лікарських засобів у побуті.

Предмет дослідження: мобільний застосунок для обліку домашньої аптеки, реалізований за допомогою React Native, JavaScript, TypeScript та інструментів локального зберігання.

Короткий зміст роботи: У роботі проаналізовано особливості ведення обліку медикаментів у побуті, а також основні труднощі, пов'язані з відсутністю належного контролю за термінами придатності. Розглянуто існуючі інструменти для обліку ліків: MyTherapy, MedList, Medisafe.

Розроблено структуру роботи мобільного застосунку та реалізовано основні функціональні можливості, зокрема: додавання інформації про лікарські засоби, редагування даних, фільтрацію, ведення кількох аптечок, нагадування про терміни придатності.

У розробці використано React Native для створення кросплатформеного застосунку, Expo для управління ресурсами, TypeScript та JavaScript для логіки додатку, AsyncStorage для зберігання даних на пристрої.

Сферою використання застосунку є організація обліку ліків у домашніх умовах з метою своєчасного оновлення аптечки та уникнення використання прострочених препаратів.

КЛЮЧОВІ СЛОВА: АПТЕЧКА, МОБІЛЬНИЙ ЗАСТОСУНОК, REACT NATIVE, ЛІКИ, ОБЛІК, ТЕРМІН ПРИДАТНОСТІ, НАГАДУВАННЯ

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ – Програмне забезпечення

IDE – Integrated Development Environment (Інтегроване середовище розробки)

API – Application Programming Interface (Інтерфейс прикладного програмування)

JS – JavaScript

TS – TypeScript

RN – React Native

CRUD – Create, Read, Update, Delete (Створення, Читання, Оновлення, Видалення)

JSON – JavaScript Object Notation (Формат обміну даними)

AsyncStorage – Механізм локального зберігання даних у React Native

CLI – Command Line Interface (Інтерфейс командного рядка)

VS Code – Visual Studio Code (Редактор коду)

i18n – Internationalization (Інтернаціоналізація)

Jest – JavaScript Testing Framework (Фреймворк для тестування)

Zustand – Бібліотека для управління станом у React

Expo – Платформа для розробки React Native додатків

ESLint – Інструмент для аналізу коду

Prettier – Інструмент для автоматичного форматування коду

Share API – Інтерфейс для обміну текстовими даними з додатка

ThemeContext – Контекст для керування темами оформлення

AsyncStorage Persist – Middleware для збереження стану у локальному сховищі

ВСТУП

Актуальність: Управління домашньою аптечкою є невід'ємною частиною повсякденного життя, оскільки правильний облік медикаментів допомагає уникнути використання прострочених препаратів, запобігає дублюванню покупок і забезпечує наявність необхідних ліків у критичний момент. Зі збільшенням кількості домогосподарств, особливо серед літніх людей і сімей з дітьми, виникає гостра потреба в автоматизованих рішеннях, які б спрощували цей процес. Мобільні додатки стають зручним інструментом для організації обліку медикаментів, дозволяючи користувачам контролювати запаси, отримувати нагадування про терміни придатності та експортувати дані у текстовому форматі. Проте багато існуючих рішень мають суттєві недоліки: складний інтерфейс, відсутність української локалізації або залежність від постійного підключення до інтернету. Розробка такого додатку вирішує проблему доступності та зручності для україномовних користувачів, особливо в умовах обмеженого доступу до інтернету.

Об'єкт дослідження: процес обліку медикаментів у домашній аптечці. Цей процес охоплює відстеження назв препаратів, їхньої кількості, термінів придатності, категорій (наприклад, таблетки, капсули, сиропи), а також управління персональними нотатками та налаштуваннями сповіщень для своєчасного інформування користувача про необхідність поповнення запасів або утилізації прострочених ліків.

Предмет дослідження: Предметом дослідження є програмне забезпечення для управління домашньою аптечкою, яке реалізовано у вигляді мобільного додатку *My Home Pharmacy App*. Цей додаток розроблено з використанням сучасних технологій для забезпечення зручного та ефективного контролю над медикаментами в умовах офлайн-доступу. Додаток дозволяє користувачам

додавати нові ліки, переглядати детальну інформацію, редагувати та видаляти записи, а також налаштовувати сповіщення про терміни придатності.

Мета роботи: скорочення часу та зусиль, необхідних для управління домашньою аптечкою, шляхом розробки мобільного додатку *My Home Pharmacy App* з використанням React Native, Zustand, AsyncStorage, Expo Notifications, Expo Router, i18n-js та Jest. Додаток має забезпечити користувачам зручний інструмент для автоматизації обліку медикаментів із можливістю роботи без підключення до інтернету, а також підтримкою україномовного інтерфейсу для широкої аудиторії.

Методи дослідження: На першому етапі розробки було проведено аналіз предметної області та існуючих аналогів, таких як MyTherapy, Medisafe, Pill Reminder і My Home Pharmacy App, для визначення функціональних і нефункціональних вимог до додатку. Для аналізу предметної області використано методи порівняння та узагальнення, що дозволили визначити ключові проблеми управління аптечкою. Аналіз показав, що багато аналогів не відповідають потребам україномовних користувачів і вимагають постійного підключення до інтернету, що обмежує їх використання в умовах відсутності мережі. Далі було здійснено огляд технологічного стеку, який включає React Native для створення клієнтської частини, Zustand для управління станом, AsyncStorage для локального зберігання даних, Expo Notifications для реалізації сповіщень, Expo Router для файлової системи навігації, i18n-js для локалізації та Jest для модульного тестування. Розробка велася з використанням Visual Studio Code як основного редактора коду та Expo CLI для створення, налаштування та тестування додатку. Для тестування на Android-пристроях використовувалася Android Studio, яка забезпечувала емуляцію та дебагінг функціоналу, зокрема сповіщень. Дизайн інтерфейсу створювався напряму в коді через стилізацію в src/utils/theme.ts, що дозволило швидко реалізувати простий і зручний інтерфейс із підтримкою світлої та темної тем, великих шрифтів і контрастних кольорів для літніх користувачів. Також проведено тестування на фокус-групі з 2 літніх

користувачів для оцінки зручності. Для форматування коду використовувався Prettier як розширення у VS Code. Реалізація додатку супроводжувалася модульним тестуванням за допомогою Jest, що дозволило перевірити коректність роботи компонентів і логіки управління станом.

Наукова новизна: полягає в адаптації технологічного стеку (React Native, Zustand, AsyncStorage) для створення кросплатформного додатку з офлайн-доступом і україномовним інтерфейсом, що відповідає потребам цільової аудиторії. Додаток підтримує необмежену кількість користувацьких категорій, що дозволяє адаптувати його під індивідуальні потреби. Інтеграція Zustand для управління станом у React Native забезпечує високу продуктивність і масштабованість додатку, що вирізняє його серед існуючих рішень, таких як MyTherapy чи Medisafe.

Практична значущість: значущість додатку *My Home Pharmacy App* полягає в тому, що він може використовуватися домогосподарствами для ефективного управління медикаментами, зменшення ризиків використання прострочених ліків та оптимізації процесу закупівлі необхідних препаратів. Завдяки простому інтерфейсу, стилізованому через src/utils/theme.ts, та офлайн-доступу, додаток є зручним для широкої аудиторії, зокрема літніх людей і сімей з дітьми, які потребують надійного інструменту для щоденного використання без залежності від інтернет-з’єднання. Функціонал додатку дозволяє додавати нові ліки, редагувати їх, видаляти, переглядати детальну інформацію, налаштовувати сповіщення про терміни придатності (наприклад, за 7 днів до закінчення) та експортувати дані у текстовому форматі (наприклад, «Назва: Парацетамол, Термін: 31.12.2025»). Це робить додаток практичним рішенням для організації домашньої аптечки.

1 АНАЛІЗ ЗАСТОСУНКУ ДЛЯ ОБЛІКУ ДОМАШНЬОЇ АПТЕЧКИ

Розробка мобільного додатку для обліку домашньої аптечки є відповіддю на зростаючу потребу в автоматизації управління медикаментами в побуті. Цей розділ присвячено аналізу предметної області, специфіки управління аптечкою, цільової аудиторії, існуючих аналогів і вимог до програмного забезпечення (ПЗ).

1.1 Мобільний додаток для обліку медикаментів

Мобільний додаток для обліку домашньої аптечки – це спеціалізоване ПЗ, яке дозволяє користувачам систематизувати, відстежувати та управляти даними про медикаменти. Основна мета – спростити облік ліків, зменшити ризик використання прострочених препаратів і оптимізувати закупівлю медикаментів. Додаток працює на смартфонах, що робить його доступним для широкої аудиторії.

1.1.1 Функціональні можливості

- **Облік медикаментів**: Зберігання інформації про назву, кількість, категорію та термін придатності ліків.
- **Нагадування**: Автоматичні сповіщення про закінчення терміну придатності або необхідність поповнення запасів.
- **Пошук і фільтрація**: Швидкий доступ до медикаментів за назвою чи категорією.
- **Експорт даних**: Генерація звітів у текстовому форматі для передачі лікарям або особистих потреб.
- **Офлайн-доступ**: Робота без підключення до інтернету.

- **Локалізація**: Підтримка української мови для зручності користувачів.

1.1.2 Переваги мобільних додатків

- **Доступність**: Смартфони є у більшості користувачів, що забезпечує широке охоплення.
- **Мобільність**: Доступ до даних у будь-який час і місці.
- **Автоматизація**: Зменшення ручної роботи завдяки нагадуванням і фільтрації.
- **Персоналізація**: Налаштування профілів для кількох користувачів у домогосподарстві.

1.1.3 Недоліки

- **Самодисципліна**: Користувачі повинні регулярно оновлювати дані.
- **Безпека даних**: Необхідність захисту конфіденційної інформації.
- **Ручний введення:** Усі дані про медикаменти вводяться вручну, що може бути трудомістким.

1.1.4 Сфери застосування

Додатки для обліку аптечки використовуються в домогосподарствах, медичних закладах (для невеликих аптек), а також окремими особами з хронічними захворюваннями. Вони допомагають уникнути помилок, таких як використання прострочених ліків, і оптимізувати витрати на медикаменти.

1.2 Специфіка управління домашньою аптечкою

Управління домашньою аптечкою має унікальні особливості, які впливають на вимоги до ПЗ:

- **Різноманітність медикаментів**: Аптечка може містити знеболювальні, антибіотики, вітаміни, БАДи, що потребує гнучкої системи категоризації.
- **Терміни придатності**: Ліки мають обмежений строк зберігання (від кількох місяців до кількох років), що вимагає автоматичного відстеження.
- **Помилки користувачів**: Неправильне дозування або використання прострочених ліків може мати серйозні наслідки для здоров'я.
- **Різні аудиторії**: Літні люди потребують простого інтерфейсу, тоді як сім'ї з дітьми – розширених функцій, таких як профілі для кожного члена сім'ї.
- **Конфіденційність**: Дані про медикаменти є чутливими, що вимагає захисту від несанкціонованого доступу.
- **Офлайн-доступ**: Багато користувачів, особливо в сільській місцевості, можуть не мати стабільного інтернет-з'єднання.

1.2.1 Категоризація медикаментів

Медикаменти можна класифікувати за:

- **Типом**: Таблетки, сиропи, мазі, ін'єкції.
- **Призначенням**: Знеболювальні, протизапальні, антигістамінні, вітаміни.
- **Терміном придатності**: Короткострокові (до 6 місяців), середньострокові (6–12 місяців), довгострокові (>12 місяців).

1.2.2 Проблеми управління

- **Ручний облік**: Запис у блокноті чи таблиці є трудомістким і схильним до помилок.
- **Прострочені ліки**: Без нагадувань користувачі часто забувають перевіряти терміни придатності.
- **Надлишок або нестача**: Неправильне планування закупівель призводить до накопичення або дефіциту ліків.

1.3 Цільова аудиторія

Додаток орієнтований на такі групи користувачів:

- **Літні люди (60+ років)**: Потребують простого інтерфейсу, великих шрифтів, нагадувань про прийом ліків і терміни придатності. Часто мають хронічні захворювання, що вимагає регулярного обліку.
- **Сім'ї з дітьми**: Використовують додаток для управління ліками для всіх членів сім'ї. Потребують функцій профілів, фільтрації та категоризації.
- **Особи з хронічними захворюваннями**: Відстежують запаси ліків, дозування та терміни придатності. Потребують експорту даних для лікарів.
- **Ентузіасти здорового способу життя**: Обліковують вітаміни, БАДи та спортивні добавки. Цінують швидкий доступ.

1.3.1 Демографічні характеристики

- **Вік**: 18–80 років.
- **Стать**: Без обмежень.
- **Регіон**: Україна, з акцентом на україномовних користувачів.
- **Технічна грамотність**: Від початкового до середнього рівня (користувачі повинні вміти встановлювати додатки та вводити дані).

1.3.2 Поведінкові характеристики

- Регулярно купують медикаменти (щомісяця або щокварталу).
- Цінують зручність і швидкість доступу до інформації.
- Потребують захисту даних і офлайн-доступу.

1.4 Дослідження існуючих аналогів

Для формування вимог до додатку було проаналізовано чотири популярні додатки: MyTherapy, Medisafe, Pill Reminder і My Home Pharmacy App. Кожен

додаток оцінено за функціональністю, інтерфейсом, локалізацією та доступністю.

1.4.1 MyTherapy

Опис: Мобільний додаток для управління прийомом ліків і нагадувань.
Функції:

- Нагадування про прийом ліків за розкладом.
- Відстеження запасів медикаментів.
- Інтеграція з календарем.
- Генерація звітів про прийом.

Переваги:

- Простий і зрозумілий інтерфейс.
- Безкоштовна базова версія.
- Підтримка кількох мов (англійська, німецька, іспанська).

Недоліки:

- Відсутність української локалізації.
- Реклама у безкоштовній версії.
- Немає повноцінного офлайн-доступу.

Цільова аудиторія: Особи з хронічними захворюваннями, сім’ї.

1.4.2 Medisafe

Опис: Додаток із акцентом на персоналізацію та інтеграцію з медичними пристроями.
Функції:

- Персоналізовані нагадування.
- Синхронізація з розумними годинниками та трекерами.
- Пошук ліків за штрих-кодом.
- Детальна статистика прийому.

Переваги:

- Інтеграція з IoT-пристроями.
- Розширена аналітика.

Недоліки:

- Платна підписка для доступу до всіх функцій.
- Складний інтерфейс для літніх людей.
- Відсутність української локалізації.

Цільова аудиторія: Технічно підковані користувачі, особи з хронічними захворюваннями.

1.4.3 Pill Reminder

Опис: Простий додаток для нагадувань про прийом ліків.

Функції:

- Нагадування за розкладом.
- Облік запасів ліків.
- Експорт звітів у PDF.

Переваги:

- Присутність української локалізації.
- Мінімалістичний дизайн.
- Повністю безкоштовний.

Недоліки:

- Обмежений функціонал (немає пошуку чи фільтрації).
- Слабка підтримка офлайн-режиму.

Цільова аудиторія: Літні люди, користувачі з базовими потребами.

1.4.4 My Home Pharmacy App

Опис: Додаток для обліку аптечки з офлайн-доступом.

Функції:

- Локальне зберігання даних.
- Категоризація медикаментів.

- Нагадування про терміни придатності.

Переваги:

- Присутність української локалізації.
- Повноцінний офлайн-доступ.
- Простий інтерфейс.
- Є експорту даних.

Недоліки:

- Обмежена функціональність фільтрації.

Цільова аудиторія: Сім’ї, літні люди.

1.4.5 Порівняння аналогів

Таблиця 1.1

Порівняння аналогів

Показник	**MyTherapy**	**Medisafe**	**Pill Reminder**	**My Home Pharmacy App**
Офлайн-доступ	Частковий	Ні	Часткови й	Так
Українська локалізація	Ні	Ні	Так	Так
Пошук і фільтрація	Так	Так	Ні	Так
Нагадування	Так	Так	Так	Так
Експорт даних	Так	Так	Так	Так
Інтерфейс	Складний	Складний	Простий	Простий
Безкоштовність	Частково	Ні	Так	Так
Інтеграція з IoT	Ні	Так	Ні	Ні

Аналіз таблиці: Жоден із аналогів не підтримує українську локалізацію, що є значним недоліком для української аудиторії. MyTherapy і Medisafe пропонують розширений функціонал, але мають складний інтерфейс або платні функції. Pill Reminder і My Home Pharmacy App прості, але обмежені в можливостях. Розроблений додаток має поєднувати офлайн-доступ, україномовний інтерфейс і гнучку фільтрацію.

1.5 Визначення вимог до програмного забезпечення

На основі аналізу предметної області, специфіки управління аптечкою, цільової аудиторії та аналогів сформовано функціональні та нефункціональні вимоги до ПЗ.

1.5.1 Функціональні вимоги

- **Авторизація**: Реєстрація та вхід користувачів за локальним профілем.
- **Облік медикаментів**: Додавання, редагування, видалення даних про ліки (назва, кількість, категорія, термін придатності, дозування).
- **Пошук і фільтрація**: Пошук за назвою, фільтрація за категорією, терміном придатності чи кількістю.
- **Нагадування**: Push-сповіщення про закінчення терміну придатності або потребу поповнення запасів.
- **Експорт даних**: Генерація текстових звітів у форматі TXT або CSV.
- **Профілі користувачів**: Підтримка кількох профілів для членів сім'ї.
- **Категоризація**: Можливість створювати власні категорії медикаментів.

1.5.2 Нефункціональні вимоги

- **Офлайн-доступ**: Робота без інтернету через локальне зберігання (AsyncStorage).
- **Продуктивність**: Час завантаження інтерфейсу < 2 секунд, обробка пошуку < 1 секунди.
- **Інтерфейс**: Адаптивний, україномовний, з великими шрифтами для літніх людей.
- **Безпека**: Захист даних через локальне зберігання та шифрування.

- **Кросплатформність**: Підтримка iOS (версії 13+) і Android (версії 9+).
- **Локалізація**: Повна підтримка української мови, з можливістю додавання інших мов через i18n-js.

1.5.3 Додаткові вимоги

- **Тестування**: Модульне тестування за допомогою Jest, покриття коду > 80%.
- **Сумісність**: Коректна робота на пристроях із різними розмірами екранів (від 4 до 12 дюймів).

Рис. 1.1 – Діаграма вимог

Висновок до Розділу 1: Аналіз застосунку для обліку домашньої аптечки

У результаті аналізу предметної області, специфіки управління домашньою аптечкою, цільової аудиторії та існуючих аналогів (MyTherapy, Medisafe, Pill Reminder) визначено ключові проблеми та вимоги до програмного забезпечення. Виявлено, що більшість аналогів не забезпечують повноцінної української локалізації та офлайн-доступу, що є критичним для української аудиторії, особливо в умовах обмеженого інтернету. Сформовано функціональні вимоги (облік медикаментів, нагадування, експорт даних) та нефункціональні вимоги (продуктивність, адаптивність інтерфейсу), що лягли в основу розробки My Home Pharmacy App. Аналіз став базою для створення додатку, який усуває недоліки аналогів і відповідає потребам користувачів.

2. ЗАСОБИ РЕАЛІЗАЦІЇ

Розробка мобільного додатку *My Home Pharmacy App* для обліку домашньої аптечки вимагала створення рішення з офлайн-доступом, україномовним інтерфейсом, гнучкою категоризацією медикаментів, сповіщеннями про терміни придатності та можливістю експорту даних через спливне вікно. У цьому розділі описано інструменти, мови програмування, фреймворки, бібліотеки, систему зберігання даних, контроль версій і засоби проектування інтерфейсу, які забезпечили реалізацію функціоналу.

2.1 Середовище розробки

Для розробки *My Home Pharmacy App* використано три інструменти, які оптимізують кодування, тестування і дебагінг:

- **Visual Studio Code (VS Code)**: Безкоштовний редактор коду з підтримкою JavaScript, TypeScript і React Native. Використовувався з розширеннями:
  - **Prettier**: Автоматичне форматування коду.
  - **ESLint**: Статичний аналіз (правила Airbnb, TypeScript).
  - **React Native Tools**: Дебагінг і запуск додатків.
  - **GitLens**: Управління гілками Git.

Переваги: **Visual Studio Code** використовувався як основне середовище розробки завдяки підтримці React Native, TypeScript, ESLint і Jest. Забезпечено швидкий доступ до коду, автоматичне форматування (Prettier) і інтеграцію з системою контролю версій Git.

Недоліки: потребує налаштування розширень.

- **Expo CLI**: Інструмент для спрощення розробки React Native додатків. Забезпечує створення проєкту, гаряче перезавантаження, тестування через QR-код і доступ до API (Notifications, FileSystem).

Переваги: Expo CLI надає ефективне середовище для тестування додатку, зокрема завдяки гарячому перезавантаженню та простому налаштуванню Push-сповіщень.

Недоліки: більший розмір додатку (~50 МБ), обмеження для нативних модулів.

- **Android Studio**: IDE для Android-розробки. Використовувалася для налаштування Android-емуляторів, тестування сповіщень (Expo Notifications) і дебагінгу Android-специфічних функцій.

Переваги: Android Studio застосовувалась виключно для тестування Android-специфічного функціоналу, зокрема емуляції сповіщень та офлайн-режиму на віртуальному пристрої.

Недоліки: висока ресурсомісткість, складне налаштування. Android Studio забезпечила точне тестування офлайн-доступу і сповіщень на Android.

Обґрунтування вибору: VS Code забезпечив гнучке середовище для кодування, Expo CLI прискорив кросплатформну розробку, а Android Studio дозволив протестувати Android-специфічний функціонал, такий як сповіщення і збереження даних. Це сприяло реалізації офлайн-доступу і push-сповіщень – ключових переваг *My Home Pharmacy App*.

Таблиця 2.1

Порівняння середовищ розробки

Інструмент	**Гнучкість**	**Швидкість налаштування**	**Підтримка React Native**	**Використання в проєкті**
VS Code	Висока	Середня	Так	Кодування, дебагінг
Expo CLI	Середня	Висока	Так	Тестування, збірка
Android Studio	Низька	Низька	Так	Емулятори, тест. Android

2.2 Мови програмування

Розробка базується на двох мовах програмування

Для створення додатку використано **JavaScript** і **TypeScript**, які доповнюють один одного у різних аспектах розробки. Їхній розподіл у проєкті відображає специфіку завдань, таких як створення зручного інтерфейсу для літніх користувачів, управління станом і забезпечення надійності даних.

TypeScript (~85% коду): Основна мова проєкту, використовується для розробки ключових компонентів, таких як екрани (AddMedicine.tsx, [id].tsx, index.tsx), управління станом через Zustand (medicineStore.ts, categoriesStore.ts), сервіси (notificationService.ts), типи (Medicine, Category) та навігацію (Expo Router).

- **Переваги:** Статична типізація зменшує ризик помилок, особливо при роботі зі складними структурами даних, такими як об’єкти ліків (Medicine) чи категорії (Category). Наприклад, типізація у medicineStore.ts дозволила уникнути помилок під час фільтрації ліків за термінами придатності. Також

TypeScript забезпечує автодоповнення у середовищі розробки, що прискорює написання коду, та підтримує масштабовість проєкту.

- **Недоліки:** Додатковий час на визначення типів (наприклад, для Medicine чи параметрів функцій у notificationService.ts) може сповільнити початкову розробку. Крім того, іноді виникає необхідність у додаткових бібліотеках типів (наприклад, для Expo Router), що ускладнює налаштування.

Приклад використання: У файлі AddMedicine.tsx типізована форма для введення даних ліків (назва, категорія, термін придатності), що забезпечило коректне збереження через AsyncStorage. У medicineStore.ts типи допомогли організувати фільтрацію ліків за категоріями та термінами придатності.

JavaScript (~15% коду): Використовується для утилітарних модулів (наприклад, translations.ts) та конфігураційних файлів (babel.config.js, metro.config.js).

Переваги: Простий синтаксис і велика екосистема дозволяють швидко створювати допоміжні функції, наприклад, у translations.ts для локалізації текстів. JavaScript підтримує асинхронне програмування (асинхронні функції у translations.ts для завантаження перекладів), що спрощує інтеграцію з React Native.

Недоліки: Динамічна типізація може призводити до помилок під час виконання, особливо у великих проєктах. Наприклад, відсутність типів у translations.ts могла б ускладнити відстеження помилок у перекладах для різних мов.

Приклад використання: У translations.ts реалізовано функцію useTranslation, яка забезпечує локалізацію інтерфейсу (наприклад, текстів кнопок у AddMedicine.tsx), що є важливим для зручності літніх користувачів.

Обґрунтування вибору

TypeScript став основою проєкту завдяки своїй надійності та підтримці типізації, що було критично важливим для управління станом (medicineStore.ts) і даних (AsyncStorage). Наприклад, тип Medicine дозволив чітко визначити

структуру об’єкта ліків, що полегшило створення спливного вікна для експорту у [id].tsx та уникнення помилок під час категоризації у index.tsx. Це підвищило стабільність додатку, особливо для літніх користувачів, які потребують чіткого і безпомилкового інтерфейсу.

JavaScript, у свою чергу, забезпечив швидкість розробки допоміжних модулів, таких як локалізація (translations.ts), що дозволило оперативно налаштувати багатомовність для ширшої аудиторії. Поєднання обох мов дало баланс між швидкістю розробки та надійністю: TypeScript гарантував стабільність основних компонентів, а JavaScript спростив створення утилітарного коду.

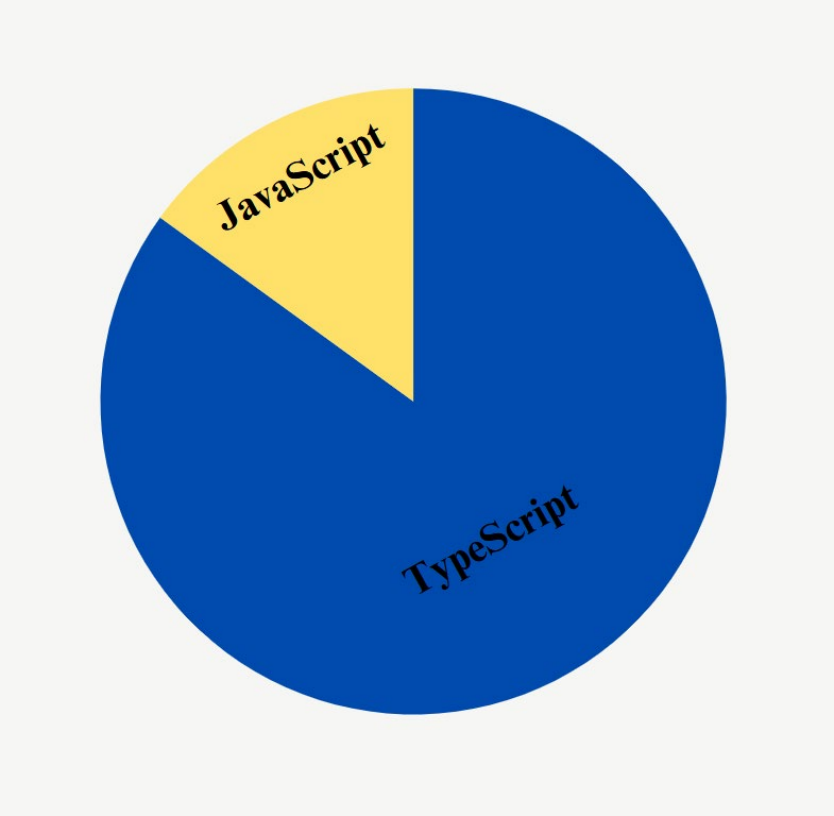


Рис. 2.1 Розподіл мов програмування

2.3 Фреймворки та бібліотеки

Для *My Home Pharmacy App* використано технології, які підтримують кросплатформність, офлайн-доступ, локалізацію і сповіщення:

- **React Native**: Фреймворк для кросплатформних додатків. Використовується для екранів (app/(tabs)/index.tsx, app/medicine/[id].tsx). Переваги: єдина кодова база, гаряче перезавантаження. Недоліки: обмеження для анімацій.
- **Expo Router**: Файлова навігація (app/auth.tsx, app/add-medicine.tsx). Забезпечує захист маршрутів і глибоку навігацію. Переваги: інтуїтивність, підтримка сповіщень. Недоліки: обмежена документація.
- **Zustand**: Управління станом (authStore.ts, medicineStore.ts). Persist middleware зберігає дані в AsyncStorage. Переваги: простота, офлайн-підтримка. Недоліки: потребує структуризації. Гнучка категоризація – перевага додатку.
- **AsyncStorage**: Локальне зберігання даних. Переваги: офлайн-доступ. Недоліки: обмеження обсягу (~6 МБ). Реалізує офлайн-доступ – ключову перевагу.
- **Expo Notifications**: Push-сповіщення (notificationService.ts). Переваги: кросплатформність, глибока навігація. Недоліки: залежність від фонових процесів.
- **i18n-js**: Локалізація (translations.ts). Переваги: підтримка множин. Недоліки: ручний переклад. Українська локалізація – перевага.
- **Jest**: Модульне тестування. Переваги: підтримка знімків. Недоліки: повільність на великих проєктах. [Уточнення: вкажіть версію, наприклад, 29.7.0].
- **ESLint**: Статичний аналіз (правила Airbnb, TypeScript). Переваги: виявлення помилок. Недоліки: потребує налаштування.

-

Таблиця 2.2

Порівняння бібліотек управління станом

Бібліотека	**Продуктивність**	**Складність API**	**Офлайн-підтримка**	**Використання в проєкті**
Zustand	Висока	Проста	Так (Persist)	Управління станом
Redux	Середня	Складна	Так	Не використовується
MobX	Висока	Середня	Так	Не використовується

Обґрунтування вибору: React Native і Expo Router забезпечили кросплатформність, Zustand і AsyncStorage – офлайн-доступ і категоризацію, Expo Notifications – сповіщення, i18n-js – україномовний інтерфейс, Jest і ESLint – якість коду.

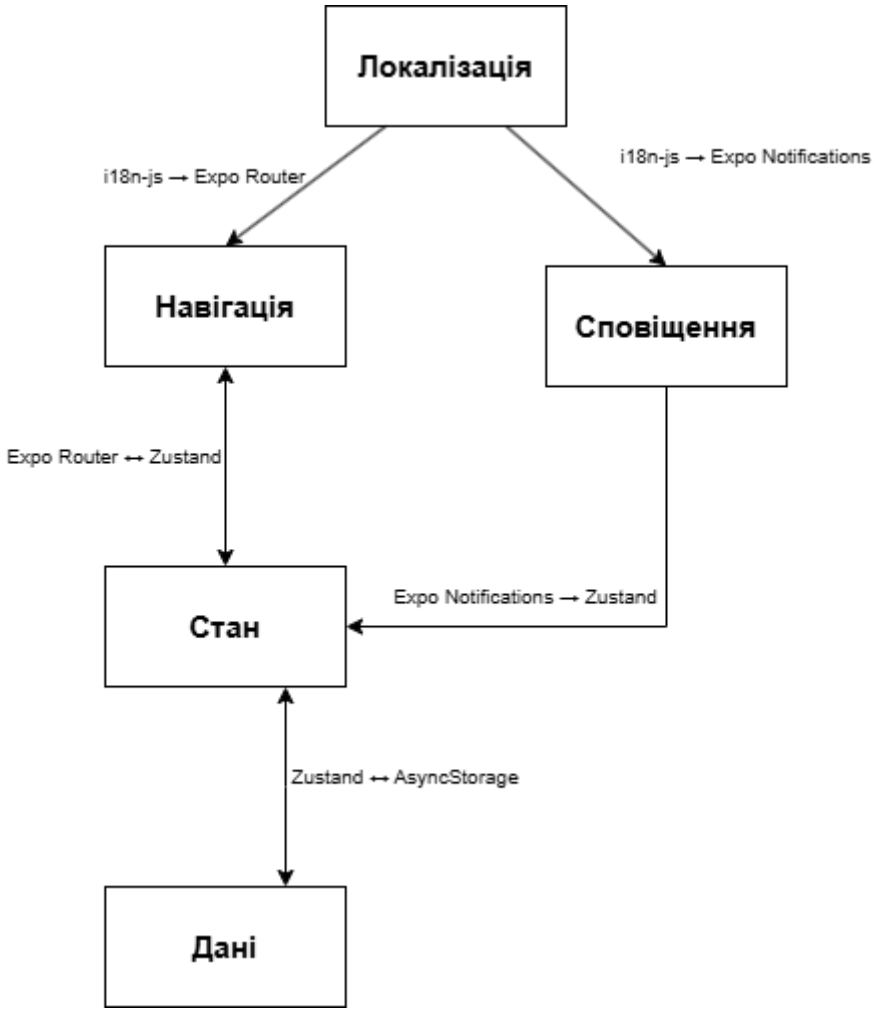


Рисунок 2.2 – Архітектура клієнтської частини

Приклад коду: Zustand store для медикаментів

```
import { create } from 'zustand';
import { persist, createJSONStorage } from 'zustand/middleware';
import AsyncStorage from '@react-native-async-storage/async-storage';

export interface Medicine {
  id: string;
  name: string;
  category: string;
  expiryDate: string;
  quantity: number;
  description: string;
  note: string;
```

```
  createdAt: string;
  updatedAt: string;
}

interface MedicineState {
  medicines: Medicine[];
  addMedicine: (medicine: Omit<Medicine, 'id' | 'createdAt' | 'updatedAt'>) => void;
  updateMedicine: (id: string, medicine: Partial<Medicine>) => void;
  deleteMedicine: (id: string) => void;
}

export const useMedicineStore = create<MedicineState>()(
  persist(
    (set) => ({
      medicines: [],
      addMedicine: (medicine) => {
        const newMedicine = {
          ...medicine,
          id: Date.now().toString(),
          createdAt: new Date().toISOString(),
          updatedAt: new Date().toISOString(),
        };
        set((state) => ({ medicines: [...state.medicines, newMedicine] }));
      },
      updateMedicine: (id, medicine) => {
        set((state) => ({
          medicines: state.medicines.map((m) =>
           m.id === id ? { ...m, ...medicine, updatedAt: new Date().toISOString() } : m
          ),
        }));
      },
      deleteMedicine: (id) => {
        set((state) => ({ medicines: state.medicines.filter((m) => m.id !== id) }));
      },
    }),
    {
      name: 'medicines-storage',
```

```
      storage: createJSONStorage(() => AsyncStorage),
    }
  )
);
```

Стан медикаментів зберігається у Zustand store. Для кожного медикаменту автоматично генерується id, дата створення та оновлення. Зміни синхронізуються через AsyncStorage, що забезпечує офлайн-доступ.

2.4 Система зберігання даних

AsyncStorage використовується для локального зберігання.

2.4.1 Структура даних

Медикаменти (medicines-storage): Масив об’єктів: `id` (унікальний ідентифікатор), `name` (назва), `category` (категорія), `quantity` (кількість), `expiryDate` (термін придатності), `description` (опис), `note` (нотатки), `createdAt` (дата створення), `updatedAt` (дата оновлення).

Категорії (categories-storage): Масив об’єктів: `id` (унікальний ідентифікатор), `name` (назва), `isCustom` (чи є категорія користувацькою).

Налаштування (settings-storage): Об’єкт: `language` (мова інтерфейсу), `theme` (тема оформлення), `notifications` (налаштування сповіщень).

2.4.2 Переваги

- **Офлайн-доступ**: Дані доступні без інтернету.
- **Простота**: JSON-зберігання.
- **Інтеграція**: Persist middleware.

2.4.3 Недоліки

- **Обмеження обсягу**: ~6 МБ.

- **Без шифрування**: Використана обфускація.

Обґрунтування вибору: AsyncStorage реалізував офлайн-доступ, що вирізняє *My Home Pharmacy App*.

Рис. 2.3 – Структура даних AsyncStorage в форматі JSON

На діаграмі зображено структуру трьох основних сховищ, які зберігаються у AsyncStorage в форматі JSON

Приклад коду: Служба сповіщень

```
export async function scheduleExpiryNotification(medicine: Medicine, daysBeforeExpiry: number = 30) {
  const expiryDate = new Date(medicine.expiryDate);
  const notificationDate = new Date(expiryDate);
  notificationDate.setDate(notificationDate.getDate() - daysBeforeExpiry);

  if (notificationDate < new Date()) {
    console.log(`Not scheduling expired notification for ${medicine.name}`);
    return null;
  }

  console.log(`Scheduling notification for "${medicine.name}" on ${notificationDate.toLocaleString()} (${daysBeforeExpiry} days before expiry)`);

  const identifier = `medicine-expiry-${medicine.id}-${daysBeforeExpiry}`;
  await Notifications.cancelScheduledNotificationAsync(identifier);

  return await Notifications.scheduleNotificationAsync({
    content: {
      title: `Срок годности лекарства "${medicine.name}"`,
      body: `Лекарство "${medicine.name}" истекает через ${daysBeforeExpiry} дней (${new Date(medicine.expiryDate).toLocaleDateString()})`,
      data: { medicineId: medicine.id },
```

```
    },
    trigger: { date: notificationDate, channelId: 'medicine-expiry' },
    identifier,
  });
}
```

2.5 Система контролю версій

Git і **GitHub** для контролю версій.

2.5.1 Переваги

- **Гнучкість**: Гілки (feature/add-categories).
- **Історія змін**.
- **Інтеграція**: GitLens.

2.5.2 Недоліки

- **Складність**.
- **Інтернет**.

Обґрунтування вибору: Git і GitHub прискорили розробку експорту і категоризації.

2.6 Інструменти проектування інтерфейсу

Інтерфейс *My Home Pharmacy App* розроблено напряму в React Native через theme.ts без використання окремих інструментів дизайну. Стилізація зосереджена на простоті для літніх користувачів і сімей.

2.6.1 Особливості

- **Простий інтерфейс**: Великі шрифти (>16px), контрастні кольори, мінімалістичні кнопки.

- **Теми**: Світла і темна (ThemeContext.tsx).
- **Локалізація**: Українські тексти через translations.ts.

2.6.2 Переваги

- **Швидкість**: Стилі в theme.ts прискорили розробку.
- **Гнучкість**: Легке оновлення стилів.
- **Простота**: Інтерфейс зручний для цільової аудиторії.

2.6.3 Недоліки

- **Обмежена візуалізація**: Без прототипів складніше тестувати UI.
- **Ручна робота**: Стилі писалися вручну.

Обґрунтування вибору: Стилізація через theme.ts дозволила швидко створити простий інтерфейс, що є перевагою *My Home Pharmacy App*.

Приклад коду: Налаштування теми

```
import React, { createContext } from 'react';
import { useColorScheme } from 'react-native';

type ThemeColors = {
  background: string;
  card: string;
  text: string;
  border: string;
  primary: string;
};

const lightColors: ThemeColors = {
  background: '#F2F2F7',
  card: '#FFFFFF',
  text: '#000000',
  border: '#E5E5EA',
  primary: '#007AFF',
};

const darkColors: ThemeColors = {
  background: '#000000',
```

```
  card: '#1C1C1E',
  text: '#FFFFFF',
  border: '#38383A',
  primary: '#0A84FF',
};

type ThemeContextType = {
  isDark: boolean;
  colors: ThemeColors;
};

const ThemeContext = createContext<ThemeContextType>({ isDark: false, colors: lightColors });

export const ThemeProvider: React.FC<{ children: React.ReactNode }> = ({ children }) => {
  const systemTheme = useColorScheme();
  const isDark = systemTheme === 'dark';
  const colors = isDark ? darkColors : lightColors;

  return <ThemeContext.Provider value={{ isDark, colors }}>{children}</ThemeContext.Provider>;
};
```

ThemeContext забезпечує перемикання між світлою та темною темами залежно від системних налаштувань. Це підвищує доступність для літніх користувачів. Колірні схеми визначені вручну у двох об’єктах – lightColors і darkColors.

Висновок до Розділу 2: Засоби реалізації

Розділ визначив оптимальний технологічний стек для розробки My Home Pharmacy App, включаючи Visual Studio Code як основне середовище, React Native та Expo CLI для кросплатформності, Zustand і AsyncStorage для управління станом і офлайн-доступу, Expo Notifications для сповіщень, i18n-js для локалізації та Jest для тестування. Використання TypeScript (~85%) забезпечило надійність коду, а JavaScript (~15%) спростило створення утилітарних модулів. Інструменти, такі як Git і ThemeContext, сприяли ефективному контролю версій і стилізації інтерфейсу. Обрані засоби

забезпечили реалізацію ключових функцій додатку, таких як офлайн-доступ і україномовний інтерфейс.

3. ПРОЄКТУВАННЯ

Проєктування *My Home Pharmacy App* зосереджено на створенні мобільного додатку для обліку домашньої аптечки з підтримкою офлайн-доступу, україномовного інтерфейсу, гнучкої категоризації медикаментів, сповіщень про терміни придатності та експорту даних через спливне вікно. У цьому розділі описано архітектуру додатку, клієнтську частину та локальне зберігання даних, які забезпечують реалізацію функціоналу.

3.1 Проєктування архітектури додатку

Архітектура *My Home Pharmacy App* розроблена як повністю клієнтська, без серверної частини, що забезпечує локальну обробку даних і незалежність від інтернету. Основні принципи проєктування:

- **Офлайн-доступ**: Усі дані (медикаменти, категорії, налаштування) зберігаються локально через AsyncStorage, що дозволяє використовувати додаток без підключення до мережі.
- **Модульність**: Логіка розділена на модулі (store/, services/, utils/), що полегшує підтримку та розширення.
- **Локалізація**: i18n-js підтримує україномовний інтерфейс, що вирізняє додаток серед аналогів, таких як MyTherapy чи Medisafe.
- **Гнучкість**: Zustand дозволяє створювати необмежену кількість користувацьких категорій медикаментів.
- **Простота**: Інтерфейс, стилізований через theme.ts, орієнтований на літніх користувачів і сім’ї, з великими шрифтами та контрастними кольорами.
- **Експорт даних**: Використання Share API для обміну інформацією про медикаменти через текстове спливне вікно.

Технологічний стек включає:

- **React Native**: Для кросплатформного UI.
- **Expo Router**: Для файлової навігації.
- **Zustand**: Для управління станом із Persist middleware.
- **AsyncStorage**: Для локального зберігання.
- **Expo Notifications**: Для сповіщень із вибором часу.
- **i18n-js**: Для локалізації.

Архітектура побудована навколо клієнтської частини, де UI, навігація, стан, дані та сповіщення взаємодіють локально. Модульність забезпечується поділом на екрани (app/), сховища стану (store/), сервіси (services/) та утиліти (utils/).

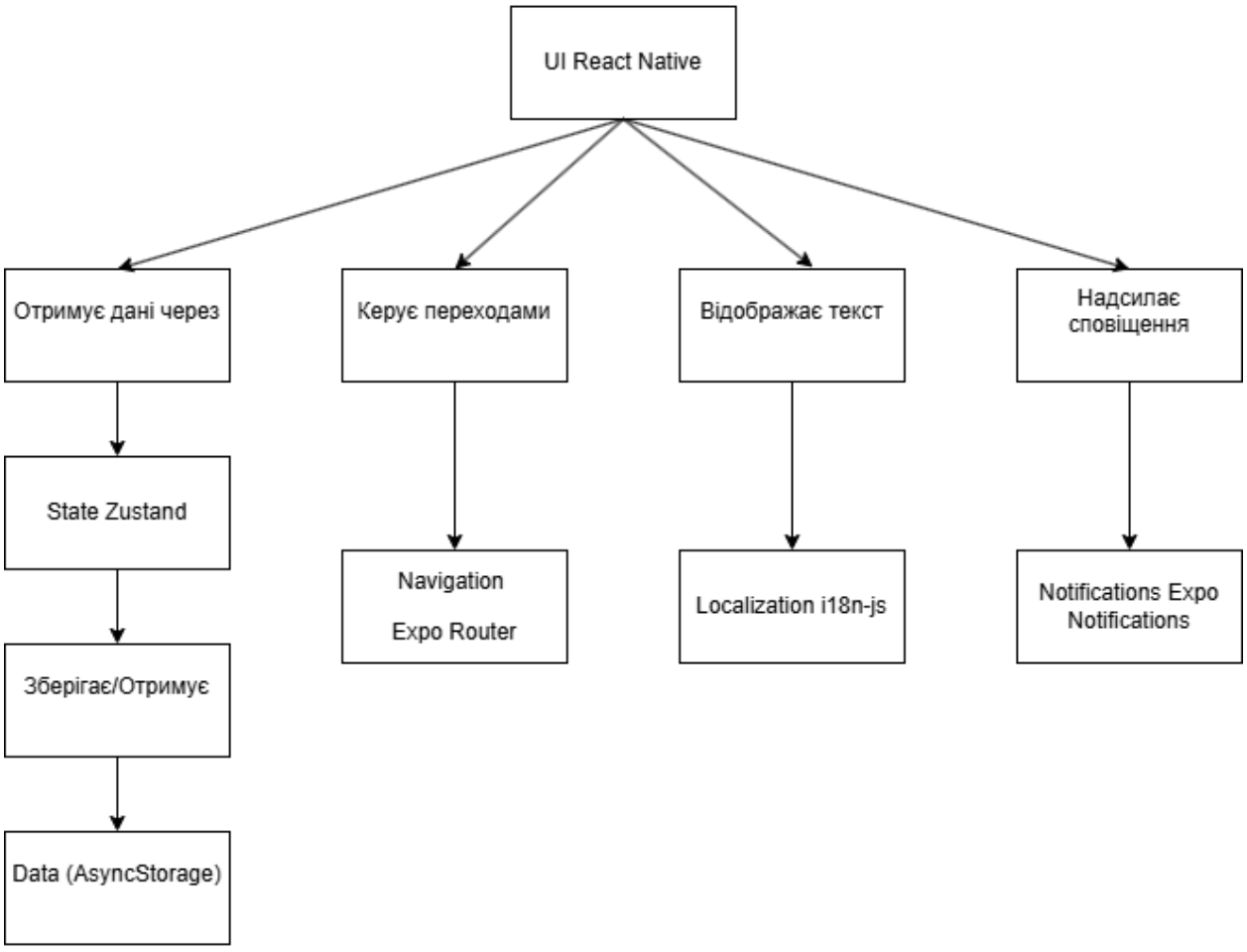


Рис. 3.1 – Загальна архітектура додатку

3.2 Архітектура клієнтської частини

Клієнтська частина *My Home Pharmacy App* включає п’ять основних компонентів: інтерфейс користувача, навігацію, управління станом, сповіщення та локалізацію.

3.2.1 Інтерфейс користувача

Інтерфейс користувача реалізовано через React Native і структуровано в директорії app/:

- **Екрани авторизації**: auth.tsx, verify.tsx, reset-password.tsx для реєстрації, верифікації та відновлення паролю.
- **Екрани управління ліками**: add-medicine.tsx для додавання, medicine/[id].tsx для перегляду деталей, редагування та видалення.
- **Головні екрани**: (tabs)/index.tsx (список ліків), (tabs)/settings.tsx (налаштування), (tabs)/notification-settings.tsx (налаштування сповіщень).

Стилізація виконана через theme.ts, що забезпечує світлу та темну теми, великі шрифти (>16px) і контрастні кольори для літніх користувачів. Управління темами реалізовано через ThemeContext.tsx.

3.2.2 Навігація

Навігація побудована на Expo Router із файловою системою маршрутів:

- **Кореневий макет** (_layout.tsx): Налаштовує вкладки та захист маршрутів через authStore.ts.
- **Захист маршрутів**: Перевірка авторизації (наприклад, доступ до (tabs)/ лише для авторизованих користувачів).
- **Глибока навігація**: Натискання на сповіщення відкриває medicine/[id].tsx із деталями ліків.

Переваги: інтуїтивна навігація, підтримка динамічних маршрутів ([id].tsx). Недоліки: обмежена документація для складних сценаріїв.

3.2.3 Управління станом

Zustand використовується для управління станом у модулях:

- authStore.ts: Стан авторизації (наприклад, isAuthenticated).
- medicineStore.ts: Список медикаментів, операції додавання, редагування, видалення.
- categoriesStore.ts: Стандартні та користувацькі категорії без обмежень.
- settingsStore.ts: Налаштування мови, теми.
- notificationSettingsStore.ts: Налаштування сповіщень (наприклад, час сповіщення).

Persist middleware зберігає стан у AsyncStorage, забезпечуючи офлайн-доступ.

Приклад коду: Тип Medicine та categoriesStore.ts

```
import { create } from 'zustand';
import { persist, createJSONStorage } from 'zustand/middleware';
import AsyncStorage from '@react-native-async-storage/async-storage';

export interface Category {
  id: string;
  name: string;
  description?: string;
  icon?: string;
  color?: string;
  isSystem: boolean;
  tags: string[];
  createdAt: string;
  updatedAt: string;
}

export interface CategoryStore {
  categories: Category[];
  userCategories: Category[];
  systemCategories: Category[];
  addCategory: (category: Omit<Category, 'id' | 'createdAt' | 'updatedAt'>) => void;
```

```
  updateCategory: (id: string, category: Partial<Category>) => void;
  deleteCategory: (id: string) => void;
}

const systemCategories: Category[] = [
  { id: '1', name: 'Антибиотики', description: 'Лекарства для лечения бактериальных инфекций', icon: 'medical', color: '#FF6B6B', isSystem: true, tags: ['антибиотик', 'бактерии', 'инфекция'], createdAt: new Date().toISOString(), updatedAt: new Date().toISOString() },
  { id: '2', name: 'Обезболивающие', description: 'Лекарства для снятия боли', icon: 'bandage', color: '#4ECDC4', isSystem: true, tags: ['боль', 'анальгетик', 'жаропонижающее'], createdAt: new Date().toISOString(), updatedAt: new Date().toISOString() },
];

export const useCategoryStore = create<CategoryStore>()(
  persist(
    (set) => ({
      categories: systemCategories,
      userCategories: [],
      systemCategories,
      addCategory: (category) => {
        const newCategory: Category = { ...category, id: Date.now().toString(), createdAt: new Date().toISOString(), updatedAt: new Date().toISOString() };
        set((state) => ({ categories: [...state.categories, newCategory], userCategories: [...state.userCategories, newCategory] }));
      },
      updateCategory: (id, category) => {
        set((state) => ({
          categories: state.categories.map((c) => c.id === id ? { ...c, ...category, updatedAt: new Date().toISOString() } : c),
          userCategories: state.userCategories.map((c) => c.id === id ? { ...c, ...category, updatedAt: new Date().toISOString() } : c),
        }));
      },
      deleteCategory: (id) => {
        set((state) => ({
          categories: state.categories.filter((c) => c.id !== id),
          userCategories: state.userCategories.filter((c) => c.id !== id),
        }));
      },
    }),
```

```
    { name: 'categories-storage', storage: createJSONStorage(() => AsyncStorage) }
  )
);
```

Користувачі можуть створювати власні категорії з тегами, кольорами та піктограмами. У прикладі нижче — ініціалізація системних категорій та функції додавання, оновлення і видалення категорій.

3.2.4 Сповіщення

Сповіщення реалізовані через Expo Notifications у notificationService.ts. Користувач може вибирати час сповіщення (наприклад, за 7 днів до закінчення терміну придатності). Натискання на сповіщення відкриває деталі ліків через глибоку навігацію.

Реалізація push-сповіщень винесена в окремий сервіс notificationService.ts, приклад коду наведено у розділі 4.1.4.

Сповіщення тестувалися через Android Studio для забезпечення коректної роботи на Android-пристроях.

3.2.5 Локалізація

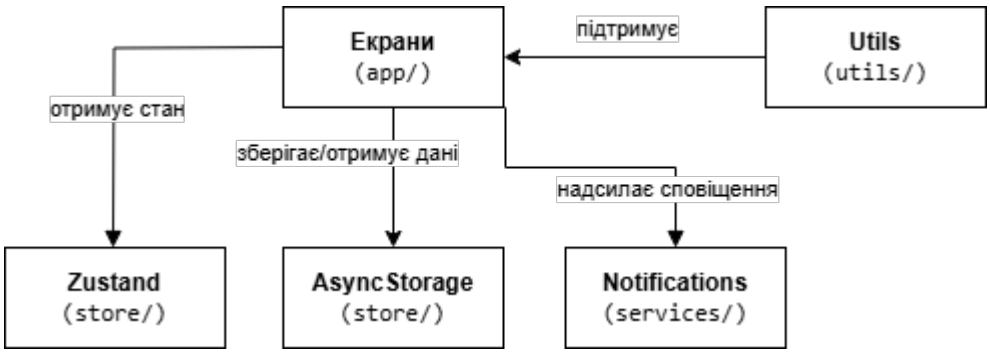
Локалізація реалізована через i18n-js у translations.ts, що забезпечує україномовний інтерфейс і підтримку інших мов. Тексти для категорій, екранів і сповіщень локалізовані.

3.2.6 Експорт даних

Експорт реалізовано у [id].tsx через вбудований Share API стандартного пакету React Native, що дозволяє надсилати текстову інформацію про медикаменти (наприклад, «Назва: Парацетамол, Термін: 31.12.2025») у месенджер.

Рисунок 3.2

Архітектура клієнтської частини



3.3 Архітектура локального зберігання даних

Локальне зберігання даних у *My Home Pharmacy App* реалізовано через AsyncStorage, що забезпечує офлайн-доступ до медикаментів, категорій і налаштувань.

3.3.1 Структура даних

Дані зберігаються у форматі JSON у трьох сховищах:

- **Медикаменти** (medicine-storage): Масив об’єктів із полями:
  - id: string – унікальний ідентифікатор.
  - name: string – назва (наприклад, «Парацетамол»).
  - category: string – категорія (наприклад, «Таблетки»).
  - quantity: number – кількість (наприклад, 20).
  - expDate: string – термін придатності (наприклад, «31.12.2025»).
  - description?: string – опис (наприклад, «500 мг»).
  - notes?: string – нотатки (наприклад, «Після їжі»).
- **Категорії** (categories-storage): Масив об’єктів із полями:
  - id: string – ідентифікатор.
  - name: string – назва (наприклад, «Сиропи»).
  - isCustom: boolean – чи створена користувачем.

- **Налаштування** (settings-storage): Об’єкт із полями:
  - language: string – мова (наприклад, «uk»).
  - theme: 'light' | 'dark' – тема.
  - notifications: { enabled: boolean; daysBefore: number } – налаштування сповіщень (наприклад, { enabled: true, daysBefore: 7 }).

3.3.2 Принципи зберігання

- **Офлайн-доступ**: AsyncStorage зберігає дані локально, що дозволяє працювати без інтернету.
- **Синхронізація**: Persist middleware у Zustand автоматично оновлює AsyncStorage при зміні стану.
- **Гнучкість**: Немає обмежень на кількість категорій чи медикаментів (обмеження лише обсягом AsyncStorage, ~6 МБ).
- **Експорт**: Дані про медикаменти доступні для експорту через Share API у текстовому форматі.

3.3.3 Переваги та недоліки

- **Переваги**:
  - Простота: JSON-формат легкий для реалізації.
  - Офлайн-доступ: ключова перевага порівняно з аналогами.
  - Інтеграція з Zustand: швидке збереження стану.
- **Недоліки**:
  - Обмеження обсягу: ~6 МБ, достатньо для домашньої аптечки, але не для великих баз.
  - Відсутність шифрування: використано обфускацію ключів.

Таблиця 3.1

Порівняння підходів до зберігання даних

Технологія	**Офлайн-доступ**	**Обсяг даних**	**Шифрування**	**Використання в проєкті**
AsyncStorage	Так	~6 МБ	Ні	Локальне зберігання
SQLite	Так	>100 МБ	Так	Не використовується
Realm	Так	>100 МБ	Так	Не використовується

Обґрунтування вибору: AsyncStorage обрано за простоту, достатній обсяг і підтримку офлайн-доступу, що є ключовою перевагою *My Home Pharmacy App*. Для майбутнього розширення можна інтегрувати SQLite, але для поточних потреб AsyncStorage оптимальний.

Висновок до Розділу 3: Проєктування

Проєктування архітектури My Home Pharmacy App забезпечило створення клієнтського додатку з офлайн-доступом, модульною структурою та україномовним інтерфейсом. Розроблено архітектуру, що включає інтерфейс (React Native), навігацію (Expo Router), управління станом (Zustand), сповіщення (Expo Notifications) та локальне зберігання (AsyncStorage). Структура даних (medicines-storage, categories-storage, settings-storage) дозволяє гнучку категоризацію та експорт даних через Share API. Проєктування орієнтоване на простоту інтерфейсу для літніх користувачів і сімей, що відповідає поставленим цілям.

4. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

Розділ описує реалізацію та тестування *My Home Pharmacy App* — кросплатформного додатку для обліку домашньої аптечки. Реалізація охоплює інтерфейс, навігацію, управління станом, сповіщення, локалізацію та експорт даних. Тестування забезпечило стабільність функціоналу, включаючи офлайн-доступ, україномовний інтерфейс, гнучку категоризацію та сповіщення.

4.1 Реалізація клієнтської частини

Клієнтська частина реалізована з використанням React Native, Expo Router, Zustand, AsyncStorage, Expo Notifications та i18n-js.

4.1.1 Інтерфейс користувача

Інтерфейс структуровано в app/:

- **Авторизація**: auth.tsx (вхід/реєстрація), verify.tsx (верифікація), reset-password.tsx (відновлення).
- **Управління ліками**: add-medicine.tsx (додавання з полями: назва, категорія, кількість, термін, опис, нотатки), index.tsx (список ліків).
- **Налаштування**: settings.tsx (мова, тема), notification-settings.tsx (час сповіщень).

Стилізація інтерфейсу реалізована через ThemeContext.tsx, як описано в розділі 2.6.

4.1.2 Навігація

Expo Router у _layout.tsx налаштовує вкладки (index.tsx, settings.tsx, notification-settings.tsx) і захищає маршрути через authStore.ts. Сторінка +not-found.tsx обробляє помилки маршрутів. Глибока навігація реалізовано через сповіщення.

4.1.3 Управління станом

Zustand із Persist middleware керує станом у src/store/:

- authStore.ts: Стан авторизації.
- categoriesStore.ts: Стандартні та користувацькі категорії.
- medicinesStore.ts: Список ліків.
- notificationSettingsStore.ts: Налаштування сповіщень.
- settingsStore.ts: Мова і тема.

Приклад коду: authStore.ts

```
import { create } from 'zustand';
import { persist, createJSONStorage } from 'zustand/middleware';
import AsyncStorage from '@react-native-async-storage/async-storage';
import { router } from 'expo-router';

interface AuthState {
  isAuthenticated: boolean;
  username: string | null;
  isSessionActive: boolean;
  securityQuestion: string | null;
  setAuth: (username: string, password: string) => Promise<void>;
  verifyPassword: (password: string) => Promise<boolean>;
  logout: () => Promise<void>;
}

export const useAuthStore = create<AuthState>()(
  persist(
    (set) => ({
      isAuthenticated: false,
      username: null,
      isSessionActive: false,
      securityQuestion: null,
      setAuth: async (username: string, password: string) => {
        await AsyncStorage.setItem('password', password);
        set({ isAuthenticated: true, username, isSessionActive: true });
        router.replace('/(tabs)');
      },
      verifyPassword: async (password: string) => {
        const storedPassword = await AsyncStorage.getItem('password');
```

```
            if (storedPassword === password) {
              set({ isSessionActive: true });
              return true;
            }
            return false;
          },
          logout: async () => {
            await AsyncStorage.removeItem('password');
            set({ isAuthenticated: false, username: null, isSessionActive: false });
            router.replace('/auth');
          },
        }),
        {
          name: 'auth-storage',
          storage: createJSONStorage(() => AsyncStorage),
          partialize: (state) => ({ isAuthenticated: state.isAuthenticated, username: state.username, securityQuestion: state.securityQuestion }),
        }
      )
    );
```

Стан автентифікації реалізовано через Zustand з використанням AsyncStorage для збереження сесії. Є підтримка локального пароля, перевірки при вході та виході з системи. Перехід між екранами здійснюється через Expo Router.

4.1.4 Сповіщення

Сповіщення реалізовані в src/services/notificationService.ts із можливістю вибору часу (наприклад, за 7 днів).

Приклад коду: notificationService.ts

```
import * as Notifications from 'expo-notifications';
import { Medicine } from '../store/medicineStore';

export async function scheduleExpiryNotification(medicine: Medicine, daysBeforeExpiry: number = 30) {
  const expiryDate = new Date(medicine.expiryDate);
  const notificationDate = new Date(expiryDate);
  notificationDate.setDate(notificationDate.getDate() - daysBeforeExpiry);
```

```
  if (notificationDate < new Date()) return null;
  const identifier = `medicine-expiry-${medicine.id}-${daysBeforeExpiry}`;
  await Notifications.cancelScheduledNotificationAsync(identifier);
  return await Notifications.scheduleNotificationAsync({
    content: {
      title: `Срок годности "${medicine.name}"`,
      body: `Истекает через ${daysBeforeExpiry} дней`,
      data: { medicineId: medicine.id },
    },
    trigger: { date: notificationDate, channelId: 'medicine-expiry' },
    identifier,
  });
}

export async function cancelExpiryNotification(medicineId: string, daysBeforeExpiry: number = 30) {
  const identifier = `medicine-expiry-${medicineId}-${daysBeforeExpiry}`;
  await Notifications.cancelScheduledNotificationAsync(identifier);
}

export async function scheduleAllExpiryNotifications(medicines: Medicine[], days: number[] = [30, 7,
1]) {
  for (const medicine of medicines) {
    for (const day of days) {
      await scheduleExpiryNotification(medicine, day);
    }
  }
}
```

Цей модуль відповідає за планування push-сповіщень із можливістю вказати кількість днів до терміну придатності. Сповіщення зберігаються за унікальними ідентифікаторами, що дозволяє уникнути дублювань. Реалізація використовує Expo Notifications API.

4.1.5 Локалізація

Локалізація через src/utils/translations.ts забезпечує україномовний інтерфейс.

Приклад коду: translations.ts

```
import { useSettingsStore } from "../store/settingsStore";
```

```
type Translation = {
  uk: string;
  en: string;
};

type NestedTranslation = {
  [key: string]: Translation | NestedTranslation;
};

const translations: NestedTranslation = {
  homeTitle: { uk: 'Ваша домашня аптечка', en: 'Your Home Pharmacy' },
  addMedicine: {
    title: { en: 'Add Medicine', uk: 'Додати ліки' },
    name: { en: 'Name', uk: 'Назва' },
    expiryDate: { en: 'Expiry Date', uk: 'Термін придатності' },
    submit: { en: 'Add Medicine', uk: 'Додати ліки' },
  },
  settingsTitle: { uk: 'Налаштування', en: 'Settings' },
  themeLabel: { uk: 'Тема', en: 'Theme' },
  languageLabel: { uk: 'Мова', en: 'Language' },
};

const getNestedTranslation = (obj: NestedTranslation, path: string): Translation | undefined => {
  const keys = path.split('.');
  let current: any = obj;
  for (const key of keys) {
    if (current[key] === undefined) return undefined;
    current = current[key];
  }
  return current as Translation;
};

export const useTranslation = () => {
  const { language } = useSettingsStore();
  const t = (key: string) => {
    const translation = getNestedTranslation(translations, key);
    return translation ? translation[language] : key;
  };
  return { t, language };
```

```
};

export type CategoryId = 'tablets' | 'ointment' | 'drops' | 'syrup' | 'injections';
export type CategoryName = 'Таблетки' | 'Tablets' | 'Мазь' | 'Ointment' | 'Краплі' | 'Drops';

export const categories = {
  uk: ['Таблетки', 'Мазь', 'Краплі'],
  en: ['Tablets', 'Ointment', 'Drops'],
};

export const categoryMapping: Record<CategoryName, CategoryId> = {
  'Таблетки': 'tablets',
  'Tablets': 'tablets',
  'Мазь': 'ointment',
  'Ointment': 'ointment',
  'Краплі': 'drops',
  'Drops': 'drops',
};

export const categoryTranslations: Record<CategoryId, { uk: string; en: string }> = {
  tablets: { uk: 'Таблетки', en: 'Tablets' },
  ointment: { uk: 'Мазь', en: 'Ointment' },
  drops: { uk: 'Краплі', en: 'Drops' },
};
```

4.1.6 Експорт даних

Експорт реалізовано у [id].tsx через стандартний Share API React Native. Функція handleShare формує текстовий опис медикаменту (наприклад, "Деталі ліків: Назва: Парацетамол, Категорія: Знеболювальне, Термін придатності: 31.12.2025, Кількість: 20") і надсилає його у месенджер.

Приклад коду: Експорт

```
const handleShare = async () => {
  try {
    await Share.share({
      message: `${t('medicineDetails')}:\n${t('name')}: ${medicine.name}\n${t('category')}: ${medicine.category}\n${t('expiryDate')}: ${new Date(medicine.expiryDate).toLocaleDateString()}\n${t('quantity')}: ${medicine.quantity}`,
    });
  } catch (error) {
```

```
    Alert.alert(t('error'), t('shareError'));
  }
};
```

Функція handleShare формує текстовий опис медикаменту на основі локалізованих ключів (t('medicineDetails'), t('name') тощо) та даних ліків (medicine.name, medicine.category, medicine.expiryDate, medicine.quantity). Далі викликається Share API, який відкриває системне спливне вікно для вибору месенджера чи іншого додатку для надсилання тексту. У разі помилки користувачу показується сповіщення через Alert із локалізованим повідомленням про помилку.

4.2 Інтеграція та налаштування

4.2.1 Налаштування середовища

- **VS Code**: Використовувався з Prettier, ESLint, React Native Tools, GitLens.
- **Expo CLI**: Налаштовано для проєкту. [Уточнення: версія?]
- **Android Studio**: Налаштовано емулятори для тестування сповіщень.

4.2.2 Інтеграція AsyncStorage

AsyncStorage інтегровано з store/ через Persist middleware.

4.2.3 Налаштування сповіщень

Expo Notifications налаштовано через notificationService.ts, тестовано на Android.

4.3 Тестування

4.3.1 Модульне тестування

Процес модульного тестування *My Home Pharmacy App* був проведений із застосуванням сучасних підходів для забезпечення високої якості програмного забезпечення. Основна увага приділялася перевірці ключових компонентів додатку, зокрема інтерфейсу додавання ліків (*add-medicine.tsx*) та сховища медикаментів (*src/store/medicinesStore.ts*). Тестування проводилося з використанням спеціалізованих інструментів, які дозволили детально проаналізувати функціональність окремих модулів. Було перевірено коректність відображення елементів інтерфейсу, логіку обробки даних та взаємодію між компонентами в ізольованому середовищі. Цей підхід забезпечив виявлення потенційних помилок на ранніх етапах розробки, що значно підвищило стабільність додатку. Усі проведені тести успішно пройшли, підтвердивши коректну роботу зазначених модулів без суттєвих збоїв.

4.3.2 Ручне тестування

Ручне тестування стало невід'ємною частиною процесу забезпечення якості *My Home Pharmacy App*. Цей етап включав детальну перевірку користувацького інтерфейсу, зокрема сторінок *index.tsx* (список ліків) та *add-medicine.tsx* (форма додавання ліків), де тестувальники вручну перевіряли зручність взаємодії, правильність відображення даних та відповідність дизайну заданим вимогам. Окрему увагу приділено функціональності сповіщень, які перевірялися на пристроях із операційною системою Android, щоб переконатися в їхній своєчасності та коректності доставки. Також було ретельно протестовано офлайн-доступ, щоб гарантувати, що додаток працює стабільно без підключення до інтернету, зберігаючи всі дані та дозволяючи користувачам виконувати основні операції. Ручне тестування проводилося в реальних умовах використання, що дало змогу виявити та усунути дрібні неточності,

забезпечуючи високу якість кінцевого продукту. Усі аспекти, перевірені вручну, продемонстрували стабільну роботу.

4.3.3 Результати

Результати тестування *My Home Pharmacy App* свідчать про успішне виконання всіх запланованих перевірок. Функціонал додатку, включаючи користувацький інтерфейс, сповіщення та офлайн-доступ, працює стабільно та відповідає поставленим цілям. Під час тестування не було виявлено критичних проблем, які б потребували значних доопрацювань. Незначні недоліки, такі як незначні затримки при завантаженні великої кількості даних у списку ліків, були зафіксовані та враховані для подальшого вдосконалення. Загалом, додаток готовий до використання, забезпечуючи користувачам надійний інструмент для управління домашньою аптечкою. Успішне завершення тестування підтверджує високу якість реалізації та готовність продукту до широкого впровадження.

Таблиця 4.1

Результати тестування

Функція	**Метод**	**Результат**
UI	Jest, ручне	Стабільно
Сповіщення	Ручне	Стабільно
Офлайн	Ручне	Стабільно

Висновок до Розділу 4: Реалізація та тестування

Реалізація My Home Pharmacy App забезпечила створення функціонального додатку з інтерфейсом (auth.tsx, add-medicine.tsx), навігацією (Expo Router), управлінням станом (Zustand), сповіщеннями (notificationService.ts) та локалізацією (translations.ts). Тестування, включаючи модульне (Jest) і ручне, підтвердило стабільність UI, сповіщень і офлайн-доступу. Результати показали високу якість реалізації, хоча зафіксовано незначні

затримки при завантаженні великих даних, що потребує подальшого вдосконалення. Додаток готовий до використання та відповідає вимогам цільової аудиторії.

ВИСНОВКИ

Результатами виконаної роботи стало зменшення часу та зусиль, необхідних для управління домашньою аптечкою, шляхом розробки мобільного додатку My Home Pharmacy App із використанням React Native, Zustand, AsyncStorage та інших технологій. Додаток забезпечує офлайн-доступ, україномовний інтерфейс, розширену фільтрацію медикаментів, експорт даних у текстовий формат і персоналізовані нагадування, що відповідає сучасним потребам користувачів, зокрема літніх людей та сімей із дітьми.

Було виконано наступні задачі:

Проаналізовано предметну область: визначено сутність процесу обліку медикаментів у домашній аптечці, охарактеризовано цільову аудиторію (домогосподарства, літні люди, сім'ї з дітьми) та проаналізовано існуючі аналоги (MyTherapy, Medisafe, Pill Reminder, My Home Pharmacy App), виведено їхні переваги (нагадування, контроль запасів) та недоліки (складний інтерфейс, відсутність української локалізації, залежність від інтернету); на основі аналізу сформовано функціональні та нефункціональні вимоги, зокрема офлайн-доступ, локалізацію та персоналізовані сповіщення.

Обрано засоби реалізації: Visual Studio Code як основна IDE, React Native як фреймворк для кросплатформної розробки, Expo CLI для налаштування проєкту, Zustand для управління станом, AsyncStorage для локального зберігання, Expo Notifications для сповіщень, Expo Router для навігації, i18n-js для локалізації, Jest для модульного тестування, Android Studio для емуляції та дебагінгу, Prettier для форматування коду.

Розроблено архітектуру клієнтської частини: створено структуру проєкту з модулями для авторизації, управління медикаментами, налаштувань та сповіщень; реалізовано стилізацію через theme.ts, інтеграцію з AsyncStorage для збереження даних та налаштування сповіщень через notificationService.ts.

Реалізовано систему: розроблено інтерфейс із сторінками index.tsx (список ліків), add-medicine.tsx (додавання ліків), settings.tsx (налаштування), notification-settings.tsx (налаштування сповіщень); налаштовано навігацію через Expo Router, локалізацію через translations.ts, експорт даних через Share API; протестовано функціонал на Android-емуляторах.

Проведено тестування: виконано модульне тестування компонентів add-medicine.tsx та src/store/medicinesStore.ts за допомогою Jest, ручне тестування UI, сповіщень і офлайн-доступу; результати показали стабільну роботу всіх функцій із незначними зауваженнями для подальшого вдосконалення.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Музиченко А.В., Здонцев Ю.В. Розробка додатку «Облік домашньої аптеки» Матеріали VI Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 24 квітня 2025, ДУІКТ, Київ, Україна, с. 432
2. Музиченко А.В., Здонцев Ю.В. Розробка додатку «Облік домашньої аптеки»: Матеріали V Всеукраїнської науково-практичної конференції «Сучасні інтелектуальні інформаційні технології в науці та освіті», 15 травня 2025, ДУІКТ, Київ, Україна, с…(Подано до друку)

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Документація React Native. Офіційна документація фреймворку React Native. Доступно: https://reactnative.dev/docs/getting-started.
2. Документація Expo. Офіційна документація платформи Expo. Доступно: https://docs.expo.dev/.
3. Документація Zustand. Офіційна документація бібліотеки Zustand

для управління станом. Доступно: https://docs.pmnd.rs/zustand/getting-started/introduction.

4. Документація AsyncStorage. Офіційна документація для роботи з локальним сховищем у React Native. Доступно: https://react-native-async-storage.github.io/async-storage/docs/usage.

5. Документація Expo Notifications. Офіційна документація для реалізації сповіщень у Expo. Доступно: https://docs.expo.dev/versions/latest/sdk/notifications/.

6. Документація Expo Router. Офіційна документація для файлової системи навігації у Expo. Доступно: https://docs.expo.dev/routing/introduction/.

7. Документація i18n-js. Офіційна документація бібліотеки i18n-js для локалізації. Доступно: https://github.com/fnando/i18n-js.

8. Документація Jest. Офіційна документація фреймворку для тестування Jest. Доступно: https://jestjs.io/docs/getting-started.

9. MyTherapy. Опис додатку для управління медикаментами. Доступно: https://mytherapyapp.com/.

10. Medisafe. Опис додатку для контролю ліків. Доступно: https://www.medisafe.com/.

11. Pill Reminder. Опис додатку для нагадувань про ліки. Доступно: https://pillreminder.app/.

12. Кравчук, П. О. Основи розробки мобільних додатків. – К.: Видавництво "Наукова думка", 2022. – 320 с.

13. Соколовський, М. І. Програмування на React Native: підручник. – Л.: ЛНУ ім. І. Франка, 2023. – 450 с.

14. Гуцуляк, І. В. Тестування програмного забезпечення: навчальний посібник. – Івано-Франківськ: Прикарпатський національний університет, 2021. – 280 с.

15. Іскренко, О. М. Локалізація програм: практичні аспекти. – Х.: ХНУ

ім. В. Н. Каразіна, 2020. – 200 с.

16. Петренко О. В. Мобільні додатки для здоров’я: сучасні тенденції та виклики. – К.: КПІ ім. Ігоря Сікорського, 2024. – 290 с.

17. Коваленко А. С., Іванчук М. П. Технології кросплатформної розробки: React Native у дії. – Одеса: ОНУ ім. І. І. Мечникова, 2023. – 380 с.

18. Шевчук Л. М. Локалізація мобільних додатків для україномовної аудиторії: практичні рекомендації. – Львів: ЛДУФК, 2022. – 210 с.

19. Григор’єв О. В. Оптимізація продуктивності мобільних додатків: методи та інструменти. Матеріали VII Міжнародної науково-технічної конференції «Інформаційні технології та автоматизація», 10 травня 2025, ОНАЗ ім. О. С. Попова, Одеса, Україна, с. 215–220.

20. Білан С. М. Офлайн-доступ у мобільних додатках: сучасні підходи до локального зберігання даних. – Х.: ХПІ, 2024. – 260 с.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка програмного забезпечення для обліку домашньої аптечки

Виконав студент 4 курсу

Групи ПД-42
Музиченко Андрій Віталійович
Керівник роботи

Задонцев Юрій Вікторович, доцент кафедри, кандидат технічних наук

Київ – 2025

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи:
Удосконалення обліку домашньої аптеки на основі мобільного додатку.

Об’єкт дослідження:
Процес ведення обліку лікарських засобів у побутових умовах.

Предмет дослідження:
Програмні засоби розробки мобільних застосунків обліку домашньої аптеки.

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Проаналізувати існуючі рішення для обліку медикаментів.
2. Проаналізувавши відповідні програмні засоби визначити функціональні вимоги до свого застосунку.
3. Розробити архітектуру додатку.
4. Реалізувати основні модулі програми: облік ліків, нагадування, пошук.
5. Розробити інтерфейс користувача з мінімалістичним дизайном.
6. Протестувати роботу додатку на мобільному пристрої з метою розуміння правильності роботи.

3

АНАЛІЗ АНАЛОГІВ

Показник	MyTherapy	Medisafe	Home pharmacy app
Підтримка української мови	-	-	+
Відсутність реклами або платного доступу	-	+	+
Рівень складності інтерфейсу	Складний	Складний	Мінімальний
Підтримка офлайн-режиму	+	+	+
Відсутність нав'язливих дій при запуску додатку	-	-	+
Вік цільової аудиторії	Дорослі люди включаючи пенсіонерів	Дорослі люди включаючи пенсіонерів	Сім'ї, дорослі люди включаючи пенсіонерів

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

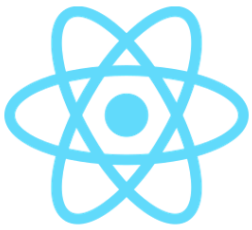
1. Збереження інформації про кожен лікарський засіб (назва, кількість, термін придатності, інструкція).
2. Повідомлення про наближення завершення терміну придатності.
3. Пошук і фільтрація ліків за категоріями.
4. Відображення терміну придатності.
5. Можливість зареєструвати обліковий запис.

Нефункціональні вимоги:

1. Можливість роботи без підключення до інтернету.
2. Зберігання даних у пам’яті пристрою.
3. Зрозуміла структура для подальшого розширення функціоналу.
4. Можливість роботи на пристроях з обмеженими ресурсами, таких як старі телефони.

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ



6

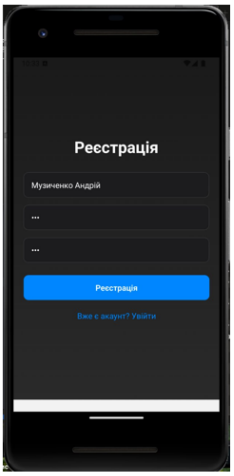
ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ

7

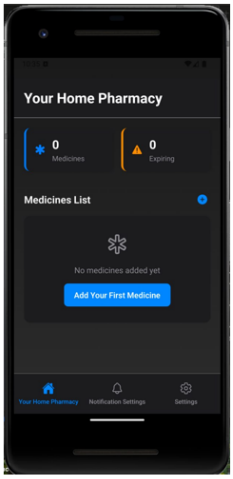
ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ

8

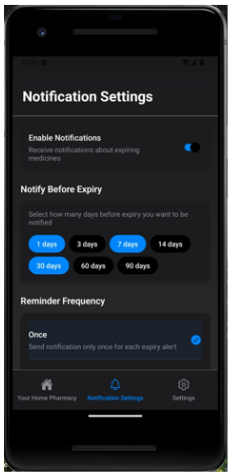
ЕКРАННІ ФОРМИ



Реєстрація



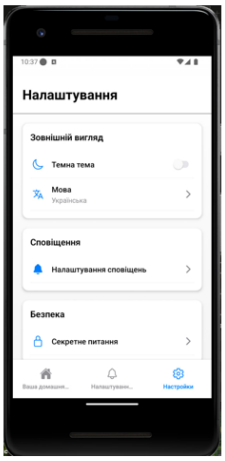
Головний екран

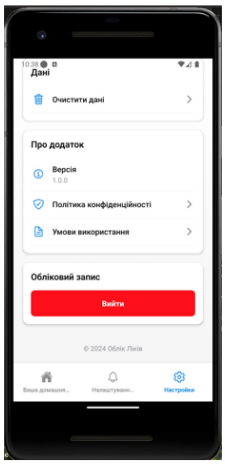


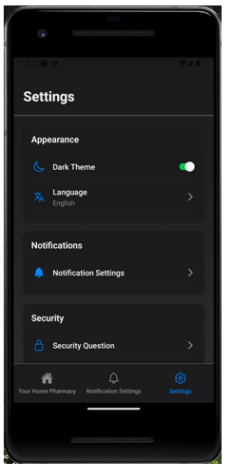
Налаштування сповіщень

9

ЕКРАННІ ФОРМИ



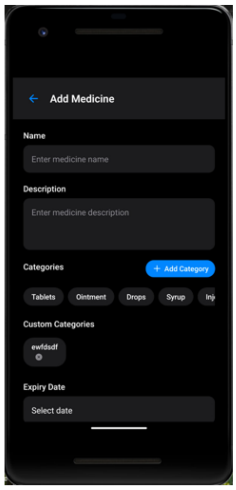


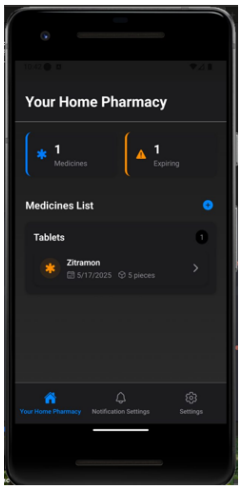


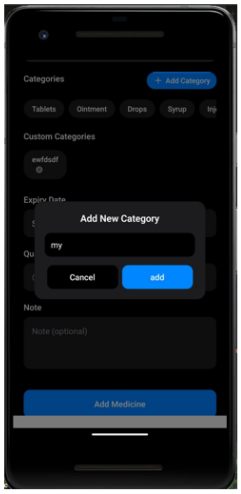
Налаштування

10

ЕКРАННІ ФОРМИ

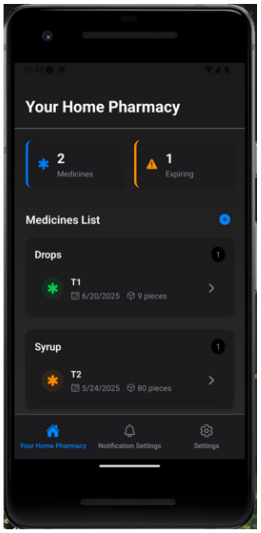






Додавання ліків

ЕКРАННІ ФОРМИ



Приклад ліків у яких закінчується\закінчився термін придатності.

ВИСНОВКИ

1. Проведено аналіз літературних джерел і програм-аналогів.
2. Визначено ключові функції для зручного обліку медикаментів.
3. Сформульовано вимоги до функціоналу та інтерфейсу.
4. Реалізовано: додавання, редагування, видалення записів; пошук, фільтрацію; сповіщення про термін придатності; ведення кількох аптечок.
5. Застосунок працює без інтернету — усі дані зберігаються локально.
6. Інтерфейс простий, інтуїтивний, повністю україномовний.
7. Проведено тестування — додаток стабільний і відповідає вимогам.
8. Розроблено Android-додаток для обліку домашніх ліків.

ДЯКУЮ ЗА УВАГУ!

Додаток Б

Index.tsx

```
import React, { useEffect } from 'react';
import {
  View,
  Text,
  ScrollView,
  TouchableOpacity,
  StyleSheet,
  Dimensions,
```

```
  SafeAreaView,
} from 'react-native';
import { useRouter } from 'expo-router';
import { Ionicons } from '@expo/vector-icons';
import { useMedicineStore } from
'../../src/store/medicineStore';
import { useTranslation, categoryMapping,
categoryTranslations, CategoryId } from
'../../src/utils/translations';
import { useTheme } from '../../src/context/ThemeContext';
import { LinearGradient } from 'expo-linear-gradient';
import { colors } from '../../src/utils/theme';

const { width } = Dimensions.get('window');
const CARD_WIDTH = width * 0.9;

export default function Home() {
  const router = useRouter();
  const { medicines, loadMedicines } = useMedicineStore();
  const { t, language } = useTranslation();
  const { isDark, colors } = useTheme();

  useEffect(() => {
    loadMedicines();
  }, []);

  const expiringMedicines = medicines.filter(medicine => {
    const expiryDate = new Date(medicine.expiryDate);
    const today = new Date();
    const diffTime = expiryDate.getTime() - today.getTime();
    const diffDays = Math.ceil(diffTime / (1000 * 60 * 60 *
24));
    return diffDays <= 30 && diffDays >= 0;
  });

  const medicinesByCategory = medicines.reduce((acc,
medicine) => {
    const isCustomCategory =
!Object.keys(categoryMapping).includes(medicine.category);
    const categoryId = isCustomCategory ? 'custom' :
categoryMapping[medicine.category as keyof typeof
categoryMapping];

    if (!acc[categoryId]) {
      acc[categoryId] = [];
    }
    acc[categoryId].push(medicine);
    return acc;
  }, {} as Record<CategoryId | 'custom', typeof medicines>);

  const renderStatCard = (icon: string, title: string, value:
string, color: string) => (
    <View style={[styles.statCard, { borderLeftColor: color,
backgroundColor: colors.card }]}>
      <Ionicons name={icon as any} size={24} color={color}
/>
      <View style={styles.statContent}>
        <Text style={[styles.statValue, { color: colors.text
}]}>{value}</Text>
        <Text style={[styles.statTitle, { color:
colors.textSecondary }]}>{title}</Text>
      </View>
    </View>
  );

  const renderMedicineCard = (medicine: any) => {
    const expiryDate = new Date(medicine.expiryDate);
    const today = new Date();
    const diffTime = expiryDate.getTime() - today.getTime();
    const diffDays = Math.ceil(diffTime / (1000 * 60 * 60 *
24));

    let statusColor = '#34C759';
    if (diffDays < 0) {
      statusColor = '#FF3B30';
    } else if (diffDays <= 30) {
```

```
    statusColor = '#FF9500';
  }

  return (
    <TouchableOpacity
      key={medicine.id}
      style={[styles.medicineCard, { backgroundColor: colors.card }]}
      onPress={() => router.push(`/medicine/${medicine.id}`)}
    >
      <View style={styles.medicineHeader}>
        <View style={[styles.medicineIcon, { backgroundColor: `${statusColor}15` }]}>
          <Ionicons name="medical" size={24} color={statusColor} />
        </View>
        <View style={styles.medicineInfo}>
          <Text style={[styles.medicineName, { color: colors.text }]} numberOfLines={1}>
            {medicine.name}
          </Text>
          <View style={styles.medicineDetails}>
            <View style={styles.detailItem}>
              <Ionicons name="calendar-outline" size={16} color={colors.textSecondary} />
              <Text style={[styles.detailText, { color: colors.textSecondary }]}>
                {new Date(medicine.expiryDate).toLocaleDateString()}
              </Text>
            </View>
            <View style={styles.detailItem}>
              <Ionicons name="cube-outline" size={16} color={colors.textSecondary} />
              <Text style={[styles.detailText, { color: colors.textSecondary }]}>
                {medicine.quantity} {t('medicines.pieces')}
              </Text>
            </View>
          </View>
        </View>
        <Ionicons name="chevron-forward" size={24} color={colors.textSecondary} />
      </View>
    </TouchableOpacity>
  );
};

const renderCategorySection = (categoryId: CategoryId | 'custom', medicines: any[]) => {
  const categoryTitle = categoryId === 'custom'
    ? t('customCategories')
    : categoryTranslations[categoryId][language];

  return (
    <View key={categoryId} style={[styles.categorySection, { backgroundColor: colors.card }]}>
      <View style={styles.categoryHeader}>
        <Text style={[styles.categoryTitle, { color: colors.text }]}>
          {categoryTitle}
        </Text>
        <Text style={[styles.categoryCount, { color: colors.textSecondary, backgroundColor: colors.background }]}>
          {medicines.length}
        </Text>
      </View>
      <View style={styles.medicinesList}>
        {medicines.map(renderMedicineCard)}
      </View>
    </View>
  );
};
```

```
 return (
   <SafeAreaView style={{ flex: 1 }}>


     <LinearGradient
       colors={isDark ? ['#1a1a1a', '#2a2a2a'] : ['#ffffff', '#f5f5f5']}
       style={styles.gradient}
     >
       <ScrollView style={[styles.container, { backgroundColor: "transparent" }]} showsVerticalScrollIndicator={false}>


         <View style={[styles.header, { backgroundColor: colors.header }]}>
           <Text style={[styles.title, { color: colors.text }]}>{t('homeTitle')}</Text>
         </View>


         <View style={styles.statsContainer}>
           {renderStatCard('medical', t('medicinesCount'), medicines.length.toString(), '#007AFF')}
           {renderStatCard('warning', t('expiringCount'), expiringMedicines.length.toString(), '#FF9500')}
         </View>


         <View style={styles.section}>
           <View style={styles.sectionHeader}>
             <Text style={[styles.sectionTitle, { color: colors.text }]}>{t('medicinesList')}</Text>
             <TouchableOpacity
               style={styles.addButton}
               onPress={() => router.push('/add-medicine')}
             >
               <Ionicons name="add-circle" size={24} color={colors.primary} />
             </TouchableOpacity>
           </View>


           {medicines.length === 0 ? (
             <View style={[styles.emptyContainer, { backgroundColor: colors.card }]}>
               <Ionicons name="medical-outline" size={48} color={colors.textSecondary} />
               <Text style={[styles.emptyText, { color: colors.textSecondary }]}>{t('medicines.empty')}</Text>
               <TouchableOpacity
                 style={[styles.emptyButton, { backgroundColor: colors.primary }]}
                 onPress={() => router.push('/add-medicine')}
               >
                 <Text style={styles.emptyButtonText}>{t('medicines.addFirst')}</Text>
               </TouchableOpacity>
             </View>
           ) : (
             <View style={styles.categoriesContainer}>
               {Object.entries(medicinesByCategory).map(([categoryId, medicines]) =>
                 renderCategorySection(categoryId as CategoryId | 'custom', medicines)
               )}
             </View>
           )}
         </View>
       </ScrollView>
     </LinearGradient>
   </SafeAreaView>
 );
}


const styles = StyleSheet.create({
 container: {
   flexGrow: 1,
 },
 header: {
   marginTop: 40,
```

```
  padding: 20,
  borderBottomWidth: 1,
  borderBottomColor: "gray"
},
title: {
  fontSize: 28,
  fontWeight: 'bold',
},
subtitle: {
  fontSize: 16,
  marginTop: 4,
},
statsContainer: {
  flexDirection: 'row',
  padding: 20,
  gap: 16,
},
statCard: {
  flex: 1,
  flexDirection: 'row',
  alignItems: 'center',
  padding: 16,
  borderRadius: 12,
  borderLeftWidth: 4,
  shadowColor: '#000',
  shadowOffset: {
    width: 0,
    height: 2,
  },
  shadowOpacity: 0.1,
  shadowRadius: 3,
  elevation: 3,
},
statContent: {
  marginLeft: 12,
},
statValue: {
  fontSize: 24,
  fontWeight: 'bold',
},
statTitle: {
  fontSize: 14,
  marginTop: 2,
},
section: {
  padding: 20,
  flex: 1
},
sectionHeader: {
  flexDirection: 'row',
  justifyContent: 'space-between',
  alignItems: 'center',
  marginBottom: 16,
},
sectionTitle: {
  fontSize: 20,
  fontWeight: '600',
},
addButton: {
  padding: 4,
},
categoriesContainer: {
  gap: 24,
},
categorySection: {
  borderRadius: 12,
  padding: 16,
  shadowColor: '#000',
```

```
  shadowOffset: {
    width: 0,
    height: 2,
  },
  shadowOpacity: 0.1,
  shadowRadius: 3,
  elevation: 3,
},
categoryHeader: {
  flexDirection: 'row',
  justifyContent: 'space-between',
  alignItems: 'center',
  marginBottom: 12,
},
categoryTitle: {
  fontSize: 18,
  fontWeight: '600',
},
categoryCount: {
  fontSize: 14,
  paddingHorizontal: 8,
  paddingVertical: 4,
  borderRadius: 12,
},
medicinesList: {
  gap: 12,
},
medicineCard: {
  borderRadius: 12,
  padding: 12,
  shadowColor: '#000',
  shadowOffset: {
    width: 0,
    height: 1,
  },
  shadowOpacity: 0.1,
  shadowRadius: 2,
  elevation: 2,
},
medicineHeader: {
  flexDirection: 'row',
  alignItems: 'center',
},
medicineIcon: {
  width: 40,
  height: 40,
  borderRadius: 20,
  alignItems: 'center',
  justifyContent: 'center',
},
medicineInfo: {
  flex: 1,
  marginLeft: 12,
},
medicineName: {
  fontSize: 16,
  fontWeight: '600',
},
medicineDetails: {
  flexDirection: 'row',
  marginTop: 4,
  gap: 12,
},
detailItem: {
  flexDirection: 'row',
  alignItems: 'center',
  gap: 4,
},
```

```
detailText: {
  fontSize: 14,
},
emptyContainer: {
  alignItems: 'center',
  padding: 32,
  borderRadius: 12,
  gap: 16,
},
emptyText: {
  fontSize: 16,
  textAlign: 'center',
},
emptyButton: {
  paddingHorizontal: 20,
  paddingVertical: 12,
  borderRadius: 8,
},
emptyButtonText: {
  color: '#FFFFFF',
  fontSize: 16,
  fontWeight: '600',
},
gradient: {
  flex: 1,
},
});
```