Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

«Мобільний застосунок для обліку домашньої аптечки з функцією нагадувань»

Назва теми

Рівень вищої освіти Перший (бакалаврський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр КвРІПЗ.2201117.01.22.ПЗ

Виконала студентка IV курсу, група ІПЗ-22-1 _____ Таїсія ШИШКА
Підпис Ім'я, ПРІЗВИЩЕ

Керівник канд. техн. наук, доцент _____ Оксана ЯШИНА
Науковий ступінь, вчене звання Підпис Ім'я, ПРІЗВИЩЕ

Нормоконтролер канд. техн. наук, доцент _____ Юрій ФОРКУН
Науковий ступінь, вчене звання Підпис Ім'я, ПРІЗВИЩЕ

До захисту допускаю:
Завідувач кафедри інженерії
програмного забезпечення _____ Леонід БЕДРАТЮК
Підпис Ім'я, ПРІЗВИЩЕ

1 червня 2026 р.

Хмельницький 2026

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій
Кафедра Інженерії програмного забезпечення
Рівень вищої освіти Перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри ІПЗ
_____ Л. П. Бедратюк
02 01 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Шишки Таїсії Віталіївни
Прізвище, ім'я, по батькові студента

1. Тема роботи «Мобільний застосунок для обліку домашньої аптечки з функцією нагадувань»

Керівник роботи Яшина Оксана Миколаївна, канд. техн. наук, доцент
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 20.01.2026 р. № 7
2. Строк подання студентом роботи на кафедру 01.06.2026 р.
3. Вихідні дані до роботи Матеріали переддипломної практики

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити): дослідження предметної області та постановка задачі; проєктування програмного забезпечення; програмна реалізація та тестування програмного забезпечення

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
1. Діаграма варіантів використання
2. Діаграма міжмодульних зв'язків
3. Діаграма послідовності

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Форкун Ю. В., доцент	25.05.26	25.05.26
Антиплагіат	Форкун Ю. В., доцент	01.05.26	25.05.26

7. Дата видачі завдання « 02 » січня 2026 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1 Ознайомлення з тематикою дипломного проєктування, визначення та узгодження індивідуальних тем кваліфікаційних робіт	01.12 – 31.12.2025	
2 Збір матеріалу за тематикою КвР; дослідження предметної області, в якій планується використання програмного забезпечення (ПЗ), визначення задач та вимог, розробка технічного завдання	01.01 – 20.02.2026	
3 Проєктування програмного забезпечення	21.02 – 20.03.2026	
4 Програмна реалізація з використанням відповідних засобів розробки. Тестування ПЗ	21.03 – 30.04.2026	
5 Написання вступу, загальних висновків, оформлення переліку джерел посилання та додатків. Оформлення пояснювальної записки КвР згідно вимог	01.05 – 25.05.2026	
6 Попередній захист КвР	Травень 2026	Згідно графіка
7 Перевірка КвР на плагіат, нормоконтроль, отримання відгуків, рецензій та інших супровідних документів. Брошурування (зшиття) пояснювальної записки	26.05 – 30.06.2026	
8 Здача КвР на кафедру; підготовка КвР для розміщення у репозитарії ХНУ; підготовка до захисту та захист КвР	З 01.06.2026	

Студентка _____ Таїсія ШИШКА
Підпис Ім'я, ПРІЗВИЩЕ

Керівник роботи _____ Оксана ЯШИНА
Підпис Ім'я, ПРІЗВИЩЕ

АНОТАЦІЯ

Тема кваліфікаційної роботи «Мобільний застосунок для обліку домашньої аптечки з функцією нагадувань».

Авторка роботи: Шишка Таїсія Віталіївна.

Керівник роботи: Яшина Оксана Миколаївна.

Пояснювальна записка: 68 с., 16 рис., 4 табл., 2 дод., 35 джерел.

Графічна частина: 3 креслення ф. А3.

АПТЕЧКА, КОНТРОЛЬ ТЕРМІНІВ ПРИДАТНОСТІ, ЛОКАЛЬНІ СПОВІЩЕННЯ, МЕДИКАМЕНТИ, МОБІЛЬНА РОЗРОБКА, МОБІЛЬНИЙ ЗАСТОСУНОК, НАГАДУВАННЯ, IOS, MVVM, SWIFT, SWIFTDATA, SWIFTUI.

Мета кваліфікаційної роботи: розроблення програмного забезпечення для обліку домашньої аптечки.

У кваліфікаційній роботі проведено аналіз предметної області та її інформаційного забезпечення, визначені функціональні та нефункціональні вимоги до програмної системи, розроблена загальна архітектура та структура застосунку.

Для реалізації програмного продукту використано мову програмування Swift, фреймворки SwiftUI та SwiftData.

Практичне значення результатів роботи полягає в тому, що використання мобільного застосунку дозволяє впровадити ефективне управління домашньою аптечкою, раціоналізувати витрати на охорону здоров'я шляхом запобігання надлишковим закупівлям препаратів та оптимізує пошук необхідних засобів у критичних ситуаціях.

27.05.2026

Дата

Підпис

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документа	Найменування документа	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	КвРІПЗ.2201117.01.22.ПЗ	Пояснювальна записка	82		
2	А4	КвРІПЗ.2201117.01.22.ПЗ	Завдання на кваліфіка-	1		
			ційну роботу			
3	А4	КвРІПЗ.2201117.01.22.ПЗ	Анотація	1		
			Графічні документи			
4	А4	КвРІПЗ.2201117.01.22.Е8	Діаграма варіантів вико-	1		
			ристання			
5	А4	КвРІПЗ.2201117.01.22.Е8	Діаграма міжмодульної	1		
			взаємодії			
6	А4	КвРІПЗ.2201117.01.22.Е8	Діаграма послідовності	1		

Змн.	Арк.	№ докум.	Підпис	Дата	КвРІПЗ.2201117.01.22.ВД			
Розроб.		Шишка Т.В.		25.05	Мобільний застосунок для обліку домашньої аптечки з функцією нагадувань	Літ.	Арк.	Аркушів
Перевір.		Яшина О.М.		25.05			1	1
Рецензн.		Нічепорук А.О.		28.05		ХНУ. ІПЗ-22-1		
Н. Контр.		Форкун Ю. В.		27.05				
Затверд.		Бедратюк Л.П.		29.05	Відомість документів			

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

КвР	-	Кваліфікаційна робота
ОС	-	Операційна система
ПЗ	-	Програмне забезпечення
DIP	-	Dependency inversion principle
ISP	-	Interface segregation principle
LSP	-	Liskov substitution principle
MVVM	-	Model-View-ViewModel
OCP	-	Open/closed principle
SRP	-	Single responsibility principle
UI	-	User interface
UML	-	Unified modeling language
UX	-	User experience

ВСТУП

Сучасний етап розвитку глобального інформаційного суспільства характеризується значною цифровізацією повсякденної діяльності людини та стрімким поширенням мобільних технологій. У цьому контексті розробка спеціалізованого програмного забезпечення для персональних мобільних пристроїв стає одним з чинників безпеки життєдіяльності людини [1]. Однією з найбільш критичних, проте часто ігнорованих проблем у сфері персонального менеджменту здоров'я, є контроль за якістю та придатністю домашніх запасів фармацевтичних препаратів.

Відомо, що після закінчення встановленого терміну придатності більшість лікарських засобів суттєво втрачають свою терапевтичну ефективність через деградацію активних компонентів [2]. Використання препаратів із порушеною стабільністю хімічного складу не лише не забезпечує очікуваного лікувального ефекту, а й створює приховано небезпечні умови, коли критично важлива терапія (наприклад, при серцево-судинних захворюваннях) виявляється недієвою [3]. Це зумовлює об'єктивну потребу в створенні інтелектуальних систем моніторингу, здатних запобігти випадкам вживання небезпечної медичної продукції.

Аналіз сучасного стану розробки програмного забезпечення свідчить про зміщення акценту від простих баз даних до проактивних систем взаємодії. Сучасні iOS-рішення все частіше інтегрують складні системи локальних push-сповіщень [4], що дозволяють підтримувати постійний зв'язок із користувачем без необхідності використання хмарних ресурсів, забезпечуючи високий рівень конфіденційності медичних даних. Окрім того, доречним буде застосування алгоритмів Deep Linking для швидкого переходу до критичних записів та візуального кодування статусів медичних препаратів, що дозволить значно знизити когнітивне навантаження на користувача та запобігти використанню небезпечних препаратів [5, 6].

Актуальність теми даної кваліфікаційної роботи обумовлена зростанням ринку медичних мобільних застосунків, потребою користувачів у систематизації домашніх аптечок та відсутності інструментів цифрового контролю за станом медикаментів у побутових умовах.

Обґрунтуванням необхідності даної розробки є відсутність серед існуючих рішень комплексного поєднання нативного інтерфейсу, автоматизованої валідації термінів придатності.

Галузь застосування розробки охоплює сферу індивідуального та сімейного менеджменту охорони здоров'я, а також може бути адаптована для потреб невеликих медичних пунктів та волонтерських організацій. Призначенням застосунку є автоматизація обліку лікарських засобів, оперативне інформування про завершення терміну їх придатності та забезпечення швидкого доступу до детальної інформації про кожен препарат через сучасні засоби системної інтеграції iOS.

Мета кваліфікаційної роботи: розроблення програмного забезпечення для обліку домашньої аптечки.

Завдання проєктування включають в себе:

– аналіз предметної області;

– аналіз наявного ПЗ;

– формування функціональних та нефункціональних вимог;

– визначення архітектурного підходу та потрібних патернів;

– проведення детального проєктування модулів;

– аналіз та вибір технологій і методів реалізації застосунку;

– реалізацію програмних модулів;

– проведення тестування застосунку;

– аналіз результатів роботи.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Змістовий аналіз предметної області, її структурних та функціональних особливостей

Предметною областю даного дослідження є процес управління персональними та сімейними медичними засобами, що зберігаються в домашніх умовах. Ефективний менеджмент домашньої аптечки є не лише питанням побутової зручності, а й критичним фактором для підтримки здоров'я та безпеки життєдіяльності людини. Проведений аналіз поточної ситуації свідчить про те, що користувачі стикаються з проблемою хаотичного накопичення медикаментів, що призводить до використання препаратів із терміном придатності, що сплив, неефективного витрачання коштів через дублювання вже наявних ліків та труднощів у швидкому пошуку необхідних засобів у критичних ситуаціях.

Традиційні методи контролю, такі як візуальний огляд або ведення паперових списків, демонструють низьку ефективність через значний вплив людського фактора. Основними недоліками неавтоматизованого підходу є складність регулярного моніторингу дат, що часто вказані дрібним шрифтом на пакуваннях, відсутність проактивного інформування про критичне наближення терміну придатності та інформаційний дефіцит у разі втрати оригінальної інструкції чи пакування препарату. Для розв'язання цих проблем необхідна розробка мобільного застосунку, який дозволить автоматизувати облік та впровадити інтелектуальну систему сповіщень.

Для чіткого визначення меж системи та ролі користувача у процесі функціонування застосунку було розроблено діаграму варіантів використання, яка візуалізує основні сервіси, що надаються програмним забезпеченням.

Відповідно до представленої моделі, взаємодія користувача із системою охоплює процеси автоматизованого додавання даних через сканування штрих-коду, що передбачає інтеграцію із зовнішніми API для отримання детального опису та зображення препарату, а також механізми ручного керування базою

даних. Особливе місце посідає функціонал опрацювання вхідних Push-сповіщень, які через механізм Deep Link забезпечують миттєву навігацію до конкретної картки медикаменту [7].

Процес трансформації вхідної інформації у вихідні керуючі сигнали та візуальні образи можна детально описати за допомогою діаграми потоків даних (Рисунок 1.1) першого рівня. Вона ілюструє шлях інформації від моменту первинного збору даних через камеру або інтерфейс введення до їх збереження та подальшого аналізу модулем моніторингу.

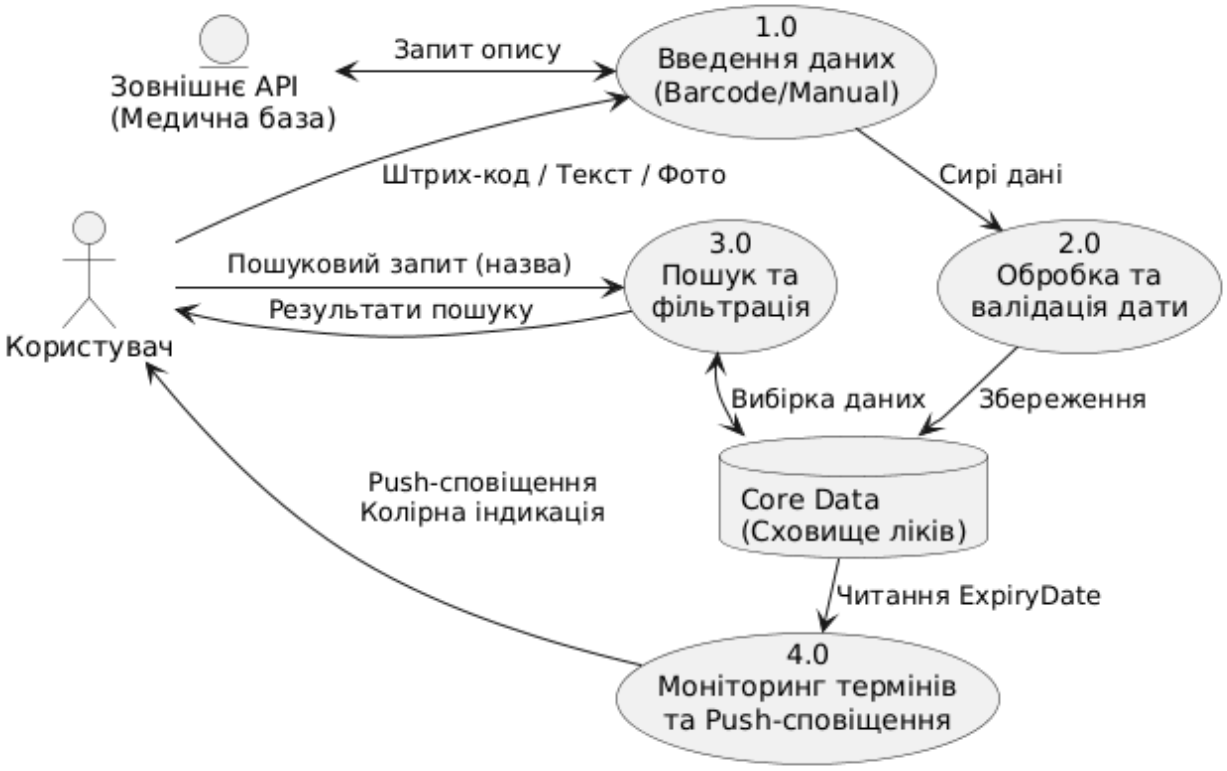


Рисунок 1.1 – Діаграма потоків даних

Вхідні реквізити препарату, такі як назва, кількість, місце зберігання та термін придатності, проходять етап валідації. Кінцевим результатом обробки є запис у локальному сховищі Core Data, дані з якого постійно аналізуються службою фонового моніторингу. Ця служба порівнює поточну системну дату з атрибутом терміну придатності, генеруючи на виході або локальні сповіщення для

операційної системи, або оновлені параметри візуалізації для інтерфейсу користувача.

Динамічний характер об’єктів у предметній області, що змінюється під впливом часу, зумовлює необхідність моделювання життєвого циклу запису про препарат. Для цього використано діаграму станів (Рисунок 1.2), яка описує логіку автоматичної зміни статусів та відповідну колірну індикацію в інтерфейсі застосунку.

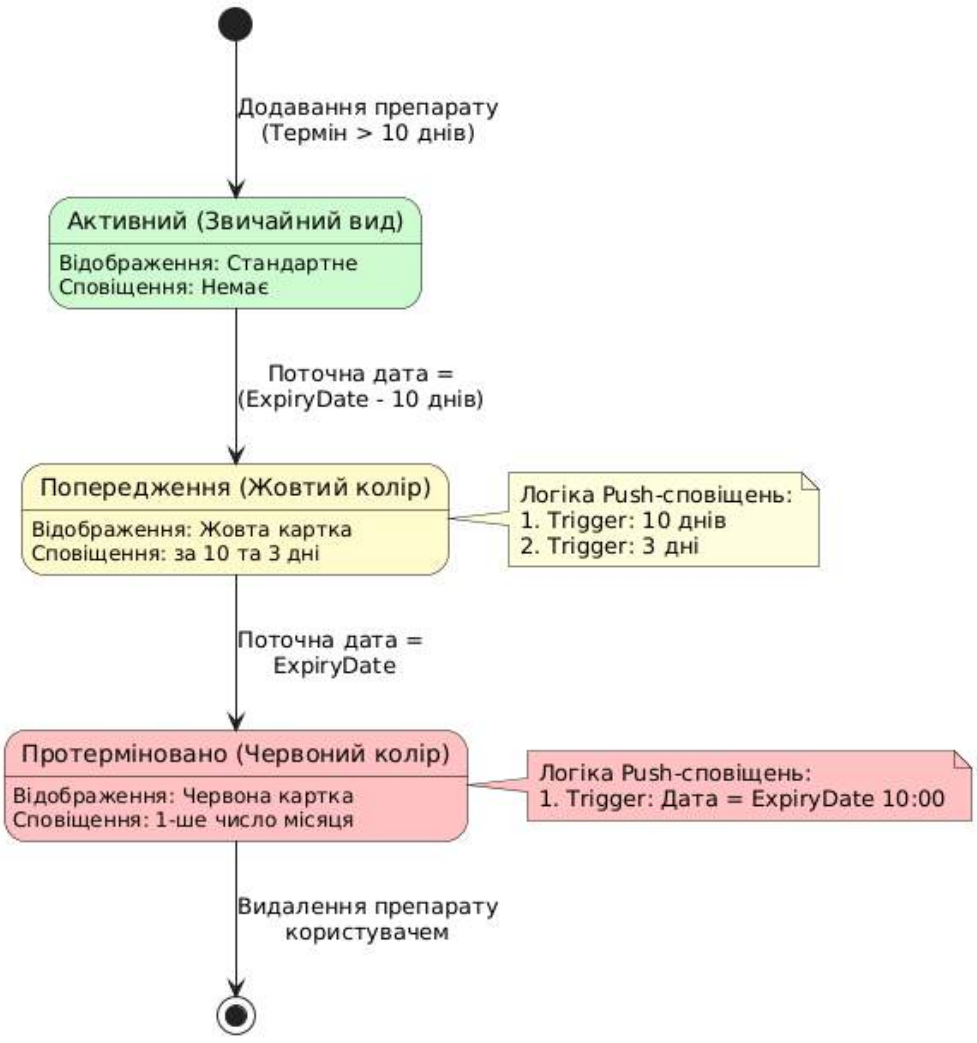


Рисунок 1.2 – Діаграма станів

Моделювання дозволяє виділити три стани: «Активний», коли препарат повністю придатний; «Попередження», що активується за 10 та 3 дні до дедлайну із відповідним зміненням кольору картки на жовтий; та «Протерміновано», що

настає першого числа місяця закінчення терміну та супроводжується червоною індикацією.

Враховуючи приватність медичної інформації, архітектура застосунку орієнтована на локальне зберігання даних на пристрої користувача.

Таким чином, проведений комплексний аналіз підтверджує доцільність розробки програмного продукту, який інтегрує базу даних, засоби комп'ютерного зору та систему планування завдань для ефективного контролю домашньої аптечки.

1.2 Аналіз наявного програмно-технічного забезпечення предметної області

На сьогодні сегмент Health & Fitness в App Store пропонує низку інструментів для обліку ліків, проте кожен із них має свої функціональні особливості та обмеження.

Одним із найбільш розповсюджених рішень є застосунок Medisafe Pill & Med Reminder від компанії Medisafe Inc [8]. Це комплексний продукт, призначений не лише для обліку наявності препаратів, а й для детального планування графіку прийому. Інтерфейс програми побудований навколо календаря прийому та детальних карток препаратів.

До переваг Medisafe можна віднести хмарну синхронізацію, можливість додавання довірених осіб та велику базу даних лікарських взаємодій. Однак основним недоліком є надмірна складність інтерфейсу для користувача (Рисунок 1.3).

Другим аналогом є застосунок MyTherapy, розроблений німецькою компанією Smartpatient GmbH [9].

Програмний продукт позиціонується як універсальний трекер здоров'я та менеджер прийому медикаментів. Призначення застосунку полягає у підвищенні прихильності пацієнта до лікування шляхом створення детального розкладу та

відстеження виконання медичних призначень. Інтерфейс програми орієнтований на щоденне використання та включає інтерактивні списки завдань, щоденник вимірювань та систему звітності (Рисунок 1.4).

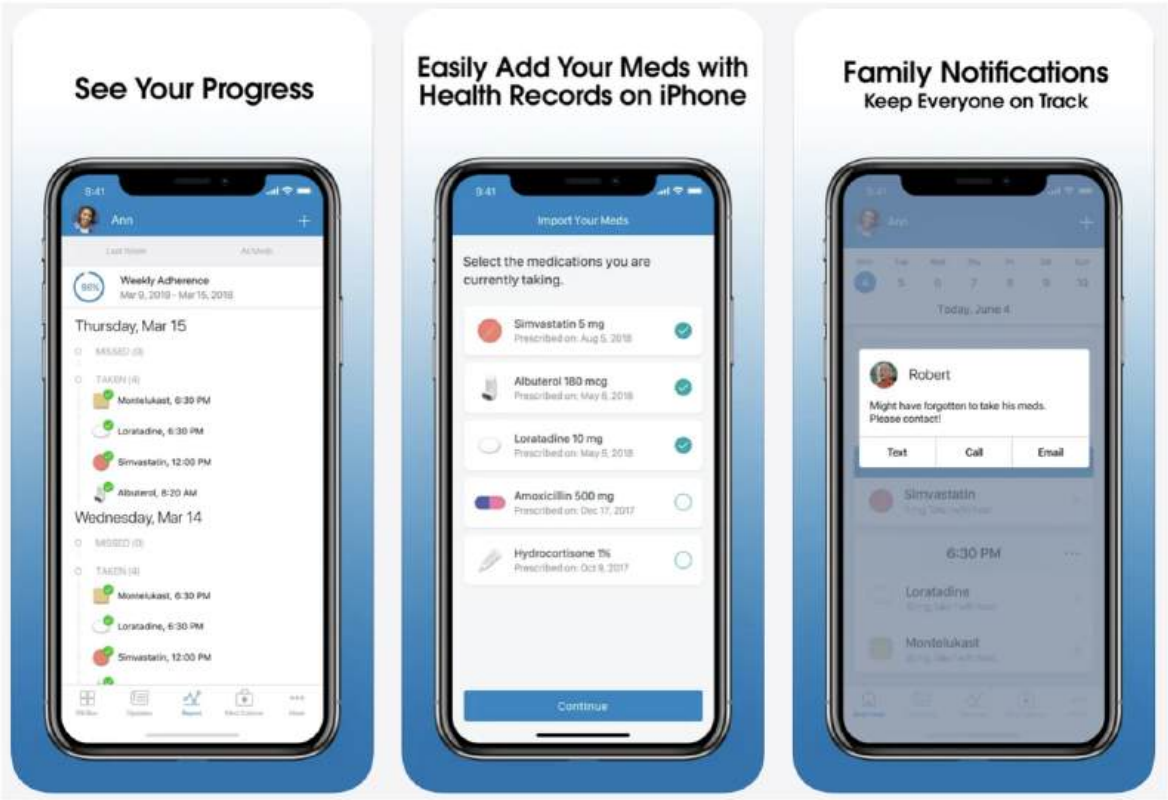


Рисунок 1.3 – Екрани застосунку Medisafe [10]

Додавання нових препаратів у MyTherapy реалізовано на високому рівні завдяки інтегрованій базі даних препаратів, доступних у обраній користувачем країні (у цьому переліку відсутня Україна), скануванню штрих-кодів (тільки для Німеччини) та/або ручному введенню. У цьому застосунку також наявна функція інвентаризації.

Безперечними перевагами продукту є його повна безкоштовність, відсутність реклами, можливість синхронізації з Apple Health та функція «командної роботи», що дозволяє родичам контролювати процес лікування користувача.

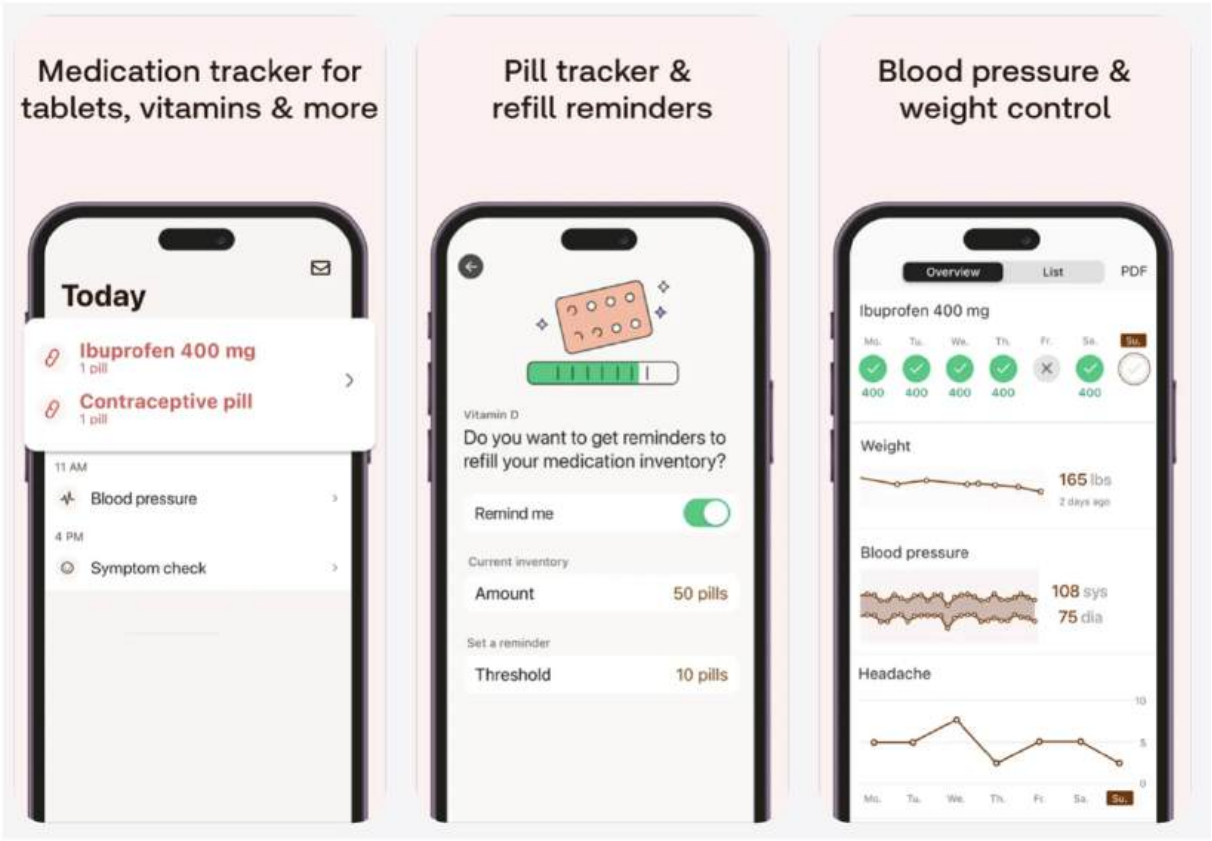


Рисунок 1.4 – Екрани застосунку MyTherapy [11]

Третім програмним продуктом для аналізу є «MedKit — Medication Reminders» від українського розробника [12]. Цей застосунок за своїм функціоналом є найбільш близьким до об’єкта розробки, оскільки надає можливість створення реєстру ліків.

Інтерфейс програми відповідає гайдлайнам Apple, пропонуючи мінімалістичний дизайн із використанням карток препаратів (Рисунок 1.5). Проте, детальний розбір виявив, що функціонал сповіщень про термін придатності у MedKit не є достатньо гнучким: він не передбачає багатокрокового алгоритму попереджень. На момент аналізу було виявлено, що застосунок надсилає лише одне сповіщення у день закінченння терміну придатності медикаменту.

Окрім того, поле «місце зберігання» є обмеженим у потенційному використанні, дозволяючи вказати лише поверхневу інформацію про розташування препарату (наприклад, «холодильник», а не «верхня полиця холодильнику»), що ускладнює навігацію у великих аптечках.

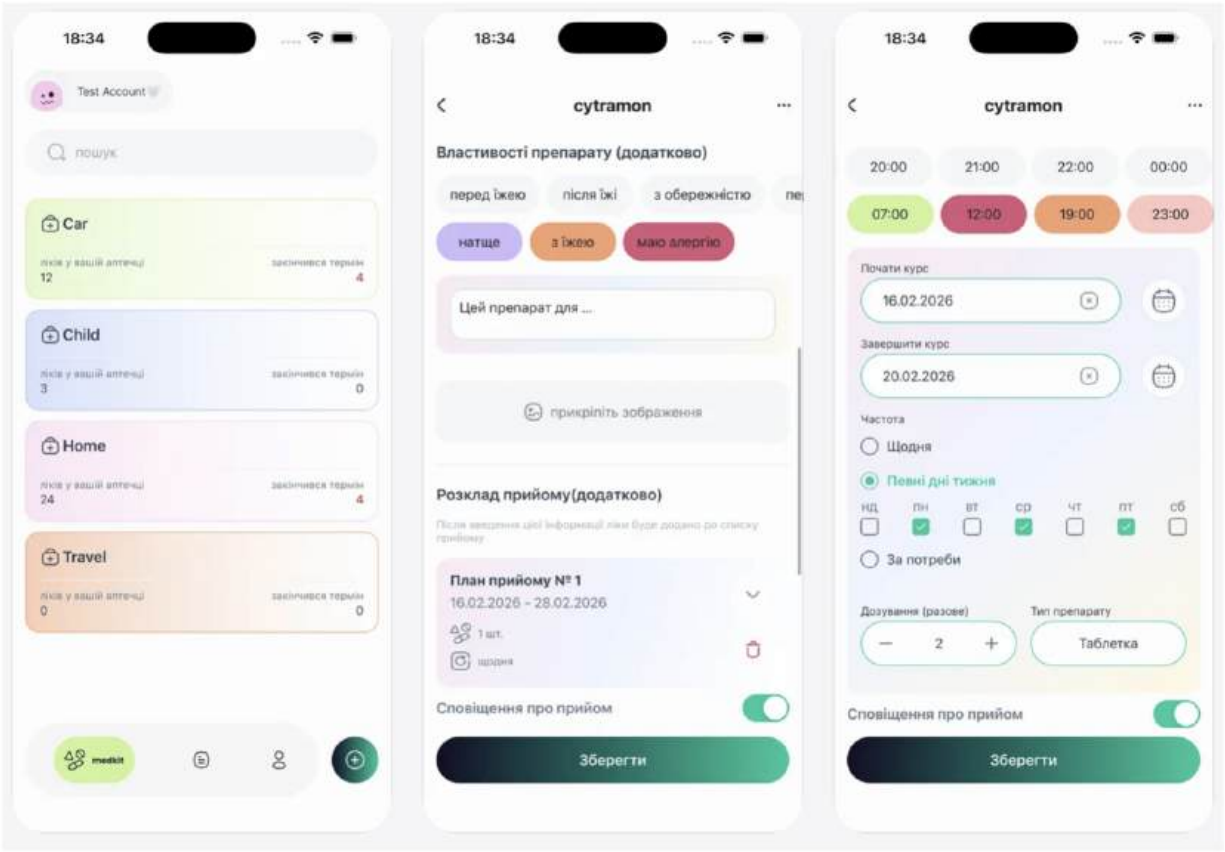


Рисунок 1.5 – Екрани застосунку MedKit [13]

Для систематизації результатів проведеного аналізу було детально розглянуто та структуровано ключові параметри обраних аналогів. Усі основні характеристики та функціональні особливості цих програмних продуктів зведені у порівняльну Таблицю 1.1.

Таблиця 1.1 Порівняльна характеристика аналогів програмного забезпечення

Критерії порівняння	Medisafe	MyTherapy	medKit	ПЗ, що розробляється у межах КвР
Платформа	iOS / Android	iOS	iOS	iOS

Кінець таблиці 1.1 Порівняльна характеристика аналогів програмного забезпечення

Критерії порівняння	Medisafe	MyTherapy	medKit	ПЗ, що розробляється у межах КвР
Облік місць зберігання	Частково	Ні	Так	Так
Складність інтерфейсу	Середня	Висока	Низька	Низька
Багаторівневі нагадування	Ні (фокус на прийом)	Ні (фокус на прийом)	Частково (разове нагадування про закінчення терміну придатності, фокус на прийом)	Так (10, 3, 0 днів)
Колірна індикація станів	Ні	Ні	Так (жовтий/ червоний)	Так (зелений/ жовтий/ червоний)
Можливість сканування штрих-кодів	Ні	Частково	Ні	Так

Аналіз наявного ПЗ показує, що більшість продуктів фокусуються або на часі прийому таблеток, або на простому переліку назв. Окрім того, не зважаючи на те, що medKit містить у собі функціонал нагадування про спливання термінів придатності, його реалізація не дозволяє користувачу заздалегідь дізнатися про подібну проблему та вжити необхідних заходів, оскільки сповіщення приходять безпосередньо у день закінчення терміну придатності.

Унікальність продукту, що розробляється, полягає у поєднанні зручного введення даних із чіткою системою візуального контролю термінів придатності. Найближчим аналогом за логікою обліку є medKit, проте у власному ПЗ буде значно вдосконалено систему сповіщень та запроваджено візуальну диференціацію станів придатності ліків для запобігання використанню небезпечних медикаментів.

Таким чином, дане ПЗ буде відповідати сучасним потребам ринку завдяки поєднанню вузької спеціалізації та простоти використання, що забезпечується використанням нативних технологій iOS.

1.3 Визначення функціональних та нефункціональних вимог до програмного забезпечення

Процес розробки мобільної системи обліку медикаментів розпочинається з етапу інженерії вимог, який є фундаментом для всієї подальшої технічної реалізації. На цій стадії проводиться декомпозиція загальної мети проєкту на набір чітко визначених функціональних можливостей та експлуатаційних обмежень, що дозволяє абстрагуватися від безпосереднього написання коду та зосередитися на побудові архітектурно правильної моделі поведінки.

В межах об’єктно-орієнтованої парадигми основним інструментом формалізації вимог виступає мова UML, яка забезпечує уніфікований спосіб візуалізації структурних та динамічних аспектів системи. Першочерговим завданням є ідентифікація акторів — зовнішніх сутностей, що взаємодіють із

програмним продуктом. У даному проєкті ключовим актором є Користувач, на якого покладено функції операційного управління аптечкою, та системний актор Зовнішнє API, що виступає вузлом у ланцюзі автоматизації введення даних. Опис цих сутностей наведено у Таблиці 1.2.

Таблиця 1.2 Актори та їх опис

Актор	Опис
Користувач	Власник домашньої аптечки. Основні цілі: інвентаризація ліків, пошук засобів, контроль термінів.
Зовнішнє API	Інформаційна система, що забезпечує ідентифікацію препаратів за штрих-кодом.

Особлива увага при аналізі вимог приділяється динаміці процесів, яка найкраще розкривається через діаграму послідовності. Ця модель дозволяє простежити життєвий цикл типового запиту в часовому розрізі, що є важливим для мобільних застосунків, де швидкість відгуку безпосередньо впливає на користувацький досвід.

На прикладі сценарію «Додавання препарату через сканування» візуалізується шлях даних: від моменту захоплення відеопотоку камерою за допомогою VisionKit [14], через розпізнавання штрих-коду та ініціацію мережевого запиту до сховища, до фінальної десеріалізації отриманого JSON-пакета та його відображення в інтерфейсі.

Моделювання такої взаємодії дозволяє заздалегідь виявити потенційні затримки або «вузькі місця» в архітектурі, наприклад, необхідність асинхронної обробки мережевих відповідей для запобігання блокуванню головного потоку інтерфейсу. Діаграму послідовності зображено на Рисунку 1.6.

Для детального опису внутрішньої логіки та алгоритмічної складової системи використовуються діаграми активності, які дозволяють візуалізувати

розгалужені процеси з урахуванням умовних переходів.

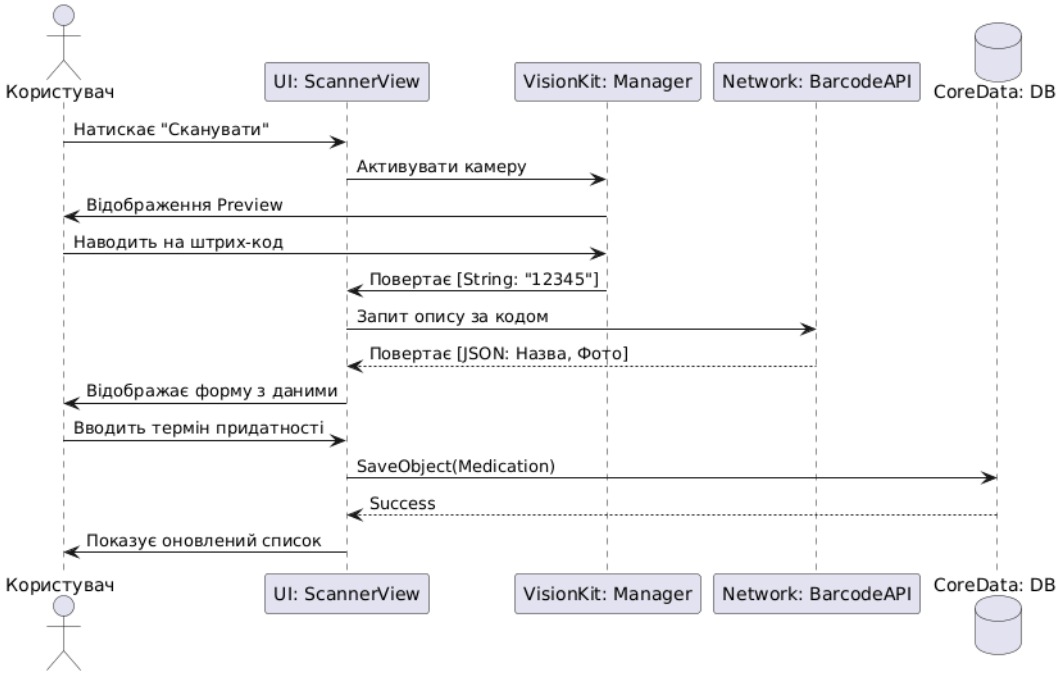


Рисунок 1.6 – Діаграма послідовності

Одним із фундаментальних процесів є алгоритм фонового моніторингу, який відповідає за безпеку використання ліків.

Система повинна безперервно аналізувати часову дельту між поточною датою та терміном придатності кожного об’єкта. Логіка передбачає наступну ієрархію перевірок: якщо термін вже минув, об’єкт переходить у критичний стан із негайною нотифікацією; якщо до завершення залишилося менше десяти днів, активується попереджувальний режим із повторними сповіщеннями за десять днів та за три дні до дедлайну (Рисунок 1.7).

Окремі діаграми активності описують процеси обробки мережевих помилок при роботі з API та алгоритми пошуку, у яких кожен введений символ ініціює новий цикл фільтрації масиву даних. Таке детальне моделювання алгоритмів дозволяє уникнути логічних помилок на етапі реалізації та забезпечує передбачуваність поведінки системи у нестандартних ситуаціях, таких як відсутність інтернет-з'єднання під час сканування або спроба введення некоректних дат. Відповідну діаграму послідовності зображено на Рисунку 1.8.

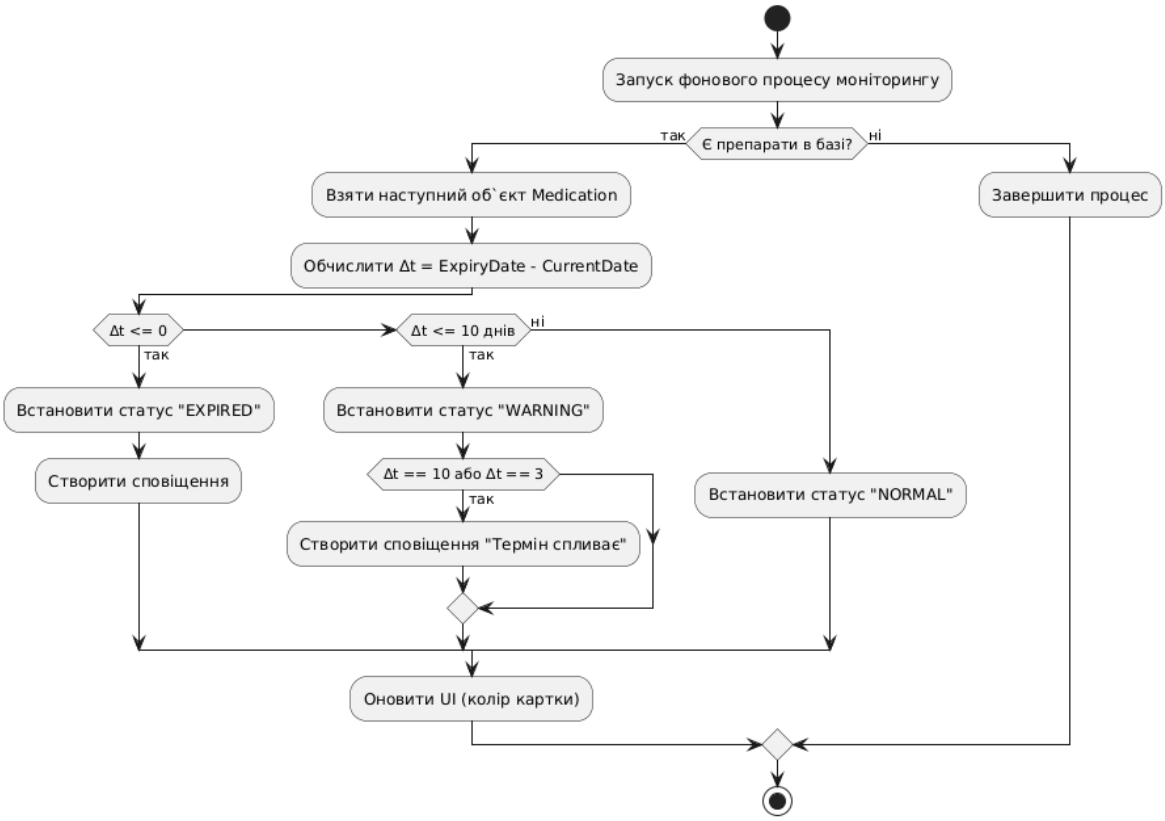


Рисунок 1.7 – Діаграма активності для процесу моніторингу термінів придатності медичних препаратів

Нефункціональні вимоги до програмного продукту визначають його якісні межі та технічний контекст функціонування. Програмне забезпечення повинно бути розроблене з урахуванням високих стандартів продуктивності, характерних для екосистеми iOS.

Вимоги до надійності включають обов’язкове локальне зберігання даних, що гарантує доступність інформації про аптечку в умовах відсутності мережі Інтернет. Швидкість ініціалізації модулів розпізнавання образів та ефективність запитів до локальної бази даних мають бути оптимізовані таким чином, щоб забезпечити плавність інтерфейсу при роботі з великими реєстрами препаратів.

Крім того, система повинна відповідати вимогам безпеки та приватності, зберігаючи всі медичні дані виключно в ізольованому середовищі застосунку без несанкціонованої передачі третім особам. Дотримання цих критеріїв дозволить

створити стабільне, відмовостійке та орієнтоване на користувача середовище, що повністю відповідає сучасним стандартам мобільної розробки Apple.

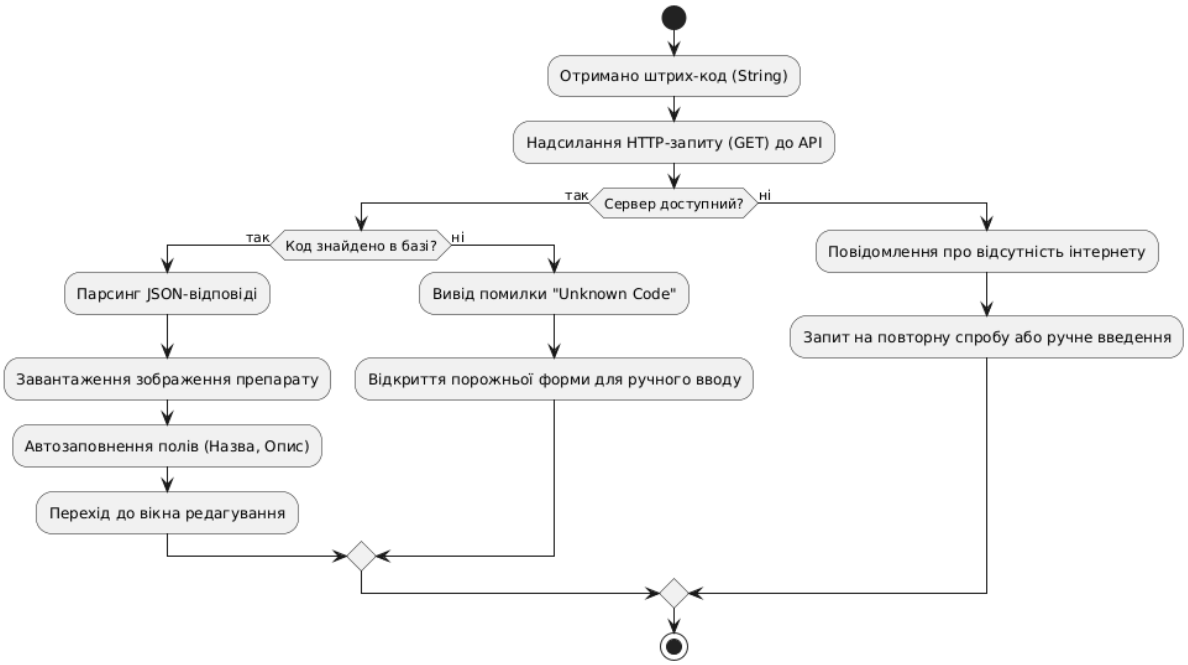


Рисунок 1.8 – Діаграма активності для процесу обробки запитів даних

Вимоги до користувацького інтерфейсу базуються на принципах доступності та інтуїтивності, що зафіксовано в Apple Human Interface Guidelines. Щонайменше, дизайн системи має підтримувати адаптивність до різних розмірів екранів та темного режиму оформлення [15].

Ключовою вимогою є впровадження системи візуальних індикаторів: це створює шар швидкої комунікації, що дозволяє користувачу оцінити стан всієї аптечки за декілька секунд. Ергономіка інтерфейсу має бути розрахована на мінімізацію кількості маніпуляцій для виконання базових завдань, забезпечуючи правило трьох кліків для доступу до будь-якої функції.

Окрім того, інтерфейс також має виконувати естетичну функцію, створюючи приємний та сучасний візуальний досвід для користувача.

Таким чином, за рахунок сукупність функціональних, алгоритмічних та якісних вимог, підкріплених UML-моделями, було сформовано повний

технологічний портрет майбутнього застосунку, забезпечуючи надійну основу для переходу до етапу безпосередньої розробки.

1.4 Висновки. Постановка задачі

У результаті проведеного дослідження було обґрунтовано необхідність автоматизації управління домашньою аптечкою, що зумовлено низькою ефективністю традиційних методів контролю та критичністю моніторингу термінів придатності ліків для безпеки користувача. Аналіз предметної області та існуючих програмних рішень, таких як Medisafe, MyTherapy та medKit, виявив функціональну прогалину на ринку: більшість аналогів орієнтовані на графік прийому препаратів, тоді як пропонована розробка фокусується на менеджменті наявності ліків та багаторівневому інформуванні.

Шляхом побудови комплексу UML-моделей, зокрема діаграм прецедентів, потоків даних, станів та послідовності, було детально формалізовано архітектуру майбутнього застосунку, визначено логіку взаємодії із зовнішніми API для сканування штрих-кодів та алгоритми фонового моніторингу станів препаратів.

Встановлено, що ключовою перевагою розробки є впровадження трирівневої системи візуальної індикації та автоматизованого обчислення дедлайнів, що у поєднанні з локальним зберігання даних та відповідністю стандартам Apple Human Interface Guidelines забезпечує створення надійного, інтуїтивно зрозумілого та високопродуктивного інструменту для підтримки здоров’я в умовах сучасної домашньої інфраструктури.

Таким чином, сформований перелік функціональних та нефункціональних вимог, підкріплений математичною та графічною логікою процесів, створює вичерпну технічну базу для подальшої програмної реалізації системи в середовищі iOS.

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Вибір типу архітектури та шаблонів проєктування

Проєктування архітектурного каркаса мобільного застосунку є фундаментальним етапом, що визначає стабільність та масштабованість системи. Для реалізації системи обліку аптечки було обрано нативну архітектуру, що базується на мові Swift та фреймворку SwiftUI. Основним архітектурним шаблоном обрано MVVM (Рисунок 2.1). Вибір цього патерну зумовлений його ефективністю при роботі з декларативними інтерфейсами, де стан синхронізується автоматично між моделлю та виглядом [16].

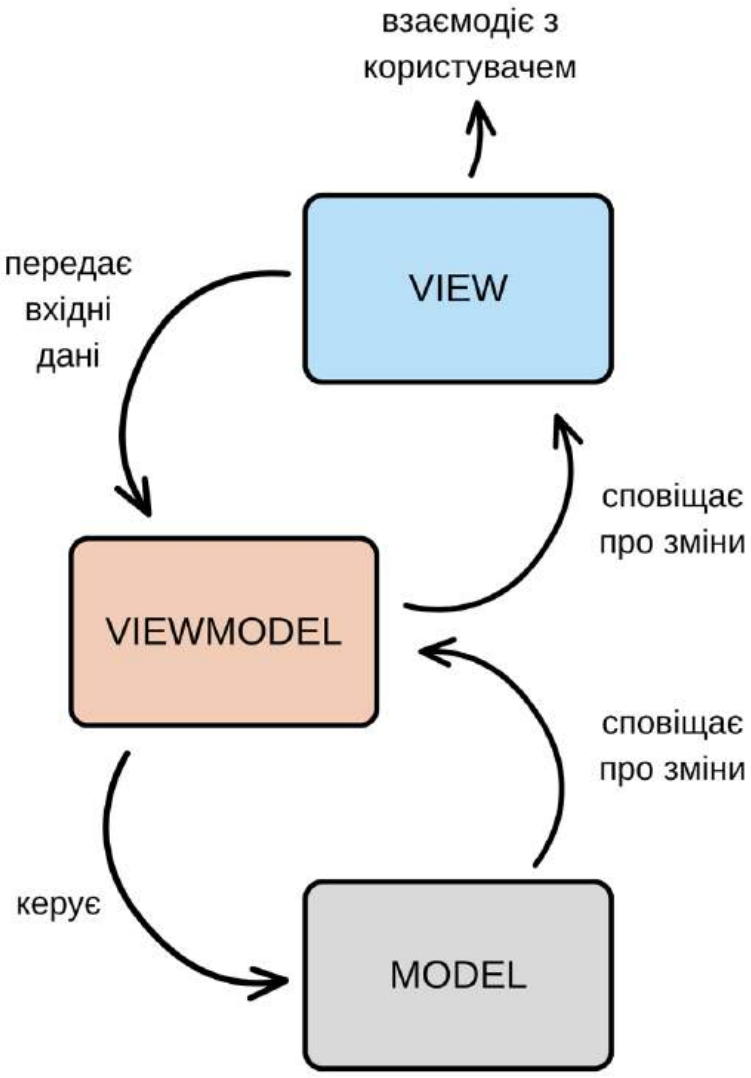


Рисунок 2.1 – Схема MVVM.

Важливим чинником вибору MVVM є позитивний вплив даної архітектури на показники продуктивності застосунку. Оскільки мобільні пристрої мають апаратні обмеження щодо енергоспоживання та обчислювальних ресурсів, застосування MVVM дозволяє оптимізувати використання центрального процесора та скоротити час виконання операцій порівняно з базовими архітектурними підходами [17].

Окрім покращення реактивності інтерфейсу, архітектурне рішення базується на принципах Clean Architecture. Функції роботи з камерою та планування сповіщень винесені в окремі сервісні модулі, що дозволяє ViewModel взаємодіяти з ними через абстракції, забезпечуючи високу модульність коду. Незважаючи на потенційне збільшення використання оперативної пам'яті через додаткові рівні абстракції та реактивні зв'язки, загальна продуктивність системи та швидкість відгуку на дії користувача (наприклад, при збереженні фото препарату) залишаються пріоритетними перевагами обраного підходу.

Архітектурне рішення базується на фундаментальних принципах проєктування компонентів SOLID, що виступають цілісною методологією формування «чистої архітектури» [18, с. 2]. Фундаментом системи є принцип єдиної відповідальності (SRP), який вимагає, щоб кожен програмний компонент, такий як сервіс для роботи з камерою чи менеджер сповіщень, був сфокусований на наданні послуг лише одному актору. Це дозволяє уникнути перевантаження класів зайвими функціями, що робить систему прозорою та простою в обслуговуванні. Наступний рівень надійності забезпечується принципом відкритості/закритості (OCP), який диктує таку структуру компонентів, за якої нові можливості додаються без модифікації вже протестованого коду, що гарантує стабільність наявних частин системи. Разом із цим, принцип підстановки Лісков (LSP) накладає суворі обмеження на ієрархію успадкування, гарантуючи, що будь-яка спеціалізована реалізація сховища даних буде повністю відповідати специфікації свого базового класу.

Для оптимізації використання ресурсів пристрою критично важливим є принцип розділення інтерфейсу (ISP), згідно з яким модулі не залежать від методів

або інтерфейсів, які вони не використовують. Такий підхід мінімізує витрати зусиль на розгортання та спрощує процес збірки проєкту. Завершальним етапом стає принцип інверсії залежностей (DIP), який розриває прямий зв'язок між високорівневою логікою інтерфейсу та низькорівневими постачальниками послуг. Замість прямої залежності вводяться проміжні інтерфейси, через які компоненти з'єднуються як незалежні модулі. Таким чином, поєднання MVVM та принципів SOLID забезпечує баланс між швидкістю розробки та технічною якістю кінцевого продукту.

Для забезпечення гнучкості, відмовостійкості та надійності архітектури, було прийнято рішення використовувати наступні шаблони проєктування: Репозиторій, Спостерігач, Одинак, Адаптер та Стратегія.

У контексті створення систем обліку з локальним збереженням даних, архітектурний шаблон Репозиторій виступає додатковим шаром абстракції, який ізолює бізнес-логіку програми від безпосередньої специфіки взаємодії з низькорівневою базою даних SQLite. Завдяки інкапсуляції логіки вибірки та збереження об'єктів у єдиному місці, репозиторій забезпечує уніфікований інтерфейс для решти компонентів системи. Це дозволяє мінімізувати жорстку залежність інтерфейсу від конкретної реалізації персистентного сховища, що спрощує потенційний перехід на альтернативні хмарні сервіси чи сторонні фреймворки в майбутньому без необхідності рефакторингу коду відображення.

Синхронізація станів у реальному часі між внутрішніми моделями даних та візуальними компонентами ефективно досягається за допомогою поведінкового патерну Спостерігач. Впровадження цього шаблону є технологічно доцільним для декларативних фреймворків, оскільки воно організує односпрямований потік даних, де графічний інтерфейс автоматично підписується на зміни у стані бізнес-моделей. Як тільки відбувається модифікація об'єктів, наприклад, додавання нового препарату або зміна його характеристик патерн ініціює автоматичне сповіщення всіх зацікавлених спостерігачів та реактивно оновлює екран користувача. Це виключає потребу в ручному управлінні життєвим циклом представлень та запобігає виникненню розсинхронізації даних.

Для централізованого управління загальносистемними службами, які повинні існувати в єдиному екземплярі протягом усього часу роботи застосунку, використовується структурний патерн Одинак. Його застосування для таких модулів, як координатор локальних сповіщень чи менеджер сховища, гарантує глобальну точку доступу до ресурсів та унеможливлює створення дубликатів об'єктів, що могло б призвести до конфліктів у системній черзі операційної системи чи перевитрати оперативної пам'яті.

Для подолання проблеми несумісності між сучасними декларативними інтерфейсами та застарілими апаратними компонентами платформи розробка спирається на патерн Adapter. Він дозволяє успішно загорнути старі системні контролери, наприклад, інтерфейс камери у сумісні зі SwiftUI структури, трансформуючи низькорівневі події пристрою у зрозумілі для нової архітектури потоки даних.

Динамічна адаптація поведінки програми та її візуального стилю реалізується за допомогою патерну Стратегія. Замість використання громіздких та важких для підтримки умовних розгалужень у коді інтерфейсу, цей шаблон дозволяє інкапсулювати алгоритми розрахунку часових інтервалів та стилізації в окремі взаємозамінні об'єкти. Залежно від поточного статусу часових мікшерів, система динамічно обирає відповідну стратегію поведінки. Поєднання цих п'яти патернів дозволяє досягти оптимального балансу між швидкодією нативного застосунку, безпекою локального збереження інформації та ергономікою взаємодії з користувачем.

2.2 Опис декомпозиції

Декомпозиція програмного забезпечення дозволяє розділити складну систему на керовані, незалежні модулі. Це спрощує паралельну розробку, тестування та покращує архітектурну цілісність ПЗ для будь-якої сфери [19].

Загальна декомпозиція системи, як і декомпозиція всіх її наступних

ієрархічних рівнів, проводиться на основі повного переліку функціональних та нефункціональних вимог, що були сформульовані на етапі аналізу предметної області. Такий підхід гарантує, що архітектура застосунку не лише забезпечуватиме виконання цільових завдань (облік ліків, фотофіксація, сповіщення), а й відповідатиме критеріям продуктивності, безпеки та надійності.

Головною ціллю загальної декомпозиції є визначення основних сутностей проєкту та детальний опис їх характеристик, що дозволяє перетворити абстрактні вимоги ТЗ у конкретну програмну структуру.

Сутність «Медикамент» є центральним елементом системи, що характеризується статичними атрибутами. Характеристикою цієї сутності є здатність до обчислення власного стану придатності залежно від системного часу.

Підсистема цифрової обробки зображень відповідає за реалізацію вимоги щодо візуального обліку препаратів. Реалізує методи стиснення даних та управління пам'яттю для збереження фотофіксації без втрати продуктивності.

Модуль обробки штрих-кодів реалізує автоматизоване розпізнавання та введення інформації. Його головною характеристикою є аналізу відеопотоку на пристрої, локальне декодування графічних ідентифікаторів у текстовий формат та передача отриманих даних до визначених полів, що мінімізує механічні помилки користувача.

Модуль управління станами даних забезпечує виконання вимог щодо локального збереження інформації. Її головною характеристикою є інкапсуляція логіки роботи зі SwiftData та гарантування цілісності об'єктів при завершенні життєвого циклу застосунку.

Сервіс сповіщень забезпечує інформування користувача завдяки автономному плануванню подій у системній черзі iOS без необхідності підключення до мережі Інтернет.

Проведення загальної декомпозиції дозволило зафіксувати межі кожної сутності та визначити їхню взаємодію, що є фундаментом для детального проєктування модулів та інтерфейсу користувача в наступних розділах.

Наступним етапом проєктування є модульна декомпозиція системи, яка

передбачає вертикальне розділення програмного забезпечення на незалежні функціональні одиниці та допоміжні інфраструктурні модулі. Такий підхід дозволяє ізолювати окремі функції застосунку, забезпечуючи їх автономність та спрощуючи процес подальшого розширення системи без ризику порушити цілісність архітектурної побудови. У межах розробки медичної аптечки реалізовано наступні ключові модулі.

Модуль навігації, що належить до шару представлення, виступає інфраструктурним фундаментом для координації переходів між компонентами системи. Він базується на принципах реактивного керування станами, забезпечуючи цілісність життєвого циклу об’єктів під час сеансу роботи та реалізуючи механізм глибокої навігації для безпосереднього доступу до деталей препарату з системних сповіщень.

У межах цього ж шару функціонують інтерфейс користувача та модуль візуального відгуку, які відповідають за інтерактивну взаємодію та відображення динамічних статусів придатності ліків. Ці компоненти забезпечують візуалізацію аналітичних даних, отриманих від нижчих рівнів, та інформують користувача про критичні зміни станів об'єктів моніторингу, синхронізуючи зміни в реальному часі та обробляючи стани помилок при некоректному введенні даних.

Шар бізнес-логіки представлено модулем сценаріїв роботи, який є операційним центром системи та організатором взаємодії між іншими компонентами. Він ініціює перевірку вводу через модуль валідації, що здійснює синтаксичний та логічний аналіз текстових рядків дати, контролюючи відповідність введених символів заданим параметрам.

Паралельно модуль сценаріїв взаємодіє з модулем захоплення зображень, який через системні інтерфейси AVFoundation забезпечує ініціалізацію камери та фотофіксацію упаковки ліків [20]. Даний модуль повністю інкапсулює логіку попередньої обробки зображення, надаючи інтерфейсу готовий масив даних для візуального підтвердження препарату.

За результатами обробки даних модуль сповіщень реалізує функцію моніторингу та планування черги локальних push-повідомлень, взаємодіючи з

системними службами iOS для автономного інформування користувача про завершення термінів придатності навіть без активного підключення до мережі.

Для забезпечення автоматизованого введення даних із сервісами захоплення зображень також інтегрується модуль обробки штрих-кодів. Основне призначення цього модуля полягає в аналізі відеопотоку або готових знімків з камери пристрою, розпізнаванні графічних ідентифікаторів (таких як штрих-коди форматів EAN-13, UPC або QR-коди) та їх декодуванні у текстовий числовий формат. Це дозволяє користувачеві уникати ручного введення назв лікарських засобів, значно прискорюючи процес інвентаризації та мінімізуючи ризик механічних помилок при заповненні картки медикаменту.

Архітектурно модуль може базуватися на нативному фреймворку Vision від Apple [21] або спеціалізованих інструментах AVCaptureMetadataOutput [22], що дозволяє виконувати розпізнавання безпосередньо на пристрої без надсилання графічних даних на зовнішні сервери. Модуль функціонує за подійною моделлю: при виявленні валідного коду в полі зору камери, він зупиняє аналіз потоку та ініціює запит до внутрішнього словника або інтегрованої бази даних для автозаповнення полів форми. Завдяки високій оптимізації алгоритмів розпізнавання, модуль стабільно працює навіть за умов недостатнього освітлення або часткового пошкодження упаковки препарату.

Наступним є шар даних, який складається з модулів керування даними та моделей даних, які в сукупності забезпечують локальну персистентність на основі технології SwiftData [23]. Модуль керування даними інкапсулює складність CRUD-операцій та виконує мапінг об’єктів у реляційну структуру локальної бази даних. Використання SwiftData дозволяє ізолювати складність операцій запису та зчитування, надаючи іншим компонентам зручний інтерфейс для доступу до списку ліків. Це гарантує швидке завантаження даних при запуску застосунку та забезпечує конфіденційність інформації, оскільки вона не залишає межі локального сховища смартфона. Така архітектурна організація забезпечує чіткий розподіл відповідальності та високу стійкість системи до змін на кожному з етапів обробки даних.

Компонентна декомпозиція графічного інтерфейсу реалізована на рівні SwiftUI-елементів. Складні екрани розділені на базові переуживані блоки, такі як уніфіковані картки ліків, спеціалізовані текстові поля з маскою введення та кнопки керування. Таке рішення забезпечує візуальну цілісність застосунку, спрощує модифікацію дизайну та підвищує стабільність роботи інтерфейсу.

2.3 Опис залежностей

Опис залежностей є етапом проєктування, який дозволяє формалізувати взаємодію між компонентами системи, визначити ієрархію модулів та встановити шлях проходження потоків даних. У контексті розробки мобільного застосунку для обліку медикаментів правильне управління залежностями гарантує стабільність роботи пристрою в умовах обмежених ресурсів, запобігає виникненню критичних помилок під час роботи з локальною пам'яттю та забезпечує високу тестованість коду.

Архітектура застосунку базується на патерні MVVM, інтегрованому з принципами «чистої архітектури», що диктує суворе правило спрямування залежностей: від зовнішніх рівнів інтерфейсу та баз даних до внутрішніх рівнів абстракції та бізнес-моделей. Це означає, що компоненти, відповідальні за збереження фото чи планування сповіщень, можуть залежати від логіки опису препарату, проте ядро системи ніколи не містить знань про технічні особливості фреймворків SwiftData чи AVFoundation.

Шар користувацького інтерфейсу, реалізований за допомогою SwiftUI, функціонує як пасивний механізм відображення станів. Екрани списку ліків та форми додавання мають жорстку залежність виключно від відповідних класів ViewModel, які виступають джерелом істини для візуальних елементів. Графічний інтерфейс лише транслює дії користувача, такі як введення текстової дати або натискання кнопки камери, до рівня ViewModel, і підписується на оновлення станів для автоматичного рендерингу індикаторів придатності.

Класи ViewModel, у свою чергу, виступають інтелектуальним мостом, що зв’язує інтерфейс із рівнем даних. Вони абстраговані від безпосередньої реалізації сховищ і залежать лише від протоколів, що описують методи збереження та отримання інформації. Наприклад, логіка додавання нового препарату має залежність від абстрактного контракту сервісу сповіщень та репозиторію даних. Завдяки такій структурі досягається низька зв'язність коду, що дозволяє замінити механізм локального збереження на хмарну синхронізацію без необхідності вносити жодної зміни у код управління станом екрана.

Ядром системи виступає шар предметної області, представлений моделями SwiftData. Цей рівень є максимально незалежним і містить у собі опис сутності медикаменту, включаючи логіку розрахунку статусів придатності. Саме ці моделі диктують правила того, як саме атрибути «Назва», «Дата» та «Фото» повинні бути структуровані для подальшої обробки. Рівень даних реалізує ці моделі через механізм персистентності, де локальні репозиторії мають пряму залежність від контейнера бази даних для фіксації змін у SQLite.

Окремої уваги заслуговує інтеграція сервісних модулів камери та сповіщень. Модуль роботи з медіа залежить від системного API iOS, проте його взаємодія з основним кодом ініціюється виключно через інтерфейс провайдера даних. Перед тим як фотографія упаковки потрапляє до сховища, ViewModel звертається до сервісу обробки для оптимізації розміру файлу. Аналогічно, модуль сповіщень залежить від фреймворку UserNotifications, але отримує необхідні часові мітки від бізнес-логіки вже після успішної валідації дати.

Для автоматизації управління життєвим циклом об'єктів у системі застосовано принцип інверсії управління. На відміну від складних зовнішніх фреймворків, у нативному середовищі Swift це реалізовано через механізм впровадження залежностей у конструктори та використання контейнерів середовища (Environment). Це дозволяє системі динамічно надавати екземпляр бази даних або сервісу камери тим компонентам, які їх потребують, без створення жорстких зв'язків між класами.

Завдяки такому підходу система поділена на ізольовані напрямки постачання залежностей. Системний модуль забезпечує глобальний доступ до контексту SwiftData, гарантуючи єдину точку входу для всіх операцій з базою даних. Сервісний модуль керує ініціалізацією камери та налаштуванням дозволів операційної системи. Модуль сповіщень забезпечує зв'язок між бізнес-правилами придатності ліків та системною чергою нагадувань. Зрештою, навігаційний модуль координує переходи між екранами, передаючи необхідні залежності від батьківських компонентів до дочірніх, що забезпечує цілісність архітектурної побудови застосунку.

2.4 Опис інтерфейсу модулів

Інтерфейси модулів забезпечують взаємодію між компонентами системи. Кожен інтерфейс розроблений таким чином, щоб надати зовнішнім об'єктам лише необхідні методи для маніпуляції даними, приховуючи внутрішню реалізацію та захищаючи приватні змінні від некоректних змін.

Для модуля інтерфейсу користувача було визначено, що програмні інтерфейси повинні забезпечувати безперебійну та безпечну взаємодію між візуальними компонентами та шаром бізнес-логіки системи. Властивості medications та filteredResults дозволяють автоматизовано отримувати актуальні дані з репозиторія через механізми @Query та виконувати миттєву фільтрацію за критеріями пошуку, не порушуючи при цьому цілісності відображення списку та зберігаючи високу швидкість відгуку інтерфейсу.

Метод navigationDestination(for:)забезпечує безпечну маршрутизацію до детальної інформації про препарат на основі унікального ідентифікатора UUID, гарантуючи точність передачі контексту між екранами, тоді як комплексна функція deleteMedication()реалізує багатокроковий сценарій очищення системи, що включає рекурсивне видалення графічних файлів через StorageService, скасування черги запланованих повідомлень у NotificationService та остаточне

вилучення об'єкта з контексту SwiftData.

У межах компонента додавання нових записів інтерфейс взаємодії побудований на методах saveMedication() та capturePhoto(), які виступають тригерами для активації відповідних сценаріїв у ViewModel та сервісах обробки медіа-контенту. Конструктор ініціалізації AddMedicineView(modelContext:) виконує критичну роль у впровадженні залежностей, що дозволяє коректно передавати контекст бази даних у логіку збереження та забезпечувати атомарність операцій запису.

Внутрішні процеси, такі як пряме керування сесією захоплення через CameraService, низькорівневі маніпуляції з буферами зображень або алгоритми синхронізації контексту, залишаються повністю прихованими від кінцевого представлення. Клієнтський код отримує лише високорівневі результати обробки, такі як сигнали про успішне завершення транзакції або необхідність візуалізації помилок валідації через інтерфейсний метод alert(), що відповідає принципам інкапсуляції та чистої архітектури.

Для модуля обробки штрих-кодів повинен бути розроблений програмний інтерфейс, який забезпечує передачу розпізнаних даних до компонентів створення препарату та керує станом процесу сканування. Центральним методом інтерфейсу є startBarcodeDetection(in:), який приймає шар попереднього перегляду камери та запускає сесію аналізу метаданих у реальному часі. Зворотний зв'язок із клієнтським кодом (наприклад, з AddMedicineViewModel) здійснюється через властивість-замикання onBarcodeDetected, яка асинхронно передає розпізнаний рядок цифр як результат успішного сканування.

Додатковий інтерфейс керування життєвим циклом процесу представлений методами pauseScanning() та resumeScanning(). Метод pauseScanning() тимчасово блокує обробку нових кадрів після першого успішного зчитування, що запобігає повторному спрацюванню тригера, тоді як resumeScanning() відновлює готовність модуля до розпізнавання. Внутрішня конфігурація, включаючи масив підтримуваних типів об'єктів (metadataObjectTypes), алгоритми фільтрації шумів зображення та логіка взаємодії з низькорівневим рушієм VNDetectBarcodesRequest,

залишається повністю інкапсульованою всередині сервісу, надаючи зовнішнім модулям лише очищений результат декодування.

Інтерфейс взаємодії з даним модулем реалізований через набір обчислювальних методів-гетерів та конструкторів візуальних компонентів. В межах перерахування ExpiryStatus визначено методи label(), color() та iconName(), які повертають локалізований текст, колірну схему та ідентифікатор системної піктограми відповідно до логічного стану препарату. Це дозволяє зовнішнім клієнтам отримувати повний пакет візуальних інструкцій через єдиний запит до об'єкта стану, забезпечуючи сувору відповідність між категорією придатності та її графічним втіленням.

Публічний інтерфейс графічного елемента StatusBadge базується на методі ініціалізації, що приймає параметр статусу для автоматичної конфігурації контейнера. Декларативний метод body інкапсулює логіку компонування стека, приховуючи деталі реалізації графічних ефектів та стилізації контурів. Таким чином, клієнтські модулі отримують готовий результат обчислень у вигляді скомпонованого компонента, що дозволяє оперувати високорівневими абстракціями без втручання у внутрішню логіку візуального відгуку

Інтерфейс модуля сценаріїв роботи має забезпечувати безпечне керування бізнес-об'єктами та надавати методи для трансформації даних згідно з потребами користувача. Ключовий метод filterMedications(_:) дозволяє отримувати відфільтрований масив об'єктів на основі введеного пошукового запиту, реалізуючи алгоритми нечутливого до регістру порівняння назв. Метод ініціалізації init(modelContext:) забезпечує впровадження контексту бази даних, що є необхідною умовою для виконання подальших транзакцій запису та зчитування.

Інтерфейс взаємодії з процесом реєстрації нових препаратів реалізується через метод saveMedication(), який ініціює ланцюжок валідації введених характеристик та виконує безпосереднє збереження об'єкта в репозиторій. Додатково модуль надає інтерфейсні точки доступу для обробки медіа-контенту через властивості захоплених зображень та методи їх асоціації з конкретними записами. При цьому внутрішня логіка маскування тексту, обробки помилок та

прямої взаємодії з системними сервісами сповіщень залишається прихованою, надаючи клієнтським View лише кінцевий результат операцій або сигнали про необхідність візуалізації станів помилок.

Модуль валідації повинен надавати високорівневі інструменти для швидкої перевірки та трансформації даних, приховуючи при цьому низькорівневі налаштування регулярних виразів та параметрів форматування. Ключовим функціональним інтерфейсом є метод validateAndParse(_:), який виступає як точка входу для сирих рядкових даних. Даний метод послідовно виконує очищення тексту від зайвих символів, перевірку за регулярним виразом dateRegex на відповідність масці «дд.мм.рррр» та фінальну конвертацію у системний об'єкт Date за допомогою форматера dateFormatter із вимкненим режимом автоматичного виправлення (lenient = false).

Додатковий інтерфейс логічного контролю реалізований через метод isReasonableDate(_:), який перевіряє отриманий об’єкт дати на відповідність встановленим бізнес-правилам щодо часових меж. Метод повертає булеве значення, підтверджуючи, чи знаходиться дата в діапазоні, прийнятному для обліку медикаментів (наприклад, у межах 20 років у минулому та 50 років у майбутньому чи іншого часового інтервалу), що дозволяє відсікати випадкові помилки вводу року. Внутрішні константи, такі як ідентифікатор локалі uk_UA або специфікації календаря, залишаються повністю прихованими від зовнішніх клієнтів, надаючи їм лише кінцевий результат у вигляді валідного об’єкта або сигналу про помилку.

Для модуля захоплення зображень було розроблено інтерфейс, що забезпечує повний контроль над життєвим циклом медіа-сесії та надає безпечні методи отримання графічного контенту. Основним функціональним інтерфейсом є метод capturePhoto(), який ініціює апаратний затвор камери з використанням специфічних налаштувань AVCapturePhotoSettings, приховуючи при цьому низькорівневі маніпуляції з вихідним потоком даних. Публічна властивість-замикання onPhotoCaptured виступає в ролі асинхронного інтерфейсного каналу, через який модуль повертає готовий об'єкт UIImage до вищих рівнів архітектури

(ViewModel) після завершення фонової обробки та перетворення сирих байтів у графічний формат.

Внутрішня конфігурація системи реалізується через методи checkPermissions() та setupSession(), які утворюють захищений інтерфейс підготовки середовища. Метод checkPermissions() забезпечує взаємодію з системною службою безпеки iOS для отримання авторизації користувача, тоді як setupSession() керує логікою підключення вхідних (AVCaptureDeviceInput) та вихідних (AVCapturePhotoOutput) вузлів до центральної сесії захоплення. Інтерфейс делегата photoOutput(_:didFinishProcessingPhoto:error:) виступає внутрішнім завершальним механізмом, який обробляє результати знімка, інкапсулює логіку обробки помилок захоплення та гарантує коректну передачу даних у головний потік для синхронізації з інтерфейсом користувача.

Для модуля керування даними було визначено набір функціональних інтерфейсів, що забезпечують безпечний доступ до дискового простору пристрою без ризику пошкодження структури каталогів. Ключовим інтерфейсом для запису інформації є метод saveImage(_:), який приймає об'єкт UIImage, виконує його конвертацію у формат JPEG із заданим коефіцієнтом стиснення та повертає унікальне ім'я файлу у форматі рядка. Цей метод інкапсулює логіку визначення шляхів до директорії Documents та механізм обробки помилок запису, надаючи клієнтським компонентам лише ідентифікатор для подальшого збереження в основній базі даних.

Для отримання та видалення медіа-ресурсів модуль надає інтерфейсні методи loadImage(fileName:) та deleteImage(fileName:). Метод loadImage реалізує безпечне зчитування даних, попередньо перевіряючи існування файлу за вказаним шляхом, що запобігає виникненню критичних помилок при відсутності контенту. Метод deleteImage виступає інструментом підтримки гігієни сховища, забезпечуючи безповоротне видалення файлів при вилученні препарату з аптечки. При цьому низькорівневі параметри, такі як ступінь компресії (0.8) або специфічні URL-адреси системних папок, залишаються повністю прихованими, надаючи

зовнішнім модулям лише результати операцій або об'єкти зображень для відображення в інтерфейсі.

Інтерфейс модуля сповіщень забезпечує повний життєвий цикл управління локальними попередженнями без необхідності прямого доступу до системного центру повідомлень з боку клієнтських модулів.

Центральним методом інтерфейсу є scheduleExpiryAlerts(for:), який приймає об'єкт моделі медикаменту та ініціює багатокроковий сценарій планування: від попереднього очищення застарілих записів до розрахунку нових точок активації за 10, 3 та 0 днів до завершення терміну придатності. Цей метод автоматично генерує унікальні ідентифікатори requestIdentifier для кожного сповіщення, пов'язуючи їх із UUID препарату, що дозволяє системі точково керувати кожним повідомленням у загальній черзі.

Інтерфейс скасування реалізований через метод cancelAlerts(for:), який надає можливість миттєвого відкликання всіх запланованих запитів для конкретного об'єкта при його видаленні або редагуванні. Метод використовує мапінг ідентифікаторів для точного визначення цільових повідомлень у UNUserNotificationCenter, гарантуючи чистоту системного реєстру та відсутність хибних спрацювань.

Внутрішня логіка, наприклад, конфігурація звукових сигналів, встановлення часу відправлення та формування текстових шаблонів тіла повідомлення, залишається повністю прихованою, надаючи зовнішнім сервісам лише високорівневі інструменти для синхронізації сповіщень із актуальним станом бази даних. Така структура створює можливість реалізації слабкої пов'язаності підсистем додатка. Такий підхід спрощує процес подальшого тестування та супроводу коду та гарантує масштабованість архітектури в ході розвитку функціональних можливостей програмного продукту.

В результаті виконаних етапів проєктування було визначено та описано ключові інтерфейси модулів проєкту, орієнтовний перелік яких наведено у Таблиці 2.1.

Таблиця 2.1 Інтерфейси ключових модулів.

Модуль	Ключові інтерфейси (Методи та властивості)
Валідації	validateAndParse(), isReasonableDate()
Інтерфейс користувача	medications, filteredMedications, navigationDestination(), deleteMedication(), saveMedication(), alert()
Візуального відгуку	label(), color(), iconName(), StatusBadge(status:)
Сценаріїв роботи	filterMedications(), saveMedication(), init(modelContext:)
Захоплення зображень	capturePhoto(), onPhotoCaptured, setupSession(), checkPermissions()
Керування даними	saveImage(), loadImage(), deleteImage()
Сповіщень	scheduleExpiryAlerts(for:), cancelAlerts(for:), requestAuthorization()
Модуль обробки штрих-кодів	startBarcodeDetection(in:), pauseScanning(), resumeScanning()

2.5 Проєктування інтерфейсу користувача

Проєктування інтерфейсу користувача (UI) та досвіду взаємодії (UX) є фінальним етапом архітектурного планування, що перетворює технічні вимоги у візуальну структуру. На цьому етапі було визначено функціональний склад екранів, розроблено логіку навігації та створено каркаси (wireframes), які

відображають розміщення елементів керування без прив’язки до фінального графічного дизайну.

Основними функціями, які повинен підтримувати інтерфейс, є: візуальне представлення списку препаратів із кольоровою індикацією їхнього стану; можливість швидкого доступу до камери для фотофіксації упаковки; ручне введення текстових даних та маскування дати у форматі «dd.mm.yyyy»; автоматизоване введення даних за допомогою сканування штрих-коду; керування наявними записами (редагування та видалення).

На основі аналізу вимог було визначено необхідність розробки трьох ключових екранів: головний, додавання та детального перегляду (картка медикаменту).

Головний екран — це центральний вузол застосунку, де відображається перелік усіх медикаментів у аптечці. Кожен елемент списку представлений у вигляді картки, яка містить назву препарату, дату закінчення терміну та мініатюру фото (якщо вона була додана). Головною особливістю є динамічна зміна кольору фону картки (білий, жовтий, червоний) залежно від близькості критичної дати. У верхній частині екрана розміщена кнопка швидкого додавання, а в самому списку підтримується жест «swipe-to-delete» для оперативного видалення ліків.

Екран додавання реалізований у вигляді модального вікна, що дозволяє користувачеві сфокусуватися на введенні даних. Він містить блок роботи з медіа (кнопка виклику камери та область перегляду знімка) та поля для введення тексту. Особлива увага приділена полю «Термін придатності», яке має текстову маску для дотримання формату «дд.мм.рррр». Нижня частина екрана містить кнопку «Зберегти», яка залишається неактивною до моменту успішного проходження валідації всіх обов'язкових полів.

Екран детального перегляду використовується для ознайомлення з повною інформацією про препарат, включаючи велике зображення упаковки та розширені текстові нотатки. Цей екран дозволяє користувачеві детально вивчити препарат перед його використанням, що особливо важливо для ідентифікації правильного дозування або призначення, записаного в нотатках.

Для візуалізації структури було розроблено відповідні макети, які демонструють ієрархію елементів та забезпечують цілісність навігації між модулями системи.

2.6 Аналіз та вибір технологій і методів реалізації застосунку

Створення мобільного застосунку для обліку медикаментів є комплексним процесом, що вимагає детального планування та обґрунтованого вибору технологічного стека. Від правильно обраних інструментів залежить не тільки стабільність кінцевого продукту в умовах обмежених ресурсів мобільного пристрою, а й ефективність розроблення, дотримання строків та можливість подальшого масштабування функціонала. У цьому розділі проведено всебічний аналіз доступних технологій, які можна використовувати для створення надійного інструмента контролю термінів придатності. Основна мета аналізу — забезпечити таку технічну основу, яка дозволить створити продукт, що відповідає високим вимогам до безпеки даних та зручності інтерфейсу.

Найважливішим архітектурним рішенням для застосунку є вибір мови програмування та моделі взаємодії з операційною системою. На сьогоднішній день ринок мобільної розробки пропонує два основні шляхи: нативний підхід та використання кросплатформових фреймворків, лідером серед яких є Flutter.

Flutter — це відкритий фреймворк від компанії Google, що використовує мову програмування Dart [24]. Головною особливістю Flutter є власний графічний двигун Impeller, який самостійно малює кожен піксель інтерфейсу, не покладаючись на стандартні системні компоненти iOS. Це забезпечує ідентичний вигляд застосунку на різних платформах і високу швидкість розробки за рахунок функції Hot Reload. Проте для даного проєкту Flutter має ряд критичних недоліків. По-перше, він створює додатковий рівень абстракції, що збільшує розмір підсумкового бінарного файлу та споживання оперативної пам'яті. По-друге, взаємодія з системними службами, такими як UserNotifications або AVFoundation,

вимагає написання додаткового нативного мосту, що нівелює переваги єдиного коду та ускладнює відлагодження системи.

Зважаючи на потребу в максимальній енергоефективності та глибокій інтеграції з екосистемою Apple, було обрано нативну мову Swift [25]. Вона забезпечує прямий доступ до апаратних потужностей пристрою без проміжних інтерпретаторів, що гарантує миттєвий відгук інтерфейсу. Для побудови візуальної частини обрано фреймворк SwiftUI. На відміну від класичного UIKit, SwiftUI базується на декларативному підході: розробник описує, що має бути зображено на екрані залежно від стану даних, а фреймворк самостійно оптимізує процес оновлення пікселів [26]. Це дозволяє легко реалізувати кольорову індикацію термінів придатності, де колір картки автоматично змінюється при настанні певних дат без ручного перемальовування компонентів.

Для управління даними було обрано новітню технологію SwiftData. Вона дозволяє описувати моделі даних за допомогою чистого коду Swift, автоматично забезпечуючи їх збереження в базі даних SQLite. SwiftData ідеально інтегрується зі SwiftUI, що дозволяє реалізувати реактивне оновлення списку ліків: як тільки в базу додається новий препарат, головний екран буде оновлюваться миттєво.

Наступним етапом є вибір інструментів для написання коду та профілювання системи. Для розроблення обрано Xcode 23.6 — єдине офіційне інтегроване середовище, яке надає повний цикл інструментів від візуального дизайну до аналізу витоків пам'яті через утиліту Instruments [27]. Хоча існують альтернативи, такі як AppCode [28], лише Xcode підтримує функцію Previews для миттєвого перегляду змін в інтерфейсі без запуску симулятора.

Для створення графічних ресурсів, іконок та макетів інтерфейсу може використовуватися Figma [29]. Це дозволить підготувати векторні ресурси, які коректно масштабуються на різних поколіннях iPhone без втрати чіткості.

Останнім аспектом є вибір методології розроблення. Традиційна каскадна модель передбачає сувору послідовність етапів, що може бути неефективним для мобільного продукту, де користувацький досвід потребує постійної корекції [30].

Зважаючи на це, для реалізації проєкту було обрано гнучкий ітеративний підхід. Він дозволяє розділити процес на короткі цикли: реалізація базового списку, додавання модуля камери, впровадження системи сповіщень тощо. Кожна ітерація завершується тестуванням, що дає змогу виявляти архітектурні помилки на ранніх стадіях та адаптувати функціонал під вимоги ТЗ [31]. Такий гібридний підхід забезпечує високу якість коду та відповідність фінального продукту очікуванням користувача.

2.7 Висновки до другого розділу

За результатами проведеного проєктування було сформовано цілісну архітектурну та технологічну концепцію мобільного застосунку для контролю термінів придатності медикаментів. У ході системного аналізу було реалізовано декомпозицію, яка дозволила розмежувати відповідальність між візуальним представленням, внутрішніми обчисленнями та механізмами персистентності. Такий підхід забезпечив високу модульність системи, де кожен функціональний блок є автономною одиницею. Це створює фундамент для стабільної роботи програмного продукту, полегшує його майбутнє супроводження та дозволяє ізолювати логіку валідації даних від особливостей графічного інтерфейсу.

Також було приділено увагу динамічній моделі системи, яка базується на опрацюванні станів кожного об’єкта. Система не просто пасивно зберігає інформацію, а активно реагує на наближення критичних дат, ініціюючи зміну маркерів та активуючи багаторівневу систему Push-сповіщень. Такий превентивний механізм є ключовим інструментом забезпечення безпеки користувача.

Архітектурна цілісність проєкту підкріплена глибоким аналізом залежностей та впровадженням сучасних патернів проєктування. Використання чистої архітектури у поєднанні з односпрямованими потоками даних дозволяє мінімізувати жорстку зв'язність між компонентами. Впровадження принципу інверсії управління забезпечило динамічне надання системних ресурсів, таких як

база даних або камера, безпосередньо у класи управління станом, що виключає виникнення конфліктів доступу та витоків пам'яті.

Проєктування користувацького інтерфейсу дозволило перевести функціональні вимоги у площину ергономічної взаємодії. Сформована структура екранів орієнтована на швидкість та зручність, де кожна дія користувача вимагає мінімальної кількості переходів. Використання макетів низької деталізації допомогло підтвердити логіку навігації та забезпечити візуальну зрозумілість кольорової індикації станів, що є важливим для швидкої ідентифікації придатних та непридатних медикаментів у реальних умовах використання.

Технологічний вибір на користь нативного стека Apple став обґрунтованим результатом порівняльного аналізу із кросплатформовими рішеннями. Відмова від використання сторонніх фреймворків, таких як Flutter, дозволила уникнути зайвих рівнів абстракції та забезпечити максимальну продуктивність системи. Використання SwiftData у поєднанні з локальною базою даних SQLite було визначено як оптимальний шлях для гарантування приватності медичних даних та автономності застосунку. Таке рішення забезпечує стабільну роботу інструментарію навіть за відсутності мережевого з’єднання, що робить його надійним помічником у будь-яких обставинах.

Обрана ітеративна методологія розробки дозволила поєднати чітке архітектурне планування із гнучкістю реалізації. Сформована проєктна документація та визначені інтерфейси модулів є вичерпною базою для переходу до безпосереднього етапу програмної реалізації, гарантуючи відповідність фінального застосунку всім технічним та функціональним стандартам.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Особливості розробки нативного застосунку в середовищі Xcode

Програмна реалізація системи обліку медикаментів базується на використанні інтегрованого середовища розроблення Xcode 23.6, що є єдиним офіційним інструментарієм для створення ПЗ під екосистему Apple (рисунок 3.1) [27]. Вибір цього середовища зумовлений потребою в оптимізації застосунку під апаратне забезпечення iPhone та необхідністю забезпечення високого рівня енергоефективності. Процес розроблення в Xcode базується на використанні мови програмування Swift 5, яка поєднує в собі продуктивність компільованих мов із лаконічністю сучасного синтаксису.

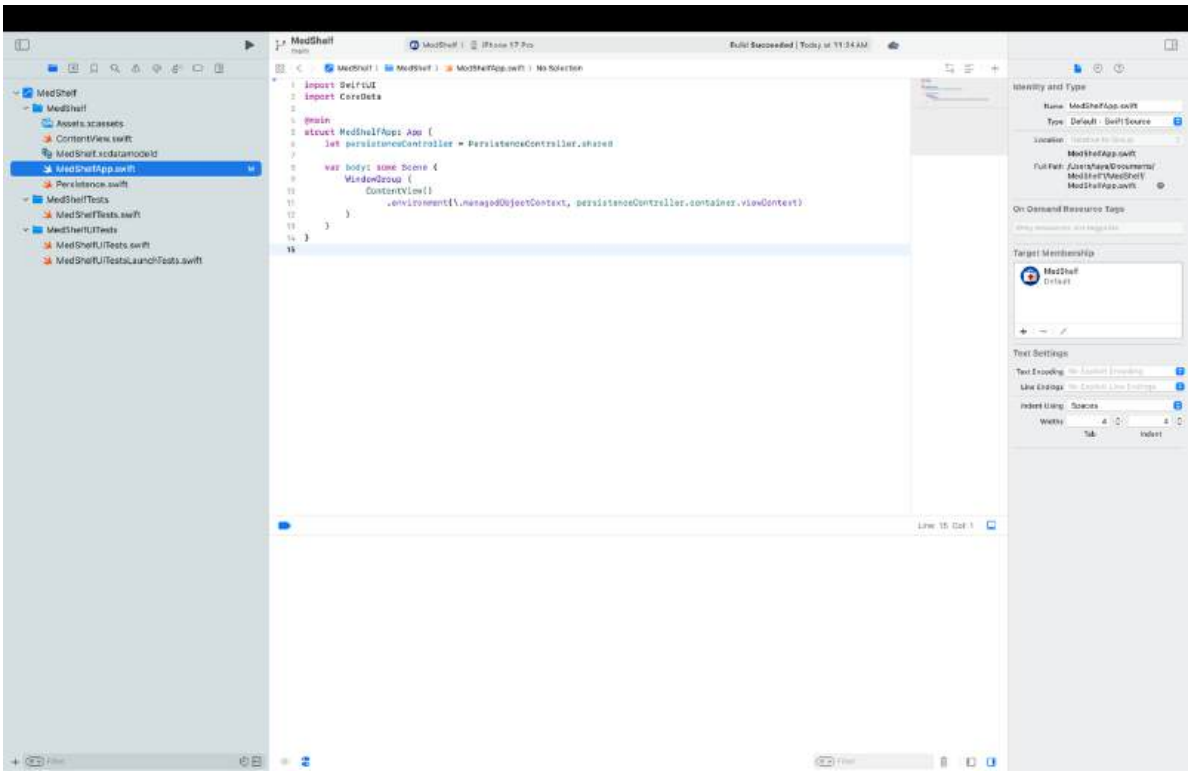


Рисунок 3.1 – Вікно Xcode

Ключовою особливістю реалізації є перехід від імперативного підходу до декларативного стилю програмування за допомогою фреймворку SwiftUI. Це

дозволяє описувати інтерфейс як функцію від стану даних, де система самостійно відстежує зміни в моделях та оновлює лише ті елементи графічної ієрархії, що безпосередньо залежать від цих змін. Такий підхід спрощує реалізацію динамічної індикації термінів придатності: при зміні дати або настанні критичного терміну, колірні маркери карток оновлюються автоматично без необхідності перемальовувати UI-компоненти.

Важливою складовою технічного процесу в Xcode є використання механізму SwiftUI Previews. Він дозволяє розробнику бачити результати змін у коді в реальному часі на симуляторах різних пристроїв одночасно, не витрачаючи час на повну перекомпіляцію та запуск проєкту [32]. Це сприяє забезпеченню адаптивності інтерфейсу під різні розміри екранів.

Окремої уваги заслуговує інтеграція з фреймворком SwiftData, який виступає основним шаром абстракції над локальною базою даних SQLite. На відміну від застарілих методів роботи з базами даних, SwiftData в середовищі Xcode дозволяє описувати схему даних безпосередньо через макроси мови Swift. Це забезпечує типобезпечність на етапі компіляції, мінімізуючи ймовірність помилок при зверненні до полів бази даних.

Xcode також надає вбудовані інструменти для дебаггінгу стану сховища, що дозволяє візуально контролювати процес запису та зчитування об’єктів під час виконання програми.

Для забезпечення надійності системи розробка в Xcode супроводжується використанням утиліти Instruments. Вона дозволяє проводити глибокий аналіз використання оперативної пам'яті, відстежувати завантаження процесора при обробці зображень з камери та виявляти потенційні «вузькі місця» в алгоритмах валідації дат.

Такий комплексний підхід до нативної розробки дозволяє створити продукт, який відчувається частиною операційної системи, забезпечуючи користувачеві звичний та плавний UX.

3.2 Реалізація програмних модулів

Для реалізації мобільного застосунку «MedShelf» було обрано модульний підхід, що базується на сучасних декларативних технологіях розроблення. Основним планом реалізації модулів є розроблення в такому порядку: архітектура бази даних та моделей, сервіси збереження та сповіщень, бізнес-логіка управління списком ліків та інтерфейс користувача.

Основою застосунку є модель даних Medication, яка була спроєктована з використанням фреймворку SwiftData. Згідно з описаним раніше функціоналом, ця модель повинна забезпечувати зберігання параметрів препарату: назви, приміток, терміну придатності та шляху до зображення упаковки. Для забезпечення цілісності даних при зміні структури бази було використано механізм автоматичної міграції схеми даних. Код, що відповідає за ініціалізацію та збереження об'єкта Medication, наведено у додатку А.

Клас AddMedicineViewModel виступає ядром на рівні представлення застосунку, безпосередньо реалізуючи концепцію моста в межах патерна MVVM. Цей компонент акумулює в собі повний стан екранної форми додавання медикаментів. Він виконує роль посередника між інтерфейсом та базою даних, забезпечуючи валідацію введеної інформації. Зв'язок із елементами інтерфейсу здійснюється за допомогою набору реактивних властивостей стану, які відстежують зміни моделі.

Особлива увага була приділена досвіду користувача (UX) під час введення даних. Для спрощення взаємодії з полем терміну придатності було розроблено кастомний компонент DateInputField. Його головна функція полягає у автоматичному маскуванні вводу: при введенні цифр символи розділювачів (крапки) розставляються автоматично після другого та четвертого символів. Цей функціонал реалізовано за допомогою аналізу зміни поточного рядка та фільтрації нечислових символів, що дозволяє уникнути помилок ручного введення та пришвидшує процес заповнення форми.

Робота з візуальним контентом забезпечується сервісом CameraService, який інтегрує системні можливості камери через обгортку CameraPreview. Користувач має можливість зробити знімок упаковки, який після підтвердження передається у ViewModel, стискається та зберігається у файловій системі пристрою через StorageService. Це дозволяє не перевантажувати базу даних великими бінарними об'єктами, зберігаючи лише посилання на шлях до файлу.

Центральним модулем відображення даних є MedicineListView, який використовує механізм реактивних запитів @Query. Це дозволяє списку ліків оновлюватися в реальному часі без необхідності ручного перезавантаження екрана після додавання чи видалення записів. Для зручності навігації впроваджено функціонал пошуку через модифікатор searchable, який фільтрує масив медикаментів за назвою. Кожен елемент списку представлений компонентом MedicineRowView, де замість дати пріоритетно виводиться місце зберігання, а стан придатності візуалізується через уніфікований елемент StatusBadge.

Для забезпечення інформативності було реалізовано перехід до детальної картки препарату через NavigationLink. Екран деталей MedicineDetailView агрегує всю наявну інформацію, включаючи розширені нотатки та повнорозмірне фото. Також у цьому модулі реалізовано функцію повного видалення об'єкта, яка включає не лише очищення запису в SwiftData, а й деструкцію пов'язаного файлу зображення та скасування запланованих локальних сповіщень через NotificationService. Такий підхід гарантує відсутність «сміттєвих» даних та некоректних нотифікацій після видалення препарату з аптечки.

Код деяких програмних модулів наведено у додатку А.

3.3 Реалізація інтерфейсу користувача

При розробленні інтерфейсу користувача застосунку «MedShelf» основний акцент було зроблено на принципах декларативності та композиції компонентів,

що надаються фреймворком SwiftUI. Побудова графічної оболонки відбувалася шляхом декомпозиції складних екранів на окремі незалежні модулі, що забезпечує гнучкість налаштування та повторне використання коду.

Центральним елементом візуальної ієрархії є головний екран списку ліків. Для його реалізації використано контейнер NavigationStack, який керує переходами між переглядами. Внутрішня логіка екрана базується на умовному розгалуженні: у разі відсутності записів у базі даних користувачеві демонструється компонент ContentUnavailableView, що виконує роль інтерактивної заглушки (рисунок 3.2). Це дозволяє уникнути порожнього простору та надає чітку інструкцію до подальших дій.

Рисунок 3.2 – Візуалізація порожнього стану аптечки

Елементи списку реалізовані через спеціалізований модуль MedicineRowView. Його структура базується на горизонтальному стеку (HStack),

що поєднує мініатюру зображення препарату, текстовий блок із назвою та місцем зберігання, а також уніфікований індикатор статусу. Особливістю реалізації є використання модифікатора swipeActions, який дозволяє викликати видалення запису безпосередньо через жест зсуву елемента вліво, що є стандартом для платформи iOS.

Для забезпечення єдиного стилю індикації термінів придатності в усіх модулях застосунку було впроваджено компонент StatusBadge. Він автоматично обчислює колірну схему та текстове значення («Придатні», «Уважно» або «Протерміновані») на основі логіки, закладеної в моделі даних. Це дозволяє централізовано змінювати дизайн індикаторів без необхідності редагування кожного екрана окремо.

Форма додавання препарату AddMedicineView побудована за допомогою системного контейнера Form, який автоматично адаптує візуальні елементи під стилістику налаштувань iOS. У цій формі інтегровано кастомний елемент DateInputField, що вирішує проблему складного введення дат на мобільних пристроях. Замість стандартного барабанного вибору дати, який потребує багато маніпуляцій, було обрано текстове поле з маскою, що автоматично форматує ввід користувача. Таке рішення мінімізує кількість помилок при перенесенні інформації з фізичної упаковки ліків у цифровий формат.

Завершальним етапом реалізації інтерфейсу став екран детальної інформації MedicineDetailView. Він використовує ScrollView для коректного відображення великих обсягів тексту та повнорозмірних фотографій на пристроях з різною діагоналлю екрана.

3.4 Інструкція користувача та вимоги до системи

Для успішної експлуатації програмного забезпечення «MedShelf» користувач повинен володіти базовими навичками роботи з пристроями на базі операційної системи iOS. Мінімальні системні вимоги включають наявність

встановленої ОС версії 17.0 або новішої, що зумовлено використанням фреймворку SwiftData [24]. Для коректної роботи застосунок потребує надання дозволу на доступ до системної камери для використання функції фотофіксації, а також дозвіл на надсилання push-сповіщень для функції нагадувань.

Взаємодія із застосунком починається з головного екрана, де відображається повний перелік медикаментів, автоматично відсортований за пріоритетністю використання. Користувач може миттєво оцінити стан аптечки завдяки колірним індикаторам StatusBadge, які візуалізують рівень придатності кожного препарату без необхідності відкриття детальної картки. Для швидкого пошуку конкретного засобу використовується інтегрований рядок пошуку у верхній частині інтерфейсу (Рисунок 3.3).

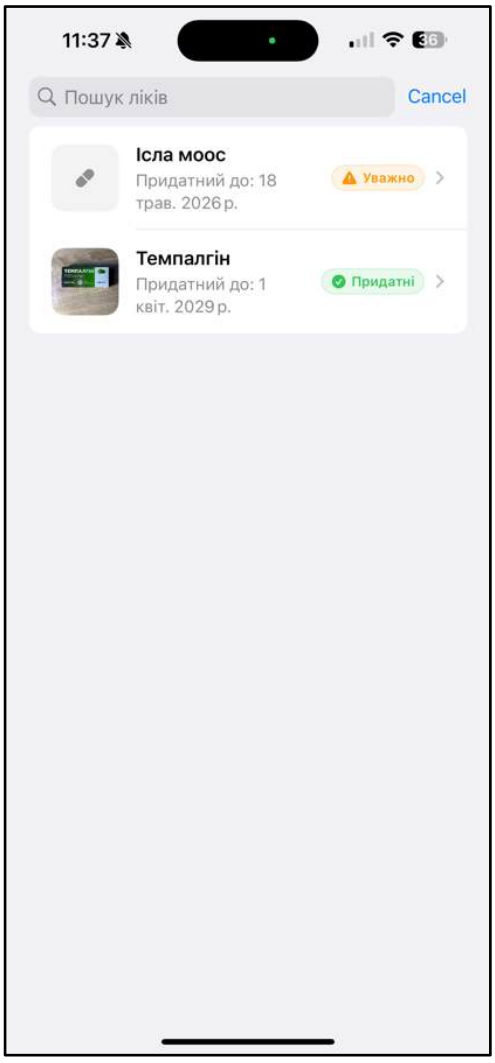


Рисунок 3.3 – Головний екран при наявності медикаментів у аптечці

Процес наповнення бази даних реалізований через інтуїтивну форму додавання, яка активується кнопкою «+». Під час введення назви та місця зберігання користувач може скористатися функцією інтелектуального вводу дати у компоненті DateInputField: система самостійно форматує цифровий ряд, додаючи крапки-розділювачі, що мінімізує ризик механічних помилок. Завершальним етапом створення запису є фотофіксація упаковки за допомогою вбудованого інтерфейсу камери, після чого дані надійно фіксуються у локальному сховищі. Рисунок 3.4 зображує екран додавання нового медичного препарату.

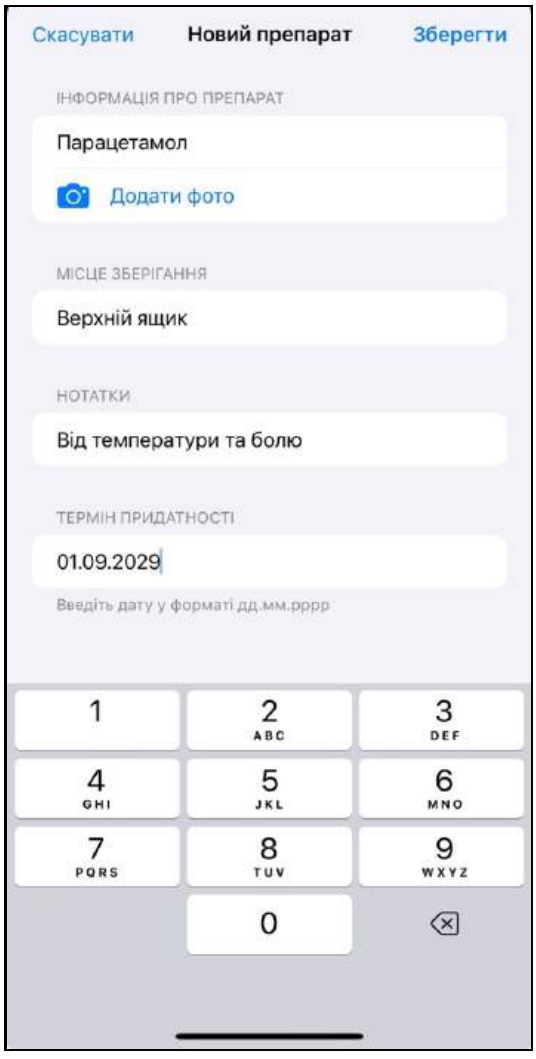
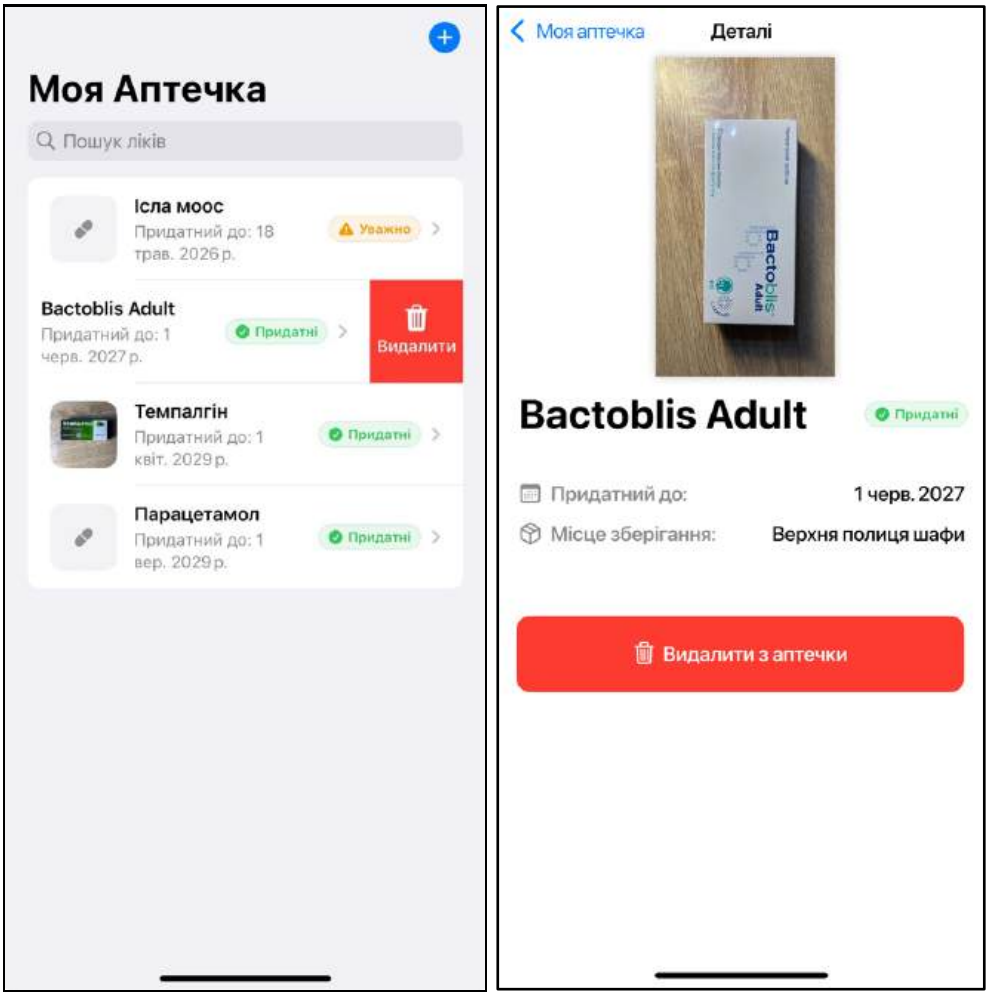


Рисунок 3.4 – Процес заповнення картки ліків

Для перегляду розширеної інформації, включаючи специфічні нотатки та повнорозмірне фото, достатньо натиснути на відповідний елемент списку. Екран

деталей MedicineDetailView дозволяє не лише ознайомитися з характеристиками препарату, а й керувати його життєвим циклом. Видалення застарілих або використаних ліків здійснюється або через спеціальну кнопку в картці об'єкта, або за допомогою швидкого жесту зсуву ліворуч безпосередньо у загальному списку (Рисунки 3.5, 3.6).



Рисунки 3.5, 3.6 – Видалення препарату зі списку за допомогою змахування та перегляд детальної інформації

Застосунок розроблений з урахуванням автономності: усі дані зберігаються локально на пристрої, що дозволяє використовувати «MedShelf» без активного підключення до мережі Інтернет, забезпечуючи конфіденційність та безпеку медичних даних користувача.

3.5 Тестування програмного забезпечення

3.5.1 Аналіз методів тестування нативних iOS застосунків

Забезпечення якості нативних iOS-застосунків є комплексним процесом, який вимагає розуміння специфіки екосистеми Apple та особливостей апаратного забезпечення мобільних пристроїв. На відміну від кросплатформових рішень, нативне тестування дозволяє максимально ефективно використовувати системні ресурси та перевіряти взаємодію програми з операційною системою на найнижчому рівні. Основним підходом до аналізу якості в даній роботі є використання концепції піраміди тестування, де нижнім рівнем виступає модульне тестування логіки, а вершиною — автоматизація користувацького інтерфейсу.

Модульне тестування в середовищі розробки Xcode реалізується за допомогою фреймворку XCTest [33] і спрямоване на ізольовану перевірку найменших логічних одиниць програмного коду. Цей метод дозволяє виявляти дефекти в алгоритмах валідації введених дат, розрахунках термінів придатності та логіці фільтрації даних. Висока швидкість виконання модульних тестів забезпечує розробнику можливість постійного рефакторингу коду без ризику порушення існуючої функціональності. Особлива увага при аналізі модульних методів приділяється потокобезпеці, оскільки сучасний стандарт Swift Concurrency вимагає перевірки виконання операцій оновлення інтерфейсу виключно у головному потоці через атрибут MainActor.

Тестування інтерфейсу користувача за допомогою XCUITest дозволяє автоматизувати перевірку цілісних сценаріїв взаємодії, імітуючи дії реальної людини та дозволяючи перевірити коректність поведінки UI [34]. Аналіз цього методу виявляє необхідність використання спеціальних ідентифікаторів доступності, які забезпечують стабільність тестів незалежно від локалізації чи візуальних змін дизайну. Важливою частиною аналізу UI-тестування є опрацювання механізмів очікування, оскільки асинхронна природа мобільних

інтерфейсів потребує врахування часу на анімації переходів та появу системної клавіатури. Це запобігає виникненню хибних результатів тестування, що пов’язані з динамічною зміною стану екрана.

Тестування продуктивності виступає окремим важливим напрямком аналізу, оскільки воно дозволяє фіксувати метрики використання пам’яті та процесорного часу. В контексті розробленого застосунку це стосується швидкості завантаження головного екрана та ефективності збереження великих об’ємів даних через SwiftData. Встановлення базових ліній продуктивності дає змогу розробнику відстежувати будь-яку деградацію швидкодії при додаванні нових функцій. Крім того, аналіз методів збереження даних включає перевірку міграцій, що підтверджує цілісність інформації при зміні версій бази даних та запобігає втраті критичних записів користувача.

Ручне тестування доповнює автоматизовані методи, фокусуючись на аспектах зручності використання та суб’єктивному сприйнятті інтерфейсу. Воно дозволяє дослідити граничні випадки, які важко передбачити в автоматизованих сценаріях, наприклад, поведінку програми під час вхідних дзвінків або при низькому рівні заряду акумулятора. Окремим аспектом є локалізаційне тестування, яке підтверджує правильність адаптації текстових елементів під українську мову, зокрема коректне відмінювання назв місяців у датах. Таким чином, поєднання різних методів аналізу якості дозволяє створити надійний програмний продукт, що відповідає високим стандартам стабільності та зручності.

3.5.2 Виконання тестування та аналіз його результатів

Процес практичної перевірки розробленого застосунку розпочався з виконання модульних тестів логічного рівня. Основним завданням на цьому етапі стала верифікація алгоритмів обробки часових інтервалів та валідації рядкових даних. Під час виконання тестування парсингу дати було підтверджено, що

система коректно інтерпретує рядок формату «дд.мм.рррр» та відхиляє некоректні значення, такі як «99.99.9999», повертаючи нульовий результат замість створення помилкового об’єкта в базі даних. Аналіз результатів тестування статусів придатності показав високу точність розрахунків: при встановленні поточної дати на 05 травня 2026 року препарати з терміном до 10 травня автоматично отримували статус протермінованих, а препарати з терміном до 07 травня успішно маркувалися попереджувальним статусом. Це підтверджує готовність бізнес-логіки до коректної роботи в реальних часових умовах.

Нижче наведено фрагменти коду модульних тестів testValidDateParsing та testExpiredStatus:

```
func testValidDateParsing() {
        // Введення "12.05.2026" має повертати коректний об'єкт Date
        let dateString = "12.05.2026"
        let formatter = DateFormatter()
        formatter.dateFormat = "dd.MM.yyyy"

        let parsedDate = formatter.date(from: dateString)

        XCTAssertNotNil(parsedDate, "Дата має бути успішно розпарсена")

        let calendar = Calendar.current
        let components = calendar.dateComponents([.day, .month, .year], from:
parsedDate!)
        XCTAssertEqual(components.day, 12)
        XCTAssertEqual(components.month, 5)
        XCTAssertEqual(components.year, 2026)
    }

    func testInvalidDateParsing() {
        // Введення "99.99.9999" має повертати nil
        let invalidDateString = "99.99.9999"
        let formatter = DateFormatter()
        formatter.dateFormat = "dd.MM.yyyy"

        let parsedDate = formatter.date(from: invalidDateString)

        XCTAssertNil(parsedDate, "Некоректна дата повинна повертати nil")
    }

    func testExpiredStatus() {
        // Якщо сьогодні 12.05.2026, а термін 10.05.2026 — статус .expired
        let calendar = Calendar.current
        let expiredDate = calendar.date(from: DateComponents(year: 2026, month:
5, day: 10))!

        let medication = Medication(name: "Тест", details: "", expiryDate:
expiredDate)

        XCTAssertEqual(medication.status, .expired, "Препарат має бути позначений
як протермінований")}
```

Наступним етапом стало виконання автоматизованого UI-тестування для перевірки цілісності користувацьких сценаріїв. Під час виконання тесту додавання препарату «Ісла Моос» система успішно пройшла всі кроки: від натискання кнопки виклику форми до введення тексту та збереження запису. Аналіз результатів виконання виявив критичну особливість взаємодії з системною клавіатурою, яка потребувала впровадження додаткових затримок та перевірок фокуса, щоб уникнути помилок при переході між текстовими полями. Нижче наведено код UI тесту testAddMedecineScenario:

```
func testAddMedicineScenario() throws {
            // 1. Відкрити форму додавання
            let app = XCUIApplication()
            app.activate()
            let addButton = app.buttons["AddMedicinePlusButton"]
            XCTAssertTrue(addButton.waitForExistence(timeout: 5))
            addButton.tap()

            // 2. Ввести назву "Ісла Моос"
            let nameField = app.textFields["MedicineNameInput"]
            XCTAssertTrue(nameField.exists)
            nameField.tap()
            nameField.typeText("Ісла Моос")

            // Закрити клавіатуру
            if app.keyboards.buttons["return"].exists {
                app.keyboards.buttons["return"].tap()
            } else {
                print("Nope")
            }

            // 3. Ввести дату "01.01.2027"
            let dateField = app.textFields["MedicineDateInput"]
            XCTAssertTrue(dateField.exists)
            dateField.tap()
            dateField.typeText("01012027")

            // Закрити клавіатуру
            if app.keyboards.buttons["return"].exists {
                app.keyboards.buttons["return"].tap()
            } else {
                print("Nope")
            }

            // 4. Ввести місце зберігання
            let storageField = app.textFields["MedicineStorageLocationInput"]
            XCTAssertTrue(storageField.exists)
            storageField.tap()
            storageField.typeText("Ящик 1")

            // Закрити клавіатуру
            if app.keyboards.buttons["return"].exists {
                app.keyboards.buttons["return"].tap()
            } else {
```

```
                print("Nope")
            }

            // 5. Натиснути "Зберегти"
            let saveButton = app.buttons["SaveMedicineButton"]
            saveButton.tap()

            // Очікуваний результат: елемент з'явився у списку
            let medicineCell = app.staticTexts["Ісла Мооc"]
            XCTAssertTrue(medicineCell.waitForExistence(timeout: 5), "Новий
препарат має з'явитися у списку")
    }
```

Результати тестування функції пошуку продемонстрували швидку реакцію інтерфейсу на введення символів, де відфільтрований список відповідав заданим критеріям, а очищення поля пошуку коректно повертало всі записи на екран (Рисунок 3.7).

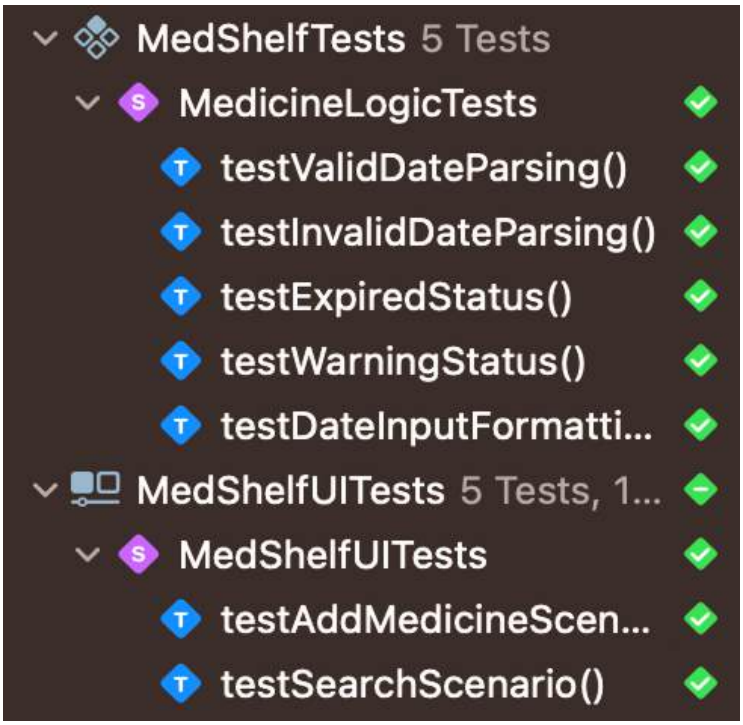


Рисунок 3.7 – Скріншот екрана Test Navigator: усі тести завершені успішно

Ручне тестування дозволило оцінити аспекти, що не піддаються автоматизації, зокрема було проведено перевірку роботи камери, яка підтвердила успішне захоплення зображення та його прив’язку до конкретного об’єкта ліків у сховищі. Під час тестування стійкості до переривань було встановлено, що дані зберігаються коректно навіть при раптовому закритті застосунку під час

заповнення форми, що свідчить про надійність інтеграції з фреймворком SwiftData. Результати ручного тестування вказані у таблиці 3.1.

Таблиця 3.1 Результати ручного тестування.

Номер ТС	Опис ТС	Очікуваний результат	Фактичний результат
ТС-01	Додавання препарату з порожнім полем назви	Кнопка збереження неактивна або система виводить попередження про обов'язковість поля	Система не дозволяє зберегти запис без назви, поле підсвічується
ТС-02	Введення дати у форматі «31.02.2026» (неіснуюча дата)	Система розпізнає дату як некоректну та не дозволяє збереження або скидає значення	Система автоматично коригує або відхиляє некоректну дату через DateFormatter
ТС-03	Зйомка фотографії препарату через камеру пристрою	Фото успішно створюється, відображається у формі та зберігається в пам'яті	Фотографія захоплена, відображена в профілі ліків та збережена
ТС-04	Зміна теми оформлення пристрою на темну	Всі текстові елементи та фони адаптуються, зберігаючи високу контрастність	Кольори інтерфейсу інвертовані, читабельність тексту збережена

Кінець таблиці 3.1 Результати ручного тестування.

Номер ТС	Опис ТС	Очікуваний результат	Фактичний результат
ТС-05	Видалення запису про ліки за допомогою жесту «свайп»	Запис зникає зі списку, дані та пов'язане фото видаляються з бази SwiftData	Запис видалено, об'єм пам'яті, зайнятий фотографією, звільнено
ТС-06	Пошук препарату за частиною назви («Іс»)	У списку залишаються лише ті препарати, назва яких містить введену комбінацію	Список миттєво відфільтровано, відображено лише «Ісла Моос»
ТС-07	Перезапуск додатка після внесення змін	Після повторного запуску всі додані раніше ліки відображаються у списку	Всі дані успішно завантажені зі сховища SwiftData, стан списку збережено

Загальний аналіз результатів проведеного тестування свідчить про високий ступінь готовності програмного продукту до експлуатації. Всі виявлені під час перевірок недоліки, пов’язані з фокусуванням елементів та анімаціями, були усунені шляхом оптимізації коду тестів та додавання ідентифікаторів доступності.

Відсутність помилок у логічних тестах та успішне проходження сценаріїв ручного тестування дозволяють стверджувати, що застосунок відповідає поставленим функціональним вимогам та забезпечує стабільну роботу в умовах нативного середовища iOS.

3.6 Висновки до третього розділу

Було виконано повний комплекс робіт, пов’язаних із програмною реалізацією, налаштуванням користувацького інтерфейсу, розробленням експлуатаційної документації та тестуванням мобільного застосунку «MedShelf».

На основі проведеного дослідження та безпосереднього написання програмного коду обґрунтовано доцільність вибору повністю нативного технологічного стеку розроблення, який базується на інтегрованому середовищі Xcode та мові програмування Swift 5.

Особливу увагу при розробленні графічної оболонки було приділено декларативному підходу, реалізованому за допомогою фреймворку SwiftUI, де інтерфейс виступає безпосередньою функцією від стану даних. Це дозволило спростити архітектуру системи та забезпечити автоматичне оновлення колірних маркерів придатності медикаментів без надлишкової перекомпіляції та перемальовування інтерфейсних компонентів. Адаптивність дизайну під різні діагоналі екранів була успішно верифікована в реальному часі за допомогою інструменту SwiftUI Previews, що суттєво пришвидшило процес верстки.

Для управління шаром даних та забезпечення стійкості інформації було інтегровано сучасний фреймворк SwiftData, який виступає об’єктно-орієнтованою абстракцією над реляційною базою даних SQLite.

При проектуванні бізнес-логіки у AddMedicineViewModel було вирішено проблему валідації вводу текстових дат шляхом розроблення кастомного компонента DateInputField, який автоматично маскує рядок та розставляє крапки-розділювачі, знижуючи ймовірність механічних помилок користувача. Окрім того, впроваджено гібридну схему збереження медіа-ресурсів через CameraService та StorageService, за якої важкі графічні знімки упаковок стискаються та записуються безпосередньо у файлову систему пристрою, а в базі даних фіксуються лише локальні шляхи до них. Такий підхід запобігає розростанню сховища та дозволяє реалізувати каскадне видалення даних, коли разом із карткою

ліків безповоротно знищуються пов'язані файли зображень та скасовуються заплановані локальні нагадування в NotificationService.

Завершальним етапом роботи стала перевірка якості створеного продукту. За допомогою фреймворку XCTest реалізовано модульні тести логічного рівня, які підтвердили правильність парсингу дат та коректність автоматичного присвоєння статусів придатності відносно поточного часу.

Автоматизоване UI-тестування за допомогою інструментарію XCUITest дозволило симулювати наскрізні сценарії додавання та пошуку препаратів із урахуванням затримок на анімацію та поведінку системної клавіатури.

Комплекс ручних функціональних тестів додатково підтвердив стійкість ПЗ до раптових переривань, безпомилкову роботу інтеграції з камерою, швидке реагування пошукових алгоритмів та коректну візуальну адаптацію всіх елементів інтерфейсу під час зміни системної теми оформлення на темну. Загальний аналіз результатів проведеного тестування свідчить про високу стабільність, надійність та готовність нативного застосунку «MedShelf» до практичної експлуатації.

ВИСНОВКИ

У результаті проведеного дослідження та розробки було створено цілісний програмний продукт — мобільний застосунок «MedShelf», призначений для інтелектуального управління домашньою аптечкою. Робота охопила повний цикл створення інформаційної системи: від аналізу проблем предметної області та моделювання бізнес-процесів до програмної реалізації нативними засобами iOS та всебічного тестування готового рішення.

У першому розділі було проведено аналіз предметної області, який визначив, що ефективний менеджмент медикаментів у домашніх умовах є одним з факторів безпеки життєдіяльності. Було виявлено, що традиційні методи контролю не забезпечують належного рівня моніторингу термінів придатності, а існуючі програмні аналоги здебільшого фокусуються на графіку прийому ліків, залишаючи поза увагою інвентаризацію. Це дозволило сформулювати концепцію нового продукту, ключовою перевагою якого стало впровадження трирівневої системи сповіщень та колірної індикації станів придатності медикаментів.

У другому розділі було викладено опис етапу проєктування, де для формалізації архітектури системи було застосовано об’єктно-орієнтований підхід та мову моделювання UML. Побудова діаграм прецедентів, потоків даних та станів забезпечила чітке розуміння функціональних меж застосунку та логіки його роботи. Вибір архітектурного шаблону MVVM у поєднанні з принципами SOLID дозволив створити гнучку та модульну систему, де бізнес-логіка розрахунку термінів відокремлена від інтерфейсу користувача, що значно спрощує подальше масштабування та супровід продукту.

Третій розділ фокусувався на програмна реалізації застосунку та його тестуванні. Було обрано сучасний технологічний стек Apple, що включає в себе мову програмування Swift, фреймворки SwiftUI та SwiftData. Такий вибір дозволив досягти максимальної продуктивності та нативності інтерфейсу.

Реалізовані модулі сценаріїв роботи, захоплення зображень, валідації,

керування даними, сповіщень та інтерфейсу користувача працюють як єдиний механізм, забезпечуючи швидкий відгук та стабільність. Особливу увагу було приділено зручності введення даних: розроблений кастомний компонент із маскуванням дати та автоматична перевірка вводу мінімізують ризик помилок користувача при наповненні бази даних.

Водночас варто зазначити, що запланований модуль сканування штрих-кодів для автоматизованого пошуку медикаментів на поточному етапі залишився нереалізованим. Це зумовлено зовнішніми чинниками, а саме дефіцитом інформаційної взаємодії зі сторони профільних суб'єктів ринку. Надіслані офіційні запити до мереж аптек та інформаційних агрегаторів щодо надання санкціонованого доступу до їхніх відкритих прикладних інтерфейсів програмування або спеціалізованих баз даних лікарських засобів не отримали позитивного відгуку, що унеможливило інтеграцію зовнішнього верифікованого контенту в архітектуру застосунку.

Результати проведеного модульного, UI та ручного тестування підтвердили якість та надійність розробленого ПЗ. Система успішно обробляє критичні сценарії, коректно інтерпретує часові дельти та забезпечує цілісність даних при перериваннях роботи. результатів тестування демонструє повну відповідність фактичної поведінки програми очікуваним результатам.

Застосунок «MedShelf» є автономним рішенням, яке гарантує приватність медичних даних користувача шляхом їх локального зберігання. Впроваджений підхід до візуалізації стану аптечки через індикатори StatusBadge та спрощена навігація дозволяють користувачу витрачати мінімум часу на облік медикаментів, отримуючи при цьому максимальний рівень контролю за їхньою безпекою. Таким чином, мета роботи досягнута повністю: створено ефективний, інтуїтивно зрозумілий та технологічно досконалий інструмент, що відповідає сучасним стандартам розробки для мобільної платформи iOS і має практичну цінність для широкого кола користувачів у сфері Health & Fitness.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Global strategy on digital health 2020-2025. Geneva: World Health Organization; 2021. Ліцензія: CC BY-NC-SA 3.0 IGO (дата звернення: 22.02.2026).
2. Don’t Be Tempted to Use Expired Medicines. U.S. Food and Drug Administration (FDA). URL: https://www.fda.gov/drugs/special-features/dont-be-tempted-use-expired-medicines (дата звернення: 22.02.2026).
3. UHBlog. Expired Medications: Dangerous or Just Less Effective?. Nationally Ranked Healthcare - Largest Network of Hospitals, Doctors & Surgeons in Cleveland & Northeast Ohio | University Hospitals. URL: https://www.uhhospitals.org/blog/articles/2023/08/expired-medications-dangerous-or-just-less-effective (дата звернення: 22.02.2026).
4. Scheduling a notification locally from your app | Apple Developer Documentation. Apple Developer Documentation. URL: https://developer.apple.com/documentation/usernotifications/scheduling-a-notification-locally-from-your-app (дата звернення: 22.02.2026).
5. Color | Apple Developer Documentation. Apple Developer Documentation. URL: https://developer.apple.com/design/human-interface-guidelines/color (дата звернення: 22.02.2026).
6. Andrzejewska M., Stolińska A., Baran W. The Effects of Colour Coding on Problem-Solving Strategies and Cognitive Engagement: Insights from Eye-Tracking Research. Applied Sciences. 2025. Т. 15, № 21. С. 11503. URL: https://doi.org/10.3390/app152111503 (дата звернення: 22.02.2026).
7. What is Deep Linking: Deep Linking Definition | Unity. Unity. URL:https://unity.com/glossary/deep-linking (дата звернення: 22.02.2026).
8. About Us. Medisafe Inc. URL: https://www.medisafe.com/about-us/ (дата звернення: 22.02.2026).

9. Medication Reminder and Pill Tracker App – MyTherapy. Medication Reminder and Pill Tracker App – MyTherapy. URL: https://www.mytherapyapp.com/ (дата звернення: 22.02.2026).

10. Medisafe Medication Management app App Store. App Store. URL: https://apps.apple.com/ua/app/medisafe-medication-management/id573916946?l=uk (дата звернення: 22.02.2026).

11. Нагадування таблетки MyTherapy. App Store. URL: https://apps.apple.com/ua/app/нагадування-таблетки-mytherapy/id662170995?l=uk (дата звернення 22.02.2026).

12. Home Medkit - Smart Pill & Reminder App. Home Medkit - Smart Pill & Reminder App. URL: https://medkit.space/#/ (дата звернення: 22.02.2026).

13. MedKit: MedicationTracker. App Store. URL: https://apps.apple.com/ua/app/medkit-medication-tracker/id6742353036?l=uk (дата звернення: 22.02.2026).

14. VisionKit | Apple Developer Documentation. Apple Developer Documentation. URL: https://developer.apple.com/documentation/visionkit (дата звернення: 14.03.2026).

15. Human Interface Guidelines | Apple Developer Documentation. Apple Developer Documentation. URL: https://developer.apple.com/design/human-interface-guidelines (дата звернення: 14.03.2026).

16. Model-View-ViewModel (MVVM) : Apple Developer Documentation. URL: https://developer.apple.com/documentation/swiftui/managing-model-data-in-your-app (дата звернення: 19.03.2026).

17. Indrawan D., Kusumo D. S., Puspitasari S. Y. ANALYSIS OF THE IMPLEMENTATION OF MVVM ARCHITECTURE PATTERN ON PERFORMANCE OF IOS MOBILE-BASED APPLICATIONS. JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika). 2023. Vol. 8, no. 1. P. 59–65. URL: https://doi.org/10.29100/jipi.v8i1.3293 (дата звернення: 19.03.2026).

18. Lano K., Yassipour Tehrani S. Introduction to Clean Architecture Concepts. Introduction to Software Architecture. Cham, 2023. С. 35–49. URL: https://doi.org/10.1007/978-3-031-44143-1_2 (дата звернення: 20.03.2026).

19. Декомпозиція в програмуванні: що це і навіщо потрібно. FoxmindEd. URL: https://foxminded.ua/dekompozytsiia-v-prohramuvanni/ (дата звернення: 29.03.2026).

20. AVFoundation | Apple Developer Documentation. Apple Developer Documentation. URL: https://developer.apple.com/documentation/avfoundation (дата звернення: 29.03.2026).

21. Vision | Apple Developer Documentation. Apple Developer Documentation. URL: https://developer.apple.com/documentation/vision (дата звернення: 22.02.2026).

22. AVCaptureMetadataOutput | Apple Developer Documentation. Apple Developer Documentation URL: https://developer.apple.com/documentation/avfoundation/avcapturemetadataoutput (дата звернення: 29.03.2026).

23. SwiftData | Apple Developer Documentation. Apple Developer Documentation. URL: https://developer.apple.com/documentation/swiftdata (дата звернення: 29.03.2026).

24. Flutter documentation. Flutter documentation. URL: https://docs.flutter.dev/ (дата звернення: 14.04.2026).

25. Swift.org. Swift.org. URL: https://www.swift.org/ (дата звернення: 14.04.2026).

26. SwiftUI - Apple Developer. Apple Developer. URL: https://developer.apple.com/swiftui/ (дата звернення: 14.04.2026).

27. Xcode - Apple Developer. Apple Developer. URL: https://developer.apple.com/xcode/ (дата звернення: 14.04.2026).

28. JetBrains. AppCode: Smart Swift/Objective-C IDE for iOS & macOS Development. JetBrains. URL: https://www.jetbrains.com/objc/ (дата звернення: 14.04.2026).

29. Figma: The Collaborative Interface Design Tool. Figma. URL: https://www.figma.com (дата звернення: 20.04.2026).

30. Agile VS Waterfall – який варіант відповідаєвашому бізнесу?. Worksection. URL: https://worksection.com/ua/blog/waterfall-vs-agile.html (дата звернення: 20.04.2026).

31. Ітеративна модель (iterative model) - QALight. QALight. URL: https://qalight.ua/baza-znaniy/iterativna-model-iterative-model/ (дата звернення: 20.04.2026).

32. Previews in Xcode | Apple Developer Documentation. Apple Developer Documentation. URL: https://developer.apple.com/documentation/swiftui/previews-in-xcode (дата звернення: 25.04.2026).

33. XCTest | Apple Developer Documentation. Apple Developer Documentation. URL: https://developer.apple.com/documentation/xctest (дата звернення: 26.04.2026).

34. XCUIAutomation | Apple Developer Documentation. Apple Developer Documentation. URL: https://developer.apple.com/documentation/xcuiautomation (дата звернення: 26.04.2026).

35. Бедратюк Л. П., Радельчук Г. І. Кваліфікаційна робота: Методичні настанови для здобувачів першого (бакалаврського) рівня вищої освіти спеціальності 121 «Інженерія програмного забезпечення». Хмельницький: ХНУ, 2023. 60 с. (дата звернення: 21.02.2026).

ДОДАТОК А

(обов'язковий)

Діаграми

А.1 Код моделі "Medication":

```
import Foundation
import SwiftData

@Model
final class Medication {
    // Унікальний ідентифікатор запису
    @Attribute(.unique) var id: UUID = UUID()

    // Основні поля даних
    var name: String
    var details: String
    var expiryDate: Date
    var createdAt: Date
    var imagePath: String?
    var storageLocation: String

    init(name: String, details: String = "", expiryDate: Date, imagePath: String?
= nil, storageLocation: String = "") {
        self.name = name
        self.details = details
        self.expiryDate = expiryDate
        self.imagePath = imagePath
        self.createdAt = Date()
        self.storageLocation = storageLocation
    }

    /// Розрахунок кількості днів, що залишилися до закінчення терміну
    var daysRemaining: Int {
        let calendar = Calendar.current
        let startOfToday = calendar.startOfDay(for: Date())
        let startOfExpiry = calendar.startOfDay(for: expiryDate)

        let components = calendar.dateComponents([.day], from: startOfToday, to:
startOfExpiry)
        return components.day ?? 0
    }

    /// Визначення поточного статусу препарату
    var status: ExpiryStatus {
        let days = daysRemaining

        if days < 0 {
            return .expired    // Червоний (дата минула)
        } else if days <= 10 {
            return .warning    // Жовтий (10 або менше днів)
        } else {
            return .active     // Зелений (більше 10 днів)
        }
```

```
    }
}
```

А.2 Код сервісу NotificationService:

```
import Foundation
import UserNotifications

struct NotificationService {

    /// Запит дозволу на сповіщення
    static func requestAuthorization() {
        UNUserNotificationCenter.current().requestAuthorization(options: [.alert,
.badge, .sound]) { granted, error in
            if granted {
                print("Дозвіл отримано")
            } else if let error = error {
                print("Помилка запиту: \(error.localizedDescription)")
            }
        }
    }

       static func scheduleExpiryAlerts(for medication: Medication) {
           cancelAlerts(for: medication)
           let alertIntervals = [10, 3, 0]

        for daysBefore in alertIntervals {
                        guard let triggerDate = Calendar.current.date(byAdding:
.day, value: -daysBefore, to: medication.expiryDate),
                  triggerDate > Date() else { continue } // Не плануємо на минуле

            let content = UNMutableNotificationContent()
            content.title = "Термін придатності закінчується"

            if daysBefore == 0 {
                content.body = "Увага! Термін придатності \(medication.name)
закінчується сьогодні!"
            } else {
                content.body = "Залишилося \(daysBefore) днів до закінчення
терміну \(medication.name)."
            }
            content.sound = .default

            content.userInfo = ["medication_id": medication.id.uuidString]

            var dateComponents = Calendar.current.dateComponents([.year, .month,
.day], from: triggerDate)
            dateComponents.hour = 10
            dateComponents.minute = 0

            let trigger = UNCalendarNotificationTrigger(dateMatching:
dateComponents, repeats: false)

            let requestIdentifier = "\(medication.id.uuidString)_\(daysBefore)"
            let request = UNNotificationRequest(identifier: requestIdentifier,
content: content, trigger: trigger)

            UNUserNotificationCenter.current().add(request)
```

```
        }
    }


    static func cancelAlerts(for medication: Medication) {
        let identifiers = ["0", "3", "10"].map {
"\(medication.id.uuidString)_\($0)" }
        UNUserNotificationCenter.current().removePendingNotificationRequests(with
Identifiers: identifiers)
    }
}
```

А.3. Код AddMedicineViewModel

```
import Foundation
import SwiftUI
import SwiftData

@Observable
final class AddMedicineViewModel {
    // Вхідні дані з полів вводу UI
    var name: String = ""
    var details: String = ""
    var dateString: String = ""
    var storageLocation: String = ""

    // Стан вибору зображення
    var capturedImage: UIImage? = nil

    // Стан валідації для відображення помилок в UI
    var errorMessage: String? = nil
    var isShowingError: Bool = false

    // Посилання на контекст бази даних (ін’єктується через ініціалізатор)
    private var modelContext: ModelContext

    init(modelContext: ModelContext) {
        self.modelContext = modelContext
    }

    /// Основна функція збереження нового препарату
    func saveMedication() -> Bool {
        // 1. Скидання попередніх помилок
        errorMessage = nil

        // 2. Валідація назви
        guard !name.trimmingCharacters(in: .whitespaces).isEmpty else {
            showError("Будь ласка, введіть назву препарату")
            return false
        }

        // 3. Валідація та парсинг дати через ValidationService
        guard let parsedDate = ValidationService.validateAndParse(dateString)
else {
            showError("Некоректний формат дати. Використовуйте дд.мм.рррр")
            return false
        }
```

```
        // 4. Додаткова бізнес-перевірка дати
        if !ValidationService.isReasonableDate(parsedDate) {
            showError("Вказано нереальну дату")
            return false
        }

        // 5. Обробка зображення (якщо воно є)
        var imagePath: String? = nil
        if let image = capturedImage {
            // Зберігаємо фізичний файл і отримуємо його ім'я
            imagePath = StorageService.saveImage(image)
        }

        // 6. Створення об'єкта моделі
        let newMedication = Medication(
            name: name,
            details: details,
            expiryDate: parsedDate,
            imagePath: imagePath,
            storageLocation: storageLocation
        )

        // 7. Збереження в SwiftData
        modelContext.insert(newMedication)

        do {
            try modelContext.save()
            NotificationService.scheduleExpiryAlerts(for: newMedication)
            return true
        } catch {
            showError("Помилка збереження в базу даних")
            return false
        }
    }

    private func showError(_ message: String) {
        self.errorMessage = message
        self.isShowingError = true
    }
}
```

А.4. Код DateInputField:

```
import SwiftUI

struct DateInputField: View {
    @Binding var dateString: String

    var body: some View {
        TextField("дд.мм.рррр", text: $dateString)
            .keyboardType(.numberPad)
            // onChange для спостереження за введенням
            .onChange(of: dateString) { oldValue, newValue in
                // 1. Якщо користувач видаляє символи, не заважаємо йому
                if newValue.count < oldValue.count {
                    return
                }
```

```
                // 2. Обмежуємо максимальну довжину
                if newValue.count > 10 {
                    dateString = String(newValue.prefix(10))
                    return
                }

                // 3. Логіка автоматичних крапок
                let filtered = newValue.filter { "0123456789".contains($0) }
                var result = ""

                for (index, char) in filtered.enumerated() {
                    // Ставимо крапки після 2-ї та 4-ї цифри
                    if index == 2 || index == 4 {
                        result.append(".")
                    }
                    result.append(char)
                }

                // 4. Оновлюємо значення тільки якщо воно змінилося,
                // щоб уникнути нескінченного циклу оновлення
                if result != newValue {
                    dateString = result
                }
            }
    }
}
```

A.5: Код ExpiryStatus:

```
import SwiftUI

enum ExpiryStatus: String, CaseIterable {
    case active
    case warning
    case expired

    // Текстове представлення для інтерфейсу
    var label: String {
        switch self {
        case .active: return "Придатні"
        case .warning: return "Уважно"
        case .expired: return "Протерміновані"
        }
    }

    // Колірна схема для індикаторів
    var color: Color {
        switch self {
        case .active: return .green
        case .warning: return .orange
        case .expired: return .red
        }
    }

    // Іконка
    var iconName: String {
        switch self {
        case .active: return "checkmark.circle.fill"
```

```
        case .warning: return "exclamationmark.triangle.fill"
        case .expired: return "xmark.octagon.fill"
        }
    }
}
```

A.6 Код MedicineRowView

```
import SwiftUI

struct MedicineRowView: View {
    let medication: Medication

    var body: some View {
        HStack(spacing: 15) {
            // 1. Блок зображення
            Group {
                if let fileName = medication.imagePath,
                   let image = StorageService.loadImage(fileName: fileName) {
                    Image(uiImage: image)
                        .resizable()
                        .scaledToFill()
                } else {
                    ZStack {
                        Color.gray.opacity(0.1)
                        Image(systemName: "pill.fill")
                            .foregroundColor(.gray)
                    }
                }
            }
            .frame(width: 60, height: 60)
            .cornerRadius(12)
            .clipped()

            // 2. Інформаційний блок
            VStack(alignment: .leading, spacing: 4) {
                Text(medication.name)
                    .font(.headline)
                    .lineLimit(1)

                Text("Придатний до: \(medication.expiryDate, format:
.dateTime.day().month().year())").environment(\.locale, Locale(identifier:
"uk_UA"))
                    .font(.subheadline)
                    .foregroundColor(.secondary)
            }

            Spacer()

            // 3. Індикатор статусу
            StatusBadge(status: medication.status)
        }
        .padding(.vertical, 4)
    }
}
```

ДОДАТОК Б

(обов’язковий)

Презентаційні матеріали

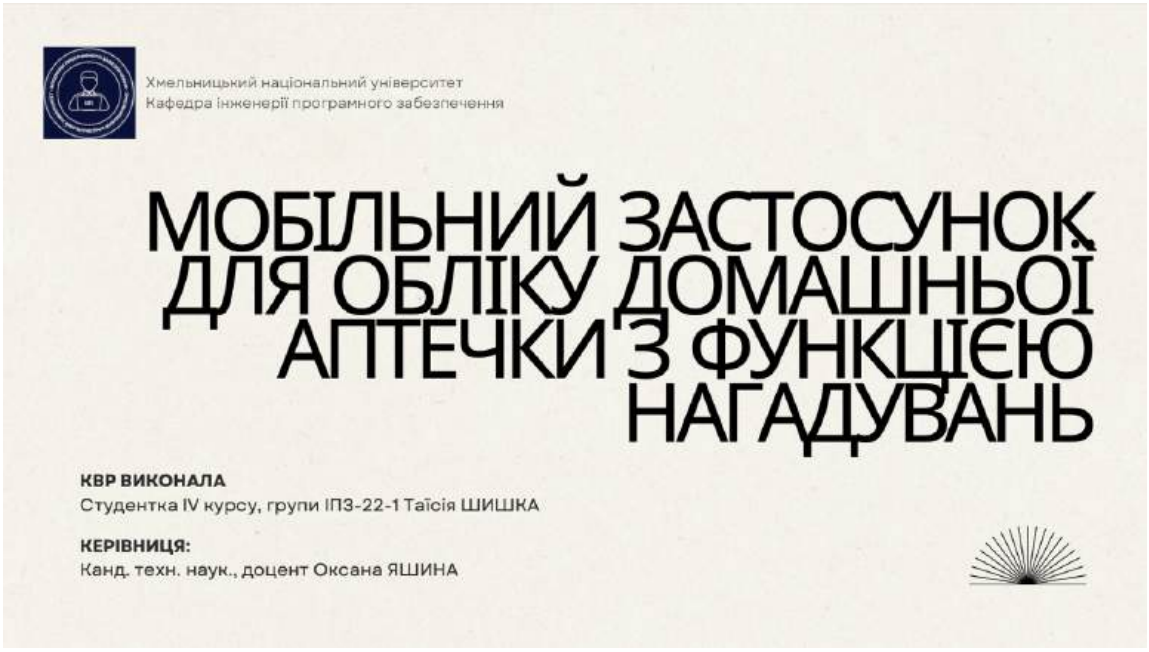


Рисунок В.1 – Титульний слайд

Рисунок В.2 – Слайд «Актуальність теми»

Рисунок В.3 – Слайд «Мета та завдання»

Рисунок В.4 – Слайд «Змістовний аналіз предметної області, її структурних та функціональних особливостей»

Рисунок В.5 – Слайд «Аналіз наявного програмно-технічного забезпечення»

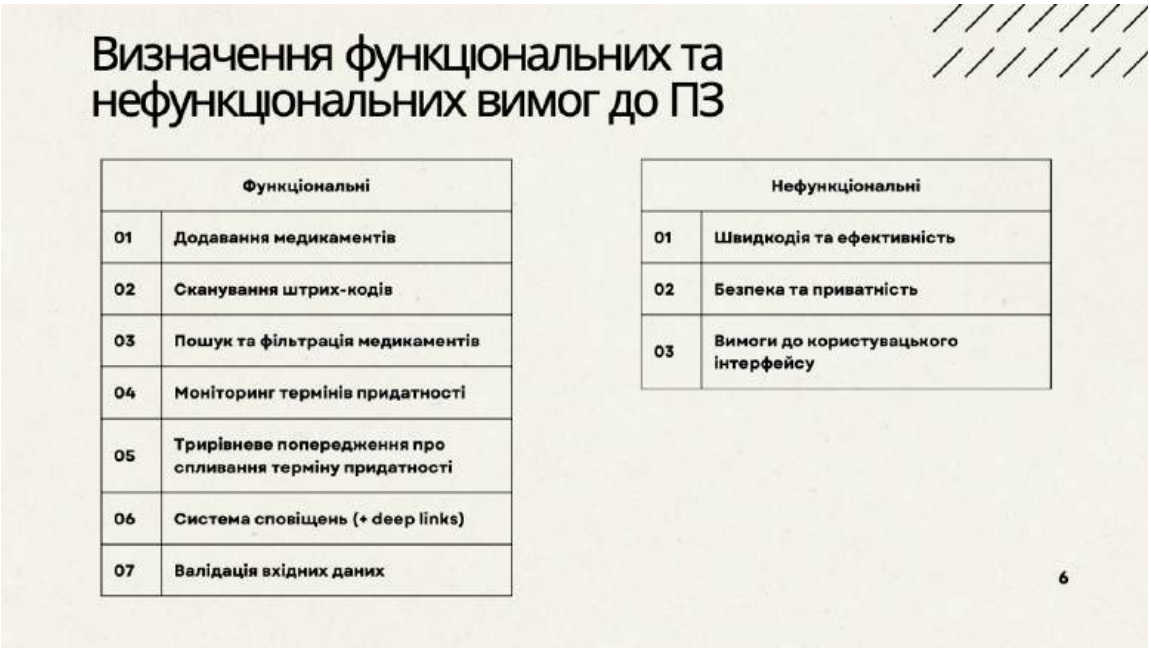


Рисунок В.6 – Слайд «Визначення функціональних та нефункціональних вимог до ПЗ»

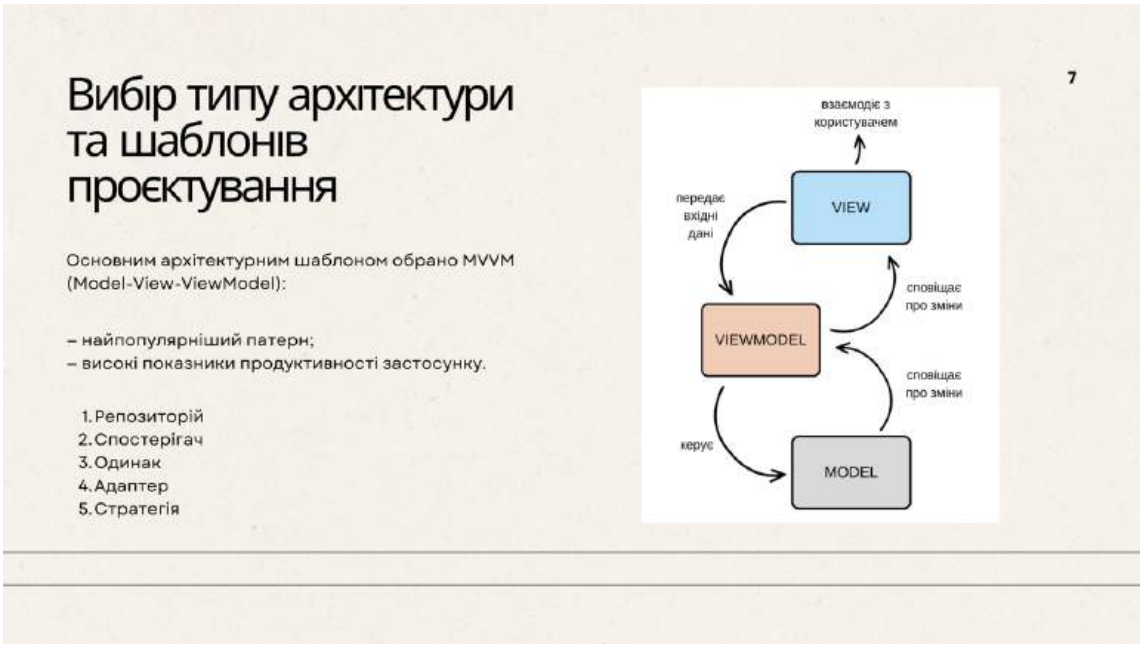


Рисунок В.7 – Слайд «Вибір типу архітектури та шаблонів проєктування»

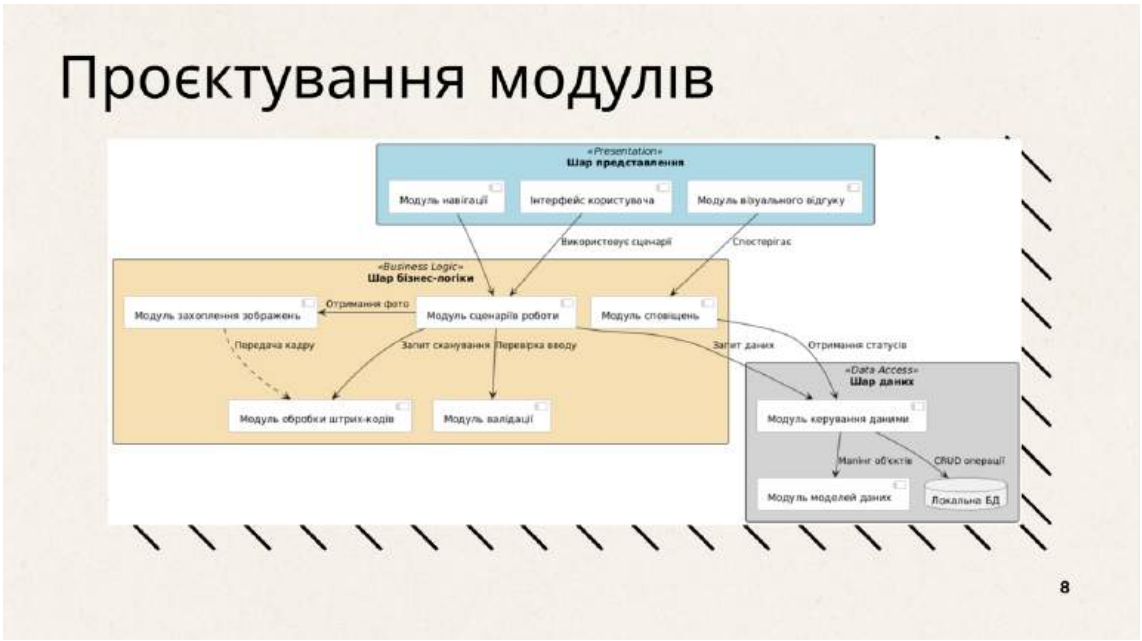


Рисунок В.8 – Слайд «Проєктування модулів»

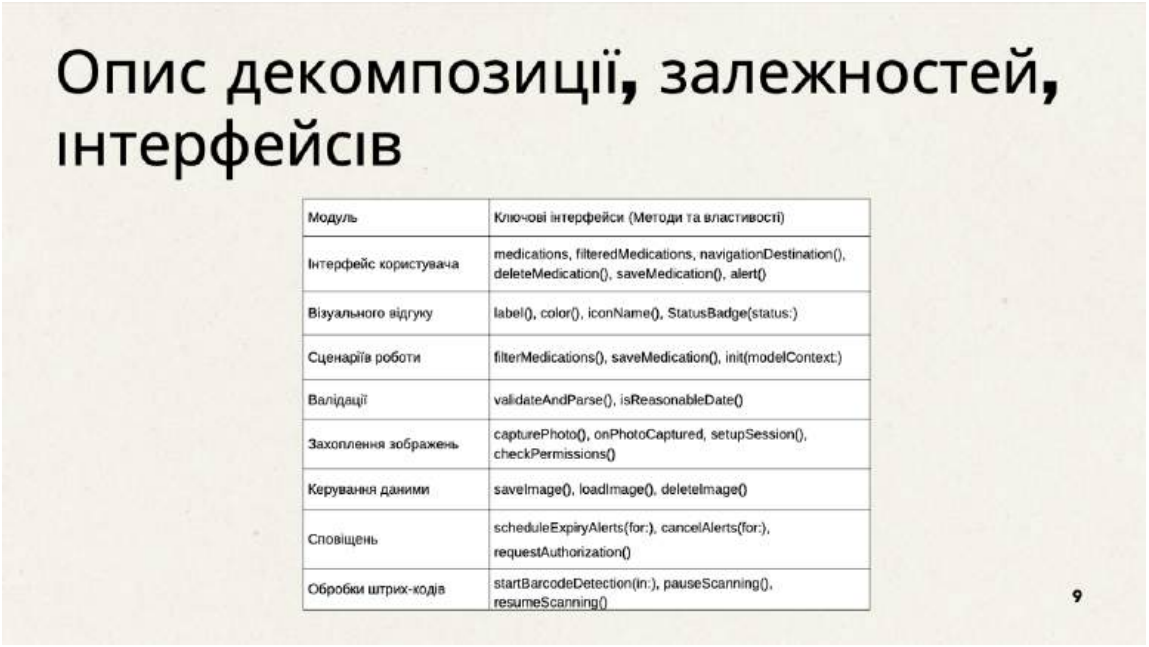


Рисунок В.9 – Слайд «Опис декомпозиції, залежностей, інтерфейсів»

Рисунок В.10 – Слайд «Аналіз та вибір технологій»

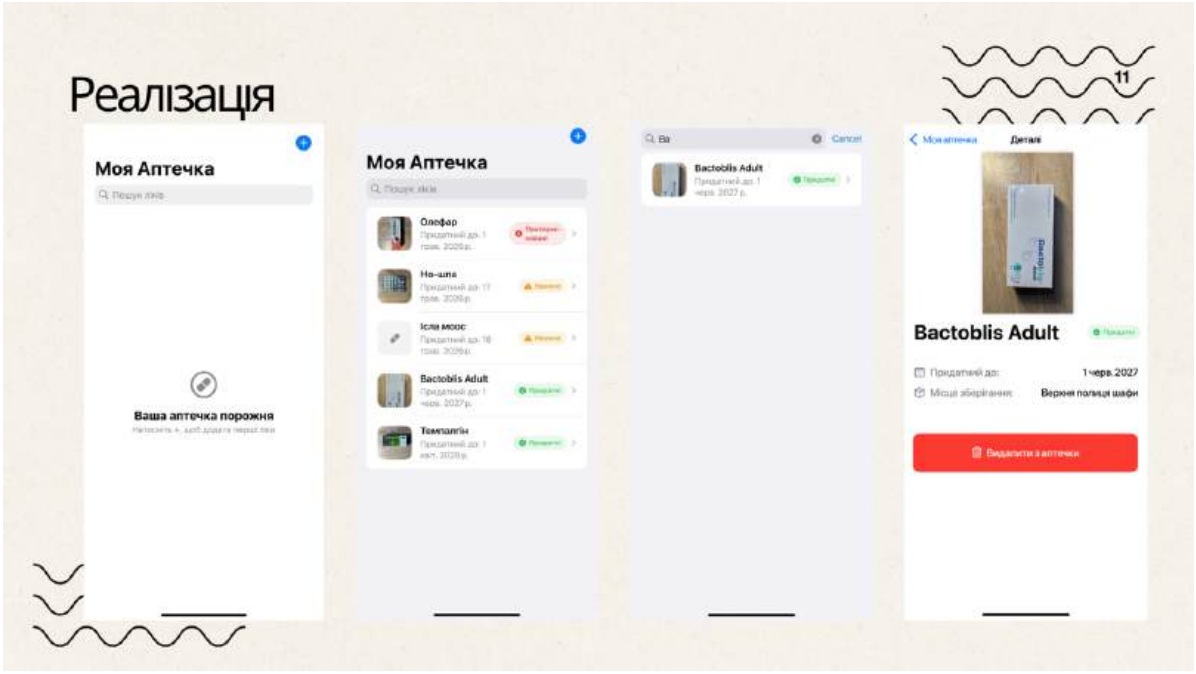


Рисунок В.11 – Слайд «Реалізація»

Рисунок В.12 – Слайд «Тестування ПЗ»

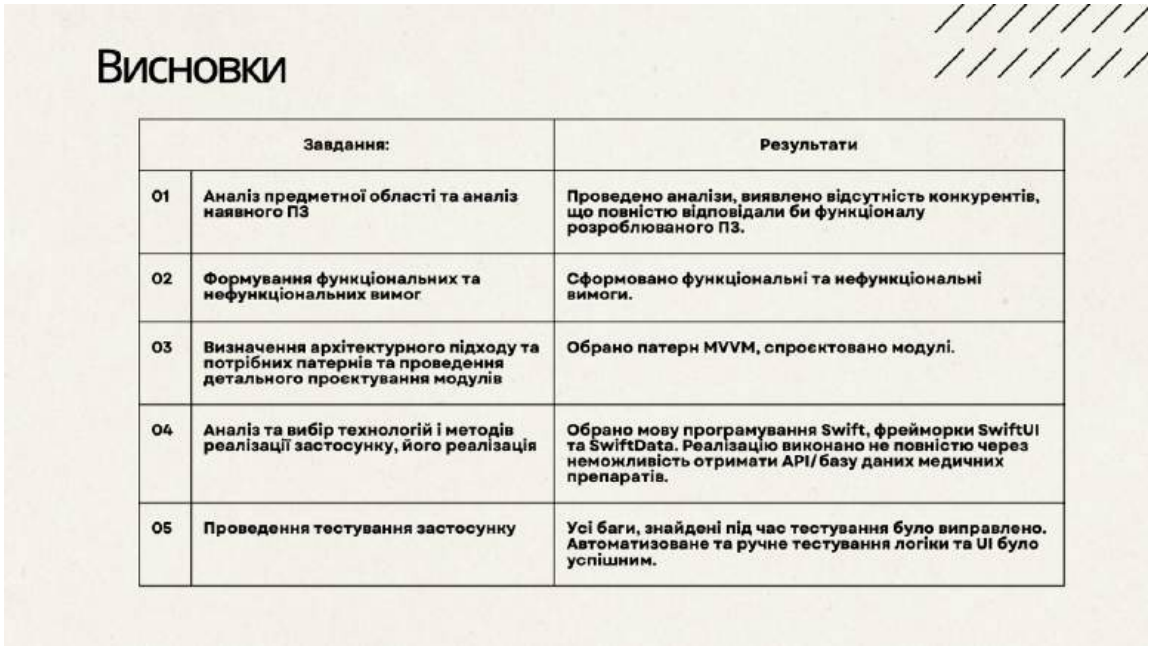
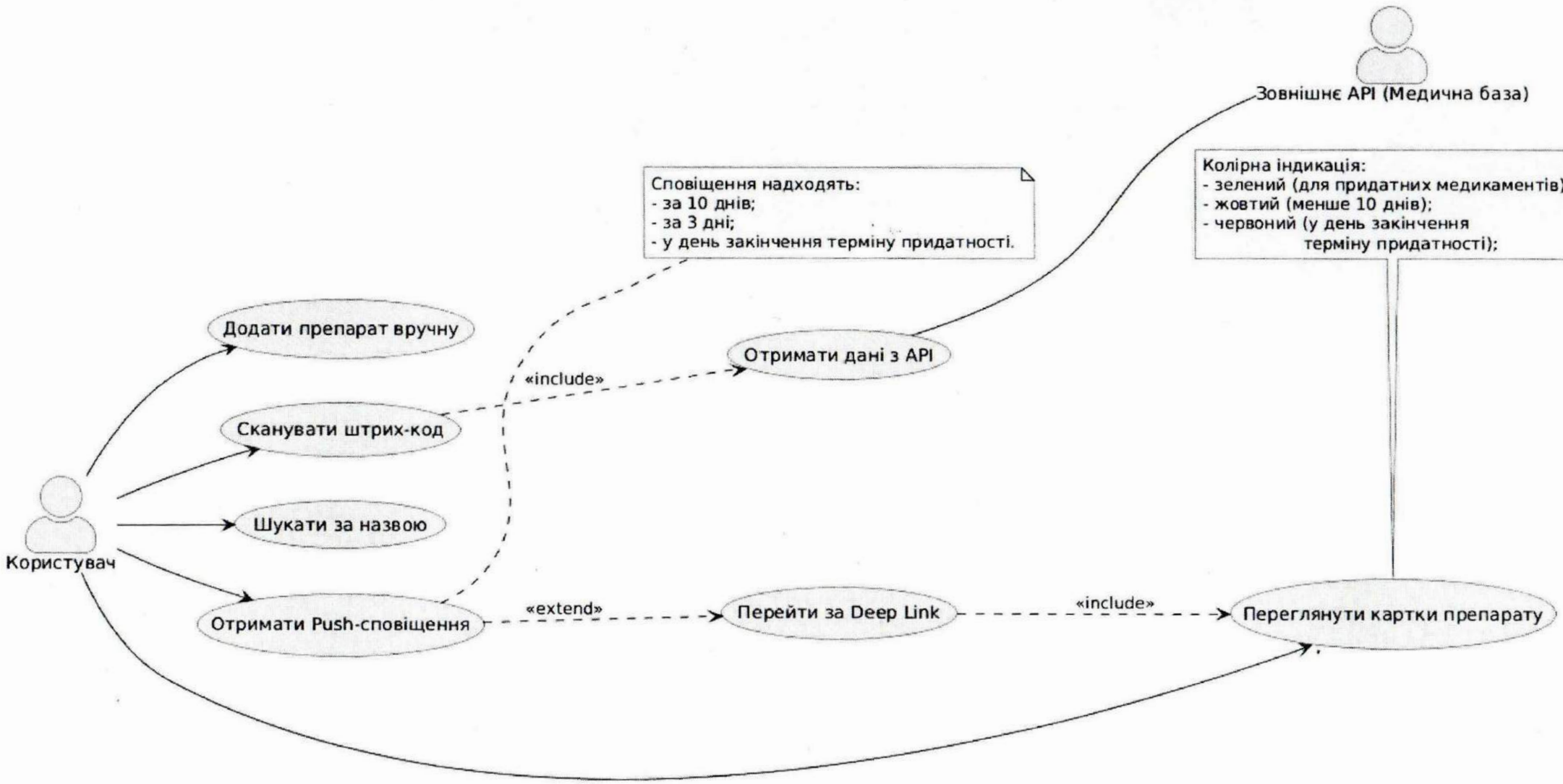


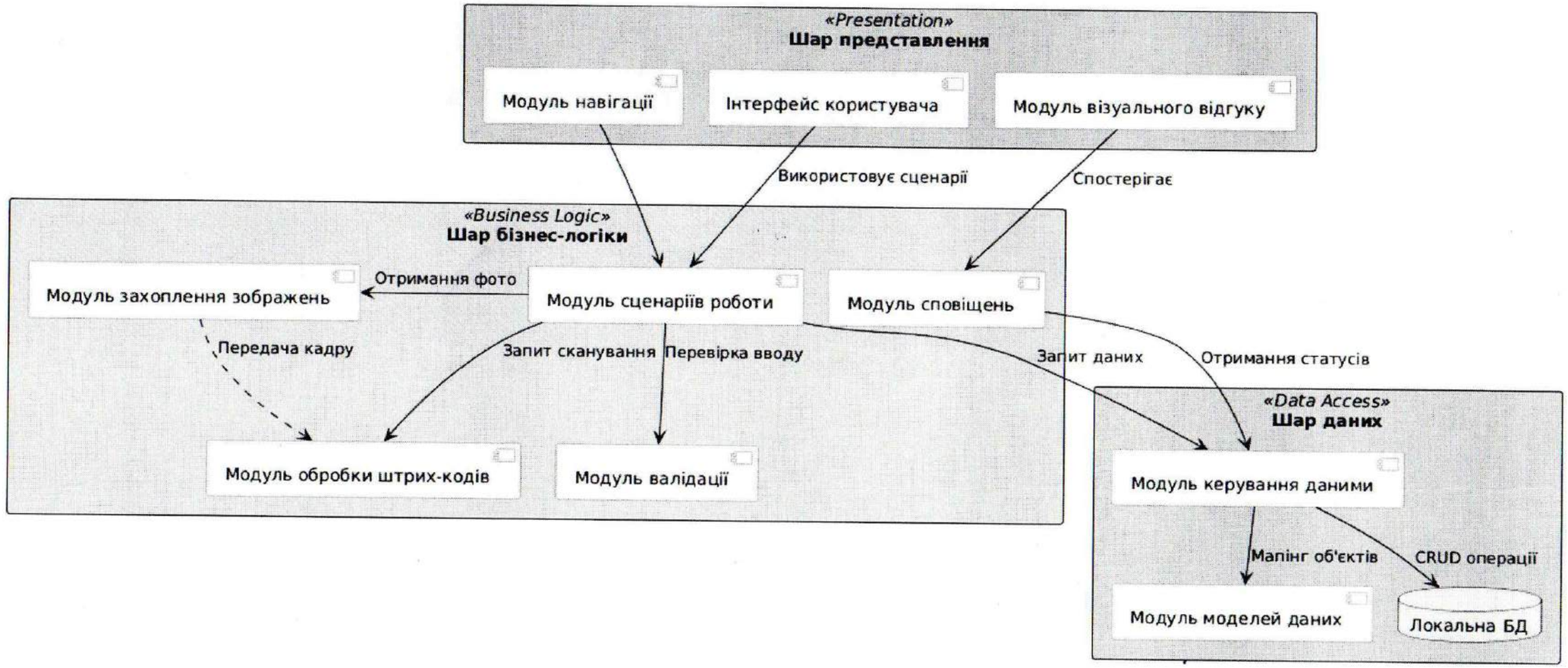
Рисунок В.13 – Слайд «Висновки»

Рисунок В.14 – Слайд «Дякую за увагу»

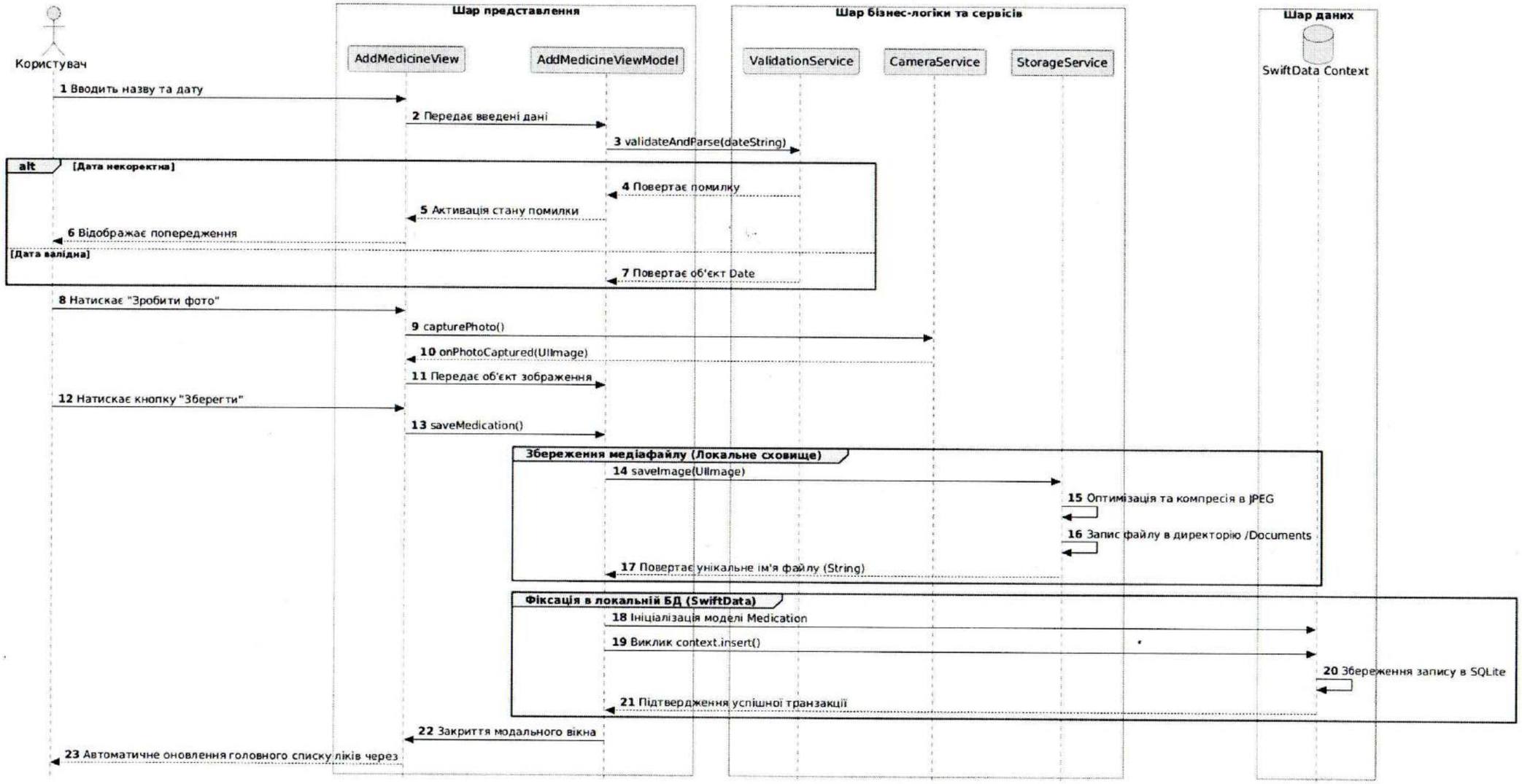
ГРАФІЧНА ЧАСТИНА



					КвРІПЗ.2201117.01.22.Е8		
Змн.	Арк	№ докум.	Підпис	Дата	Мобільний застосунок для обліку домашньої аптечки з функцією нагадувань	Літ.	Маса	Масштаб
Розробив		Шишка Т.В.		15.05				
Керівник		Яшина О.М.		15.05				
					Діаграма варіантів використання	Аркуш 1	Аркушів 3	
Н. Контр.		Форкун Ю. В.		20.05		ХНУ,ІПЗ-22-1		
Затверд.		Бедратюк Л. П.		23.05				



					КвРІПЗ.2201117.01.22.Е8		
Змн.	Арк	№ докум.	Підпис	Дата	Мобільний застосунок для обліку домашньої аптечки з функцією нагадувань Діаграма міжмодульної взаємодії	Літ.	Маса / Масштаб
Розробив		Шишка Т.В.		15.05			
Керівник		Яшина О.М.		15.05		Аркуш 2	Аркушів 3
Н. Контр.		Форкун Ю. В.		23.05		ХНУ,ІПЗ-22-1	
Затверд.		Бедратюк Л. П.		23.05			



					КвРІПЗ.2201117.01.22.Е8		
Змн.	Арк	№ докум.	Підпис	Дата	Мобільний застосунок для обліку домашньої аптечки з функцією нагадувань Діаграма послідовності	Літ.	Маса / Масштаб
Розробив		Шишка Т.В.		25.05			
Керівник		Яшина О.М.		25.05		Аркуш 3	Аркушів 3
Н. Контр.		Форкун Ю. В.		25.05		ХНУ,ІПЗ-22-1	
Затверд.		Бедратюк Л. П.		25.05			

Завідувачу кафедри інженерії програмного
забезпечення проф. Леоніду БЕДРАТЮКУ
здобувача вищої освіти
Шишки Таїсії Віталіївни
факультет ІТ, IV курс, група ІПЗ-22-1

ЗАЯВА

З правилами чинного Положення про систему забезпечення академічної доброчесності в Хмельницькому національному університеті, згідно з яким виявлення академічного плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів дисциплінарної та академічної відповідальності, ознайомлена. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність академічного плагіату оповіщена та надаю свою згоду на обробку й збереження університетом моєї роботи в інституційному репозитарії Хмельницького національного університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-обчислювального комплексу StrikePlagiarism та/або програмно-технічного засобу AntiPlagiarism і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення текстових збігів у роботах.

Робота надається для перевірки в електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

10.05.2026
дата

підпис

Anti-Plagiarism (http://ap.km.ua) v-16.718

Максимальне співпадіння з одним документом 14.0%

Словники перевірки: UA, US, RU. **Помилок в документах: 14%**

ID: 272082 Назва: Мобільний застосунок для обліку домашньої аптечки з функцією нагадувань Додано в БД: 2026-05-25 Автора: Таїсія ШИШКА Керівники: канд. техн. наук, доцент Оксана ЯШИНА Консультанти: Опоненти:	Документ		Сумарний збіг по Базі Даних	
	Символи	Лексеми	Символи	Лексеми
	89380	609	13297 (15%)	112 (18%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми
269624	Назва: Переддипломна практика Додано в БД: 2026-03-02 Автора: Таїсія ШИШКА Керівники: Яшина О. М., канд. техн. наук, доцент Консультанти: Опоненти:	12124 (14.0%)	97 (16.0%)

Протокол аналізу звіту подібності науковим керівником

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Таїсія ШИШКА

Співавтор:

Назва: Мобільний застосунок для обліку домашньої аптечки з функцією нагадувань

Науковий керівник: канд. техн. наук, доцент Оксана ЯШИНА

Підрозділ: Кафедра інженерії програмного забезпечення

Коефіцієнт подібності 1:1.29%

Коефіцієнт подібності 2:0%

Мікропробіли: 158

Заміна букв: 11

Інтервали: 0

Білі знаки: 0

Дата створення звіту: 2026-05-25 01:36:45.0

Після аналізу Звіту подібності констатую наступне:

☑ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

Дата 25.05.26

експерт Ворона Ю.В.

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ
освітнього ступеня «Бакалавр»

Дипломник: Шишка Таїсія Віталіївна

Тема: «Мобільний застосунок для обліку домашньої аптечки з функцією нагадувань»

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг кваліфікаційної роботи:

Кількість листів креслень 3; кількість сторінок записки 68

1.Короткий зміст пояснювальної записки та прийнятих рішень: у кваліфікаційній роботі досліджено і проаналізовано предметну область, визначено функціональні та нефункціональні вимоги. Було проведено аналіз існуючих програм на ринку, розглянуто їх переваги і недоліки, та доведено актуальність розробки нового програмного забезпечення. Було розглянуто інструменти для реалізації спроєктованих рішень, після чого було безпосередньо створено програмне забезпечення. Також було проведено тестування програми, за результатами якого доведено, що розроблене програмне забезпечення працює коректно та готове до експуатації.

2. Висновок про відповідність роботи поставленому завданню: кваліфікаційна робота виконана відповідно до поставленого завдання та з дотриманням всіх вимог.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки та передових методів роботи: у вступі доведено актуальність теми, визначено мету та завдання дипломного проектування. У першому розділі проведено аналіз предметної області, розглянуто існуючі рішення та визначені функціональні і нефункціональні вимоги до розроблюваного програмного забезпечення. У другому розділі було обрано патерни: проєктування відбувалося на базі шаблону MVVM. Окрім того, було спроєктовано модулі, їхні інтерфейси та описано їх взаємодію. У третьому розділі було описано створення програмного продукту, написано інструкцію користувача та проведено модульне, автоматизоване та мануальне тестування системи, в результаті було підтверджено коректну роботу програми.

4. Позитивні сторони роботи: тематика кваліфікаційної роботи є актуальною, оскільки на сьогодні мобільні застосунки для контролю медикаментів переважно надають функціонал сфокусований на графіку прийому. Також було застосовано новітні технології та актуальні архітектурні рішення на базі шаблону MVVM.

5. Негативні сторони роботи: у роботі архітектура застосунку орієнтована лише на локальне збереження даних на пристрої. Було б доцільно впровадити хмарну синхронізацію для захисту інформації від втрати. Крім того, у фінальній версії ПЗ не вдалося реалізувати автоматизоване додавання препаратів через сканування штрих-кодів.

6. Оцінка графічного оформлення та пояснювальної: графічне оформлення виконано відповідно до теми кваліфікаційної роботи та подано у вигляді діаграм і рисунків. Пояснювальна записка оформлена згідно вимог чинних стандартів.

7. Відгук про кваліфікаційну роботу в цілому: кваліфікаційна робота заслуговує позитивної оцінки. Матеріал пояснювальної записки структурований, послідовний, чіткий та простий, що дозволяє чітко зрозуміти викладений матеріал у рамках тематики проектування. Графічний матеріал дає можливість наочно побачити деталі проектування системи.

8. Інші зауваження

9. Оцінка кваліфікаційної роботи: кваліфікаційна робота виконана у повному обсязі, відповідає поставленій задачі та заслуговує на оцінку «добре».

РЕЦЕНЗЕНТ Нічепорук Андрій Олександрович, кандидат технічних наук, доцент кафедри комп'ютерної інженерії та інформаційних систем

"28" 05 2026 р. _____
(підпис)

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ
КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатами звіту/звітів перевірки роботи, продукованими програмно-технічним засобом (ами), на наявність текстових збігів.
Назва кваліфікаційної роботи: «Мобільний застосунок для обліку домашньої аптечки з функцією нагадувань»
Автор: Шишка Таїсія Віталіївна
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Спеціальність: 121 – Інженерія програмного забезпечення
Науковий керівник: Яшина Оксана Миколаївна, кандидат технічних наук, доцент
Після аналізу звіту/звітів зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається до захисту.	**відповідає**
2	Виявлені запозичення не є академічним плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована.	
3	Виявлені запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Виявлені запозичення частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнуті. Робота може бути допущена до захисту після того, як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття текстових запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:
Запозичення, виявлені у роботі, є законними і не є плагіатом, оскільки:
1) у тексті кваліфікаційної роботи системою перевірки на плагіат Anti-Plagiarism виявлено схожість з деякими документами у частині загальновживаних обов'язкових словосполучень у стандартних бланках, у структурі змісту, назвах розділів/підрозділів, рамках форм, у назвах та URL-адресах публікацій переліку джерел посилання;
2) запозичення, виявлені у тексті роботи, є фрагментарними.
Максимальний обсяг запозичень, визнаний системою Anti-Plagiarism, складає 14.0% з одного джерела. Загальна сумарна подібність у базі даних складає 15% за символами та 18% за лексемами. Крім того, за результатами додаткового аналізу коефіцієнт подібності 1 становить 1.29% коефіцієнт подібності 2 – 0%. Виявлено 158 мікропробілів, проте не виявлено зайвих білих знаків або маніпуляцій з інтервалами. З урахуванням наведених обґрунтувань, відповідає характеру теми і свідчить на користь кваліфікаційної роботи.

Дата 27 травня 2026 р.

Завідувач кафедри _____ Леонід БЕДРАТЮК
Гарант освітньої програми _____ Леонід БЕДРАТЮК
Керівник кваліфікаційної роботи _____ Оксана ЯШИНА