

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

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка iOS-застосунку для щоденної
саморефлексії та отримання персональних рекомендацій на
основі штучного інтелекту»

на здобуття освітнього ступеня бакалавра

зі спеціальності 121 Інженерія програмного забезпечення

освітньо-професійної програми «Інженерія програмного забезпечення»

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

_____ Всеволод ЧИКАЛО
(підпис)

Виконав: здобувач вищої освіти групи ПД-43

_____ Всеволод ЧИКАЛО

Керівник: _____ Віталій ЗАЛИВА
доктор філософії (PhD)

Рецензент: _____

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

Інженерії програмного забезпечення

_____ Ірина ЗАМРІЙ

« ____ » _____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Чикалу Всеволоду Денисовичу

1. Тема кваліфікаційної роботи: «Розробка iOS-застосунку для щоденної саморефлексії та отримання персональних рекомендацій на основі штучного інтелекту»

керівник кваліфікаційної роботи доктор філософії (PhD) Віталій ЗАЛИВА, затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026 року № 51.

2. Строк подання кваліфікаційної роботи «29» травня 2026 року.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про процес щоденної саморефлексії та психологічні аспекти ведення особистого щоденника, відомості про методології структурованих ретроспектив у командній розробці ПЗ, технічна документація з мови програмування Swift, фреймворків SwiftUI та SwiftData, а також документація OpenAI API.

4. Зміст розрахунково-пояснювальної записки

1. Аналіз предметної області та постановка задачі. Огляд існуючих рішень.

2. Засоби реалізації — обґрунтування вибору технологічного стека.

3. Проектування та реалізація iOS-застосунку Diary AI.

4. Демонстрація застосунку.

5. Перелік графічного матеріалу: презентація

1. Аналіз аналогів.

2. Вимоги до програмного забезпечення.

3. Програмні засоби реалізації.

4. Архітектура застосунку.

5. Діаграма варіантів використання.

6. Карта екранів.

7. Екранні форми.

8. Апробація результатів дослідження.

6. Дата видачі завдання «23» лютого 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	23.02-28.02.2026	
2	Аналіз та дослідження існуючих аналогів	01.03-07.03.2026	
3	Огляд існуючих методів та технологій для розробки iOS-застосунків з інтеграцією штучного інтелекту	08.03 – 14.03.2026	
4	Проектування застосунку для щоденної саморефлексії з AI-аналізом записів	15.03 – 28.03.2026	
5	Програмна реалізація застосунку	29.03 – 19.04.2026	
6	Тестування застосунку	20.04 – 26.04.2026	
7	Оформлення роботи: вступ, висновки, реферат	27.04 – 03.05.2026	
8	Розробка демонстраційних матеріалів	04.05 – 10.05.2026	
9	Попередній захист роботи	11.05-22.05.2026	

Здобувач вищої освіти

Всеволод ЧИКАЛО

Керівник

кваліфікаційної роботи

Віталій ЗАЛИВА

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 83 стор., 1 табл., 27 рис., 24 джерел.

Мета роботи — підвищення ефективності процесу щоденної саморефлексії користувача шляхом структурованого ведення особистого щоденника та автоматизованого аналізу записів з використанням штучного інтелекту для формування персоналізованих рекомендацій щодо покращення якості життя.

Об'єкт дослідження — процес щоденної саморефлексії та самоаналізу особистості.

Предмет дослідження — засоби, технології та підходи щодо розробки iOS-застосунку для ведення щоденних записів та їх інтелектуального аналізу для виявлення позитивних і проблемних аспектів життя та формування персональних рекомендацій..

Короткий зміст роботи: У роботі розглянуто проблематику цифрових щоденників та існуючі підходи до автоматизованого аналізу персональних текстових даних. Проведено огляд сучасних мобільних застосунків аналогічного призначення, зокрема Day One, Reflectly, Journey та Daylio, виявлено їхні функціональні обмеження. Обґрунтовано доцільність комбінування методики структурованих ретроспективних запитань, адаптованих з підходу Scrum-ретроспектив, із обробкою записів за допомогою великої мовної моделі. Сформовано вимоги до програмного забезпечення, спроєктовано архітектуру за патерном MVVM та реалізовано застосунок із використанням мови Swift, фреймворку SwiftUI, локального сховища SwiftData та зовнішнього сервісу OpenAI API. Розроблений застосунок Diary AI підтримує реєстрацію користувача, ведення щоденних записів за чотирма структурованими запитаннями, перегляд історії, формування тижневого AI-аналізу з виокремленням позитивних аспектів, труднощів і персональних

рекомендацій, а також налаштування нагадувань через push-сповіщення. Виконано функціональне тестування основних сценаріїв роботи та перевірку адаптивності інтерфейсу під різні моделі пристроїв Apple.

Сферою застосування розробленого продукту є мобільні застосунки категорії здоров'я та особистого розвитку, зокрема цифрові щоденники й інструменти для ментального самоконтролю.

КЛЮЧОВІ СЛОВА: IOS-ЗАСТОСУНОК, SWIFT, SWIFTUI, SWIFTDATA, ШТУЧНИЙ ІНТЕЛЕКТ, OPENAI API, ЦИФРОВИЙ ЩОДЕННИК, САМОРЕФЛЕКСІЯ, MVVM, МОБІЛЬНА РОЗРОБКА.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	10
ВСТУП	11
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ	15
1.1 Цифрові щоденники як інструмент саморефлексії	15
1.2 Застосування великих мовних моделей для аналізу персональних текстових даних	17
1.3 Цільова аудиторія застосунку	19
1.4 Аналіз існуючих застосунків-аналогів	20
1.4.1 Day One	21
1.4.2 Reflectly	22
1.4.3 Journey	23
1.4.4 Daylio	24
1.4.5 Stoic	26
1.4.6 Порівняльний аналіз існуючих рішень	27
1.5 Формування вимог до програмного забезпечення	28
1.5.1 Функціональні вимоги	28
1.5.2 Нефункціональні вимоги	30
2 ЗАСОБИ РЕАЛІЗАЦІЇ	32
2.1 Платформа iOS та екосистема Apple	32
2.2 Мова програмування Swift	34
2.3 Фреймворк SwiftUI	35
2.4 SwiftData як локальне сховище	37
2.5 Архітектурний патерн MVVM	39
2.6 Інтеграція з OpenAI API	40
2.7 Середовище розробки Xcode	41
2.8.1 UserNotifications	43
2.8.2 CryptoKit	44
3 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ЗАСТОСУНКУ DIARY AI	46
3.1 Загальна архітектура застосунку	46
3.2 Структура моделей даних	48
3.2.1 Модель UserProfile	50
3.2.2 Модель DayEntry	50
3.2.3 Модель WeeklyAnalysis	52
3.3 Реалізація рівня ViewModel	53
3.3.1 AuthViewModel	53
3.3.2 EntryViewModel	55

3.3.3 AIAnalysisViewModel	55
3.4 Реалізація користувацького інтерфейсу	57
3.4.1 Екрани автентифікації	58
3.4.2 Головний екран та навігація	59
3.4.3 Екран створення запису	61
3.4.4 Перегляд історії та деталей запису	62
3.4.5 Екран AI-аналізу та архів	64
3.4.6 Екран налаштувань	67
3.5 Сервісний шар	69
3.5.1 OpenAIService	69
3.5.2 NotificationService	72
3.6 Безпека збереження облікових даних	74
4 ДЕМОНСТРАЦІЯ ТА ТЕСТУВАННЯ	76
4.1 Демонстрація основних сценаріїв роботи	76
4.1.1 Реєстрація нового користувача	76
4.1.2 Створення першого щоденного запису	77
4.1.3 Перегляд історії записів	80
4.1.4 Налаштування OpenAI API ключа та сповіщень	82
4.1.5 Виконання AI-аналізу	84
4.2 Тестування застосунку	86
4.2.1 Тестування модуля криптографічного хешування	87
4.2.2 Тестування модуля роботи з датами	88
4.2.3 Тестування логіки роботи з записами	88
4.2.4 Тестування розбору відповіді AI-аналізу	89
ВИСНОВКИ	90
ПЕРЕЛІК ПОСИЛАНЬ	94
ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ	97
ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ	103

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AI — Artificial Intelligence (штучний інтелект)
API — Application Programming Interface
GPT — Generative Pre-trained Transformer
HTTP — HyperText Transfer Protocol
IDE — Integrated Development Environment
JSON — JavaScript Object Notation
LLM — Large Language Model (велика мовна модель)
MVVM — Model-View-ViewModel
OS — Operating System
REST — Representational State Transfer
SDK — Software Development Kit
SHA — Secure Hash Algorithm
UI — User Interface
URL — Uniform Resource Locator
UX — User Experience

ВСТУП

Актуальність роботи. Сучасний ритм життя, постійне інформаційне навантаження та зростаючий рівень повсякденного стресу формують стійкий запит на інструменти, які допомагають людині краще розуміти власний емоційний стан, відслідковувати динаміку настрою та усвідомлювати причини власних реакцій. Особливо гостро ця проблема стоїть серед молодшої аудиторії — за даними різних досліджень, саме у віковій групі 18–30 років фіксується найбільший приріст випадків емоційного вигорання та тривожних розладів. Практика щоденних записів, відома як ведення особистого щоденника, є одним із найдавніших і водночас найдосліджуваніших методів психологічної саморегуляції. Сучасні дослідження у галузі позитивної психології підтверджують, що регулярна письмова рефлексія сприяє зниженню тривожності, покращенню якості сну та формуванню навичок усвідомленого ставлення до подій власного життя.

З поширенням смартфонів традиційний паперовий щоденник масово трансформувався у формат мобільного застосунку. На ринку iOS наявна низка популярних рішень для ведення цифрових щоденників — Day One, Journey, Reflectly, Daylio. Однак переважна більшість таких продуктів пропонує користувачу вільну форму запису без чіткої методичної основи: людина залишається сам на сам із порожнім полем введення тексту, що часто стає психологічним бар'єром для регулярної практики. Окремо існують застосунки, що пропонують відстеження настрою у вигляді емодзі або числової шкали, проте така фіксація є надто поверхневою та не дає матеріалу для змістовної рефлексії.

Окрема проблема полягає в тому, що навіть за умови регулярного ведення записів користувач рідко повертається до них для аналізу та виявлення повторюваних патернів. Самостійний аналіз десятків текстових нотаток є трудомістким і часто не виконується через брак часу або відсутності

навичок. Поява в широкому доступі великих мовних моделей, таких як GPT-4 та її похідних, відкрила можливість автоматизованого опрацювання таких даних — модель здатна виокремити в текстах ключові теми, повторювані труднощі, позитивні моменти та сформулювати рекомендації природною мовою.

Поєднання двох підходів — структурованих запитань для зниження бар'єру входу та подальшого AI-аналізу для виявлення інсайтів — лежить в основі розробленого у роботі застосунку Diary AI. Запитання для щоденного запису адаптовані з методології Scrum-ретроспектив, яка зарекомендувала себе у сфері командної розробки програмного забезпечення для регулярного аналізу процесів. Перенесення цього підходу у площину індивідуальної рефлексії дозволяє отримувати конкретні, тематично сфокусовані записи замість абстрактного «потoku свідомості».

Окремою складовою актуальності є відсутність на ринку україномовних рішень такого типу. Більшість популярних AI-щоденників орієнтовані на англomовну аудиторію, що ускладнює їх використання для україномовних користувачів — як з огляду на мову інтерфейсу, так і на якість AI-відповідей рідною мовою.

Об'єктом дослідження є процес щоденної саморефлексії та самоаналізу особистості.

Предметом дослідження є засоби, технології та підходи щодо розробки iOS-застосунку для ведення щоденних записів та їх інтелектуального аналізу для виявлення позитивних і проблемних аспектів життя та формування персональних рекомендацій.

Метою роботи є підвищення ефективності процесу щоденної саморефлексії користувача шляхом структурованого ведення особистого щоденника та автоматизованого аналізу записів з використанням штучного інтелекту для формування персоналізованих рекомендацій щодо покращення якості життя.

Для досягнення поставленої мети у роботі вирішуються такі завдання:

- проаналізувати предметну область цифрових щоденників та існуючі підходи до автоматизованого аналізу персональних текстових даних;
- виконати огляд популярних мобільних застосунків-аналогів, виявити їхні сильні сторони та функціональні прогалини;
- сформулювати функціональні та нефункціональні вимоги до програмного забезпечення;
- обґрунтувати вибір технологічного стека для нативної розробки під платформу iOS;
- спроектувати архітектуру застосунку за патерном MVVM із локальним сховищем даних;
- реалізувати ключові модулі застосунку: автентифікацію, ведення записів, перегляд історії, тижневий AI-аналіз, систему сповіщень та налаштування;
- інтегрувати застосунок із зовнішнім сервісом OpenAI API для виконання аналітичних запитів до мовної моделі;
- продемонструвати роботу реалізованого застосунку на основних сценаріях використання.
- виконати модульне тестування ключових компонентів реалізованого застосунку.

Наукова новизна роботи полягає у поєднанні методики структурованих ретроспективних запитань, адаптованих із практики Scrum-ретроспектив, із автоматизованим тижневим аналізом записів засобами великої мовної моделі.

Такий підхід відрізняється від існуючих рішень одночасною наявністю чіткої структури введення даних та змістовної AI-обробки, що формує для користувача матеріал для рефлексії у вигляді трьох тематичних блоків: позитивні сторони періоду, труднощі та персональні рекомендації.

Додатково запропоноване рішення орієнтоване на україномовну аудиторію, що становить окремий аспект новизни порівняно з наявними англomовними AI-щоденниками.

Практична значущість роботи полягає у створенні готового до використання нативного iOS-застосунку, який може застосовуватися широким колом користувачів для регулярної самопостережувальної практики. Реалізована архітектура та підхід до інтеграції з мовною моделлю можуть бути використані як основа для подальших розширень — додавання групових режимів, синхронізації між пристроями, інтеграції з інструментами здоров'я Apple HealthKit, а також для адаптації під інші платформи.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Цифрові щоденники як інструмент саморефлексії

Саморефлексія — це свідомо робота людини над власними думками, почуттями, вчинками й переживаннями, мета якої — краще зрозуміти себе та визначити напрямки подальшого розвитку. На відміну від спонтанного обдумування, структурована саморефлексія передбачає регулярне повернення до конкретних запитань про власне життя у письмовій або мовленнєвій формі. Саме структурованість і регулярність відрізняють її від просто «думання про себе»: коли записи накопичуються протягом тижнів або місяців, у них починають виявлятися повторювані поведінкові й мисленнєві патерни, які людина в моменті часто не помічає.

Ведення щоденника як практика самоаналізу має багатовікову історію — від античних філософських «гіпомнем» Марка Аврелія до особистих нотатників видатних науковців, письменників і політиків XIX–XX століть. Сучасна психологія повернулася до цієї практики у форматі експериментально перевірених методик. Дослідження у галузі позитивної психології, експресивного письма та когнітивно-поведінкової терапії стабільно підтверджують, що письмова рефлексія стресових подій знижує рівень тривожності, допомагає опрацьовувати негативні емоції та підвищує загальну стресостійкість [1]. Поза суто терапевтичним ефектом регулярна письмова рефлексія сприяє покращенню якості сну, формуванню навичок усвідомленого ставлення до подій та підвищенню рівня життєвого задоволення.

Окремо заслуговує на увагу так званий ефект самодистанціювання — психологічний феномен, при якому людина дивиться на власну ситуацію ніби з боку, з позиції стороннього спостерігача. Дослідження показують, що такий погляд знижує емоційне «провалювання» в проблему, дозволяє формувати

більш зважене ставлення до події та сприяє пошуку конструктивних рішень [2]. Письмова форма рефлексії підтримує цей ефект краще, ніж усне обдумування — сам акт викладення думки на «папері» вимагає її структурування й уточнення.

Важливим аспектом є те, що різні форми рефлексії дають різний ефект. Просте описове ведення щоденника — фіксація фактів про події дня — працює слабше, ніж відповіді на конкретні аналітичні запитання. Останній підхід змушує людину шукати причинно-наслідкові зв'язки, виокремлювати ключові моменти й формулювати власну позицію щодо них. Саме тому в сучасних методиках психологічного консультування дедалі частіше застосовуються структуровані шаблони щоденникового запису [1, 3, 4].

Із поширенням смартфонів традиційний паперовий щоденник масово трансформувався у формат мобільного застосунку. Перенесення практики на мобільний пристрій має очевидні переваги: смартфон постійно з користувачем, тож запис можна зробити в будь-який момент і в будь-якому місці; цифрова форма забезпечує миттєвий пошук по архіву записів; механізми push-сповіщень підтримують регулярність практики через автоматичні нагадування. Водночас цифровий формат відкриває принципово нові можливості, недоступні паперовому щоденнику: автоматизований аналіз накопичених текстів, виявлення статистичних патернів настрою, інтеграція з іншими сервісами здоров'я.

Незважаючи на наявність значної кількості мобільних застосунків для ведення щоденника, більшість з них залишаються в парадигмі «цифрового аналога паперового зошита» — пропонують користувачу порожнє поле введення тексту й набір тегів. Фактичний зсув у бік структурованих методик відбувається повільно, а інтеграція з технологіями штучного інтелекту лише починає з'являтися в окремих продуктах. Саме ця комбінація — структурованість + AI-аналіз — і складає основу пропонованого у роботі рішення.

1.2 Застосування великих мовних моделей для аналізу персональних текстових даних

Технологія великих мовних моделей (Large Language Models, LLM) — одна з найбільш помітних ІТ-новацій останніх років. Архітектура трансформерів, представлена у 2017 році [5], стала основою для цілої низки моделей, серед яких GPT-3 [6], GPT-4 [7], Claude, Gemini, LLaMA та інші. Принцип роботи таких моделей полягає у прогнозуванні найбільш імовірного наступного токена у тексті на основі контексту попередніх — однак за рахунок мільярдів параметрів і навчання на колосальних обсягах текстових даних мовні моделі демонструють здатність до якісної обробки природної мови, узагальнення, тематичного аналізу та формулювання змістовних відповідей.

Для задач щоденникового аналізу LLM мають низку важливих переваг порівняно з традиційними методами обробки природної мови. По-перше, мовна модель не вимагає попереднього навчання під конкретну предметну область — достатньо коректно сформулювати інструкцію (системний промпт). По-друге, модель здатна виокремлювати неявні теми та емоційні відтінки, що для класичних методів NLP (на кшталт TF-IDF чи тематичного моделювання LDA) є непосильною задачею. По-третє, відповідь модель формулює природною мовою, у форматі, придатному для безпосереднього подання користувачу [8, 9].

Серед основних задач, які мовні моделі ефективно розв’язують у контексті аналізу персональних записів, варто виділити такі:

- виявлення повторюваних тем у наборі текстів (наприклад, систематична згадка робочих перевантажень, проблем зі сном, конфліктів у певних сферах життя);
- аналіз емоційного тону записів (sentiment analysis) із визначенням переважаючих емоційних станів за період;
- підсумовування великого обсягу тексту до короткого огляду ключових моментів;

- формулювання індивідуальних рекомендацій на основі виявлених патернів;
- виявлення позитивних аспектів, які користувач може недооцінювати у власному житті.

Важливим питанням при застосуванні LLM до персональних даних є питання конфіденційності. Записи у щоденнику за визначенням містять чутливу інформацію — переживання, оцінки інших людей, події особистого життя. У зв'язку з цим розробникам застосунків такого типу необхідно ретельно опрацьовувати архітектуру взаємодії з мовною моделлю: мінімізувати обсяг даних, що передається на зовнішні сервери, забезпечувати надійні канали передачі (HTTPS), уникати тривалого зберігання запитів на стороні провайдера API. У рамках цієї роботи прийнято рішення зберігати всі записи виключно локально на пристрої користувача — на сервер OpenAI передаються лише тексти за конкретний тиждень у момент виконання запиту на аналіз; передача відбувається по захищеному каналу.

Ще один аспект — якість роботи моделі з україномовним текстом. Сучасні моделі сімейства GPT-4 та їх «полегшені» версії (gpt-4o-mini) показують високу якість як розуміння, так і генерації українською мовою. Тестування у процесі розробки підтвердило, що модель коректно опрацьовує україномовні записи, виявляє в них тематику й емоційні відтінки, а відповіді формулює грамотно українською мовою з природним стилем. Цей результат критично важливий для проєкту, оскільки україномовні AI-щоденники практично відсутні на ринку.

Окрема увага у роботі приділена структуруванню відповіді мовної моделі. У вільному форматі LLM могла б відповідати по-різному — нерегулярні відповіді ускладнили б обробку та відображення на стороні застосунку. Тому в системному промті закладено вимогу повертати відповідь у форматі JSON із трьома фіксованими блоками: позитивні аспекти періоду, виявлені труднощі та персональні рекомендації. Такий підхід дозволяє

надійно розпарсити відповідь і коректно відобразити її у відповідних розділах інтерфейсу.

1.3 Цільова аудиторія застосунку

Розуміння цільової аудиторії — необхідний крок перед формуванням вимог до застосунку. Diary AI орієнтований на людей віком від 18 до 35 років, які прагнуть більш усвідомленого ставлення до власного життя, але потребують зовнішньої структури для регулярної рефлексивної практики.

За даними соціологічних і медичних досліджень останніх років [10], саме у віковій групі 18–35 років фіксується найбільший приріст випадків емоційного вигорання, тривожних розладів та психоемоційного навантаження, пов'язаного з робочим графіком, інформаційним перевантаженням і зміною життєвих орієнтирів. Користувачі цієї групи активно споживають мобільний контент, мають високий рівень технічної грамотності та готові експериментувати з новими інструментами для покращення якості життя.

З досвіду спостереження за поведінкою користувачів існуючих щоденникових застосунків можна виокремити дві ключові проблеми, які заважають регулярній рефлексивній практиці.

Проблема порожньої сторінки. Коли застосунок пропонує порожнє поле для запису і єдину інструкцію «опиши свій день», значна частина користувачів просто не починає писати або обмежується кількома загальними фразами. Психологічний бар'єр виникає через невизначеність — людина не знає, з чого почати, що саме вважати важливим, як структурувати думки. Цей бар'єр настільки серйозний, що вже після кількох спроб користувач залишає застосунок, навіть якщо мав щирі мотивації працювати з ним.

Проблема відсутності синтезу. Навіть у тих, хто регулярно веде записи, стикається з другою проблемою — накопичений масив тексту лежить без використання. Самостійно перечитувати десятки записів, виявляти у них

патерни й узагальнювати — задача трудомістка, тож на практиці вона не виконується. У результаті щоденник перетворюється на «сховище», з якого неможливо дістати корисні висновки.

Diary AI цілеспрямовано закриває обидві проблеми. Першу — через структуровані запитання, адаптовані з методології Scrum-ретроспектив [11]: користувач не починає з порожнього аркуша, а відповідає на конкретні запитання, які звужують і спрямовують думку. Другу — через автоматизований AI-аналіз записів за тиждень, що формує для користувача готовий синтез без необхідності самотійно перерахувати архів.

Окремою рисою цільової аудиторії є мовний фактор. Україномовні користувачі, які цінують роботу з рідною мовою, на сьогодні фактично позбавлені якісних AI-щоденників: домінуючі на ринку рішення (Reflectly, Stoic, Rosebud) орієнтовані на англomовну аудиторію. Відсутність повноцінної української локалізації та якісних AI-відповідей рідною мовою формує окрему ринкову нішу, на яку безпосередньо орієнтований проєкт Diary AI.

1.4 Аналіз існуючих застосунків-аналогів

Перш ніж формулювати вимоги до власного застосунку, доцільно проаналізувати наявні рішення на ринку. У сегменті мобільних застосунків для саморефлексії та ведення щоденника можна виділити кілька основних типів продуктів: класичні цифрові щоденники з вільним форматом запису, трекери настрою з мінімалістичним інтерфейсом, спеціалізовані застосунки для медитації та рефлексії, а також AI-щоденники, що тільки починають з'являтися як окремий клас. Для аналізу обрано п'ять найпопулярніших представників ринку: Day One, Reflectly, Journey, Daylio та Stoic.

1.4.1 Day One

Day One — один із найвідоміших цифрових щоденників на платформах iOS, macOS та Android [12]. Розробники позиціонують застосунок як універсальне рішення для ведення особистих нотаток та зберігання спогадів. На рисунку 1.1 показано головний екран застосунку.

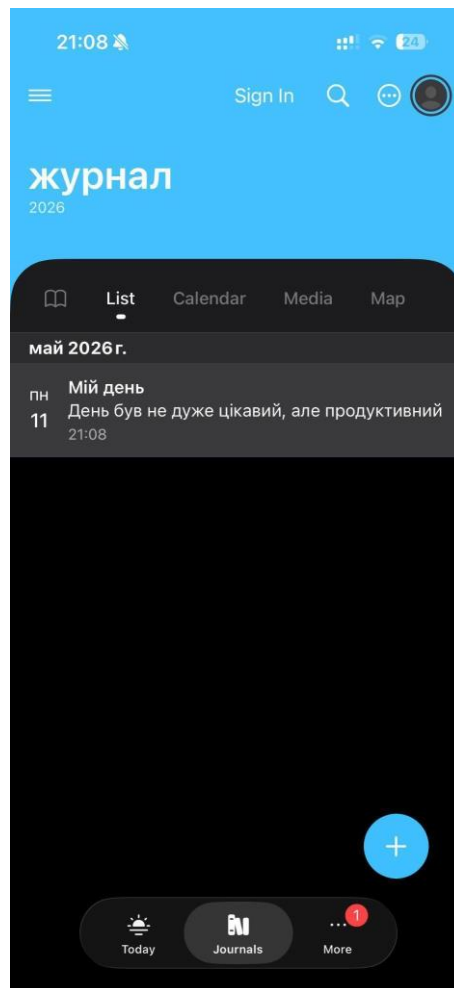


Рис. 1.1 Головний екран застосунку Day One

Серед сильних сторін Day One варто відзначити такі функціональні особливості. Застосунок підтримує мультимедійні вкладення — фото, відео, аудіозаписи, що дозволяє створювати багатий контекстуальний запис. Реалізовано наскрізне шифрування (end-to-end encryption) даних, що забезпечує високий рівень конфіденційності. Розвинена система тегів і повнотекстового пошуку дозволяє швидко знаходити потрібні записи в архіві. Хмарна синхронізація через власний сервіс Day One забезпечує доступ до

записів з різних пристроїв. Підтримка Markdown-розмітки дає можливість зручно структурувати текст записів.

Однак для задач саморефлексії Day One має суттєві обмеження. По-перше, застосунок повністю орієнтований на вільну форму запису — заготовлених запитань, шаблонів чи підказок не передбачено, що ставить користувача перед класичною проблемою «порожньої сторінки». По-друге, AI-обробка чи автоматичний аналіз записів відсутні: застосунок виступає виключно сховищем, а не інструментом аналізу. По-третє, основний функціонал прихований за платною підпискою, що звужує коло потенційних користувачів. Україномовної локалізації не передбачено.

1.4.2 Reflectly

Reflectly позиціонується розробниками як «AI-щоденник для ментального здоров'я» і представлений на iOS та Android [13]. На відміну від Day One, тут процес запису є значно більш керованим: застосунок задає користувачу серію запитань про настрій та події дня, ведучи його через рефлексію у форматі діалогу. На рисунку 1.2 показано приклад такого діалогу.

До сильних сторін Reflectly належать наявність структурованих запитань, що знижує бар'єр входу для нових користувачів, а також вбудовані AI-функції — модель аналізує емоційні патерни та формує персональні підказки. Інтерфейс реалізований у мінімалістичному, дружньому стилі, що сприяє регулярному використанню. Застосунок надсилає щоденні нагадування про необхідність зробити запис.

Водночас Reflectly має ряд суттєвих недоліків. Запитання, які пропонує застосунок, є доволі поверхневими — здебільшого вони стосуються загального настрою та коротких описів дня, але не виводять користувача на глибший аналіз труднощів, ресурсів і життєвих уроків. AI-функціонал зводиться переважно до візуалізації динаміки настрою, а повноцінних тижневих підсумків з конкретними поведінковими рекомендаціями застосунок не пропонує.



Рис. 1.2 Діалоговий інтерфейс застосунку Reflectly

Більшість можливостей доступні лише за платною підпискою. Інтерфейс англomовний — української локалізації не передбачено, що знижує цінність застосунку для україномовної аудиторії.

1.4.3 Journey

Journey — кросплатформний цифровий щоденник, доступний на iOS, Android та через веббраузер [14]. Застосунок поєднує можливості традиційного щоденника з низкою додаткових функцій: підтримка різних типів медіавкладень, прив'язка записів до геолокації, інтеграція з сервісами Google Fit та Apple Health, синхронізація через Google Drive. Передбачена також система «підказок» (prompts) — заготовлених запитань, що частково розв'язують проблему відсутності структури. Головний екран застосунку наведено на рисунку 1.3.

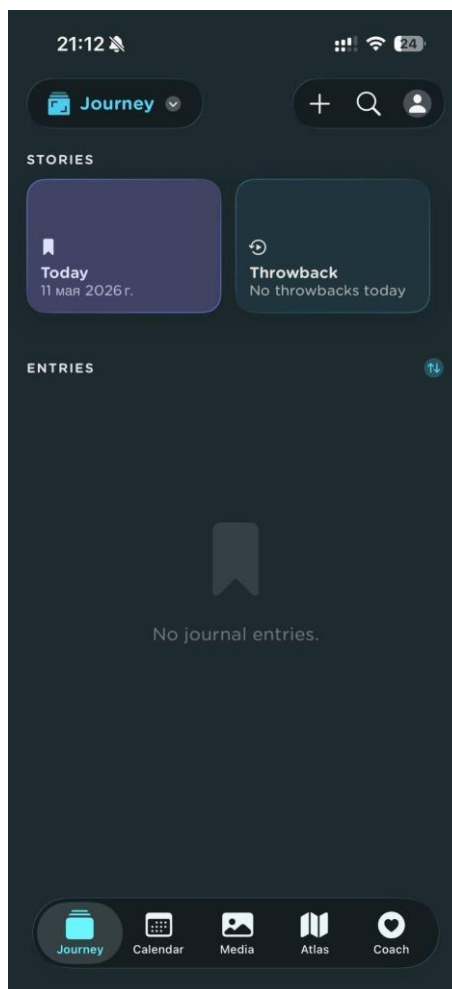


Рис. 1.3 Головний екран застосунку Journey

Серед переваг Journey — широкий функціональний набір, кросплатформність, підтримка хмарної синхронізації, наявність системи тематичних підказок. Застосунок підходить як для ведення подієвого щоденника (фіксація поїздок, важливих моментів життя), так і для регулярних записів про переживання.

Втім, Journey за своєю суттю, як і Day One, лишається класичним щоденником без серйозної інтеграції з ІШ. Підказки тут — додаткова, необов'язкова функція, а не основний спосіб взаємодії. AI-аналізу записів немає. Сценарій щоденної рефлексії не є для застосунку основним: радше він придатний для нерегулярних, подієвих записів.

Повний функціонал, як і у випадку з Day One, доступний лише за платною підпискою.

1.4.4 Daylio

Daylio — застосунок для трекінгу настрою і коротких щоденних відміток, доступний на iOS та Android [15]. Головна риса Daylio — максимально швидке введення: користувач обирає піктограму настрою з фіксованого набору, опціонально додає коротку текстову нотатку та проставляє теги з переліку активностей. Завдяки такому мінімалістичному підходу один запис у Daylio займає менше хвилини часу. Головний екран застосунку наведено на рисунку 1.4.

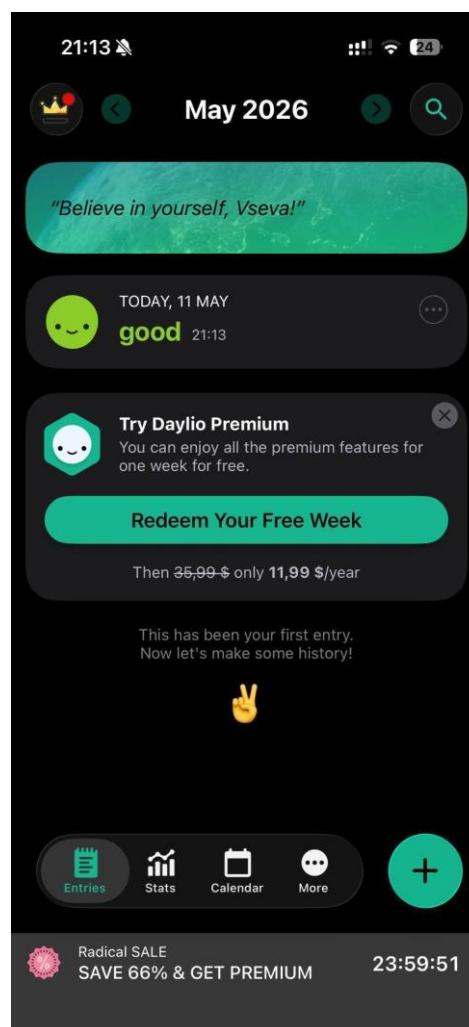


Рис. 1.4 Головний екран застосунку Daylio

Сильні сторони Daylio — простота використання, низький бар'єр входу, наочна статистика настрою у вигляді графіків і календарних теплових карт, можливість виявляти кореляції між активностями та настроєм. Застосунок

добре підходить для тих, кому важливо відстежувати загальну динаміку емоційного стану в часі.

Проте сама ця простота і є основним обмеженням Daylio з точки зору повноцінної рефлексії. Застосунок дозволяє зафіксувати настрій і активності, але не допомагає глибше осмислити день: запитання типу «що саме вплинуло на настрій», «з якими труднощами я зіткнувся» чи «що допомогло їх подолати» виходять за межі основного сценарію. AI-аналізу та тижневих рекомендацій у Daylio немає. Україномовний інтерфейс також відсутній.

1.4.5 Stoic

Stoic — застосунок для рефлексії та ментального здоров'я, побудований на основі філософії стоїцизму [16]. На відміну від класичних щоденників, Stoic пропонує користувачу структуровані ранкові та вечірні рутини: ранкові запитання про наміри на день, вечірні — про події та висновки. Застосунок включає медитативні практики, добірки цитат стоїчних філософів, базові статистичні візуалізації. Головний екран застосунку наведено на рисунку 1.5.

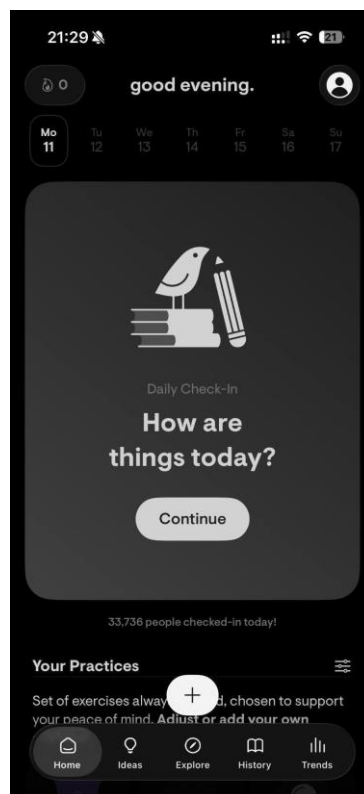


Рис. 1.5 Головний екран застосунку Stoic

Серед сильних сторін Stoic — продумана структура запитань на основі сталої методології, інтеграція рефлексії з медитативними практиками, високоякісний візуальний дизайн. Stoic найбільш близький за концепцією до пропонованого Diary AI з точки зору структурованості запитань.

Водночас, Stoic не надає повноцінного AI-аналізу записів — статистичні візуалізації обмежуються графіками настрою без змістовних висновків та рекомендацій. Інтерфейс англomовний; україномовна локалізація відсутня. Більша частина функціоналу доступна за платною підпискою. Застосунок орієнтований на філософську рамку стоїцизму, що може не імпонувати частині користувачів, які шукають нейтральний, не ідеологічно навантажений інструмент рефлексії.

1.4.6 Порівняльний аналіз існуючих рішень

Зведений результат аналізу розглянутих застосунків наведено у таблиці 1.1. У ній подано порівняння за ключовими функціональними критеріями, важливими для задач щоденної структурованої рефлексії.

Таблиця 1.1

Порівняння існуючих застосунків-аналогів

Критерій порівняння	Day One	Reflectly	Journey	Daylio	Stoic	Diary AI
Структуровані запитання для рефлексії	—	—	—	—	+	+
AI-аналіз записів	—	+	—	—	—	+
Тижневі підсумки з рекомендаціями	—	—	—	—	—	+
Україномовний інтерфейс	—	—	—	—	—	+
Локальне зберігання даних	—	—	+	+	—	+

Умовні позначення: «+» — функція повністю реалізована; «±» — функція реалізована частково або доступна лише в платній версії; «-» — функція відсутня.

Аналіз таблиці 1.1 дозволяє зробити такі висновки. Жоден з розглянутих застосунків одночасно не поєднує усіх ключових характеристик, важливих для задач щоденної структурованої рефлексії з AI-підтримкою. Day One і Journey є потужними, але «нейтральними» щоденниками без структури і без AI-аналізу. Reflectly має часткову AI-функціональність, але обмежується аналізом настрою без розгорнутих рекомендацій. Daylio орієнтований на швидку фіксацію настрою без глибокого аналізу. Stoic найближчий до концепції Diary AI з точки зору структурованості, але не пропонує повноцінного AI-аналізу записів.

Окремо варто наголосити на двох спільних рисах усіх розглянутих рішень: відсутність україномовної локалізації та переважання платної моделі монетизації як обов'язкової умови повноцінного використання. Українська аудиторія, яка користується мобільним контентом рідною мовою, на сьогодні фактично не має на ринку якісного спеціалізованого AI-щоденника. Це формує окрему ринкову нішу, на яку орієнтований проєкт Diary AI.

1.5 Формування вимог до програмного забезпечення

На основі проведеного аналізу предметної області, цільової аудиторії та існуючих рішень сформовано перелік вимог до розроблюваного застосунку Diary AI. Вимоги поділено на функціональні (опис того, що має робити система) та нефункціональні (опис того, як система має працювати — характеристики продуктивності, безпеки, зручності використання).

1.5.1 Функціональні вимоги

Функціональні вимоги до застосунку Diary AI:

- підтримка реєстрації нового користувача за email-адресою та паролем з валідацією введених даних на стороні застосунку;
- автентифікація зареєстрованого користувача за email і паролем з відповідною перевіркою облікових даних;
- безпечне зберігання облікових даних з використанням криптографічного хешування паролів;
- створення нового щоденного запису шляхом відповіді на чотири структуровані запитання, адаптовані з методології Scrum-ретроспектив:

- 1) як пройшов день у цілому;
- 2) які виникли труднощі чи проблеми;
- 3) що допомогло впоратися з труднощами;
- 4) вільні думки, спостереження та інсайти;

- обмеження на створення лише одного запису на день для забезпечення дисципліни рефлексивної практики;
- перегляд хронологічної історії всіх записів користувача з можливістю переходу до повного тексту будь-якого запису;
- виконання тижневого AI-аналізу записів за допомогою OpenAI API за умови наявності щонайменше трьох записів за тижневий період;
- подання результату AI-аналізу у структурованому вигляді — три тематичні блоки: позитивні моменти періоду, виявлені труднощі, персональні рекомендації;
- локальне збереження виконаних AI-аналізів для повторного перегляду без додаткових звернень до зовнішнього API;
- перегляд архіву виконаних аналізів з можливістю переходу між тижнями;
- надсилання push-сповіщень для нагадування про необхідність зробити щоденний запис у заданий користувачем час;
- налаштування часу нагадувань через інтерфейс застосунку;
- можливість введення власного API-ключа OpenAI у налаштуваннях застосунку;

– вихід із облікового запису з автоматичним перенаправленням на екран автентифікації.

1.5.2 Нефункціональні вимоги

Нефункціональні вимоги до застосунку Diary AI:

- платформа та сумісність — нативний застосунок для iOS з мінімально підтримуваною версією iOS 17.0;
- підтримка моделей iPhone від iPhone SE до моделей лінійки Pro Max;
- продуктивність — час відгуку інтерфейсу на дії користувача не повинен перевищувати 200 мс;
- час очікування відповіді AI-аналізу не повинен перевищувати 30 секунд за умови стабільного інтернет-з'єднання;
- безпека — паролі користувачів зберігаються виключно у вигляді криптографічних хешів за алгоритмом SHA-256 [17];
- передача даних до зовнішнього API виконується по захищеному протоколу HTTPS; персональні записи зберігаються локально на пристрої та не передаються на жодні зовнішні сервери, окрім тимчасової передачі тижневого набору на OpenAI API у момент виконання запиту на аналіз;
- зручність використання — інтуїтивний мінімалістичний інтерфейс з підтримкою стандартних патернів iOS-навігації;
- підтримка темної теми (Dark Mode) та автоматичне перемикання відповідно до системних налаштувань; підтримка адаптивного збільшення шрифтів (Dynamic Type) для покращення доступності;
- локалізація — повна україномовна локалізація інтерфейсу та AI-відповідей;
- надійність — автономна робота основних функцій (створення запису, перегляд історії) без необхідності в інтернет-з'єднанні;
- коректна обробка помилок мережі при зверненні до OpenAI API з інформативним повідомленням користувачу;
- архітектура — реалізація за патерном MVVM (Model-View-ViewModel) для забезпечення тестованості та подальшого розширення функціоналу.

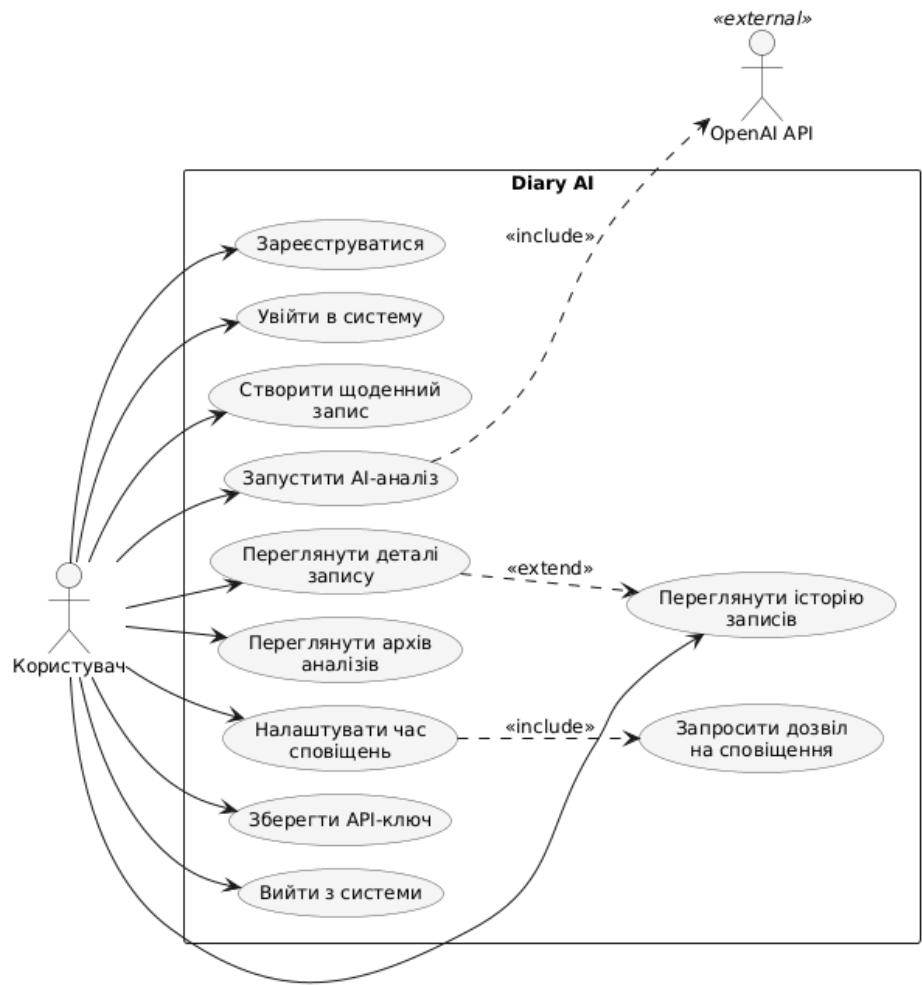


Рис. 1.6 Діаграма варіантів використання застосунку Diary AI

Сформовані на основі попередньо визначених варіантів використання застосунку (див. рис. 1.6) функціональні та нефункціональні вимоги становлять основу для подальшого проєктування архітектури застосунку та вибору технологічного стека для його реалізації, що детально розглядається у наступному розділі.

2 ЗАСОБИ РЕАЛІЗАЦІЇ

Вибір технологічного стека для розробки мобільного застосунку є одним із ключових архітектурних рішень, що визначає характеристики кінцевого продукту — від продуктивності та якості користувацького досвіду до зручності подальшого супроводу й розширення функціональності. У цьому розділі обґрунтовується вибір кожного компонента використаного у проєкті стека: платформи iOS як цільового середовища, мови Swift, декларативного фреймворку SwiftUI для побудови інтерфейсу, локального сховища SwiftData, архітектурного патерну MVVM, інтеграції з OpenAI API та середовища розробки Xcode. Окремо розглядаються допоміжні фреймворки UserNotifications та CryptoKit, що використовуються для реалізації системи нагадувань і криптографічного хешування.

2.1 Платформа iOS та екосистема Apple

Платформа iOS, розроблена компанією Apple, є однією з двох домінуючих мобільних операційних систем у світі. Перша версія iOS (тоді iPhone OS) була випущена у 2007 році разом із першим iPhone, а до моменту виконання цієї роботи актуальною є iOS 18. Екосистема Apple характеризується низкою особливостей, які роблять її привабливою як для користувачів, так і для розробників.

З точки зору користувача, iOS відрізняється високою стабільністю роботи, плавністю анімацій, тривалим терміном підтримки навіть старих моделей пристроїв (типово 6–7 років оновлень), а також жорстким контролем якості застосунків через App Store. Ці фактори формують аудиторію з вищим середнім чеком та готовністю платити за якісні застосунки, що особливо важливо для нішевих продуктів.

З точки зору розробника, ключовою перевагою iOS є фрагментація пристроїв на порядок нижча, ніж у Android: незначна кількість моделей iPhone із подібними характеристиками і єдина версія операційної системи на більшості активних пристроїв спрощують тестування й дозволяють використовувати найновіші API без потреби підтримувати застарілі.

Для проєкту Diary AI як мінімальна підтримувана версія обрано iOS 17.0. Цей вибір обґрунтований трьома причинами. По-перше, iOS 17.0 вийшла у вересні 2023 року, і станом на початок 2026 року частка пристроїв із цією або новішою версією перевищує 90% активних iPhone, тобто обмеження аудиторії є мінімальним. По-друге, саме починаючи з iOS 17 став доступним фреймворк SwiftData, який є основою локального сховища застосунку. По-третє, iOS 17 принесла низку покращень SwiftUI, зокрема у роботі з анімаціями, навігацією та Observable-моделями, що спрощує розробку.

Серед альтернатив iOS можна було б розглянути:

Android — більший охоплення аудиторії, але вища фрагментація пристроїв, нижча середня якість мобільного контенту в певних сегментах та складніша монетизація для нішевих продуктів;

кросплатформні фреймворки (Flutter, React Native) — швидша розробка під обидві платформи одразу, але типово гірша інтеграція з нативними API, обмежені можливості роботи з системними фреймворками та компроміс у якості користувацького досвіду;

гібридні рішення (PWA, Capacitor) — найшвидша розробка, але неможливість використання push-сповіщень у повному обсязі та обмежений доступ до системних можливостей.

З огляду на специфіку Diary AI як застосунку для щоденного використання, де критично важливими є плавність інтерфейсу, надійність push-сповіщень та якісна локальна робота з даними, нативна iOS-розробка є оптимальним вибором. Орієнтація на одну платформу також дозволила сконцентрувати зусилля на якості реалізації, замість компромісів кросплатформності.

2.2 Мова програмування Swift

Swift — сучасна мультипарадигмова мова програмування, представлена компанією Apple у 2014 році на конференції WWDC та відкрита у 2015 році [18]. Swift створювалася як заміна Objective-C — мови, яка від 1980-х років була основною для розробки під платформи Apple, але мала ряд застарілих рішень, успадкованих з епохи C та Smalltalk.

Серед ключових характеристик Swift, важливих для проєкту Diary AI, варто виділити такі.

Типобезпека. Swift є строго типізованою мовою з потужною системою виведення типів. Більшість помилок, пов'язаних із некоректним використанням типів, виявляються на етапі компіляції, а не у час виконання. Опціональні типи (Optional) явно розділяють значення, які можуть бути nil, від тих, що завжди мають значення, що практично виключає клас помилок, відомий як "null pointer exception".

Автоматичне керування пам'яттю. Swift використовує механізм Automatic Reference Counting (ARC), що автоматично звільняє об'єкти, на які більше немає посилань, без потреби у "збиранні сміття" (garbage collection) із його властивими затримками. Це робить роботу застосунків більш плавною та передбачуваною за швидкістю.

Підтримка функціональної парадигми. Swift надає першокласну підтримку замикань (closures), функцій вищих порядків (map, filter, reduce), імутабельних структур даних. Це дозволяє писати лаконічний і виразний код, особливо у роботі з колекціями.

Протокол-орієнтоване програмування. На відміну від класичного об'єктно-орієнтованого підходу, в Swift акцент зроблено на протоколи — інтерфейси, що описують функціональність, незалежну від конкретної реалізації. Це сприяє написанню гнучкого, тестованого коду та зменшенню зайвої складності спадкування класів.

Сучасні асинхронні механізми. Починаючи з версії Swift 5.5 (2021 рік), мова підтримує механізм `async/await` — синтаксичну конструкцію для написання асинхронного коду у послідовному, легкому для читання вигляді. Цей механізм критично важливий для реалізації мережевих запитів до OpenAI API, які за визначенням є асинхронними.

Відкритий код. Компілятор та стандартна бібліотека Swift розповсюджуються під ліцензією Apache 2.0. Це гарантує довгострокову підтримку мови та можливість використання у проєктах різного типу.

У застосунку Diary AI Swift використовується як єдина мова реалізації — від моделей даних до сервісного шару. Жодних додаткових мов (наприклад, Objective-C для роботи з застарілими API) у проєкті не задіяно. Це забезпечує однорідність кодової бази та спрощує її подальшу підтримку.

2.3 Фреймворк SwiftUI

SwiftUI — декларативний фреймворк для побудови користувацьких інтерфейсів, представлений Apple у 2019 році на конференції WWDC [19]. SwiftUI став принципово новим підходом до розробки інтерфейсів у платформах Apple, прийшовши на зміну імперативному фреймворку UIKit, що використовувався з 2008 року.

Принципова відмінність між цими підходами полягає у способі опису інтерфейсу. У імперативному підході (UIKit) розробник створює об'єкти інтерфейсу, додає їх до ієрархії перегляду та вручну оновлює їхні властивості при зміні даних. Декларативний підхід (SwiftUI) натомість передбачає опис того, як має виглядати інтерфейс при певному стані застосунку — фреймворк сам приводить інтерфейс до описаного вигляду при кожній зміні даних.

Серед переваг SwiftUI для проєкту Diary AI варто відзначити такі.

Лаконічність коду. Кількість рядків коду, потрібних для реалізації типового екрана, у SwiftUI зазвичай у 2–3 рази менша, ніж в еквівалентній

реалізації UIKit. Для невеликого проєкту з обмеженим часом розробки це є критичною перевагою [20, 21].

Автоматична реактивність. Завдяки властивостям-обгорткам (property wrappers) `@State`, `@Binding`, `@StateObject`, `@ObservedObject`, `@Query`, інтерфейс автоматично перебудовується при зміні даних. Розробнику не потрібно вручну відстежувати, які елементи треба оновити — SwiftUI робить це самостійно та оптимально.

Готова підтримка системних можливостей. SwiftUI з коробки підтримує темну тему (Dark Mode), адаптивне збільшення шрифтів (Dynamic Type), локалізацію, доступність (Accessibility), різні розміри екранів. Усі ці можливості активуються автоматично без додаткових зусиль розробника.

Інтеграція з SwiftData. Властивість-обгортка `@Query` дозволяє отримувати дані з локальної бази SwiftData у вигляді, що автоматично оновлюється при зміні даних у сховищі. Це усуває необхідність у написанні класичного коду оновлення таблиць чи списків.

Прев'ю в Xcode. Canvas-режим Xcode дозволяє бачити результат рендерингу екранів у реальному часі без потреби запускати симулятор, що значно прискорює ітеративну розробку інтерфейсу.

Водночас SwiftUI має певні обмеження. Він доступний лише на нових версіях iOS (iOS 13+), а певні API працюють лише починаючи з iOS 17. Деякі складні сценарії інтерфейсу — кастомні переходи, складні взаємодії з жестами, низькорівневий контроль над візуалізацією — можуть бути реалізовані простіше у UIKit. Однак для застосунку категорії "цифровий щоденник" із достатньо стандартним набором екранів усі ці обмеження несуттєві.

У Diary AI на основі SwiftUI побудовано всі без винятку екрани застосунку: вхід та реєстрація, головний екран, екран створення запису, історія записів, екран AI-аналізу, архів аналізів, налаштування. Загалом проєкт містить більше десяти окремих екранів, побудованих на SwiftUI-компонентах.

2.4 SwiftData як локальне сховище

SwiftData — фреймворк для роботи з локальними базами даних, представлений Apple у 2023 році на WWDC разом з iOS 17 [22]. Він є модернізованою обгорткою над Core Data — фреймворком об’єктно-реляційного відображення (ORM), який Apple розробляє з 2005 року. SwiftData зберігає всю функціональність Core Data, але пропонує значно простіший і виразніший API на основі макросів Swift.

Ключовою особливістю SwiftData є те, що моделі даних описуються прямо у Swift-кодi через атрибут `@Model`. Це позбавляє необхідності в окремому редакторі схеми (як це було в Core Data із файлами `.xcdatamodeld`) і робить визначення моделі компактним та зрозумілим. Приклад моделі у SwiftData для проєкту Diary AI має такий вигляд:

```
@Model
class DayEntry {
    var id: UUID
    var date: Date
    var userEmail: String
    var question1: String
    var question2: String
    var question3: String
    var question4: String
    var createdAt: Date
    // ...
}
```

Усе, що потрібно для перетворення цього класу на повноцінний об’єкт ORM-сховища — це атрибут `@Model`. SwiftData автоматично генерує необхідну схему бази даних SQLite та забезпечує операції збереження, отримання й видалення об’єктів.

Серед інших переваг SwiftData для проєкту:

Реактивні запити. Властивість-обгортка `@Query` отримує дані з бази та автоматично оновлює інтерфейс при зміні цих даних. Це усуває необхідність у "ручному" оновленні переглядів — приклад використання `@Query(sort: \DayEntry.date, order: .reverse)` у кодi списку записів дозволяє автоматично

відображати найновіші записи зверху, причому при додаванні нового запису список оновлюється без додаткового коду.

Типобезпечні предикати. Складні запити пишуться через макрос `#Predicate`, який забезпечує перевірку синтаксису на етапі компіляції. Помилка у назві поля або у типі значення буде виявлена ще до запуску застосунку.

Тісна інтеграція з SwiftUI. Властивості, що змінюються у моделях `SwiftData`, автоматично запускають оновлення інтерфейсу через механізми реактивності SwiftUI.

Можливість синхронізації через iCloud. Хоча в поточній версії `Diary AI` ця функція не задіяна, `SwiftData` підтримує синхронізацію даних між пристроями користувача через `CloudKit` без додаткового коду — це може бути додано у майбутніх версіях.

У застосунку `Diary AI` на основі `SwiftData` реалізовано три моделі:

- `UserProfile` — профіль зареєстрованого користувача (ім'я, email, хеш пароля);
- `DayEntry` — щоденний запис, що містить дату, email власника та чотири текстові відповіді;
- `WeeklyAnalysis` — результат AI-аналізу за певний період, що містить діапазон дат і три текстові блоки (позитиви, труднощі, рекомендації).

Контейнер моделей (`modelContainer`) ініціалізується на рівні всього застосунку, що робить дані доступними з будь-якого екрана через властивість оточення `@Environment(\.modelContext)`. Детальний опис структури моделей та операцій з ними наведено у розділі 3.

2.5 Архітектурний патерн MVVM

Model-View-ViewModel (MVVM) — архітектурний патерн, призначений для відокремлення логіки роботи з даними від коду користувацького інтерфейсу. MVVM був запропонований у 2005 році для платформи Windows Presentation Foundation (WPF) і з того часу набув широкого розповсюдження у багатьох екосистемах розробки.

Класична структура MVVM включає три рівні.

Model — рівень даних та бізнес-логіки. У контексті Diary AI до моделей належать класи UserProfile, DayEntry, WeeklyAnalysis, а також логіка роботи з зовнішніми сервісами (OpenAIService).

View — рівень користувацького інтерфейсу. Реалізується через структури SwiftUI, які описують вигляд кожного екрана. View має виконувати лише задачу подання — тобто описувати, як виглядають дані, та обробляти користувацькі дії, передаючи їх до ViewModel.

ViewModel — посередник між Model і View. ViewModel містить стан, що відображається у View (зазвичай через властивості з обгорткою @Published), та методи, що обробляють користувацькі дії й оновлюють стан. View "не бачить" Model напряду — вся взаємодія йде через ViewModel.

Переваги MVVM для проєкту Diary AI:

- тестованість — ViewModel містить логіку у вигляді звичайних класів Swift і може тестуватися незалежно від інтерфейсу;
- чітка відповідальність компонентів — кожен файл проєкту має зрозумілу роль, що спрощує орієнтацію у кодовій базі;
- природна сумісність зі SwiftUI — ViewModel реалізує протокол ObservableObject, властивості з @Published автоматично сповіщають View про зміни.

У Diary AI MVVM реалізовано через три ключові ViewModel-класи:

- AuthViewModel — обробляє реєстрацію, вхід та вихід користувача;

- `EntryViewModel` — відповідає за створення нових щоденних записів і перевірку, чи вже існує запис на сьогодні;
- `AIAnalysisViewModel` — управляє процесом запиту AI-аналізу, обробляє стани завантаження та помилок, працює з вибраним діапазоном дат.

Усі `ViewModel`-класи позначено атрибутом `@MainActor` — це механізм `Swift Concurrency`, що гарантує виконання їхнього коду у головному потоці. Оскільки `ViewModel`-и оновлюють властивості, що впливають на інтерфейс, виконання їхніх методів саме у головному потоці є необхідною умовою коректної роботи `SwiftUI` [21].

2.6 Інтеграція з OpenAI API

OpenAI API — програмний інтерфейс для роботи з мовними моделями компанії OpenAI, серед яких сімейство GPT-4 та його похідні [23]. API надає розробникам доступ до моделей через стандартні HTTP-запити, що повертають структуровані JSON-відповіді.

У застосунку `Diary AI` використовується модель `gpt-4o-mini` — полегшена версія GPT-4o, представлена OpenAI у 2024 році. Вибір саме цієї моделі обґрунтований кількома міркуваннями.

Якість роботи з україномовними текстами. Тестування у процесі розробки показало, що `gpt-4o-mini` коректно розуміє україномовні записи, виявляє в них тематику, емоційні відтінки, а відповіді формулює грамотною українською мовою з природним стилем.

Економічна доступність. `gpt-4o-mini` має значно нижчу вартість запиту порівняно з повноцінними моделями GPT-4, що робить її придатною для застосунків, де користувач може запитувати аналіз багаторазово без надмірних витрат.

Швидкість відповіді. За рахунок меншого розміру модель відповідає швидше, що покращує користувацький досвід — типовий час очікування відповіді в Diary AI становить 5–15 секунд залежно від обсягу записів.

Достатня для задачі продуктивність. Задача узагальнення тижневих записів, виявлення тем і формулювання порад не вимагає максимально потужних моделей. Тестові порівняння з повноцінною GPT-4o показали, що для цієї конкретної задачі різниця у якості незначна.

Взаємодія із OpenAI API у проєкті виконується через стандартний URLSession — мережевий клієнт, вбудований у iOS. Жодних сторонніх SDK для OpenAI у проєкті не використовується — це робить кодову базу повністю незалежною від зовнішніх бібліотек і спрощує її обслуговування.

Запит надсилається методом POST на endpoint `https://api.openai.com/v1/chat/completions` із заголовком `Authorization: Bearer {API_KEY}`. Тіло запиту — JSON-об’єкт з полями `model` (назва моделі), `temperature` (параметр випадковості відповіді, у проєкті встановлений у 0.7 для балансу між послідовністю та різноманітністю), `max_tokens` (максимальна довжина відповіді — 1000 токенів) та `messages` (список повідомлень діалогу).

Структура промπτу, що формується для моделі, описана у розділі 3.5.1. Принципово важливо, що у промπτі явно вказано вимогу повернути відповідь у форматі JSON-об’єкта з трьома фіксованими полями (`positives`, `challenges`, `recommendations`), що дозволяє надійно розпарсити відповідь на стороні застосунку.

2.7 Середовище розробки Xcode

Xcode — офіційне інтегроване середовище розробки (IDE) від Apple для створення застосунків під усі платформи компанії: iOS, iPadOS, macOS, watchOS, tvOS, visionOS. Перша версія Xcode була випущена у 2003 році, актуальна станом на 2026 рік версія — Xcode 16. У роботі над Diary AI використовувалася версія Xcode 15.

Серед ключових можливостей Xcode, важливих для проєкту Diary AI, варто відзначити такі.

Редактор коду. Xcode надає повноцінний редактор Swift-коду з автодоповненням, перевіркою синтаксису у реальному часі, рефакторингом і інтегрованою документацією Apple API.

SwiftUI Canvas. Інтерактивне попередньо переглядання інтерфейсів SwiftUI просто з коду, без потреби запускати симулятор. У Diary AI використовується саме Canvas, оскільки весь інтерфейс побудовано на SwiftUI.

Симулятор iOS. Xcode постачається з повноцінним симулятором, що дозволяє запускати застосунок на віртуальних моделях iPhone та iPad різних поколінь без потреби у фізичному пристрої. Це значно прискорює тестування — особливо для перевірки адаптивності інтерфейсу під різні розміри екранів.

Налагоджувач (Debugger). Інтегрований налагоджувач LLDB дозволяє ставити точки зупинки, інспектувати значення змінних, виконувати код покроково. Окремо існує Memory Graph Debugger — інструмент для виявлення витоків пам'яті та циклів утримання посилань.

Instruments. Окремий додаток у складі Xcode для профілювання продуктивності, аналізу витрат пам'яті, мережевої активності, енергоспоживання застосунку. У Diary AI Instruments використовувався для перевірки, що список великої кількості записів плавно прокручується без затримок, та для контролю мережевих запитів до OpenAI API.

Swift Package Manager. Вбудований у Xcode менеджер залежностей для Swift-проєктів. У проєкті Diary AI зовнішні бібліотеки не використовуються — всі компоненти базуються на стандартних фреймворках Apple, що забезпечує мінімальний розмір застосунку та відсутність ризиків, пов'язаних із залежностями.

Безкоштовне розповсюдження Xcode (через App Store на macOS), наявність повної офіційної документації та регулярні оновлення роблять

Xcode стандартним і єдиним розумним вибором для нативної розробки під iOS.

2.8 Допоміжні фреймворки

Окрім основних компонентів стека, у проєкті Diary AI використовуються два допоміжні фреймворки Apple, що відповідають за окремі специфічні задачі.

2.8.1 UserNotifications

UserNotifications — фреймворк для роботи з локальними та віддаленими сповіщеннями на пристроях iOS. У застосунку Diary AI використовується тільки локальні сповіщення (без push-сервера), які плануються самим застосунком і доставляються в задані моменти часу.

Робота з фреймворком будується навколо центрального об'єкта `UNUserNotificationCenter`, через який виконуються всі операції. Послідовність типового сценарію планування сповіщення в Diary AI така:

- запит дозволу від користувача через метод `requestAuthorization` із зазначенням бажаних опцій (`alert`, `sound`, `badge`);
- створення вмісту сповіщення через `UNMutableNotificationContent` — заголовок, текст, звук;
- створення тригера `UNCalendarNotificationTrigger`, що спрацьовує у задані компоненти часу (година, хвилина) з повторенням щодня;
- створення запиту `UNNotificationRequest` із унікальним ідентифікатором (у проєкті — рядок `"daily_reflection"`) та реєстрація його у центрі через метод `add`.

При зміні часу нагадування користувачем у налаштуваннях усі попередньо заплановані сповіщення скасовуються (`removeAllPendingNotificationRequests`) і планується нове з оновленим часом.

2.8.2 CryptoKit

CryptoKit — сучасний фреймворк Apple для роботи з криптографічними примітивами, представлений на WWDC 2019. CryptoKit пропонує безпечні Swift-обгортки навколо алгоритмів симетричного та асиметричного шифрування, цифрового підпису, хешування і автентифікації повідомлень.

У застосунку Diary AI з усього функціоналу CryptoKit використовується лише алгоритм хешування SHA-256 [17], призначений для безпечного зберігання паролів користувачів. Замість збереження пароля у відкритому вигляді обчислюється його криптографічний хеш, який і записується у локальну базу даних. При вході користувача введений ним пароль повторно хешується тим самим алгоритмом і порівнюється зі збереженим хешем.

Реалізація хешування у проєкті займає лише кілька рядків коду:

```
import CryptoKit

enum CryptoHelper {
    static func sha256(_ input: String) -> String {
        let data = Data(input.utf8)
        let hashed = SHA256.hash(data: data)
        return hashed.compactMap { String(format: "%02x", $0) }.joined()
    }
}
```

Такий підхід відповідає базовим практикам безпеки: навіть якщо зловмисник отримає доступ до локальної бази даних застосунку, він не зможе прочитати паролі у відкритому вигляді. Більш повноцінне рішення передбачало б додавання "соління" (salt) до паролів та використання спеціалізованих алгоритмів сповільненого хешування (наприклад, bcrypt чи Argon2), однак для застосунку, де чутливі дані зберігаються виключно локально на пристрої користувача і не передаються нікуди, обраний рівень захисту є достатнім. Цей аспект детальніше розглядається у розділі 3.6.

Обраний у розділі стек технологій — нативна iOS-розробка на Swift і SwiftUI з локальним сховищем SwiftData, архітектурою MVVM та інтеграцією OpenAI API через стандартний URLSession — повністю покриває функціональні та нефункціональні вимоги до застосунку Diary AI, сформовані у попередньому розділі. Усі компоненти стека є офіційними фреймворками

Apple або широко апробованими сторонніми сервісами, що гарантує їхню довгострокову підтримку та якісну документацію. Конкретна реалізація застосунку на основі обраного стека детально описується у наступному розділі.

3 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ЗАСТОСУНКУ DIARY AI

У цьому розділі описано практичну реалізацію застосунку Diary AI на основі обраного у попередньому розділі технологічного стека. Наводиться загальна архітектура застосунку, структура моделей даних, реалізація рівня ViewModel, опис ключових екранів користувацького інтерфейсу та сервісного шару. Окремо розглянуто питання безпеки збереження облікових даних користувача.

3.1 Загальна архітектура застосунку

Застосунок Diary AI побудовано за архітектурним патерном MVVM з чітким поділом на три рівні (див. рис. 3.1): моделі даних (Models), користувацький інтерфейс (Views) та посередницький рівень логіки (ViewModels). Окремо виділено шар сервісів (Services), що відповідає за взаємодію із зовнішніми системами — OpenAI API та підсистемою локальних сповіщень iOS, а також модуль допоміжних утиліт (Helpers) із функціоналом для криптографічного хешування й роботи з датами.

Структура каталогів проєкту відображає його архітектурні рівні:

```

REFLEX/
├── REFLEXApp.swift           // Точка входу застосунку
├── ContentView.swift        // Кореневий View – маршрутизація
├── Models/                  // Моделі даних SwiftData
│   ├── UserProfile.swift
│   ├── DayEntry.swift
│   └── WeeklyAnalysis.swift
├── ViewModels/              // Бізнес-логіка
│   ├── AuthViewModel.swift
│   ├── EntryViewModel.swift
│   └── AIAalysisViewModel.swift
├── Views/                   // Користувацький інтерфейс
│   ├── Auth/
│   │   ├── LoginView.swift
│   │   └── RegisterView.swift
│   └── Main/
│       ├── HomeView.swift
│       ├── NewEntryView.swift
│       └── HistoryView.swift

```

```

├── EntryDetailView.swift
├── AIAalysisView.swift
├── AnalysisArchiveView.swift
├── SettingsView.swift
├── MainTabView.swift
├── Services/                // Зовнішні сервіси
│   ├── OpenAIService.swift
│   └── NotificationService.swift
├── Helpers/                 // Утиліти
│   ├── CryptoHelper.swift
│   └── DateHelper.swift

```

Така структура дозволяє швидко орієнтуватися у кодовій базі: за назвою каталогу зрозуміла роль кожного файлу, а взаємозв'язки між компонентами слідує єдиному напрямку — View звертається до ViewModel, ViewModel працює з Models та Services, Services взаємодіють із зовнішніми системами.

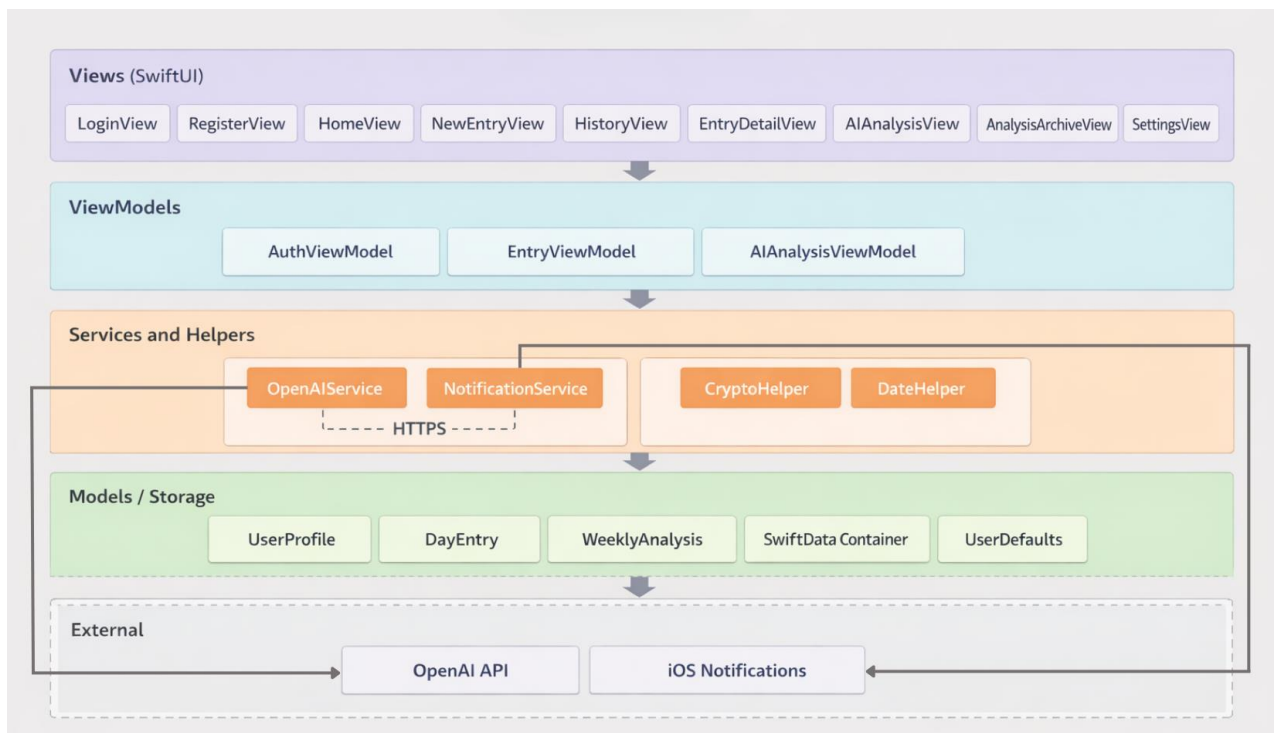


Рис. 3.1 Архітектурна діаграма застосунку Diary AI

Точка входу застосунку реалізована у файлі REFLEXApp.swift. Вона налаштовує контейнер моделей SwiftData, що міститиме сховища всіх трьох сутностей застосунку, та визначає кореневий View:

```

@main
struct ReflexApp: App {
    var body: some Scene {
        WindowGroup {
            ContentView()
        }
    }
}

```

```

    }
    .modelContainer(for: [DayEntry.self, UserProfile.self,
WeeklyAnalysis.self])
    }
}

```

Кореневий ContentView виконує функцію маршрутизації між двома сценаріями застосунку — автентифікацією та основним функціоналом. Прийняття рішення базується на значенні стану isLoggedIn, що зберігається у UserDefaults:

```

struct ContentView: View {
    @AppStorage("isLoggedIn") private var isLoggedIn: Bool = false

    var body: some View {
        if isLoggedIn {
            HomeView()
        } else {
            LoginView()
        }
    }
}

```

При запуску застосунку, якщо користувач не зареєстрований, відображається екран входу (див. рис. 3.2). Після успішної автентифікації стан змінюється — і SwiftUI автоматично перебудовує інтерфейс, відображаючи головний екран. Аналогічно при виході з облікового запису користувач повертається на екран входу.

3.2 Структура моделей даних

Усі моделі даних у Diary AI реалізовані як SwiftData-сутності, позначені атрибутом @Model. Усього у застосунку три моделі: UserProfile для зберігання профілю користувача, DayEntry для щоденних записів та WeeklyAnalysis для збережених AI-аналізів.

Зв'язки між моделями реалізовані через поле userEmail — текстовий ідентифікатор, що відповідає email-адресі власника. Такий підхід використано замість прямих зв'язків через зовнішні ключі (relationships в Core Data) з міркувань простоти: оскільки в одному пристрої зазвичай використовується один обліковий запис, складна реляційна структура не потрібна.

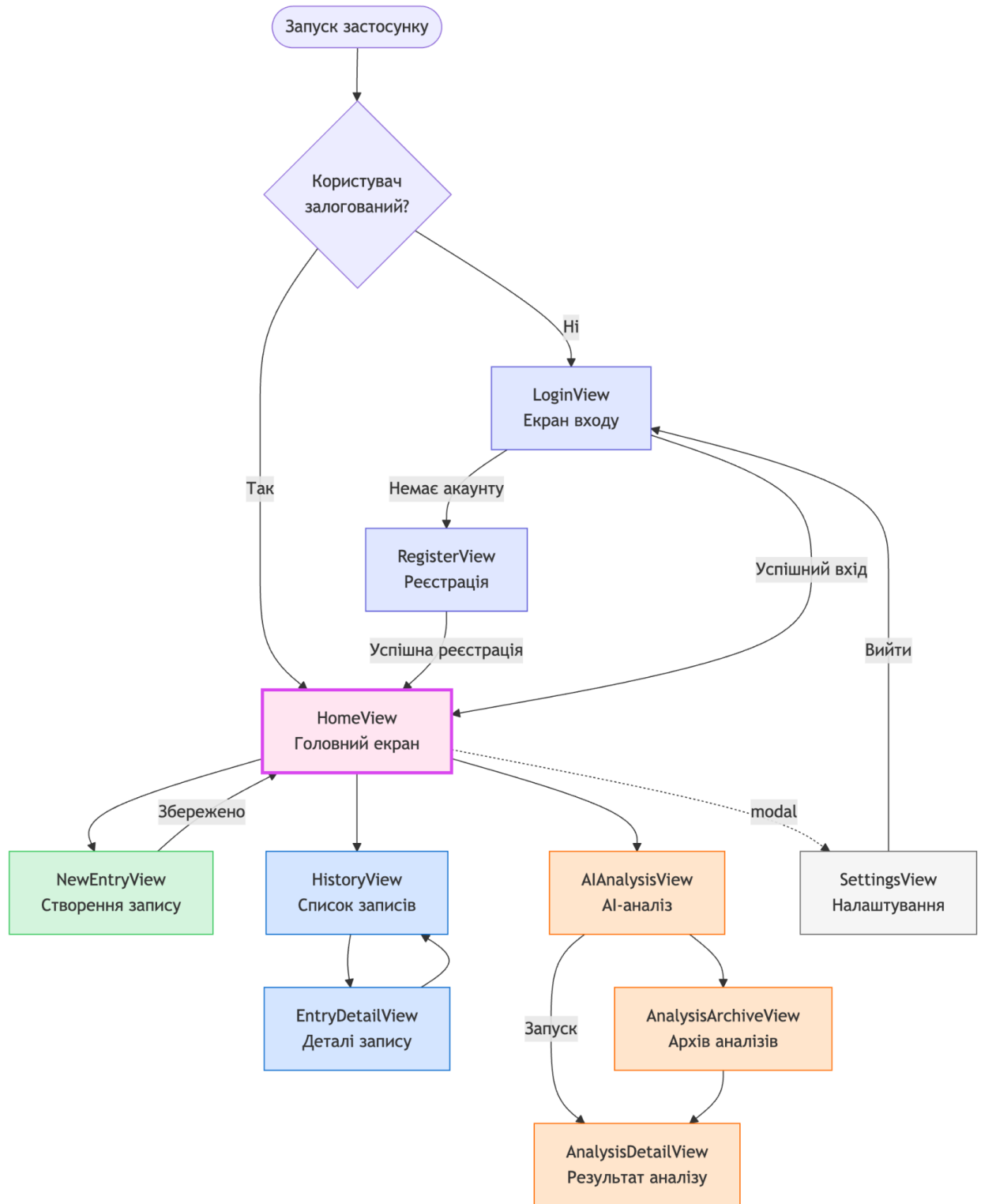


Рис. 3.2 Карта екранів застосунку Diary AI

3.2.1 Модель UserProfile

Модель UserProfile зберігає дані зареєстрованого користувача. Її структура така:

```

@Model
class UserProfile {
    var id: UUID
    var name: String
    var email: String
    var passwordHash: String
    var createdAt: Date

    init(name: String, email: String, passwordHash: String) {
        self.id = UUID()
        self.name = name
        self.email = email
        self.passwordHash = passwordHash
        self.createdAt = Date()
    }
}

```

Поля моделі:

- id — унікальний ідентифікатор запису у форматі UUID, генерується автоматично при створенні;
- name — ім'я користувача, що вводиться при реєстрації;
- email — електронна адреса, що використовується як логін; нормалізується до нижнього регістру для уникнення проблем із чутливістю до регістру;
- passwordHash — криптографічний хеш пароля за алгоритмом SHA-256, представлений у вигляді шістнадцяткового рядка;
- createdAt — дата створення облікового запису, фіксується автоматично у момент реєстрації.

Принципово важливо, що поле password у моделі відсутнє — пароль у відкритому вигляді ніколи не зберігається у базі даних.

3.2.2 Модель DayEntry

Модель DayEntry зберігає щоденний запис користувача з відповідями на чотири структуровані запитання:

```

@Model
class DayEntry {
    var id: UUID
    var date: Date
    var userEmail: String
    var question1: String
    var question2: String
    var question3: String
}

```

```

var question4: String
var createdAt: Date

init(date: Date, userEmail: String,
    question1: String, question2: String,
    question3: String, question4: String = "") {
    self.id = UUID()
    self.date = date
    self.userEmail = userEmail
    self.question1 = question1
    self.question2 = question2
    self.question3 = question3
    self.question4 = question4
    self.createdAt = Date()
}
}

```

Поля моделі:

- id — унікальний ідентифікатор запису;
- date — дата, до якої належить запис, нормалізується до початку дня (00:00:00) для забезпечення коректних запитів за діапазоном;
- userEmail — email власника запису;
- question1 — відповідь на запитання «Як пройшов мій день?»;
- question2 — відповідь на запитання «З якими труднощами ти сьогодні стикнувся?»;
- question3 — відповідь на запитання «Що допомогло тобі пережити цей день?»;
- question4 — відповідь на необов'язкове запитання «Твої думки (за бажанням)»;
- createdAt — фактична дата та час створення запису.

Розділення date (день, до якого належить запис) і createdAt (момент створення) зроблено навмисно — це дозволяє коректно обробляти випадки, коли користувач робить запис, наприклад, опівночі, але хоче, щоб він стосувався вчорашнього дня. У поточній реалізації запис автоматично прив'язується до поточного дня, проте архітектура моделі залишає можливість розширення.

Чотири запитання, на які відповідає користувач, адаптовані з методології Scrum-ретроспектив, що детально обґрунтовано у розділі 1.1.

Перші три запитання є обов'язковими — без відповіді на них запис не зберігається. Четверте запитання необов'язкове і виконує роль вільного простору для думок, які не вписалися в попередні три категорії.

3.2.3 Модель WeeklyAnalysis

Модель WeeklyAnalysis зберігає результат AI-аналізу записів за певний період:

```
@Model
class WeeklyAnalysis {
    var id: UUID
    var userEmail: String
    var weekStartDate: Date
    var weekEndDate: Date
    var positives: String
    var challenges: String
    var recommendations: String
    var createdAt: Date
    // ...
}
```

Поля моделі:

- id — унікальний ідентифікатор аналізу;
- userEmail — email власника;
- weekStartDate — дата початку аналізованого періоду (нормалізована до початку дня);
- weekEndDate — дата завершення аналізованого періоду;
- positives — текстовий блок із позитивними аспектами періоду, виокремленими мовною моделлю;
- challenges — текстовий блок із виявленими труднощами;
- recommendations — текстовий блок із персональними рекомендаціями;
- createdAt — дата та час створення аналізу.

Збереження результату аналізу у локальній базі дозволяє користувачу повертатися до раніше згенерованих аналізів без повторних звернень до OpenAI API. Це має дві переваги: економію витрат на запити та можливість роботи з раніше отриманими аналізами офлайн. Назва «Weekly» є умовною — насправді користувач може обирати будь-який діапазон дат через відповідні елементи інтерфейсу.

3.3 Реалізація рівня ViewModel

ViewModel-класи в Diary AI реалізують основну бізнес-логіку застосунку та відіграють роль посередників між View та моделями. Усі три ViewModel-класи позначені атрибутом `@MainActor`, що забезпечує виконання їхнього коду у головному потоці — необхідна умова для коректної взаємодії зі SwiftUI. Кожен ViewModel реалізує протокол `ObservableObject` та містить посилання на `ModelContext SwiftData` для виконання операцій з базою даних.

3.3.1 AuthViewModel

Клас `AuthViewModel` відповідає за обробку процесів автентифікації — реєстрації нового користувача, входу та виходу з облікового запису. Структура класу:

```
@MainActor
final class AuthViewModel: ObservableObject {
    var modelContext: ModelContext

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

    func login(email: String, password: String) -> Bool { /* ... */ }
    func register(name: String, email: String, password: String) -> Bool {
/* ... */ }
    func logout() { /* ... */ }
}
```

Метод `login` виконує таку послідовність дій. По-перше, `email` нормалізується — приводиться до нижнього регістру та обрізаються пробіли. По-друге, пароль хешується за алгоритмом SHA-256 через утиліту `CryptoHelper`. По-третє, виконується запит до бази даних — пошук профілю з відповідним `email`. Якщо профіль знайдено і збережений хеш пароля збігається з обчисленим, метод зберігає сесію (через `UserDefaults`) і повертає `true`. В іншому випадку повертається `false`, на основі чого View відображає помилку входу.

Ключовий фрагмент реалізації:

```

func login(email: String, password: String) -> Bool {
    let norm = email.lowercased().trimmingCharacters(in: .whitespaces)
    let hash = CryptoHelper.sha256(password)
    let desc = FetchDescriptor<UserProfile>(
        predicate: #Predicate { $0.email == norm }
    )
    guard
        let list = try? modelContext.fetch(desc),
        let profile = list.first,
        profile.passwordHash == hash
    else { return false }
    saveSession(email: profile.email, name: profile.name)
    return true
}

```

Метод `register` реалізує реєстрацію нового користувача. Послідовність дій така: нормалізація email, перевірка унікальності email у базі (повернення `false` при наявності збігу), створення нового екземпляра `UserProfile` з обчисленим хешем пароля, збереження у базу та автоматичний вхід — без необхідності повторно вводити дані на екрані входу.

Метод `logout` очищає дані сесії з `UserDefaults`, що призводить до автоматичного повернення на екран входу через реактивну природу `SwiftUI`.

Сесія користувача зберігається у `UserDefaults` через ключі `isLoggedIn`, `currentUserEmail` та `currentUserName`. Цей механізм був обраний як компроміс між простотою реалізації та достатньою функціональністю — для застосунку, що працює виключно локально на одному пристрої та не передає чутливі дані на сторонні сервери, такого рівня збереження сесії достатньо. Альтернативою могло б бути використання `Keychain Services`, що забезпечує апаратно-захищене зберігання, проте враховуючи, що паролі у відкритому вигляді не зберігаються у жодному місці застосунку (зберігаються тільки хеші), такий рівень захисту вважається надмірним.

3.3.2 EntryViewModel

Клас `EntryViewModel` відповідає за роботу з щоденними записами:

```

@MainActor
final class EntryViewModel: ObservableObject {
    var modelContext: ModelContext

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

```

```

    }

    func hasEntryToday(for email: String) -> Bool { /* ... */ }
    func save(q1: String, q2: String, q3: String, q4: String, email:
String) { /* ... */ }
}

```

Метод `hasEntryToday` перевіряє наявність запису на поточний день для заданого користувача. Реалізація використовує предикат `SwiftData` з фільтрацією за `email` і діапазоном дат:

```

func hasEntryToday(for email: String) -> Bool {
    let today = Calendar.current.startOfDay(for: Date())
    let tomorrow = Calendar.current.date(byAdding: .day, value: 1, to:
today)!
    let desc = FetchDescriptor<DayEntry>(
        predicate: #Predicate {
            $0.userEmail == email &&
            $0.date >= today &&
            $0.date < tomorrow
        }
    )
    return !((try? modelContext.fetch(desc)) ?? []).isEmpty
}

```

Цей метод використовується для визначення стану кнопки створення нового запису на головному екрані: якщо запис на сьогодні вже існує, кнопка переходить у вимкнений стан із відповідним повідомленням.

Метод `save` створює новий екземпляр `DayEntry` з переданими відповідями та поточною датою, додає його до контексту бази даних і виконує збереження. Дата запису нормалізується до початку дня для коректної фільтрації при подальших запитах.

3.3.3 `AIAnalysisViewModel`

Клас `AIAnalysisViewModel` є найбільш складним з трьох — він керує процесом виконання AI-аналізу, обробляє стани завантаження, помилок та працює з вибраним діапазоном дат. Структура класу:

```

@MainActor
final class AIAnalysisViewModel: ObservableObject {
    @Published var startDate: Date = Calendar.current.date(
        byAdding: .day, value: -6, to: Date())!
    @Published var endDate: Date = Date()
    @Published var isLoading: Bool = false
    @Published var errorMessage: String?

    var modelContext: ModelContext?
}

```

```

func entries(for email: String) -> [DayEntry] { /* ... */ }
func existingAnalysis(for email: String) -> WeeklyAnalysis? { /* ...
*/ }
func runAnalysis(entries: [DayEntry], email: String) async { /* ... */
}
}

```

Чотири властивості з обгорткою `@Published` забезпечують реактивне оновлення інтерфейсу:

- `startDate` і `endDate` — діапазон дат, обраний користувачем; за замовчуванням встановлюються як останні 7 днів;
- `isLoading` — індикатор виконання запиту до OpenAI API; коли він `true`, на екрані відображається індикатор завантаження;
- `errorMessage` — текст помилки, що відображається у разі збою.

Метод `entries(for:)` повертає список записів за вибраний діапазон дат для заданого користувача, відсортований за датою:

```

func entries(for email: String) -> [DayEntry] {
    guard let ctx = modelContext else { return [] }
    let start = Calendar.current.startOfDay(for: startDate)
    let end = Calendar.current.date(byAdding: .day, value: 1,
                                    to: Calendar.current.startOfDay(for:
endDate))!
    let desc = FetchDescriptor<DayEntry>(
        predicate: #Predicate {
            $0.userEmail == email &&
            $0.date >= start &&
            $0.date < end
        },
        sortBy: [SortDescriptor(\.date)]
    )
    return (try? ctx.fetch(desc)) ?? []
}

```

Метод `runAnalysis(entries:email:)` — основний метод класу, що ініціює процес AI-аналізу. Він є асинхронним (`async`), що дозволяє виконувати мережевий запит без блокування інтерфейсу. Послідовність дій така.

По-перше, отримується API-ключ OpenAI зі сховища `UserDefaults`. Якщо ключ відсутній — операція переривається з відповідною помилкою.

По-друге, встановлюється стан завантаження (`isLoading = true`), а у блоці `defer` гарантовано скидається у `false` при виході з методу — незалежно від того, успішне виконання чи виняткове.

По-третє, перевіряється наявність попереднього аналізу за той самий діапазон дат. Якщо такий є, він видаляється — щоб уникнути дублювання та забезпечити, що останній виконаний аналіз буде актуальним. Цей підхід обраний як компроміс між зручністю користувача (можливість перезапустити аналіз для оновленого набору записів) та збереженням історії (попередні аналізи за інші діапазони залишаються доступними в архіві).

По-четверте, виконується сам запит до OpenAI API через сервісний клас `OpenAIService` (детально описаний у розділі 3.5.1). Виклик через `try await` дозволяє чекати відповіді, не блокуючи головний потік.

По-п'яте, отриманий результат огортається у новий екземпляр `WeeklyAnalysis` і зберігається у локальну базу.

При виникненні помилки на будь-якому з етапів вона перехоплюється блоком `catch`, її локалізований опис записується у властивість `errorMessage`, що автоматично призводить до показу алерта на екрані.

3.4 Реалізація користувацького інтерфейсу

Користувацький інтерфейс `Diary AI` побудовано повністю на `SwiftUI`. Загалом застосунок містить десять основних екранів, поділених на дві групи: екрани автентифікації (вхід та реєстрація) та основні екрани (головна, новий запис, історія записів, деталі запису, AI-аналіз, архів аналізів, налаштування).

Інтерфейс виконано у мінімалістичному стилі з використанням стандартних компонентів `SwiftUI`, доповнених кастомними картковими компонентами для головного екрана. Як акцентний колір використовується насичений синьо-фіолетовий (RGB: 0.345, 0.337, 0.839), що задає індивідуальність застосунку, не перевантажуючи інтерфейс. Усі екрани автоматично підтримують темну тему, яка вмикається через системні налаштування iOS.

3.4.1 Екрани автентифікації

Екран входу наведено на рисунку 3.3 (LoginView) та екран реєстрації рисунок 3.4 (RegisterView) є точками входу для нових і повертаючих користувачів. Обидва побудовані за подібним візуальним шаблоном: заголовок із назвою застосунку, поля введення з лейблами, основна кнопка дії, посилання на альтернативний екран (з LoginView — на RegisterView і навпаки).

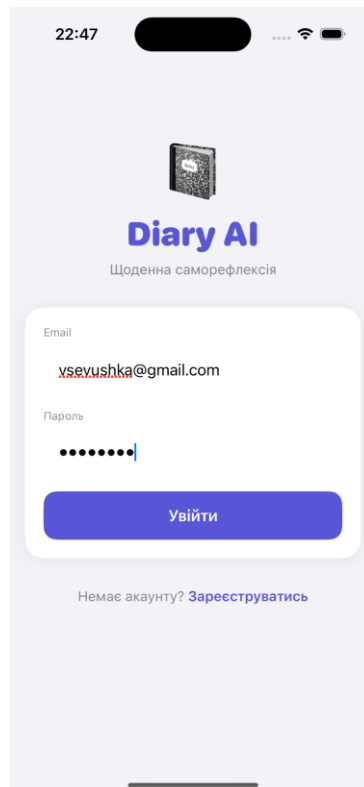


Рис. 3.3 Екран входу до застосунку

Екран реєстрації включає три поля введення: ім'я, email і пароль. Поле пароля використовує SwiftUI-компонент SecureField, що автоматично приховує введені символи. Перед збереженням виконуються базові валідації — непорожні поля, коректний формат email, мінімальна довжина пароля.

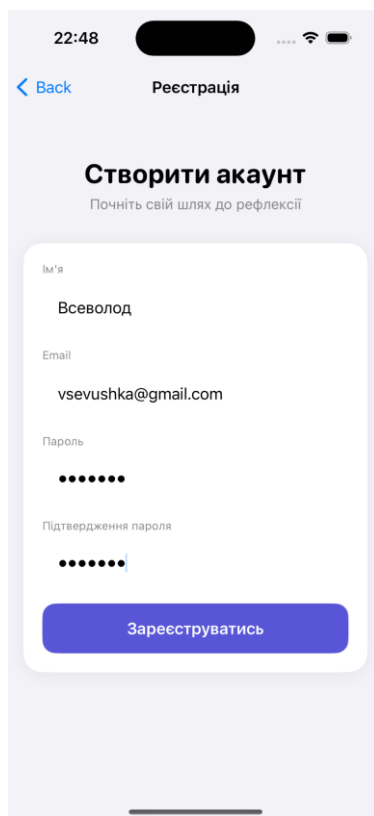


Рис. 3.4 Екран реєстрації нового користувача

При успішній реєстрації користувач автоматично переходить на головний екран без потреби повторно вводити облікові дані — це вдосконалення стандартного UX-сценарію, що зменшує тертя для нових користувачів.

3.4.2 Головний екран та навігація

Головний екран наведено на рисунку 3.5 (HomeView) — центральний хаб застосунку, з якого користувач переходить до основних сценаріїв. Дизайн екрана побудовано навколо трьох великих карток (FullCard) із градієнтними фонами, що візуально відрізняються одна від одної кольорами:

- Новий запис — зелено-блакитний градієнт, асоціюється з продуктивністю та активною дією;
- Мої записи — синьо-фіолетовий градієнт, асоціюється з накопиченим архівом;

— AI-аналіз — пурпурно-фіолетовий градієнт, асоціюється з технологічністю та інсайтами.

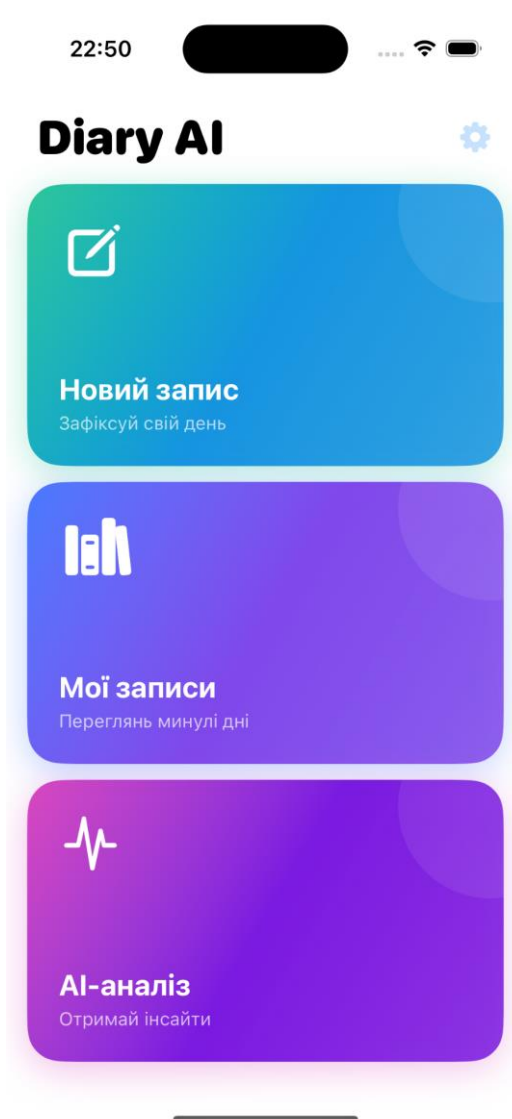


Рис. 3.5 Головний екран застосунку

Картки реалізовані як кастомний компонент `FullCard`, що приймає параметри: іконку (з `SF Symbols` — стандартної бібліотеки іконок Apple), заголовок, підзаголовок, набір кольорів градієнта та опціональний прапорець `disabled`. Останній використовується для картки нового запису, коли запис на сьогодні вже зроблено — у цьому випадку вона стає неактивною з відповідним підписом «Запис сьогодні зроблено / Повернись завтра».

У правому верхньому куті розміщено кнопку переходу до екрана налаштувань, реалізовану як іконка-шестерня. При натисканні відкривається модальний `SettingsView`.

Перевірка наявності запису на сьогодні виконується реактивно через `@Query`-запит, що повертає всі записи поточного користувача. Це гарантує, що картка автоматично оновлюється відразу після створення нового запису, без потреби у явному перезавантаженні екрана.

3.4.3 Екран створення запису

Екран створення нового запису наведено на рисунку 3.6 (`NewEntryView`) реалізує покрокову форму відповіді на чотири запитання. Замість традиційного підходу з показом усіх запитань на одному екрані з прокручуванням, у `Diary AI` використано формат «майстра» (wizard) — кожне запитання займає окремий слайд, а перехід між слайдами здійснюється кнопками «Далі» та «Назад» внизу екрана.

Такий підхід має дві переваги. По-перше, користувач концентрується на одному запитанні за раз, що сприяє якіснішим, обдуманним відповідям. По-друге, інтерфейс залишається чистим і не перевантаженим, що особливо важливо на менших екранах `iPhone SE` або `mini`-моделей.

Зверху екрана відображається індикатор прогресу (`ProgressView`), що показує поточний крок із чотирьох. У головній частині екрана реалізовано `TabView` зі стилем `.page(indexDisplayMode: .never)` — це створює слайдовий інтерфейс без видимих індикаторів пагінації. Системний жест свайпу між сторінками примусово вимкнено через `.scrollDisabled(true)`, щоб користувач не міг пропустити обов'язкові запитання, не давши на них відповідь.

Кожен слайд містить запитання та поле введення (`TextEditor`) для відповіді. Перші три запитання є обов'язковими (позначені червоною зірочкою), четверте — необов'язкове. Кнопка «Далі» блокується, якщо поточне обов'язкове запитання залишається без відповіді — це реалізовано

через обчислювальну властивість `canGoNext`. На останньому кроці кнопка «Далі» замінюється на «Зберегти запис».

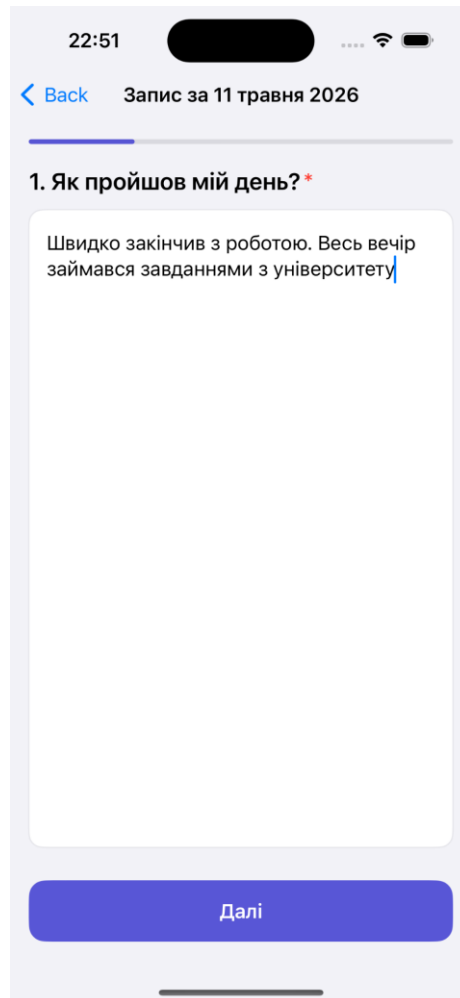


Рис. 3.6 Екран створення нового запису

Перед збереженням ще раз перевіряється наявність запису на сьогодні (через властивість `hasEntryToday`) — якщо такий запис є, виводиться попередження і користувач не може створити дублікат. Це додатковий захист від випадкового створення кількох записів на один день.

3.4.4 Перегляд історії та деталей запису

Екран історії наведено на рисунку 3.7 (`HistoryView`) відображає хронологічний список усіх записів користувача, відсортованих від

найновіших до найстаріших. Кожен елемент списку показує дату запису, день тижня та фрагмент першої відповіді як попередній перегляд.

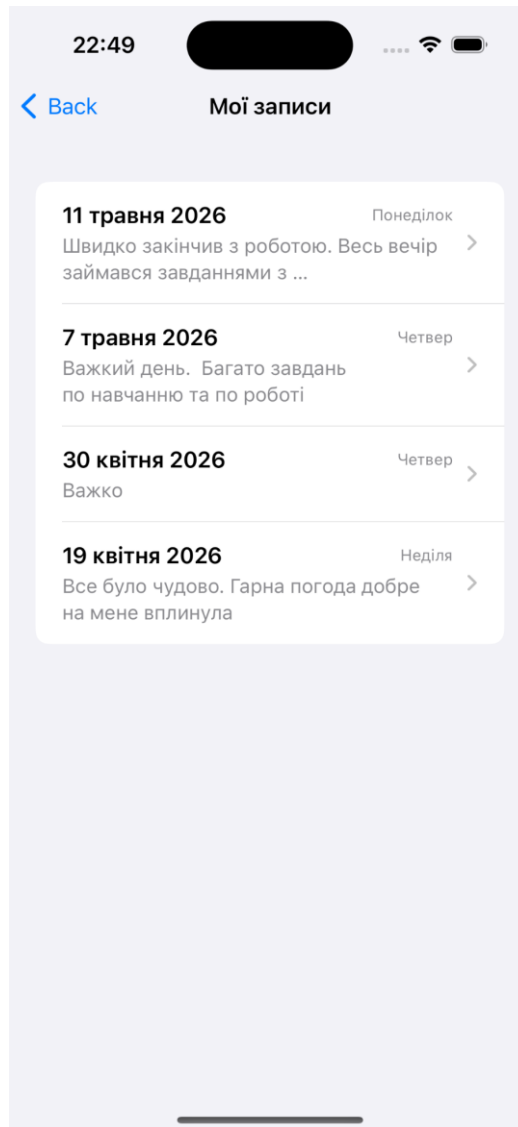


Рис. 3.7 Екран історії записів

Список реалізовано через `@Query(sort: \DayEntry.date, order: .reverse)`. Завдяки реактивності `SwiftData` при додаванні нового запису він з'являється у списку без явного оновлення.

При натисканні на елемент списку відбувається перехід на екран деталей запису наведеному на рисунку 3.8 (`EntryDetailView`), що показує повну інформацію: дату, день тижня та повні тексти всіх чотирьох відповідей. Кожне запитання та відповідь винесені у окремий блок із заголовком запитання, що покращує читабельність архівних записів.

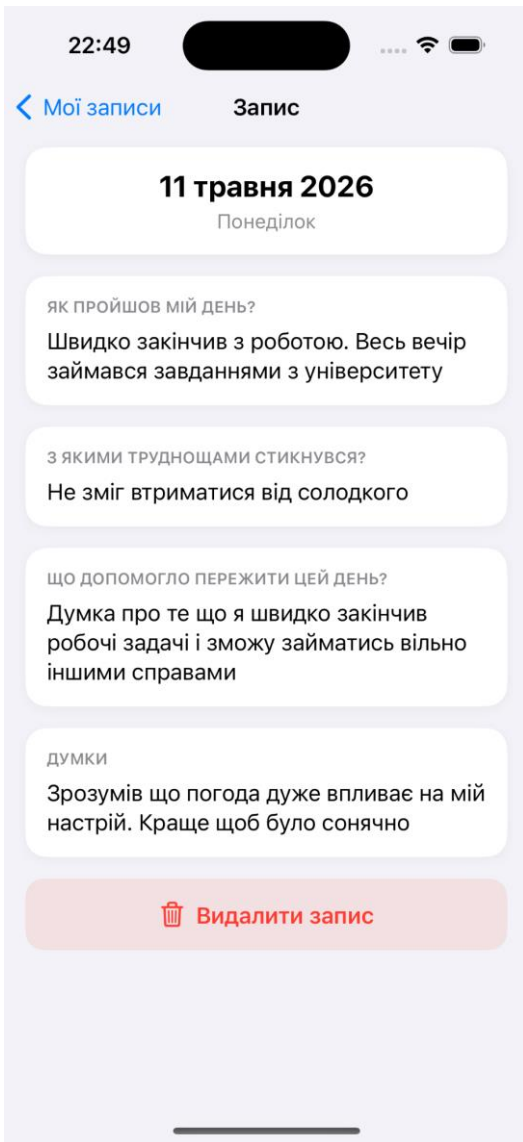


Рис. 3.8 Екран деталей запису

3.4.5 Екран AI-аналізу та архів

Екран AI-аналізу наведено на рисунку 3.9 (AIAnalysisView) — найбільш функціонально насичений екран застосунку. Він складається з кількох логічних блоків, які з'являються залежно від поточного стану.

Зверху розміщені два компоненти DatePicker для вибору початкової та кінцевої дати періоду аналізу. За замовчуванням обрано останні 7 днів, але користувач може змінити діапазон у межах від найранішого запису до сьогоднішнього дня.

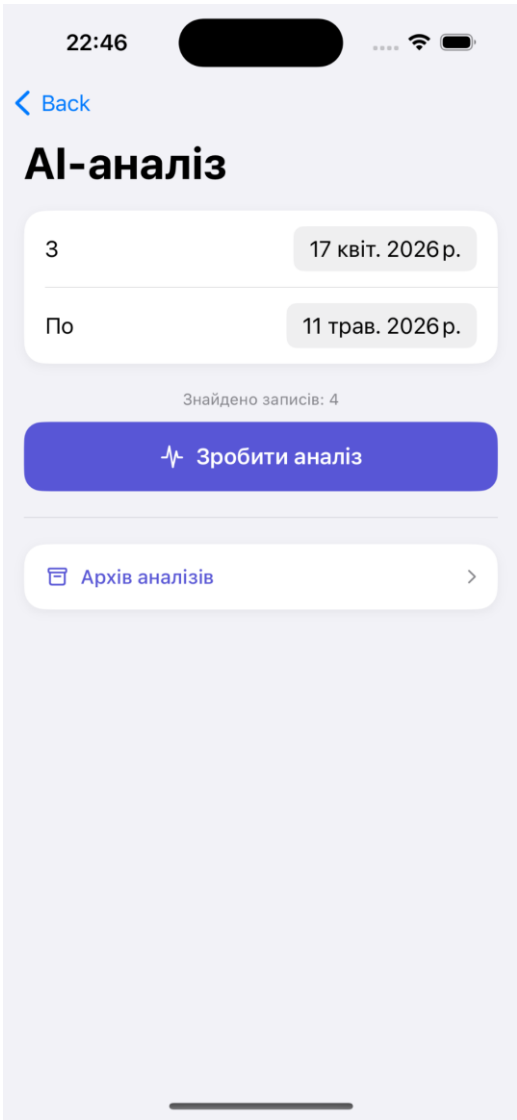


Рис. 3.9 Екран AI-аналізу

Нижче відображається інформаційний блок із кількістю знайдених записів за вибраний період. Якщо записів менше трьох, замість кнопки запуску аналізу показується повідомлення про недостатню кількість даних з конкретними цифрами (наприклад, «Потрібно мінімум 3, знайдено: 1»). Це обмеження запроваджено для забезпечення достатньої кількості матеріалу для якісного аналізу — на одному-двох записах мовна модель не може виявити змістовні патерни.

Якщо записів достатньо, відображається кнопка «Зробити аналіз». При натисканні запускається асинхронний процес: на екрані з'являється індикатор

завантаження з підписом «Аналізую записи...», а після отримання відповіді автоматично відкривається модальний екран з результатом аналізу.

Результат аналізу подається у вигляді трьох карток із візуальним розрізненням за кольорами (див. рис. 3.10):

- Позитивні сторони — зелена картка з іконкою ✓;
- Виявлені труднощі — жовта картка з іконкою ⚠;
- Рекомендації — синя картка з іконкою 💡.

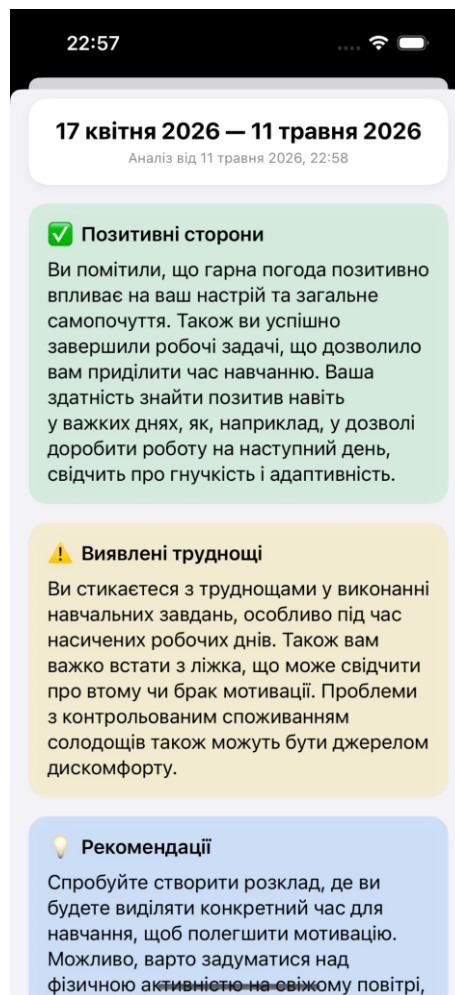


Рис. 3.10 Результат AI-аналізу

У нижній частині екрана AI-аналізу розміщено посилання на архів аналізів (див. рис. 3.11). При натисканні відкривається модальний екран AnalysisArchiveView зі списком усіх раніше виконаних аналізів. Кожен елемент списку показує діапазон дат аналізованого періоду, дату виконання аналізу та фрагмент блоку «Позитивні сторони» як попередній перегляд.

Вбудована функція видалення (onDelete) дозволяє прибирати непотрібні аналізи свайпом.

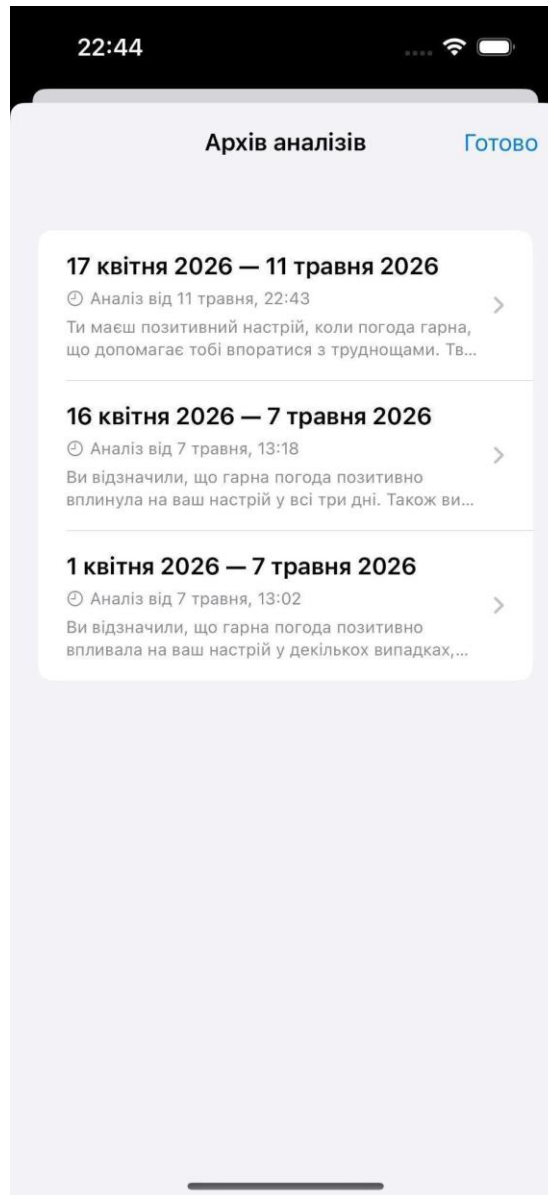


Рис. 3.11 Архів виконаних AI-аналізів

3.4.6 Екран налаштувань

Екран налаштувань наведено на рисунку 3.12 (SettingsView) реалізовано через стандартний компонент List зі стилем .insetGrouped, що відповідає типовому iOS-патерну для екранів налаштувань. Усі опції згруповані в чотири логічні розділи.

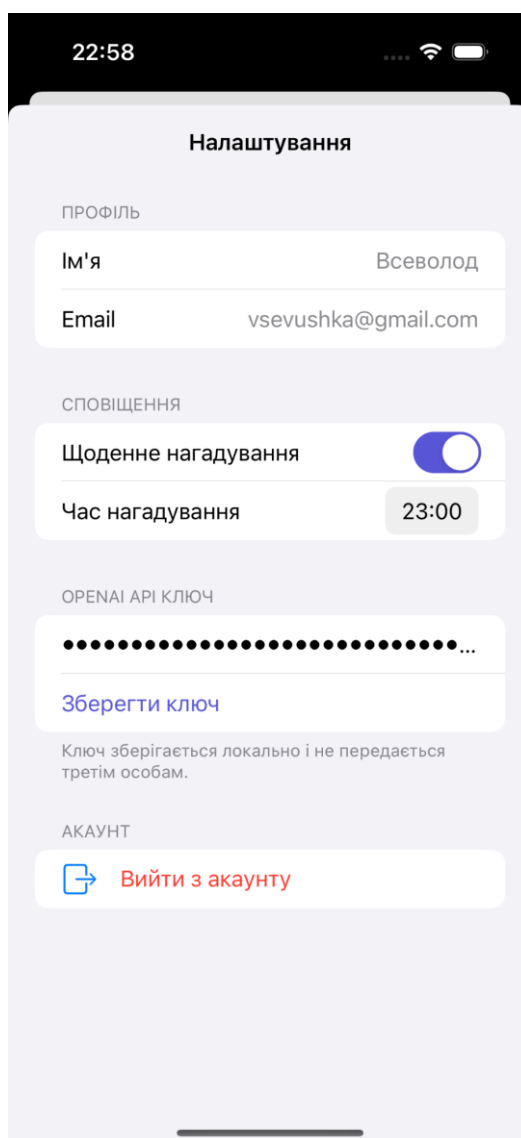


Рис. 3.12 Екран налаштувань

Розділ «Профіль» — інформаційний блок із даними поточного користувача (ім'я, email). Поля доступні лише для перегляду.

Розділ «Сповіщення» — перемикач щоденних нагадувань і вибір часу. При увімкненні перемикача застосунок запитує дозвіл на показ сповіщень через системний діалог. Якщо дозвіл надано, нижче з'являється DatePicker для вибору часу нагадування. При зміні часу попередньо заплановані сповіщення скасовуються та плануються нові з оновленим часом — це реалізовано у сервісному класі NotificationService (детальніше у розділі 3.5.2). За замовчуванням час нагадування встановлено на 21:00 — типовий вечірній час, коли користувач має змогу спокійно зробити запис.

Розділ «OpenAI API ключ» — поле введення (SecureField), кнопка збереження ключа та інформаційний підпис про те, що ключ зберігається локально і не передається третім особам. Передбачено візуальний фідбек — при успішному збереженні поряд із кнопкою на 2 секунди з'являється зелена іконка-галочка. Можливість використання власного API-ключа є важливим архітектурним рішенням: вона дозволяє не приховувати ключ у коді застосунку (що було б небезпечно), а водночас дає кожному користувачу контроль над власними витратами на запити.

Розділ «Акаунт» — кнопка виходу з облікового запису. Виконана у червоному кольорі (role: .destructive) як деструктивна дія. При натисканні з'являється підтверджуюче діалогове вікно. Після підтвердження сесія очищається, і застосунок автоматично повертається до екрана входу.

3.5 Сервісний шар

Сервісний шар застосунку відповідає за взаємодію із зовнішніми системами та інкапсулює технічні деталі цієї взаємодії, надаючи решті коду застосунку прості, ясні інтерфейси. Реалізовано два сервіси: OpenAIService для звернень до AI-моделі та NotificationService для роботи зі сповіщеннями. Обидва реалізовано через Swift enum без екземплярів — як набори статичних методів. Такий підхід обрано замість класів-сервісів, оскільки сервіси не мають внутрішнього стану, який треба зберігати між викликами.

3.5.1 OpenAIService

OpenAIService — ключовий сервіс застосунку, що виконує запити до OpenAI API для отримання AI-аналізу записів. Він складається з основного методу analyze, допоміжного методу формування промпту, типу результату AnalysisResult та переліку специфічних помилок OpenAIError.

Тип результату декларується як простий Decodable-об'єкт із трьома полями, що відповідають структурі очікуваної JSON-відповіді від моделі:

```
struct AnalysisResult: Decodable {
```

```

    let positives: String
    let challenges: String
    let recommendations: String
  }

```

Перелік `OpenAIError` визначає можливі помилки на різних етапах виконання запиту, з локалізованими описами для відображення користувачу:

- `noAPIKey` — API-ключ не введено в налаштуваннях;
- `invalidResponse` — сервер повернув некоректну відповідь;
- `httpError(Int)` — HTTP-статус відмінний від 200;
- `apiMessage(String)` — помилка, повернута самим OpenAI API (наприклад, неправильний ключ або вичерпана квота);
- `parsingFailed(String)` — не вдалося розпарсити відповідь моделі.

Реалізація основного методу `analyze` виконує таку послідовність дій.

Перевірка API-ключа. Якщо ключ порожній, викидається помилка `noAPIKey` з дружнім повідомленням, що спрямовує користувача в налаштування.

Формування HTTP-запиту. Створюється `URLRequest` із методом `POST`, цільовим URL `https://api.openai.com/v1/chat/completions`, заголовками авторизації та `Content-Type`:

```

var req = URLRequest(url: URL(string:
  "https://api.openai.com/v1/chat/completions")!)
req.httpMethod = "POST"
req.setValue("Bearer \(apiKey)", forHTTPHeaderField: "Authorization")
req.setValue("application/json", forHTTPHeaderField: "Content-Type")

```

Формування тіла запиту. JSON-об'єкт містить параметри моделі та повідомлення користувача — текст промпту, що генерується методом `prompt(entries):`

```

let body: [String: Any] = [
  "model": "gpt-4o-mini",
  "temperature": 0.7,
  "max_tokens": 1000,
  "messages": [{"role": "user", "content": prompt(entries)}]
]
req.httpBody = try JSONSerialization.data(withJSONObject: body)

```

Виконання запиту. Використовується сучасний асинхронний API `URLSession`:

```

let (data, response) = try await URLSession.shared.data(for: req)

```

Обробка HTTP-помилки. Якщо статус-код відмінний від 200, спроба витягти повідомлення про помилку з тіла відповіді (OpenAI повертає JSON з полем `error.message` у випадку помилки). Якщо це вдається — викидається `apiMessage`, в іншому разі — `httpError`.

Розпарсювання відповіді. Послідовно витягуються поля з JSON: `choices` → перший елемент → `message` → `content`. Якщо на будь-якому з кроків структура не відповідає очікуваній — викидається `invalidResponse`.

Очищення відповіді. Незважаючи на явну вказівку в промті повертати тільки чистий JSON, мовна модель іноді обгортає відповідь у `markdown`-блоки. Тому виконується додаткове очищення:

```
let cleaned = content
    .replacingOccurrences(of: "`json", with: "")
    .replacingOccurrences(of: "`", with: "")
    .trimmingCharacters(in: .whitespacesAndNewlines)
```

Декодування у структуру. Очищений рядок декодується у `AnalysisResult` через `JSONDecoder`. У разі помилки декодування викидається `parsingFailed` з описом проблеми.

Окремо варто розглянути формування промту. Метод `prompt(entries:)` будує текст у такому форматі:

Ти – персональний асистент для саморефлексії.
Ось записи користувача за тиждень:

☐ Понеділок, 15 квітня
– Як пройшов день: [текст]
– Труднощі: [текст]
– Що допомогло: [текст]
– Думки: [текст]

☐ Вівторок, 16 квітня
...

Проаналізуй ці записи та надай відповідь СТРОГО у форматі JSON (без `markdown`, без `markdown`-блоків, тільки чистий JSON об'єкт):
{`"positives": "...", "challenges": "...", "recommendations": "..."`}
Відповідай українською мовою. Будь конкретним і персональним.

Декілька деталей у структурі промту мають принципове значення.

Роль асистента. Перше речення формує контекст — модель отримує «роль» персонального асистента саморефлексії, що налаштовує тон і характер відповіді.

Структуроване подання записів. Кожен день розділений емодзі-маркером, а кожна відповідь — мітками з тире. Це допомагає моделі чітко розрізняти, де закінчується один запис і починається інший.

Локалізована дата. Дата кожного запису форматується через DateFormatter з локаллю uk_UA, що дає українські назви днів тижня та місяців («Понеділок, 15 квітня»). Це покращує орієнтацію моделі в часі та дозволяє їй посилалися на конкретні дні у відповіді.

Опціональний блок «Думки». Поле question4 додається до промπτу лише якщо воно непорожнє — це уникає зайвого шуму у запиті.

Чітка інструкція щодо формату. Слово «СТРОГО» та подальші деталі написані з намірено імперативним тоном — практика показала, що це підвищує ймовірність отримання чистого JSON без обгортки.

Вказівка щодо мови. Явна команда відповідати українською гарантує україномовну відповідь незалежно від того, якою мовою написані конкретні записи (адже користувач може вставити окремі англійські слова чи фрази).

Команда на конкретність. Інструкція «Будь конкретним і персональним» спонукає модель уникати загальних, шаблонних рекомендацій типу «приділяйте більше уваги собі» і натомість давати поради, що базуються на конкретному вмісті записів.

3.5.2 NotificationService

NotificationService інкапсулює логіку роботи з підсистемою сповіщень iOS через фреймворк UserNotifications. Сервіс надає три статичні методи: запит дозволу, планування щоденного нагадування та скасування всіх запланованих сповіщень.

Метод requestPermission(completion:) запитує у користувача дозвіл на показ сповіщень через системний діалог. Запитуються опції alert (показ повідомлення), sound (програвання звуку) та badge (індикатор на іконці застосунку):

```
static func requestPermission(completion: @escaping (Bool) -> Void) {
    UNUserNotificationCenter.current()
```

```

        .requestAuthorization(options: [.alert, .sound, .badge]) {
    granted, _ in
        DispatchQueue.main.async { completion(granted) }
    }
}

```

Виклик `DispatchQueue.main.async` гарантує, що замикання з результатом виконається у головному потоці — це необхідно, оскільки результат використовується для оновлення інтерфейсу.

Метод `scheduleDailyReminder(at:)` планує щоденне нагадування у заданий час:

```

static func scheduleDailyReminder(at time: Date) {
    let center = UNUserNotificationCenter.current()
    center.removeAllPendingNotificationRequests()

    let content = UNMutableNotificationContent()
    content.title = "Diary AI"
    content.body = "Час для рефлексії 📖 Як пройшов твій день?"
    content.sound = .default

    let comps = Calendar.current.dateComponents([.hour, .minute], from:
time)
    let trigger = UNCalendarNotificationTrigger(dateMatching: comps,
repeats: true)
    let request = UNNotificationRequest(
        identifier: "daily_reflection",
        content: content,
        trigger: trigger
    )
    center.add(request)
}

```

Кілька деталей реалізації заслуговують на увагу.

Перед плануванням нового сповіщення викликається `removeAllPendingNotificationRequests()` — це очищає всі попередні заплановані сповіщення. Такий підхід гарантує, що при зміні часу нагадування старе сповіщення не залишиться у системі.

Тригер створюється з компонентами часу (година і хвилина) та параметром `repeats: true` — це означає, що сповіщення буде надходити щодня у заданий час, а не одноразово.

Унікальний ідентифікатор сповіщення («daily_reflection») дозволяє при потребі звертатися саме до цього сповіщення (наприклад, для його окремого

скасування), не зачіпаючи інших сповіщень, які можуть бути додані в майбутніх версіях.

Текст сповіщення містить емодзі-місяць як візуальний акцент, що робить нагадування більш привітним і менш формальним.

Метод `cancelAll()` скасовує всі заплановані сповіщення — використовується при вимиканні нагадувань у налаштуваннях.

3.6 Безпека збереження облікових даних

Питання безпеки в застосунку Diary AI має певну специфіку: усі персональні дані (записи, аналізи, профіль) зберігаються виключно локально на пристрої користувача, без передачі на будь-які зовнішні сервери. Виняток складає лише тимчасова передача тижневого набору записів на OpenAI API в момент виконання запиту аналізу — і ця передача йде по захищеному каналу HTTPS.

Така архітектурна модель — «local-first» — суттєво змінює профіль безпекових загроз. Класичні ризики, пов'язані з компрометацією серверної бази даних, перехопленням мережевого трафіку або витоком даних на стороні провайдера, є для Diary AI неактуальними. Натомість основні ризики — це фізичний доступ до пристрою користувача та зловмисне програмне забезпечення, що отримало доступ до пісочниці застосунку.

Враховуючи цю специфіку, у Diary AI реалізовано такі заходи безпеки.

Хешування паролів. Паролі користувачів ніколи не зберігаються у відкритому вигляді [24]. Перед збереженням пароль обчислюється алгоритмом SHA-256 [17] через системний фреймворк CryptoKit і у базу записується тільки шістнадцятковий рядок хешу. Алгоритм SHA-256 є криптографічно стійкою однонаправленою функцією — навіть маючи хеш, обчислювально неможливо відновити вихідний пароль за прийнятний час.

Базова валідація на стороні застосунку. Перед обробкою email і пароль проходять базові перевірки: обрізання пробілів, нормалізація регістру для

email, перевірка на непорожні значення. Це захищає від випадкових некоректних введенень.

Захищена передача даних до OpenAI. Усі звернення до OpenAI API виконуються по HTTPS-протоколу. URLSession iOS автоматично виконує перевірку TLS-сертифіката сервера, що захищає від атак типу «man-in-the-middle».

Локальне зберігання API-ключа. OpenAI API-ключ, наданий користувачем, зберігається у UserDefaults поточного застосунку. Ця область пам'яті ізольована за рахунок iOS-механізму пісочниці (App Sandbox) — інші застосунки на пристрої не мають до неї доступу.

Ізоляція даних SwiftData. Локальна база SwiftData розташована у каталозі застосунку, що також захищений механізмом пісочниці. На сучасних пристроях iOS це сховище додатково шифрується системою Apple Data Protection при заблокованому пристрої.

Серед потенційних напрямків посилення безпеки в наступних версіях застосунку можна виділити такі:

- використання Keychain Services для зберігання API-ключа та токенів сесії — це додало б апаратний рівень захисту через Secure Enclave;
- використання сповільненого хешування (наприклад, PBKDF2, scrypt або Argon2) із «солінням» для паролів — захист від атак перебором, особливо актуальний у разі компрометації базової файлової системи;
- шифрування контенту записів окремим ключем, отриманим з пароля користувача — захист самого вмісту записів від читання навіть при фізичному доступі до пристрою.

У контексті бакалаврської роботи реалізований рівень безпеки є достатнім — він демонструє розуміння базових принципів захисту облікових даних та охоплює основні ризики, актуальні для архітектури «local-first». Розширені механізми безпеки залишаються відкритим напрямком для подальшого розвитку проєкту.

4 ДЕМОНСТРАЦІЯ ТА ТЕСТУВАННЯ

У цьому розділі представлено результати практичної роботи розробленого застосунку Diary AI. Виконано покрокову демонстрацію основних користувацьких сценаріїв роботи — від реєстрації нового облікового запису до отримання AI-аналізу.

4.1 Демонстрація основних сценаріїв роботи

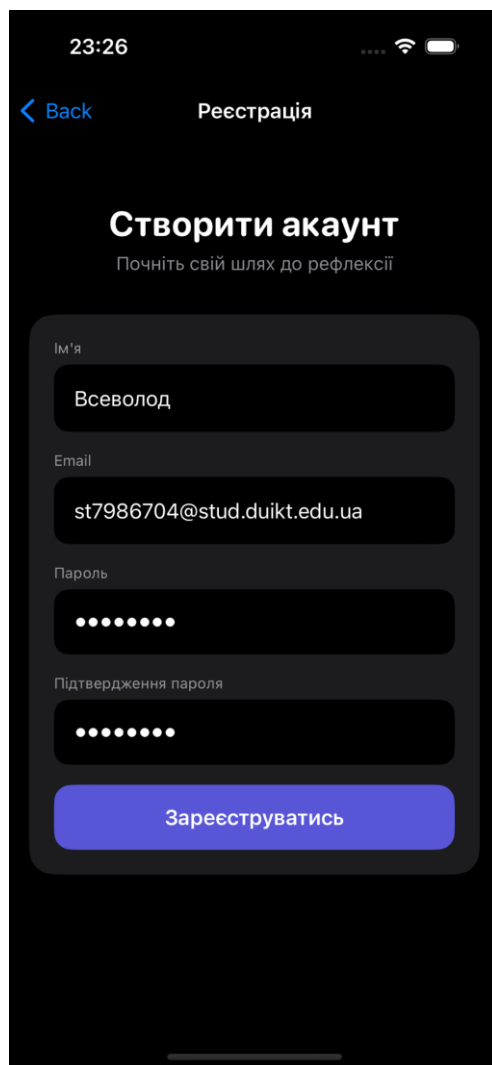
Для демонстрації роботи застосунку розглянемо повний сценарій взаємодії нового користувача з Diary AI — від першого запуску до отримання тижневого AI-аналізу. Усі скріншоти зроблено на симуляторі iPhone 15 Pro в актуальній версії iOS 17.

4.1.1 Реєстрація нового користувача

При першому запуску застосунку відкривається екран входу (див. рис. 4.1). Оскільки обліковий запис ще не створено, користувач переходить за посиланням внизу екрана на форму реєстрації.

Користувач вводить ім'я, email та пароль. Після натискання кнопки «Зареєструватися» застосунок виконує валідацію введених даних: перевіряє непорожність полів, коректність формату email, мінімальну довжину пароля.

У разі успішної валідації обліковий запис створюється у локальній базі (з обчисленням SHA-256-хешу пароля), і користувач автоматично переходить на головний екран без потреби повторно вводити дані для входу.



23:26

< Back Реєстрація

Створити акаунт

Почніть свій шлях до рефлексії

Ім'я

Всеволод

Email

st7986704@stud.duikt.edu.ua

Пароль

••••••••

Підтвердження пароля

••••••••

Зареєструватись

Рис. 4.1 Процес реєстрації нового користувача

4.1.2 Створення першого щоденного запису

Після успішної реєстрації на головному екрані відображаються три картки (див. рис. 4.2). Картка «Новий запис» виділена яскравим зелено-блакитним градієнтом, що візуально підказує користувачу, з чого почати.

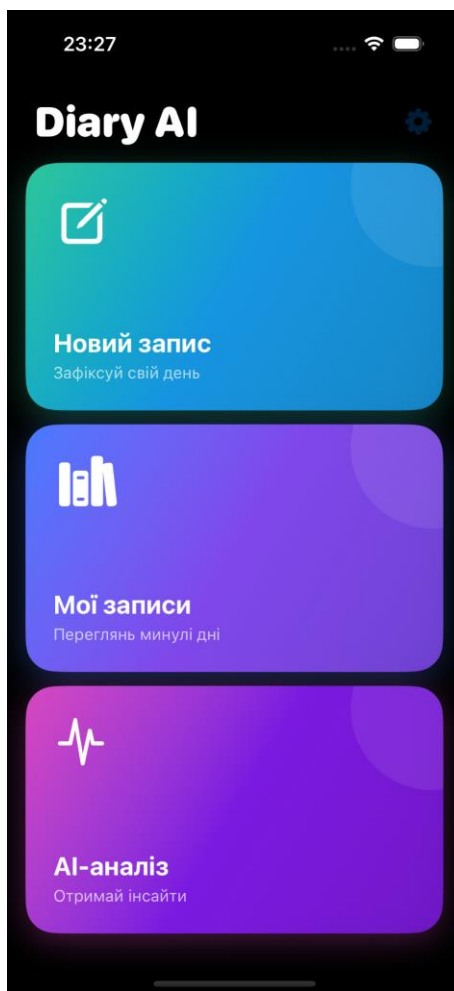


Рис. 4.2 Головний екран при першому використанні

При натисканні на картку відкривається покрокова форма створення запису (див. рис. 4.3). Користувач послідовно проходить чотири запитання:

- Як пройшов мій день? — головна відповідь, що описує загальне сприйняття дня;
- З якими труднощами ти сьогодні стикнувся? — фіксація викликів і проблем;
- Що допомогло тобі пережити цей день? — виявлення власних ресурсів і стратегій подолання;
- Твої думки (за бажанням) — вільне поле для додаткових нотаток.

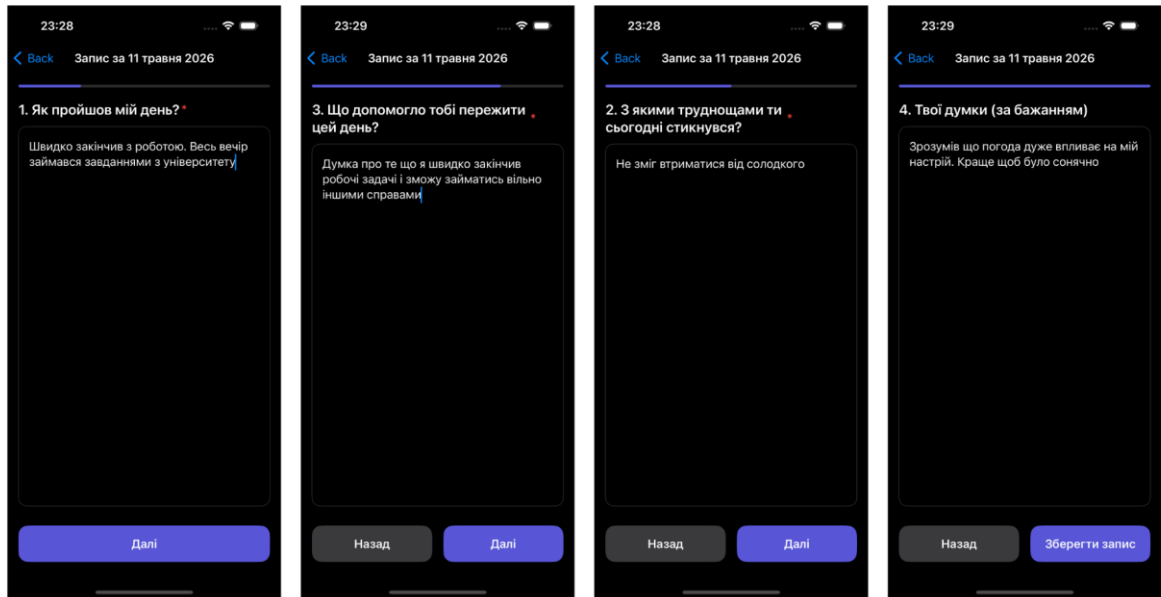


Рис. 4.3 Покрокове заповнення щоденного запису

Перші три запитання обов'язкові — кнопка «Далі» залишається неактивною, поки відповідь порожня.

Після відповіді на четверте запитання кнопка змінюється на «Зберегти запис».

Після збереження користувач бачить підтверджуюче повідомлення та повертається на головний екран, де картка «Новий запис» тепер відображається у неактивному стані з підписом «Запис сьогодні зроблено / Повернись завтра» (див. рис. 4.4).

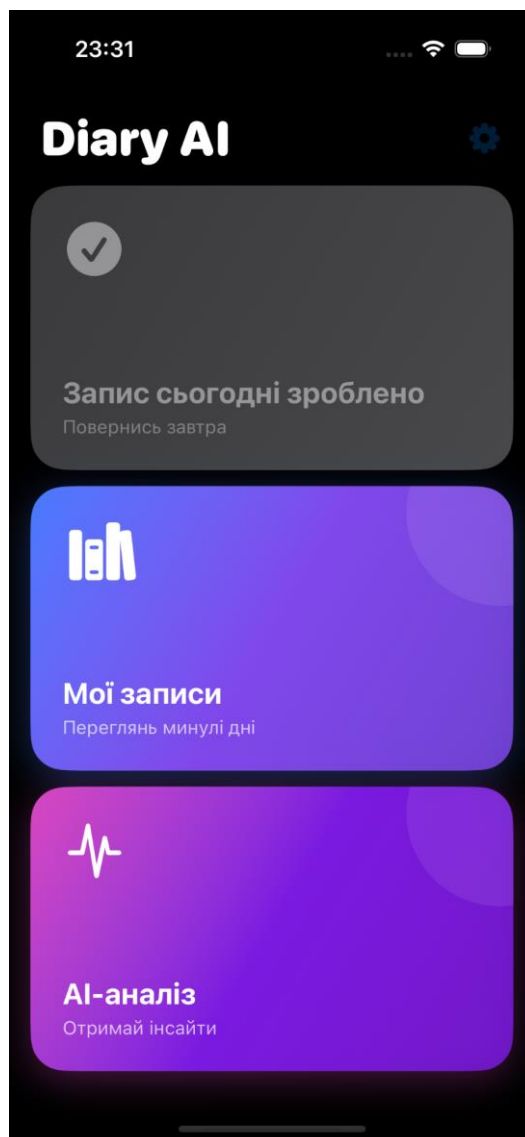


Рис. 4.4 Головний екран після створення запису на сьогодні

4.1.3 Перегляд історії записів

Через кілька днів регулярного використання у користувача накопичується історія записів, яку можна переглянути через картку «Мої записи» на головному екрані (див. рис. 4.5).

Список відображає всі записи у хронологічному порядку від найновіших до найстаріших, з відображенням дати, дня тижня та фрагмента першої відповіді як попереднього перегляду.

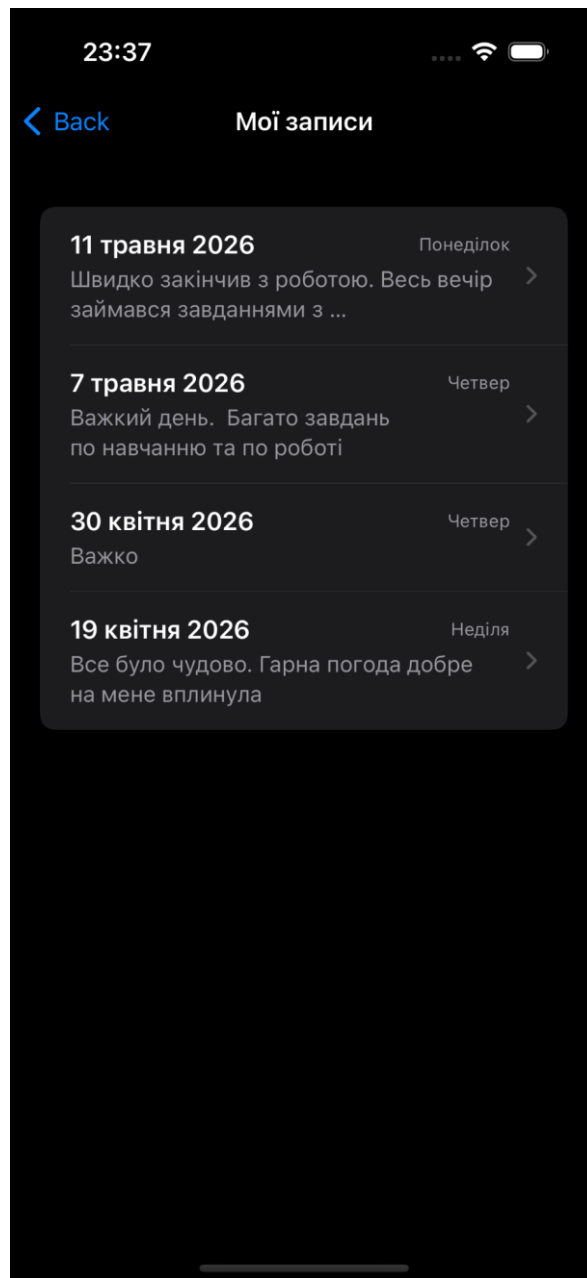


Рис. 4.5 Перегляд історії записів

При натисканні на елемент списку відкривається детальний перегляд запису з повним текстом усіх відповідей (див. рис. 4.6).

Кожне запитання та відповідь винесені у окремий блок із заголовком, що покращує читабельність архівних записів.

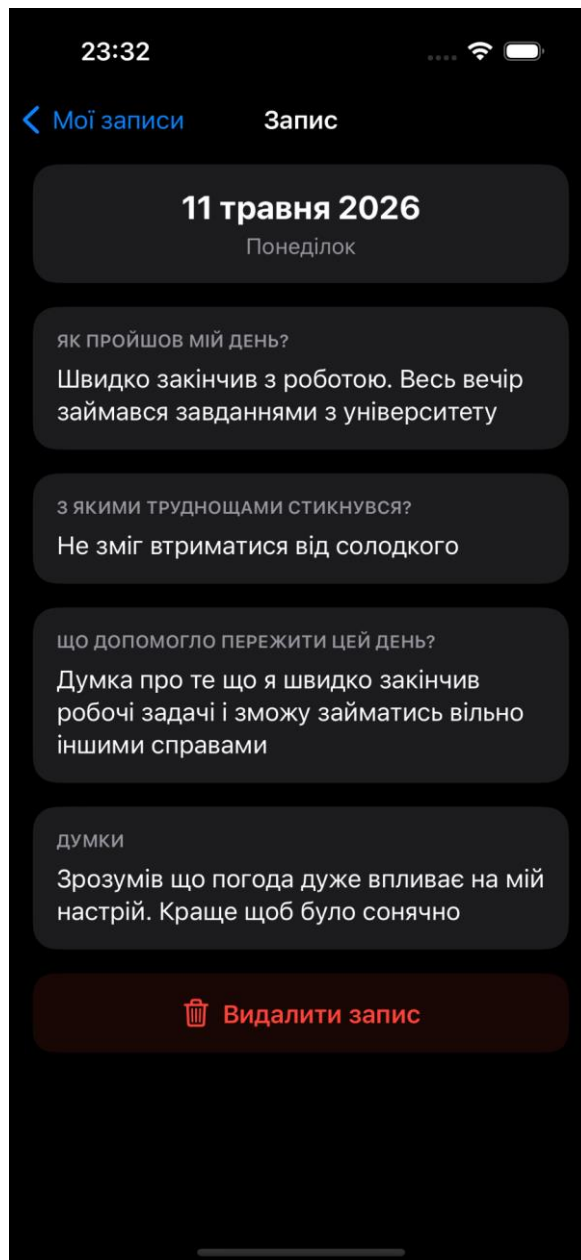


Рис. 4.6 Детальний перегляд щоденного запису

4.1.4 Налаштування OpenAI API ключа та сповіщень

Перед першим використанням AI-аналізу користувач має ввести власний OpenAI API-ключ через екран налаштувань (див. рис. 4.7). Це необхідна умова для виконання запитів до моделі.

Налаштування відкриваються через іконку шестерні у правому верхньому куті головного екрана.

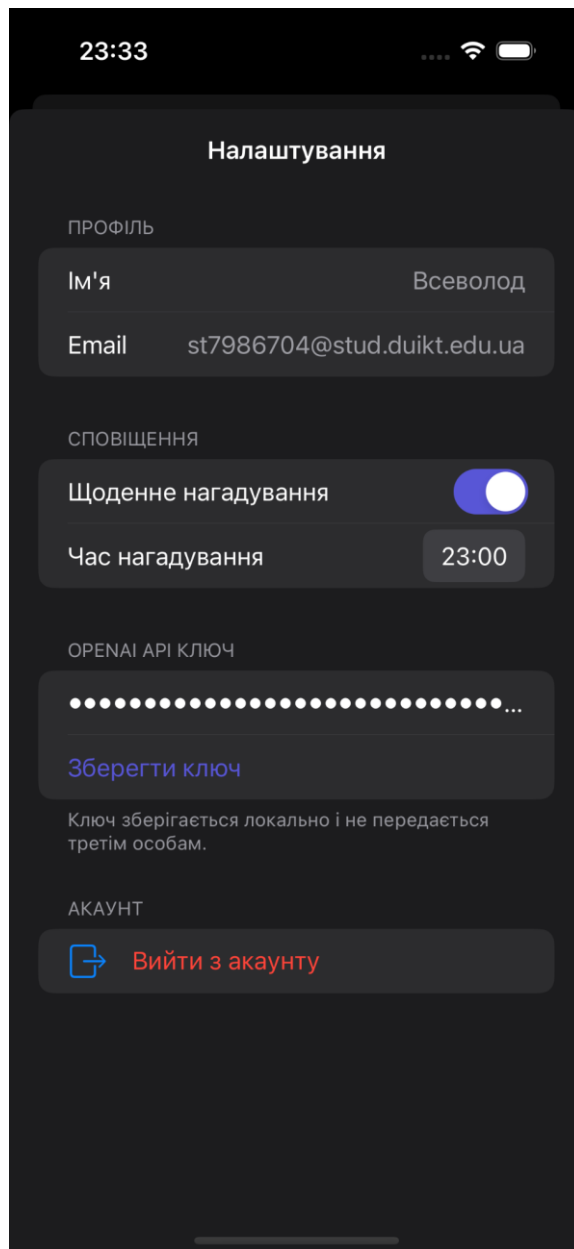


Рис. 4.7 Екран налаштувань застосунку

У розділі «OpenAI API ключ» користувач вводить свій ключ (отриманий на сайті OpenAI Platform) і натискає «Зберегти ключ». Поряд із кнопкою на 2 секунди з'являється зелена іконка-галочка, що підтверджує успішне збереження.

У тому ж екрані можна увімкнути щоденні нагадування. При увімкненні перемикача застосунок запитує системний дозвіл на показ сповіщень, після чого з'являється елемент вибору часу нагадування.

4.1.5 Виконання AI-аналізу

Після накопичення достатньої кількості записів (мінімум трьох) користувач може запустити AI-аналіз. Перехід виконується через картку «AI-аналіз» на головному екрані.

На екрані аналізу користувач обирає діапазон дат для аналізу (див. рис. 4.8). За замовчуванням обрано останні 7 днів. Якщо за вибраний період знайдено недостатньо записів, замість кнопки запуску відображається інформаційне повідомлення.

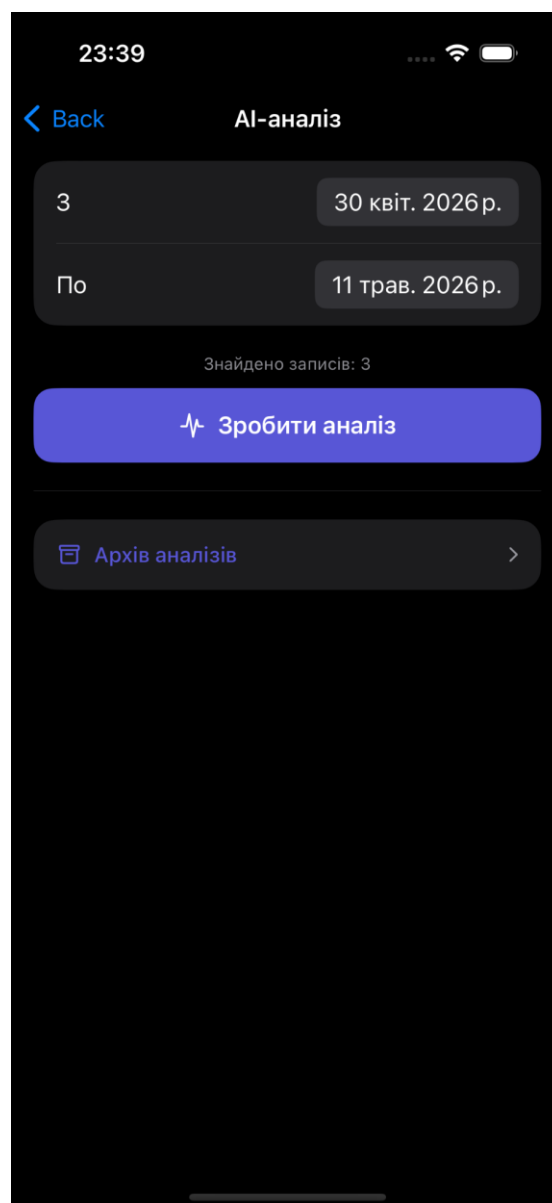


Рис. 4.8 Екран запуску AI-аналізу

При натисканні кнопки «Зробити аналіз» застосунок надсилає запит до OpenAI API. На екрані з'являється анімований індикатор завантаження з підписом «Аналізую записи...». Після отримання відповіді (типово 5–15 секунд) відкривається модальний екран із результатом, що містить три тематичні блоки (див. рис. 4.9).

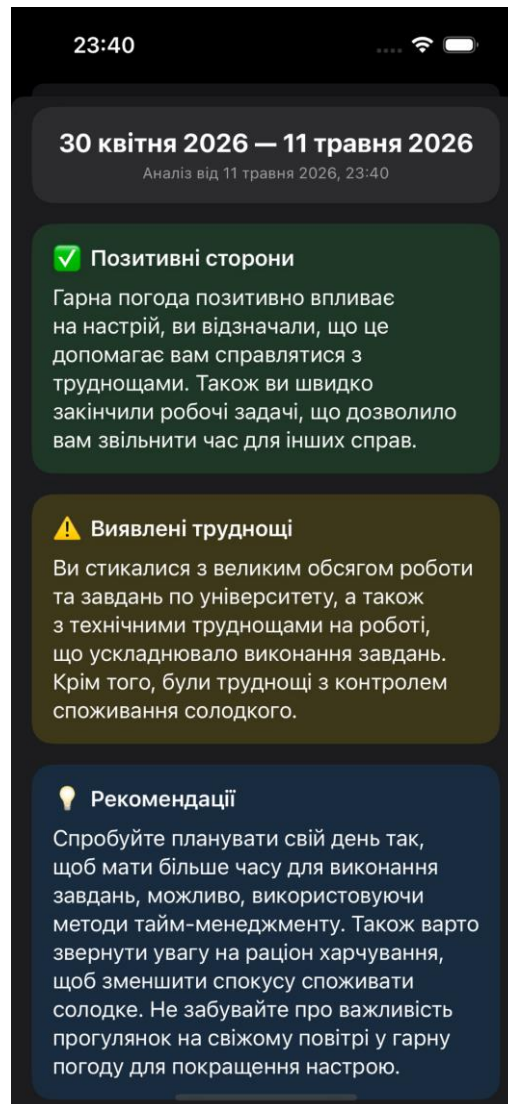


Рис. 4.9 Результат AI-аналізу

Виконані аналізи автоматично зберігаються у локальній базі та доступні для повторного перегляду через посилання «Архів аналізів» внизу екрана (див. рис. 4.10). Архів дозволяє переглядати всі попередні аналізи, відкривати їх повний вміст і видаляти непотрібні свайпом.

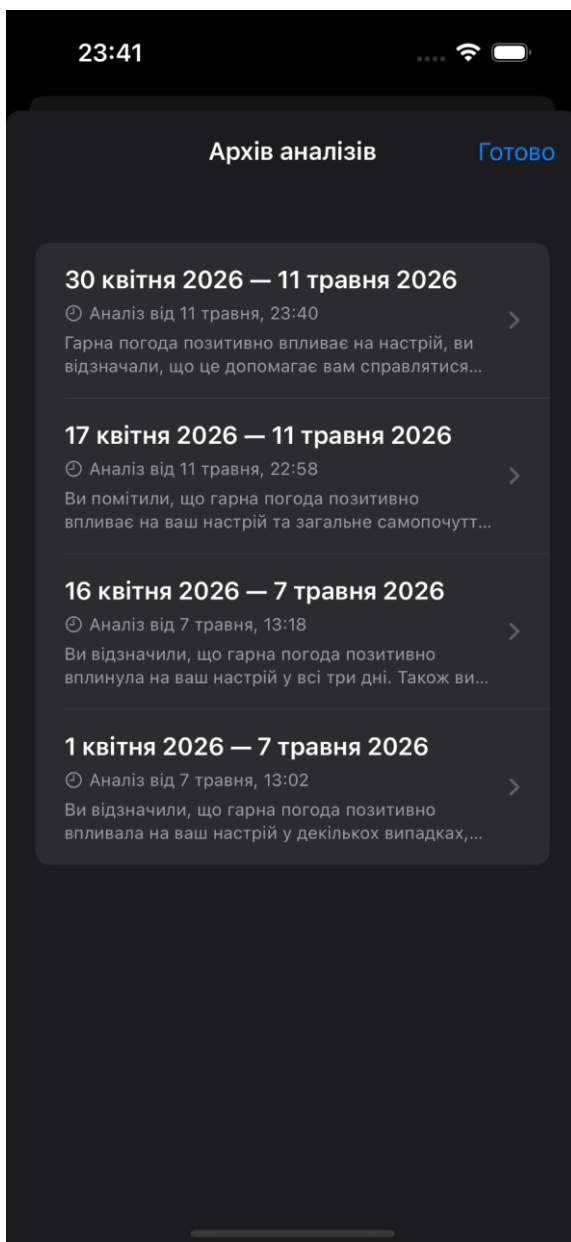


Рис. 4.10 Архів виконаних AI-аналізів

4.2 Тестування застосунку

Для перевірки коректності роботи ключових компонентів застосунку Diary AI було проведено модульне тестування (unit-тестування). Цей вид тестування передбачає автоматизовану перевірку окремих функцій та методів у ізоляції від решти системи, що дозволяє виявляти помилки на ранніх етапах та гарантувати стабільність базової логіки при подальшому розширенні проєкту.

Тестування виконано засобами фреймворку XCTest — стандартного інструменту для тестування у середовищі розробки Xcode. Тести оформлено у вигляді окремої цілі (test target) проєкту, що відокремлює тестовий код від основного коду застосунку. Кожен тест являє собою метод, у якому викликається перевіряюча функція з певними вхідними даними, після чого фактичний результат порівнюється з очікуваним за допомогою спеціальних макросів-тверджень (XCTAssertEqual, XCTAssertTrue, XCTAssertNil тощо).

Для тестування обрано компоненти, що реалізують основну логіку застосунку та не залежать від зовнішніх сервісів чи користувацького інтерфейсу. Такий підхід є типовим для модульного тестування, оскільки дозволяє перевіряти логіку детерміновано — результат тесту не залежить від стану мережі, відповіді зовнішнього API чи дій користувача.

4.2.1 Тестування модуля криптографічного хешування

Утиліта CryptoHelper відповідає за обчислення SHA-256-хешу паролів. Для неї перевірено дві ключові властивості. По-перше, детермінованість: однаковий вхідний рядок завжди повинен давати однаковий хеш — це критично для коректної автентифікації, оскільки при вході пароль хешується повторно й порівнюється зі збереженим значенням. По-друге, унікальність: різні вхідні рядки повинні давати різні хеші. Також перевірено, що довжина результату завжди становить 64 символи, що відповідає шістнадцятковому представленню 256-бітного хешу.

```
swift
func testSHA256IsDeterministic() {
    let hash1 = CryptoHelper.sha256("password123")
    let hash2 = CryptoHelper.sha256("password123")
    XCTAssertEqual(hash1, hash2)
}
func testSHA256DifferentInputs() {
    let hash1 = CryptoHelper.sha256("password123")
    let hash2 = CryptoHelper.sha256("password124")
    XCTAssertNotEqual(hash1, hash2)
}
func testSHA256Length() {
    let hash = CryptoHelper.sha256("test")
```

```

    XCTAssertEqual(hash.count, 64)
}

```

4.2.2 Тестування модуля роботи з датами

Утиліта `DateHelper` містить функції форматування дат та визначення меж тижня. Для неї перевірено коректність обчислення початку тижня — функція `startOfWeek` повинна повертати понеділок для будь-якої дати в межах одного тижня. Також перевірено, що методи форматування повертають непорожній рядок та коректно працюють з українською локаллю.

```

swift
func testStartOfWeekReturnsMonday() {
    let calendar = Calendar.current
    let date = calendar.date(from: DateComponents(year: 2026, month: 5, day:
14))!
    let weekStart = DateHelper.startOfWeek(for: date)
    let weekday = calendar.component(.weekday, from: weekStart)
    XCTAssertEqual(weekday, 2) // 2 – понеділок
}
func testFormattedDateNotEmpty() {
    let date = Date()
    let formatted = DateHelper.formattedDate(date)
    XCTAssertFalse(formatted.isEmpty)
}

```

4.2.3 Тестування логіки роботи з записами

Перевірено основну логіку класу `EntryViewModel`, зокрема метод `hasEntryToday`, що визначає наявність запису користувача за поточний день. Тестування виконано з використанням ізольованого тестового контейнера `SwiftData`, створеного в оперативній пам'яті. Це дозволяє тестувати роботу з базою даних без впливу на реальні дані застосунку. Перевірено два сценарії: за відсутності записів метод повинен повертати `false`, а після додавання запису на поточну дату — `true`.

```

swift
func testHasEntryTodayWhenEmpty() {
    let viewModel = EntryViewModel(modelContext: testContext)
    let result = viewModel.hasEntryToday(for: "test@example.com")
    XCTAssertFalse(result)
}
func testHasEntryTodayAfterSave() {
    let viewModel = EntryViewModel(modelContext: testContext)

```

```
viewModel.save(q1: "Добре", q2: "Немає", q3: "Відпочинок",
              q4: "", email: "test@example.com")
let result = viewModel.hasEntryToday(for: "test@example.com")
XCTAssertTrue(result)
}
```

4.2.4 Тестування розбору відповіді AI-аналізу

Окремо перевірено коректність декодування JSON-відповіді у структуру `AnalysisResult`. Тест перевіряє, що коректно сформований JSON-рядок з трьома полями успішно перетворюється на об'єкт, а всі поля заповнюються відповідними значеннями. Це гарантує, що відповідь від мовної моделі буде правильно опрацьована застосунком.

```
swift
func testAnalysisResultDecoding() {
    let json = """
    {"positives": "Гарний тиждень", "challenges": "Втома",
    "recommendations": "Більше відпочинку"}
    """

    let data = json.data(using: .utf8)!
    let result = try? JSONDecoder().decode(AnalysisResult.self, from: data)
    XCTAssertNotNil(result)
    XCTAssertEqual(result?.positives, "Гарний тиждень")
}
```

За результатами проведеного модульного тестування підтверджено коректність роботи ключових компонентів застосунку `Diary AI`: криптографічного хешування паролів, обробки дат, логіки роботи зі щоденними записами та розбору відповіді AI-аналізу. Усі сформовані тести виконуються успішно, що свідчить про надійність базової логіки застосунку. Застосування фреймворку `XCTest` та архітектури `MVVM`, яка відокремлює бізнес-логіку від інтерфейсу, забезпечує можливість подальшого розширення набору тестів у міру розвитку проєкту.

ВИСНОВКИ

У результаті виконання кваліфікаційної бакалаврської роботи розроблено нативний iOS-застосунок Diary AI, що поєднує методику структурованих ретроспективних запитань з автоматизованим аналізом записів засобами великої мовної моделі для підтримки практики щоденної саморефлексії. Виконано всі поставлені у вступі завдання, що дозволяє говорити про досягнення мети роботи.

У першому розділі проаналізовано предметну область цифрових щоденників та психологічні основи практики саморефлексії. Розглянуто сучасні підходи до автоматизованого аналізу персональних текстових даних на основі великих мовних моделей. Визначено цільову аудиторію застосунку та виокремлено дві ключові проблеми, що заважають регулярній рефлексивній практиці — проблема «порожньої сторінки» і проблема відсутності синтезу накопичених записів. Виконано порівняльний аналіз п'яти найпопулярніших застосунків-аналогів — Day One, Reflectly, Journey, Daylio та Stoic — за дев'ятьма функціональними критеріями. За результатами аналізу встановлено, що жоден з розглянутих застосунків не поєднує одночасно структуровані запитання для рефлексії, AI-аналіз тижневих записів з персональними рекомендаціями та орієнтацію на україномовну аудиторію — це обґрунтовує доцільність створення власного рішення. На основі проведеного аналізу сформовано перелік функціональних і нефункціональних вимог до застосунку.

У другому розділі обґрунтовано вибір технологічного стека для розробки. Цільовою платформою обрано iOS 17.0 і вище. Як основну мову програмування використано Swift, для побудови інтерфейсу — декларативний фреймворк SwiftUI, для локального зберігання даних — SwiftData, для AI-функціональності — OpenAI API з моделлю gpt-4o-mini. Архітектурною основою обрано патерн MVVM, що забезпечує чіткий поділ компонентів за відповідальністю та сумісний з реактивним підходом SwiftUI. Розгляд

кожного компонента стека включав не лише опис його можливостей, але й обґрунтування вибору саме цього інструменту з огляду на потреби проєкту. Проаналізовано також допоміжні фреймворки UserNotifications для роботи зі сповіщеннями та CryptoKit для криптографічного хешування паролів.

У третьому розділі описано практичну реалізацію застосунку. Спроектовано загальну архітектуру з чотирма рівнями: моделями даних (Models), посередницькою логікою (ViewModels), користувацьким інтерфейсом (Views) та сервісним шаром (Services). Реалізовано три SwiftData-моделі для зберігання профілів користувачів, щоденних записів та результатів AI-аналізів. Розроблено три ViewModel-класи для обробки автентифікації, роботи з записами та виконання AI-аналізу — усі позначені атрибутом @MainActor для гарантованої роботи у головному потоці. Реалізовано десять екранів інтерфейсу, що покривають усі функціональні вимоги: екрани автентифікації, головний екран із картковою навігацією, покрокову форму створення запису, історію та деталі записів, екран AI-аналізу з вибором діапазону дат, архів виконаних аналізів і екран налаштувань.

Окрему увагу приділено сервісному шару — реалізовано клас OpenAIService, що інкапсулює весь процес взаємодії з API, включаючи формування промпту з україномовною локалізацією дат і обробку структурованої JSON-відповіді, а також клас NotificationService для роботи зі щоденними нагадуваннями. Розглянуто питання безпеки збереження облікових даних: паролі зберігаються у вигляді SHA-256-хешів, передача до зовнішнього API виконується по HTTPS, персональні записи розташовуються виключно локально на пристрої користувача.

У четвертому розділі продемонстровано роботу застосунку на повному сценарії використання — від реєстрації нового користувача до отримання тижневого AI-аналізу. Розглянуто покрокове виконання основних функцій: реєстрацію, створення щоденних записів, перегляд історії, налаштування API-ключа OpenAI та системи сповіщень, а також виконання AI-аналізу за вибраний період. Окрему увагу приділено перевірці коректності роботи

застосунку. Виконано модульне тестування ключових компонентів за допомогою фреймворку XCTest. Перевірено роботу криптографічного хешування паролів, утиліт обробки дат, логіки роботи зі щоденними записами та механізму розбору JSON-відповіді AI-аналізу. Усі сформовані тести виконуються успішно, що підтверджує надійність базової логіки застосунку. Використання архітектури MVVM, яка відокремлює бізнес-логіку від користувацького інтерфейсу, забезпечує можливість подальшого розширення набору тестів у міру розвитку проєкту.

Розроблений застосунок Diary AI успішно поєднує методичну основу (структуровані запитання, адаптовані з практики Scrum-ретроспектив) із сучасними технологіями штучного інтелекту. Така комбінація закриває обидві ключові проблеми ринку — знижує бар'єр входу для регулярної рефлексивної практики через структурованість запитань і автоматизує синтез накопичених записів через AI-аналіз. Орієнтація на україномовну аудиторію формує окрему ринкову нішу, не зайняту існуючими рішеннями.

Наукова новизна роботи полягає у запропонованому підході до поєднання методики Scrum-ретроспектив з автоматизованим аналізом записів великою мовною моделлю в межах одного мобільного застосунку. На відміну від існуючих рішень, що пропонують або лише структуру, або лише AI-аналіз, у Diary AI ці два підходи інтегровано в єдиний користувацький сценарій. Додатково запропонована архітектура взаємодії з мовною моделлю — структурований промпт, що гарантує відповідь у форматі JSON з трьома фіксованими блоками — дозволяє надійно обробляти результат на стороні застосунку.

Практична значущість роботи полягає у створенні готового до використання нативного iOS-застосунку, який може застосовуватися широким колом україномовних користувачів для регулярної рефлексивної практики. Реалізована архітектура локального зберігання даних, інтеграції з зовнішнім AI-сервісом та обробки помилок може бути використана як основа для подальших розширень застосунку — додавання синхронізації між пристроями

через iCloud, інтеграції з Apple HealthKit для зіставлення записів із фізіологічними показниками, реалізації біометричної автентифікації, додавання шифрування контенту записів окремим ключем користувача, розширення набору запитань через персональні шаблони. Реалізовану модель також можна адаптувати до інших платформ — Android, web — зі збереженням ключових архітектурних рішень.

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

1. Чикало В.Д., Залива В.В. Забезпечення безпеки інформації в iOS-застосунку для щоденної саморефлексії. Матеріали VII Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 23 квітня 2026 року, Київ, ДУІКТ. Збірник тез.— К.: ДУІКТ, 2026, С. 291-293.

2. Чикало В.Д., Залива В.В. Практичні питання вбудовування великих мовних моделей у мобільні iOS-застосунки. Матеріали VI Всеукраїнської науково-технічної конференції «Сучасні інтелектуальні інформаційні технології в науці та освіті», 15 травня 2026 року, Київ, ДУІКТ. Збірник тез.(подано до друку).

ПЕРЕЛІК ПОСИЛАНЬ

1. Ullrich P. M., Lutgendorf S. K. Journaling about stressful events: Effects of cognitive processing and emotional expression. *Annals of Behavioral Medicine*. 2002. Vol. 24, No. 3. P. 244–250. URL: https://doi.org/10.1207/S15324796ABM2403_10
2. Kross E., Ayduk O. Self-distancing: Theory, research, and current directions. *Advances in Experimental Social Psychology*. 2017. Vol. 55. P. 81–136. URL: <https://doi.org/10.1016/bs.aesp.2016.10.002>
3. Pennebaker J. W., Smyth J. M. *Opening Up by Writing It Down: How Expressive Writing Improves Health and Eases Emotional Pain*. 3rd ed. New York: Guilford Press, 2016. 240 p.
4. Smyth J. M., Johnson J. A., Auer B. J., Lehman E., Talamo G., Sciamanna C. N. Online positive affect journaling in the improvement of mental distress and well-being in general medical patients with elevated anxiety symptoms: A preliminary randomized controlled trial. *JMIR Mental Health*. 2018. Vol. 5, No. 4. e11290. URL: <https://doi.org/10.2196/11290>
5. Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A. N., Kaiser Ł., Polosukhin I. Attention is all you need. *Advances in Neural Information Processing Systems*. 2017. Vol. 30. P. 5998–6008. URL: <https://doi.org/10.48550/arXiv.1706.03762>
6. Brown T., Mann B., Ryder N., Subbiah M., Kaplan J. D., Dhariwal P. et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*. 2020. Vol. 33. P. 1877–1901. URL: <https://doi.org/10.48550/arXiv.2005.14165>
7. OpenAI. GPT-4 Technical Report. arXiv preprint. 2023. URL: <https://doi.org/10.48550/arXiv.2303.08774>

8. Hodson N., Williamson S. Can large language models replace therapists? Evaluating performance at simple cognitive behavioral therapy tasks. *JMIR AI*. 2024. Vol. 3. e52500. URL: <https://doi.org/10.2196/52500>
9. Stade E. C., Stirman S. W., Ungar L. H., Boland C. L., Schwartz H. A., Yaden D. B. et al. Large language models could change the future of behavioral healthcare: A proposal for responsible development and evaluation. *npj Mental Health Research*. 2024. Vol. 3, No. 12. URL: <https://doi.org/10.1038/s44184-024-00056-z>
10. Linardon J., Cuijpers P., Carlbring P., Messer M., Fuller-Tyszkiewicz M. The efficacy of app-supported smartphone interventions for mental health problems: A meta-analysis of randomized controlled trials. *World Psychiatry*. 2019. Vol. 18, No. 3. P. 325–336. URL: <https://doi.org/10.1002/wps.20673>
11. Derby E., Larsen D. *Agile Retrospectives: Making Good Teams Great*. Raleigh: Pragmatic Bookshelf, 2006. 200 p.
12. Day One — Your Journal for Life. URL: <https://dayoneapp.com/> (дата звернення: 21.04.2026).
13. Reflectly — AI Journal & Diary. URL: <https://reflectly.app/> (дата звернення: 21.04.2026).
14. Journey — Diary, Journal. URL: <https://journey.cloud/> (дата звернення: 21.04.2026).
15. Daylio — Journal, Diary, Mood. URL: <https://daylio.net/> (дата звернення: 21.04.2026).
16. Stoic — Mental Health Journal. URL: <https://www.getstoic.com/> (дата звернення: 22.04.2026).
17. National Institute of Standards and Technology. Secure Hash Standard (SHS). FIPS PUB 180-4. Gaithersburg: NIST, 2015. 36 p. URL: <https://doi.org/10.6028/NIST.FIPS.180-4>

18. Apple Inc. The Swift Programming Language. Swift 5.9. URL: <https://docs.swift.org/swift-book/documentation/the-swift-programming-language/> (дата звернення: 22.04.2026).
19. Apple Inc. SwiftUI Documentation. URL: <https://developer.apple.com/documentation/swiftui> (дата звернення: 22.04.2026).
20. Hoyt M. Pro iPhone Development with SwiftUI: Design and Manage Top-Quality Apps. 4th ed. New York: Apress, 2022. 540 p.
21. Neuburg M. SwiftUI for Masterminds: How to Take Advantage of SwiftUI to Create Insanely Great Apps. 4th ed. Boston: Addison-Wesley Professional, 2023. 712 p.
22. Apple Inc. SwiftData Documentation. URL: <https://developer.apple.com/documentation/swiftdata> (дата звернення: 22.04.2026).
23. OpenAI. API Reference. URL: <https://platform.openai.com/docs/api-reference> (дата звернення: 22.04.2026).

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ



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



Розробка iOS-застосунку для щоденної саморефлексії та отримання персональних рекомендацій на основі штучного інтелекту

Виконав студент 4 курсу
групи ПД-43
Чикало Всеволод Денисович
Керівник роботи
доцент кафедри, доктор філософії
Залива Віталій Вікторович

Київ – 2026

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: покращення процесу щоденної саморефлексії користувача шляхом структурованого ведення особистого щоденника та автоматизованого аналізу записів з використанням штучного інтелекту для формування персоналізованих рекомендацій щодо покращення якості життя

Об'єкт дослідження: процес щоденної саморефлексії та самоаналізу особистості.

Предмет дослідження: засоби, технології та підходи щодо розробки iOS-застосунку для ведення щоденних записів та їх інтелектуального аналізу для виявлення позитивних і проблемних аспектів життя та формування персональних рекомендацій.

АНАЛІЗ АНАЛОГІВ

Критерій порівняння	Day One	Reflectly	Journey	Daylio	Stoic	Diary AI
Структуровані запитання для рефлексії	–	–	–	–	+	+
AI-аналіз записів	–	+	–	–	–	+
Тижневі підсумки з рекомендаціями	–	–	–	–	–	+
Вбудований україномовний інтерфейс	–	–	–	–	–	+
Локальне зберігання даних	–	–	+	+	–	+

4

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Проаналізувати предметну область та підходи до AI-аналізу текстових даних.
2. Виконати огляд застосунків-аналогів та виявити їхні функціональні прогалини.
3. Сформулювати функціональні та нефункціональні вимоги до програмного забезпечення.
4. Обґрунтувати вибір технологічного стеку.
5. Спроектувати архітектуру застосунку.
6. Реалізувати та протестувати iOS-застосунок.

3

ВИМОГИ ДО ДОДАТКУ

Функціональні вимоги:

1. Реєстрація та автентифікація користувача за email і паролем.
2. Щоденне ведення записів за структурованими запитаннями.
3. Перегляд історії записів та аналізів.
4. Генерація AI-аналізу тижневих записів з рекомендаціями (OpenAI API).
5. Налаштування часу push-сповіщень для нагадування про запис.

Нефункціональні вимоги:

1. Безпека: шифрування даних, захищене зберігання облікових даних (Keychain).
2. Усі записи користувача зберігаються локально на пристрої засобами SwiftData; створення та перегляд записів доступні без підключення до інтернету.
3. Особисті записи користувача не передаються на сторонні сервери, окрім моменту надсилання тексту записів до OpenAI API для генерації тижневого аналізу за явною ініціативою користувача.

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ



XCode



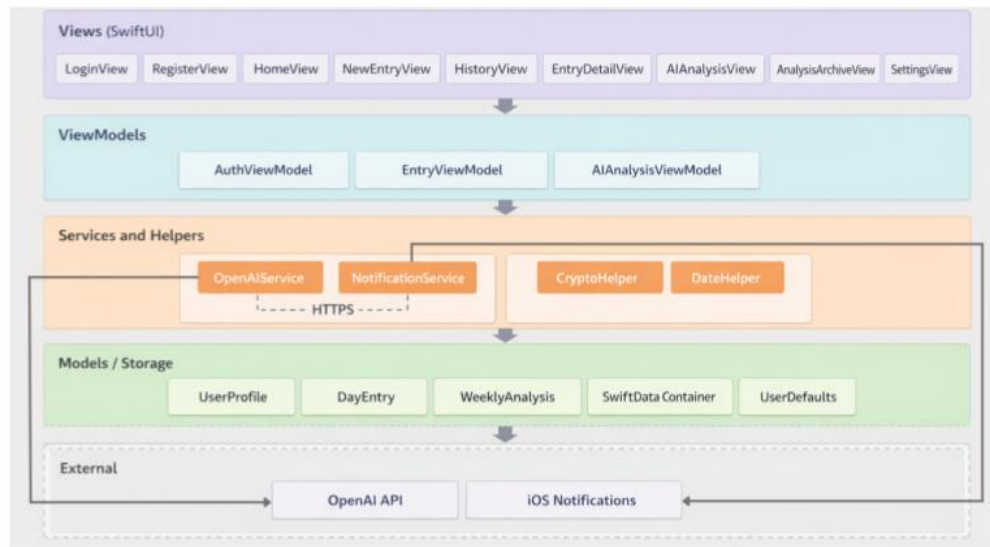
Swift



SwiftUI

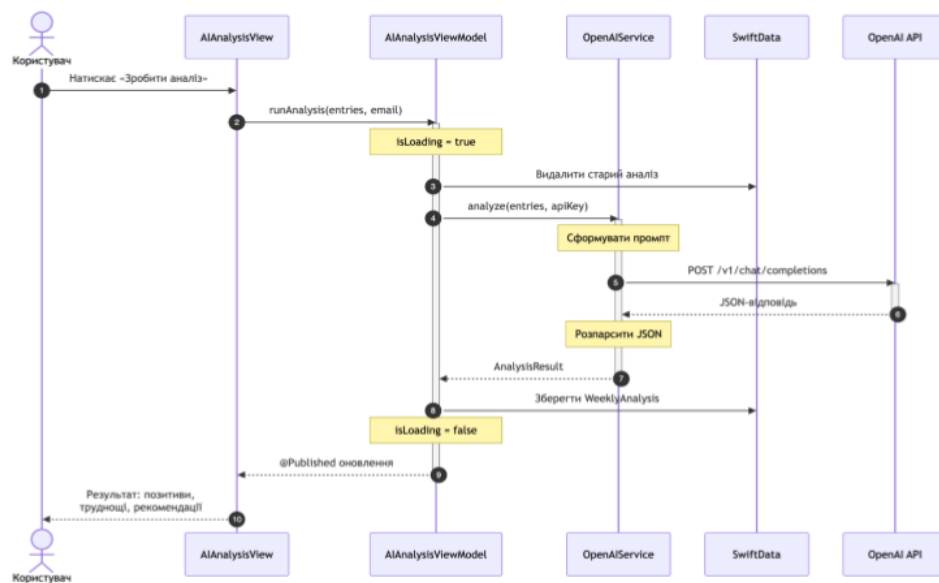
6

АРХІТЕКТУРА ЗАСТОСУНКУ DIARY AI



7

ДІАГРАМА ПОСЛІДОВНОСТІ ВИКОНАННЯ АІ-АНАЛІЗУ



8

ЕКРАННІ ФОРМИ

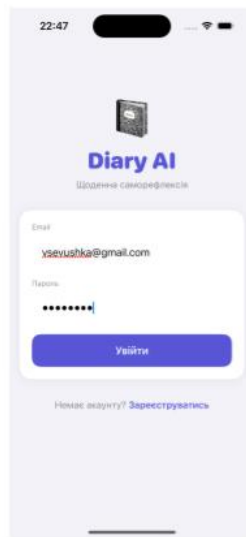


рис.1. - Вхід у додаток

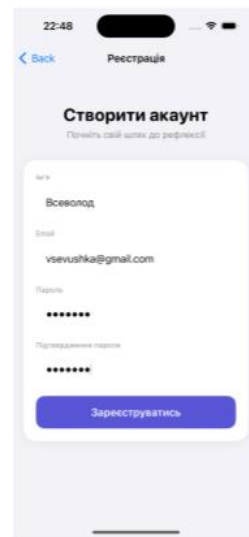


рис.2. - Реєстрація у додатку

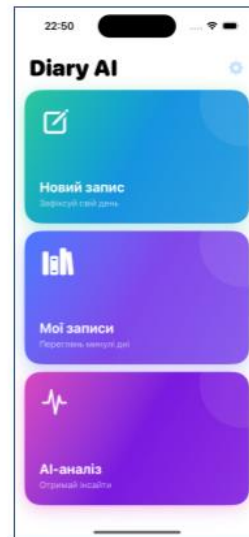


рис.3. - Головне меню

9

ЕКРАННІ ФОРМИ

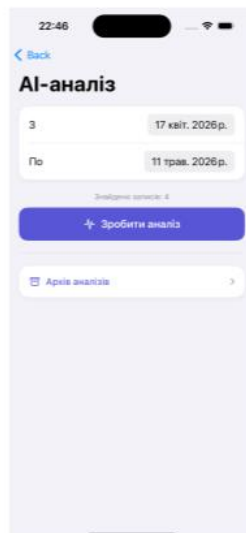


рис.4. - Меню AI аналізу



рис.5. - Готовий аналіз

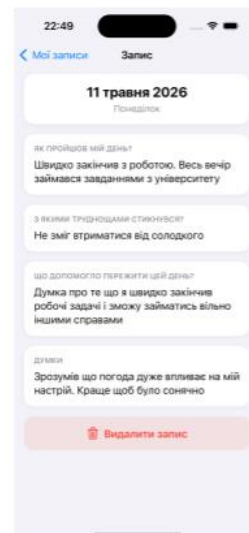


рис.6. - Архів щоденних записів

10

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

Тези доповідей:

1. Чикало В.Д., Залива В.В. Забезпечення безпеки інформації в iOS-застосунку для щоденної саморефлексії. Матеріали VII Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 23 квітня 2026 року, Київ, ДУІКТ. Збірник тез.— К.: ДУІКТ, 2026, С. 291-293.
2. Чикало В.Д., Залива В.В. Практичні питання вбудовування великих мовних моделей у мобільні iOS-застосунки. Матеріали VI Всеукраїнської науково-технічної конференції «Сучасні інтелектуальні інформаційні технології в науці та освіті», 15 травня 2026 року, Київ, ДУІКТ. Збірник тез.(подано до друку).

11

ВИСНОВКИ

1. Проаналізовано предметну область та підходи до AI-аналізу текстових даних, на основі чого визначено ключові проблеми, тренди та недоліки в галузі ведення цифрових щоденників.
2. Виконано огляд 5 сучасних застосунків-аналогів, в результаті якого виявлено їхні функціональні прогалини, що дозволило встановити напрямки вдосконалення власного програмного забезпечення.
3. На основі аналізу предметної області та виявлених недоліків існуючих рішень сформовано функціональні та нефункціональні вимоги до розроблюваного застосунку.
4. Проаналізовано існуючі інструменти та обґрунтовано вибір технологічного стека (Swift, SwiftUI, SwiftData, OpenAI API, паттерн MVVM), який є оптимальним для вирішення поставлених завдань.
5. Відповідно до визначених вимог спроєктовано чотиришарову архітектуру застосунку, що забезпечує гнучкість, масштабованість та безпеку збереження даних.
6. Реалізовано всі ключові модулі iOS-застосунку «Diary AI». За результатами проведеного тестування визначено, що застосунок працює коректно, стабільно та повністю відповідає всім поставленим вимогам.

12

ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ

Б.1 Модель UserProfile (UserProfile.swift)

```
import Foundation
import SwiftData

@Model
class UserProfile {
    var id: UUID
    var name: String
    var email: String
    var passwordHash: String
    var createdAt: Date

    init(name: String, email: String,
passwordHash: String) {
        self.id = UUID()
        self.name = name
        self.email = email
        self.passwordHash = passwordHash
        self.createdAt = Date()
    }
}
```

Б.2 Модель DayEntry (DayEntry.swift)

```
import Foundation
import SwiftData

@Model
class DayEntry {
    var id: UUID
    var date: Date
    var userEmail: String
    var question1: String
    var question2: String
    var question3: String
    var question4: String
    var createdAt: Date

    init(date: Date,
        userEmail: String,
        question1: String,
        question2: String,
        question3: String,
        question4: String = "") {
        self.id = UUID()
        self.date = date
        self.userEmail = userEmail
        self.question1 = question1
        self.question2 = question2
        self.question3 = question3
        self.question4 = question4
        self.createdAt = Date()
    }
}
```

Б.3 Модель WeeklyAnalysis (WeeklyAnalysis.swift)

```
import Foundation
import SwiftData

@Model
class WeeklyAnalysis {
    var id: UUID
```

```
    var userEmail: String
    var weekStartDate: Date
    var weekEndDate: Date
    var positives: String
    var challenges: String
    var recommendations: String
    var createdAt: Date
```

```
    init(userEmail: String,
        weekStartDate: Date,
        weekEndDate: Date,
        positives: String,
        challenges: String,
        recommendations: String) {
        self.id = UUID()
        self.userEmail = userEmail
        self.weekStartDate = weekStartDate
        self.weekEndDate = weekEndDate
        self.positives = positives
        self.challenges = challenges
        self.recommendations =
recommendations
        self.createdAt = Date()
    }
}
```

Б.4 ViewModel автентифікації (AuthViewModel.swift)

```
import SwiftUI
import SwiftData

@MainActor
final class AuthViewModel:
ObservableObject {
    var modelContext: ModelContext

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

    func login(email: String, password:
String) -> Bool {
        let norm =
email.lowercased().trimmingCharacters(in:
.whitespaces)
        let hash =
CryptoHelper.sha256(password)
        let desc =
FetchDescriptor<UserProfile>(
            predicate: #Predicate {
                $0.email == norm
            }
        )
        guard
            let list = try?
modelContext.fetch(desc),
            let profile = list.first,
            profile.passwordHash == hash
        else { return false }
        saveSession(email: profile.email,
name: profile.name)
        return true
    }

    func register(name: String, email:
String, password: String) -> Bool {
```

```

        let norm =
email.lowercased().trimmingCharacters(in:
.whitespaces)
        let desc =
FetchDescriptor<UserProfile>(
        predicate: #Predicate {
$0.email == norm }
    )
        if let existing = try?
modelContext.fetch(desc),
!existing.isEmpty {
            return false
        }

        let profile = UserProfile(
            name: name,
            email: norm,
            passwordHash:
CryptoHelper.sha256(password)
        )
        modelContext.insert(profile)
        try? modelContext.save()
        saveSession(email: norm, name:
name)
        return true
    }

    func logout() {
        UserDefaults.standard.set(false,
forKey: "isLoggedIn")

        UserDefaults.standard.removeObject(forKey:
"currentUserEmail")

        UserDefaults.standard.removeObject(forKey:
"currentUserName")
    }

    private func saveSession(email:
String, name: String) {
        UserDefaults.standard.set(true,
forKey: "isLoggedIn")
        UserDefaults.standard.set(email,
forKey: "currentUserEmail")
        UserDefaults.standard.set(name,
forKey: "currentUserName")
    }
}

```

Б.5 ViewModel роботи з записами (EntryViewModel.swift)

```

import SwiftUI
import SwiftData

@MainActor
final class EntryViewModel:
ObservableObject {
    var modelContext: ModelContext

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

    func hasEntryToday(for email: String)
-> Bool {
        let today =
Calendar.current.startOfDay(for: Date())
        let tomorrow =
Calendar.current.date(byAdding: .day,
value: 1, to: today)!
        let desc =
FetchDescriptor<DayEntry>(

```

```

        predicate: #Predicate {
            $0.userEmail == email &&
            $0.date >= today &&
            $0.date < tomorrow
        }
    )
    return !((try?
modelContext.fetch(desc)) ?? []).isEmpty
}

    func save(q1: String, q2: String, q3:
String, q4: String, email: String) {
        let entry = DayEntry(
            date:
Calendar.current.startOfDay(for: Date()),
            userEmail: email,
            question1: q1,
            question2: q2,
            question3: q3,
            question4: q4
        )
        modelContext.insert(entry)
        try? modelContext.save()
    }
}

```

Б.6 ViewModel AI-аналізу (AIAnalysisViewModel.swift)

```

import SwiftUI
import SwiftData

@MainActor
final class AIAnalysisViewModel:
ObservableObject {
    @Published var startDate: Date =
Calendar.current.date(byAdding: .day,
value: -6, to: Date())!
    @Published var endDate: Date = Date()
    @Published var isLoading: Bool = false
    @Published var errorMessage: String?

    var modelContext: ModelContext?

    func entries(for email: String) ->
[DayEntry] {
        guard let ctx = modelContext else
{ return [] }
        let start =
Calendar.current.startOfDay(for:
startDate)
        let end =
Calendar.current.date(byAdding: .day,
value: 1,
to: Calendar.current.startOfDay(for:
endDate))!
        let desc =
FetchDescriptor<DayEntry>(
            predicate: #Predicate {
                $0.userEmail == email &&
                $0.date >= start &&
                $0.date < end
            },
            sortBy:
[SortDescriptor(\.date)]
        )
        return (try? ctx.fetch(desc)) ??
[]
    }

    func existingAnalysis(for email:
String) -> WeeklyAnalysis? {

```

```

        guard let ctx = modelContext else
        { return nil }
        let start =
Calendar.current.startOfDay(for:
startDate)
        let end =
Calendar.current.startOfDay(for: endDate)
        let desc =
FetchDescriptor<WeeklyAnalysis>(
    predicate: #Predicate {
        $0.userEmail == email &&
        $0.weekStartDate == start
&&
        $0.weekEndDate == end
    }
)
        return (try?
ctx.fetch(desc))?.first
    }

    func runAnalysis(entries: [DayEntry],
email: String) async {
        guard let ctx = modelContext else
        { return }
        let key =
UserDefaults.standard.string(forKey:
"openai_api_key") ?? ""
        isLoading = true
        errorMessage = nil
        defer { isLoading = false }

        do {
            let start =
Calendar.current.startOfDay(for:
startDate)
            let end =
Calendar.current.startOfDay(for: endDate)
            let existing =
FetchDescriptor<WeeklyAnalysis>(
                predicate: #Predicate {
                    $0.userEmail == email
&&
                    $0.weekStartDate ==
start &&
                    $0.weekEndDate == end
                }
            )
            if let old = try?
ctx.fetch(existing) {
                old.forEach {
ctx.delete($0) }
            }

            let result = try await
OpenAIService.analyze(entries: entries,
apiKey: key)
            let analysis = WeeklyAnalysis(
                userEmail: email,
                weekStartDate: start,
                weekEndDate: end,
                positives:
result.positives,
                challenges:
result.challenges,
                recommendations:
result.recommendations
            )
            ctx.insert(analysis)
            try? ctx.save()
        } catch {
            errorMessage =
error.localizedDescription

```

```

    }
}
}

```

B.7 Сервіс OpenAI (OpenAIService.swift)

```

import Foundation

struct AnalysisResult: Decodable {
    let positives: String
    let challenges: String
    let recommendations: String
}

enum OpenAIError: LocalizedError {
    case noAPIKey, invalidResponse
    case httpError(Int)
    case apiMessage(String)
    case parsingFailed(String)

    var errorDescription: String? {
        switch self {
            case .noAPIKey:
                return "API ключ не введено.
Перейдіть у Налаштування та збережіть ваш
OpenAI API ключ."
            case .invalidResponse:
                return "Невалідна відповідь
від сервера."
            case .httpError(let c):
                return "HTTP помилка: \(c)"
            case .apiMessage(let m):
                return "Помилка OpenAI: \(m)"
            case .parsingFailed(let m):
                return "Помилка парсингу:
\(m)"
        }
    }
}

enum OpenAIService {

    static func analyze(entries:
[DayEntry], apiKey: String) async throws -
> AnalysisResult {
        guard
!apiKey.trimmingCharacters(in:
.whitespaces).isEmpty else {
            throw OpenAIError.noAPIKey
        }

        var req = URLRequest(url:
URL(string:
"https://api.openai.com/v1/chat/completion
s")!)
        req.httpMethod = "POST"
        req.setValue("Bearer \(apiKey)",
forHTTPHeaderField: "Authorization")
        req.setValue("application/json",
forHTTPHeaderField: "Content-Type")

        let body: [String: Any] = [
            "model": "gpt-4o-mini",
            "temperature": 0.7,
            "max_tokens": 1000,
            "messages": [["role": "user",
"content": prompt(entries)]]
        ]
        req.httpBody = try
JSONSerialization.data(withJSONObject:
body)

```

```

        let (data, response) = try await
URLSession.shared.data(for: req)
        guard let http = response as?
HTTPURLResponse else {
            throw
OpenAIError.invalidResponse
        }

        if http.statusCode != 200 {
            let msg = (try?
JSONSerialization.jsonObject(with: data)
as? [String: Any])
                .flatMap { $0["error"] as?
[String: Any] }
                .flatMap { $0["message"]
as? String }
            throw msg.map {
OpenAIError.apiMessage($0) } ??
OpenAIError.httpError(http.statusCode)
        }

        guard
            let json = (try?
JSONSerialization.jsonObject(with: data)
as? [String: Any],
                let choices = json["choices"]
as? [[String: Any]],
                let msg =
choices.first?["message"] as? [String:
Any],
                let content = msg["content"]
as? String
            else { throw
OpenAIError.invalidResponse }

            let cleaned = content
                .replacingOccurrences(of:
"```json", with: "")
                .replacingOccurrences(of:
"```", with: "")
                .trimmingCharacters(in:
.whitespacesAndNewlines)

            guard let jsonData =
cleaned.data(using: .utf8) else {
                throw
OpenAIError.parsingFailed("Не вдалося
конвертувати відповідь")
            }

            do {
                return try
JSONDecoder().decode(AnalysisResult.self,
from: jsonData)
            } catch {
                throw
OpenAIError.parsingFailed(error.localizedDescription)
            }
        }

        private static func prompt(_ entries:
[DayEntry]) -> String {
            let f = DateFormatter()
            f.locale = Locale(identifier:
"uk-UA")
            f.dateFormat = "EEEE, d MMMM"

            var t = "Ти – персональний
асистент для саморефлексії. Ось записи
користувача за тиждень:\n\n"
            for e in entries {

```

```

                t += "📅 \ \(f.string(from:
e.date).capitalized)\n"
                t += "– Як пройшов день:
\ \(e.question1)\n"
                t += "– Труднощі:
\ \(e.question2)\n"
                t += "– Що допомогло:
\ \(e.question3)\n"
                if !e.question4.isEmpty { t +=
"– Думки: \ \(e.question4)\n" }
                t += "\n"
            }
            t += ""
            Проаналізуй ці записи та надай
відповідь СТРОГО у форматі JSON \
(без markdown, без ```json, тільки
чистий JSON об'єкт):

{"positives":"...", "challenges":"...", "rec
ommendations":"..."}

Відповідай українською мовою. Будь
конкретним і персональним.
            ""
            return t
        }
    }
}

```

Б.8 Хелпер дат (DateHelper.swift)

```

import Foundation

enum DateHelper {

    static func startOfWeek(for date:
Date) -> Date {
        var cal = Calendar.current
        cal.firstWeekday = 2
        let c =
cal.dateComponents([.yearForWeekOfYear,
.weekOfYear], from: date)
        return cal.date(from: c) ?? date
    }

    static func endOfWeek(for date: Date)
-> Date {
        let start = startOfWeek(for: date)
        return
Calendar.current.date(byAdding: .day,
value: 6, to: start) ?? date
    }

    static func formattedDate(_ date:
Date) -> String {
        let f = DateFormatter()
        f.locale = Locale(identifier:
"uk-UA")
        f.dateFormat = "d MMMM yyyy"
        return f.string(from: date)
    }

    static func dayOfWeek(_ date: Date) ->
String {
        let f = DateFormatter()
        f.locale = Locale(identifier:
"uk-UA")
        f.dateFormat = "EEEE"
        return f.string(from:
date).capitalized
    }
}

```

Б.9 Головний екран (HomeView.swift) — фрагмент

```

import SwiftUI
import SwiftData

struct HomeView: View {
    @AppStorage("currentUserEmail")
    private var userEmail: String = ""
    @Query(sort: \DayEntry.date, order:
.reverse) private var allEntries:
[DayEntry]
    @State private var showSettings =
false

    private var hasEntryToday: Bool {
        let today =
Calendar.current.startOfDay(for: Date())
        let tomorrow =
Calendar.current.date(byAdding: .day,
value: 1, to: today)!
        return allEntries.contains {
            $0.userEmail == userEmail &&
            $0.date >= today && $0.date < tomorrow
        }
    }

    var body: some View {
        NavigationStack {
            VStack(spacing: 0) {
                HStack(alignment: .center)
            {
                Text("Diary AI")

                .font(.system(size: 36, weight: .black,
design: .rounded))
                Spacer()
                Button { showSettings
= true } label: {
                    Image(systemName:
"gearshape.fill")
                }
                .font(.system(size: 22, weight: .medium))
                .foregroundColor(.primary.opacity(0.22))
            }
            .padding(.horizontal, 24)
            .padding(.top, 16)
            .padding(.bottom, 14)

            VStack(spacing: 12) {
                if hasEntryToday {
                    FullCard(
                        icon:
"checkmark.circle.fill",
                        title: "Запис
сьогодні зроблено",
                        subtitle:
"Повернись завтра",
                        colors:
[Color(.systemGray4),
Color(.systemGray3)],
                        disabled: true
                    )
                } else {
                    NavigationLink(destination:
NewEntryView()) {
                        FullCard(
                            icon:
"square.and.pencil",
                            title:
"Новий запис",

```

```

                        subtitle:
"Зафіксуй свій день",
                        colors: [
Color(red: 0.18, green: 0.78, blue: 0.60),
Color(red: 0.09, green: 0.58, blue: 0.88)
                        ]
                    )
                }
                .buttonStyle(.plain)
            }
            // ... аналогічні
            NavigationLink для HistoryView та
            AIAnalysisView
        }
        .padding(.horizontal, 16)
        .padding(.bottom, 20)
        .frame(maxHeight:
.infinity)
    }
    .navigationBarHidden(true)
    .sheet(isPresented:
$showSettings) { SettingsView() }
}

```

Б.10 Екран створення запису (NewEntryView.swift) — фрагмент

```

import SwiftUI
import SwiftData

struct NewEntryView: View {
    @Environment(\.modelContext) private
var modelContext
    @Environment(\.dismiss) private var
dismiss
    @AppStorage("currentUserEmail")
    private var userEmail: String = ""
    @Query(sort: \DayEntry.date, order:
.reverse) private var allEntries:
[DayEntry]

    @State private var q1 = ""
    @State private var q2 = ""
    @State private var q3 = ""
    @State private var q4 = ""
    @State private var currentStep = 1
    @State private var showSaved = false
    @State private var showExists = false

    private let accent = Color(red: 0.345,
green: 0.337, blue: 0.839)

    private var canGoNext: Bool {
        switch currentStep {
            case 1: return
!q1.trimmingCharacters(in:
.whitespaces).isEmpty
            case 2: return
!q2.trimmingCharacters(in:
.whitespaces).isEmpty
            case 3: return
!q3.trimmingCharacters(in:
.whitespaces).isEmpty
            case 4: return true
            default: return false
        }
    }
}

```

```

        var body: some View {
            ZStack {

                Color(.systemGroupedBackground).ignoresSafeArea()

                VStack(spacing: 20) {
                    ProgressView(value:
                        Double(currentStep), total: 4.0)
                        .tint(accent)
                        .padding(.horizontal)
                        .padding(.top, 16)

                    TabView(selection:
                        $currentStep) {
                        qPage(n: 1, question:
                            "Як пройшов мій день?",
                            binding: $q1,
                            required: true).tag(1)
                        qPage(n: 2, question:
                            "З якими труднощами ти сьогодні
                            стикнувся?",
                            binding: $q2,
                            required: true).tag(2)
                        qPage(n: 3, question:
                            "Що допомогло тобі пережити цей день?",
                            binding: $q3,
                            required: true).tag(3)
                        qPage(n: 4, question:
                            "Твої думки (за бажанням)",
                            binding: $q4,
                            required: false).tag(4)
                    }

                    .tabViewStyle(.page(indexDisplayMode:
                        .never))
                        .scrollDisabled(true)

                    // ... нижні кнопки
                    навігації Назад/Далі/Зберегти
                }

                .navigationTitle("Запис за
                    \(DateHelper.formattedDate(Date()))")

                .navigationBarTitleDisplayMode(.inline)
            }

            @MainActor private func saveEntry() {
                EntryViewModel(modelContext:
                    modelContext)
                    .save(q1: q1, q2: q2, q3: q3,
                        q4: q4, email: userEmail)
                showSaved = true
            }
        }
    }

```