Форма № 24

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій
(повне найменування факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

магістр
(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ РОЗУМНОГО ПЛАНУВАЛЬНИКА ДЛЯ IOS

RESEARCH AND SOFTWARE IMPLEMENTATION OF SMART SCHEDULER FOR IOS

Виконав(ла): студент(ка) 2 курсу, групи КНТ-214м

Спеціальності 122 Комп'ютерні науки
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Системи штучного інтелекту

ГРИЩЕНКОВ М.С.
(ПРІЗВИЩЕ та ініціали)

Керівник ФЕДОРОНЧАК Т.В.
(ПРІЗВИЩЕ та ініціали)

Рецензент БАБЕНКО Н.В.
(ПРІЗВИЩЕ та ініціали)

2025

Форма № 25

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 122 Комп'ютерні науки
(код і найменування)
Освітня програма (спеціалізація) Системи штучного інтелекту
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
_______________Сергій СУББОТІН
“____” _______________2025 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ГРИЩЕНКОВА Максима Сергійовича
(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація розумного планувальника для iOS. Research and Software Implementation of Smart Scheduler for iOS
керівник проєкту (роботи) к.т.н., доцент, ФЕДОРОНЧАК Тетяна Василівна ,
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)
затверджені наказом закладу вищої освіти від “ 30 ” вересня 2025 року № 447
2. Строк подання студентом проєкту (роботи) 01 грудня 2025 року
3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження. 2. Матеріали і методи. 3. Розробка архітектури програми. 4. Основні рішення щодо реалізації компонентів системи. 5. Експлуатація, тестування та експериментальне дослідження програми.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	ФЕДОРОНЧАК Т.В., доцент		
Нормоконтроль	БЄЛОВА А.В., асистент		

7. Дата видачі завдання « 30 » вересня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2-3 тижні	Розділ 1
3	Розробка та удосконалення методів, моделей й	4-5 тижні	Розділ 2
	алгоритмів вирішення задачі.		
4	Вибір мови програмування та інших технологій	6 тиждень	Розділ 3
	розробки.		
5	Розробка архітектури програми.	6 тиждень	Розділ 3
6	Розробка програми.	7-8 тижні	Розділ 4
7	Тестування та експериментальне дослідження	9 тиждень	Розділ 5
	програмного забезпечення.		
8	Оформлення пояснювальної записки та документів	10-11 тижні	Додатки
	до неї.		
9	Нормоконтроль та рецензування.	12 тиждень	
10	Захист роботи.	12 тиждень	

Студент(ка) __________ Максим ГРИЩЕНКОВ
(підпис) (Ім’я ПРІЗВИЩЕ)

Керівник проєкту (роботи) __________ Тетяна ФЕДОРОНЧАК
(підпис) (Ім’я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра: 117 с., 14 табл., 45 рис., 3 дод., 22 джерел.

МОБІЛЬНИЙ ЗАСТОСУНОК, IOS, ПЛАНУВАЛЬНИК, НАГАДУВАННЯ, SWIFTUI, SWIFT, COREML, REMINDERS.

Об'єкт дослідження – алгоритми для реалізації розумних планувальників задач.

Предмет дослідження – алгоритми фільтрації та штучного інтелекту для створення та керування нагадуваннями в мобільному телефоні.

Мета роботи – реалізація мобільного застосунку для створення та керування нагадуваннями за допомогою алгоритмів фільтрації та штучного інтелекту для платформи iOS.

Матеріали, методи та технічні засоби: мови програмування Swift, фреймворки SwiftUI та UIKit, бібліотеки SwiftData, CoreML та EventKit, середовище розробки Xcode, персональний комп'ютер з процесором M1 Pro під управлінням операційної системи macOS Tahoe, мобільний пристрій з операційною системою iOS 26.

Результати. Створено мобільний застосунок, який дозволяє створювати та керувати нагадуваннями за допомогою алгоритмів фільтрації та штучного інтелекту.

Висновки. Розроблено мобільний застосунок, що дозволяє створювати та керувати нагадуваннями за допомогою алгоритмів фільтрації та штучного інтелекту, що полегшує користувачу планувати свої справи.

Галузь використання – користувачі, яким потрібно створити, керувати та організувати нагадування.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 117 pages, 14 tables, 45 figures, 3 appendices, 22 sources.

MOBILE APPLICATION, IOS, PLANNER, REMINDERS, SWIFTUI, SWIFT, COREML, REMINDERS.

The object of the study is algorithms for implementing intelligent task schedulers.

The subject of the study is filtering and artificial intelligence algorithms for creating and managing reminders on a mobile phone.

The purpose of the work is to implement a mobile application for creating and managing reminders using filtering algorithms and artificial intelligence for the iOS platform.

Materials, methods, and technical means: the Swift programming language, the SwiftUI and UIKit frameworks, the SwiftData, CoreML, and EventKit libraries, the Xcode development environment, a personal computer with an M1 Pro processor running the macOS Tahoe operating system, a mobile device with iOS 26.

Results. A mobile application has been created that allows users to create and manage reminders using filtering algorithms and artificial intelligence.

Conclusions. A mobile application has been developed that allows users to create and manage reminders using filtering algorithms and artificial intelligence, making it easier for users to plan their tasks.

Field of use – users who need to create, manage, and organize reminders.

ЗМІСТ

С.

Перелік скорочень та умовних познак 8
Вступ 9
1 Аналіз проблеми та постановка завдань роботи 10
  1.1 Аналіз предметної області 10
  1.2 Приклади існуючих аналогів 10
    1.2.1 Reminders 10
    1.2.2 Google Tasks 12
    1.2.3 Microsoft To Do 13
    1.2.4 Порівняльна таблиця існуючих аналогів з розроблюваним програмним продуктом 14
  1.3 Постановка завдання для роботи 16
  1.4 Висновки до розділу 17
2 Матеріали і методи 18
  2.1 Огляд основних алгоритмів для класифікації текстів 18
    2.1.1 MaxEnt 18
    2.1.2 CRF 18
    2.1.3 BERT 19
  2.2 Створення датасету для навчання та тестування моделі 20
  2.3 Висновки до розділу 22
3 Розробка архітектури програми 23
  3.1 Функціональні вимоги до програми 23
  3.2 Вибір програмних ресурсів 24
    3.2.1 Вибір фреймворку та мови програмування 24
    3.2.2 Вибір середовища розробки 26
    3.2.3 Вибір інструментарію для побудови нейромережі 27
  3.3 Вибір типу бази даних та бібліотеки для її побудови 28
  3.4 Архітектура розроблювального програмного продукту 30

3.5 Керування станом застосунку ... 31
3.6 Висновки до розділу ... 32
4 Основні рішення щодо реалізації компонентів системи ... 34
4.1 Схема функціонування програмного продукту ... 34
4.2 Структура бази даних ... 35
4.3 Опис структури програмного продукту ... 39
4.3.1 Опис пакету Data ... 40
4.3.2 Опис пакету Domain ... 41
4.3.3 Опис пакету Presentation ... 42
4.3.4 Опис пакету Resources ... 43
4.3.5 Опис пакету ReminderTests ... 44
4.4 Опис алгоритмів роботи програмного продукту ... 44
4.4.1 Алгоритм синхронізації з EventKit ... 44
4.4.2 Алгоритм створення та редагування списку ... 45
4.4.3 Алгоритм створення та редагування нагадування ... 46
4.4.4 Алгоритм створення та редагування AI-списку ... 47
4.4.5 Алгоритм створення та редагування смарт-списку ... 48
4.4.6 Алгоритм фільтрації нагадувань у смарт-списку ... 48
4.5 Висновки до розділу ... 49
5 Експлуатація, тестування та експериментальне дослідження програми ... 50
5.1 Призначення програмного продукту та вимоги до виконання ... 50
5.2 Експлуатація програмного продукту ... 50
5.3 Проведення тестування програмного продукту та аналіз отриманих результатів ... 58
5.4 Висновки до розділу ... 60
Висновки ... 61
Перелік джерел посилання ... 62
Додаток А Технічне завдання ... 64
Додаток Б Текст програми ... 67
Додаток В Слайди презентації ... 109

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

AI – Artificial Intelligence;

API – Application Programming Interface;

JSON – JavaScript Object Notation;

BERT – Bidirectional Encoder Representations from Transformers;

MaxEnt – Maximum Entropy Classifier;

CRF – Conditional Random Fields;

SQL – Structured Query Language;

IDE – Integrated Development Environment;

NoSQL – Not only SQL;

RAM – Random Access Memory;

БД – База даних;

Мб – Мегабайт;

UI – User Interface.

ВСТУП

У сучасному зайнятому світі відстеження завдань та термінів може бути приголомшливим. Нагадування стали важливим інструментом для того, щоб залишатися організованим і бути на вершині свого графіка [1].

Опираючись на результати дослідження щодо зниження проспективної пам'яті, з'ясувалося, що нагадування не знижують проспективну пам'ять, а, навпаки, нівелюють її зниження з віком, допомагаючи здебільшого літнім людям підтримувати високу продуктивність [2].

Тому, розуміючи всю важливість нагадувань у житті кожної людини, розробниками зі всього світу було створено безліч мобільних додатків, які зробили нагадування ще зручнішими, додавши різні види автоматизацій їх спрацьовування та повторення.

Зважаючи на це, ця робота буде присвячена не стільки створенню нагадувань, скільки їх сортуванню за створеними користувачем списками різних типів. Цей мобільний застосунок забезпечить потужну фільтрацію нагадувань за різними їх ознаками як за допомогою алгоритмів фільтрації, так і штучного інтелекту, що зможе ще більше покращити керування нагадуваннями для користувача.

1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАВДАНЬ РОБОТИ

1.1 Аналіз предметної області

Планувальник нагадувань являє собою тип застосунку чи системи що допомагає користувачу організовувати нагадування про події у визначений час. В якості аналогу для визначення вимог до планувальника було обрано застосунок компанії Apple «Reminders».

Нагадування (англ. Reminders) — програма керування завданнями, розроблена компанією Apple Inc. для своїх платформ iOS, macOS та watchOS, яка дозволяє користувачам створювати списки та встановлювати для себе сповіщення [3].

У сучасному ритмі життя кількість інформації, яку потрібно утримувати в пам’яті, постійно зростає, тому системи нагадувань стали невід’ємною частиною цифрової екосистеми користувача. Вони підвищують продуктивність, знижуючи когнітивне навантаження, та дозволяють ефективніше керувати часом. Завдяки інтеграції з календарями, сповіщеннями та штучним інтелектом, нагадування еволюціонували від простих повідомлень до «розумних» планувальників, здатних самостійно структурувати завдання й підлаштовуватись під звички користувача.

1.2 Приклади існуючих аналогів

Серед всіх мобільних додатків у «App Store» багато надає функціонал створення та керування нагадуваннями де найпопулярнішими рішеннями залишаються «Reminders», «Google Tasks» та «Microsoft To Do».

1.2.1 Reminders

Нативний застосунок від Apple який має найглибшу інтеграцію з iOS девайсами.

За допомогою програми «Reminders» на iOS та iPadOS ви можете створювати нагадування з підзавданнями та вкладеннями, а також встановлювати сповіщення на основі часу та місцезнаходження [4]. Приклад зображено на рис. 1.1.

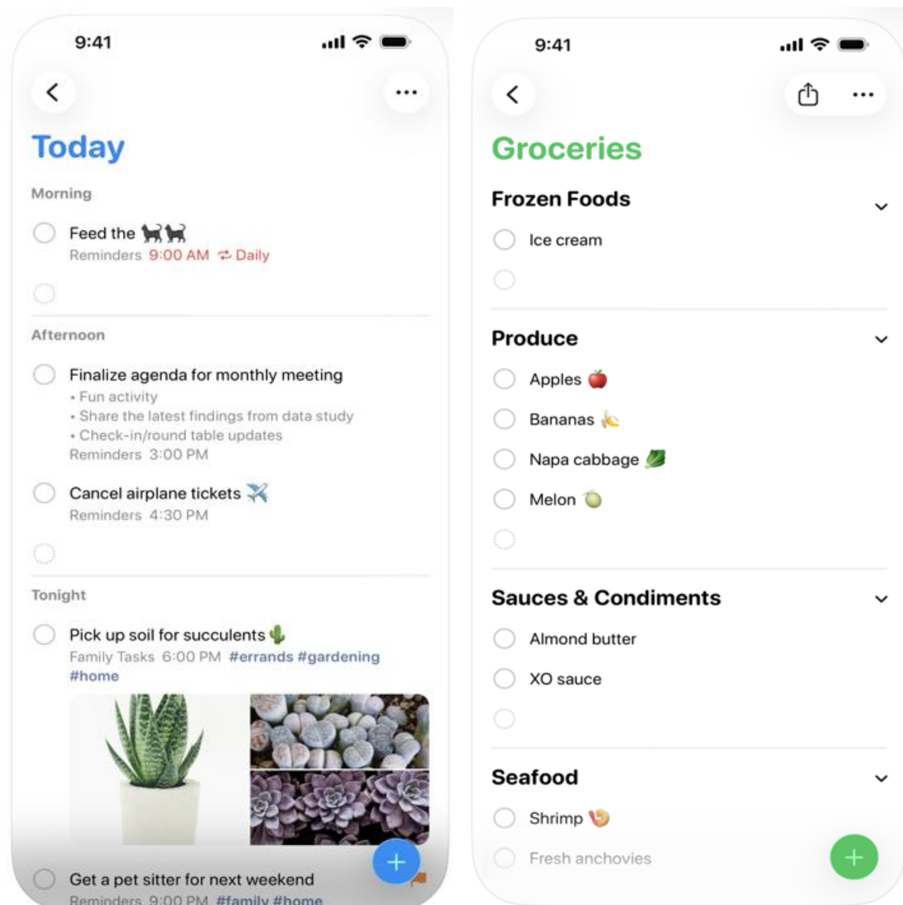

Рисунок 1.1 – Мобільний застосунок Reminders [5]

Основні переваги сервісу:

– інтеграція в екосистему Apple;
– простий та інтуїтивний інтерфейс згідно гайдлайнів;
– потужний механізм повторюваних нагадувань;
– підтримка розумних списків з фільтрацією.

Основні недоліки:

– доступний лише в екосистемі Apple;
– немає розумного сортування нагадувань за допомогою нейромереж.

1.2.2 Google Tasks

Google Tasks – це програма для керування завданнями, розроблена компанією Google і включена до Google Workspace.

Спочатку Google Tasks було включено як функцію в Gmail і Google Календар, а в 2018 році було запущено як основний продукт з окремим додатком. Вона доступна для Android та iOS, а також на правій бічній панелі в додатках Google Workspace у веб-версії та в Google Календарі [6]. Приклад зображено на рис. 1.2.

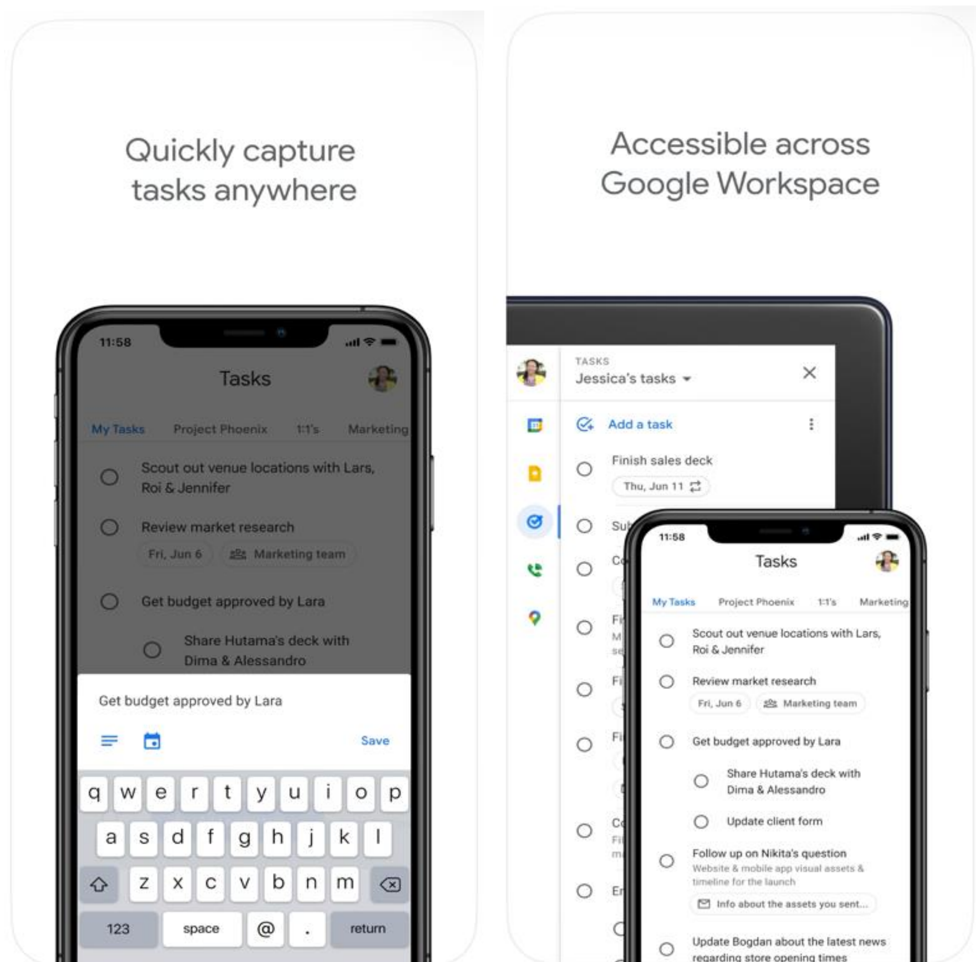

Рисунок 1.2 – Мобільний застосунок Google Tasks [7]

Основні переваги сервісу:

– інтеграція з сервісами Google;

– кросплатформність;

– підтримка дедлайнів;
– гнучкий механізм повторюваних нагадувань.

Основні недоліки:

– відсутність нагадувань за локацією;
– немає тегів та пріоритетів;
– дизайн не зовсім відповідає гайдлайнам Apple;
– немає розумних списків.

1.2.3 Microsoft To Do

Microsoft To Do – це програма для управління завданнями на основі хмарних сервісів. Дозволяє своїм користувачам керувати завданнями зі смартфона, планшета та комп’ютера. Цю технологію розроблено командою Wunderlist, яку придбала Microsoft, а окремі програми вводяться в існуючу функцію завдань в асортименті продуктів Outlook [8]. Приклад зображено на рис. 1.3.

Основні переваги сервісу:

– глибока інтеграція з Microsoft Office;
– спільні списки;
– кросплатформність;
– можливість створювати списки, пріоритети, підзадачі.

Основні недоліки:

– перевантажений інтерфейс з слабкою відповідністю гайдлайнам;
– недостатньо гнучкий механізм повтору нагадування;
– немає розумних списків.

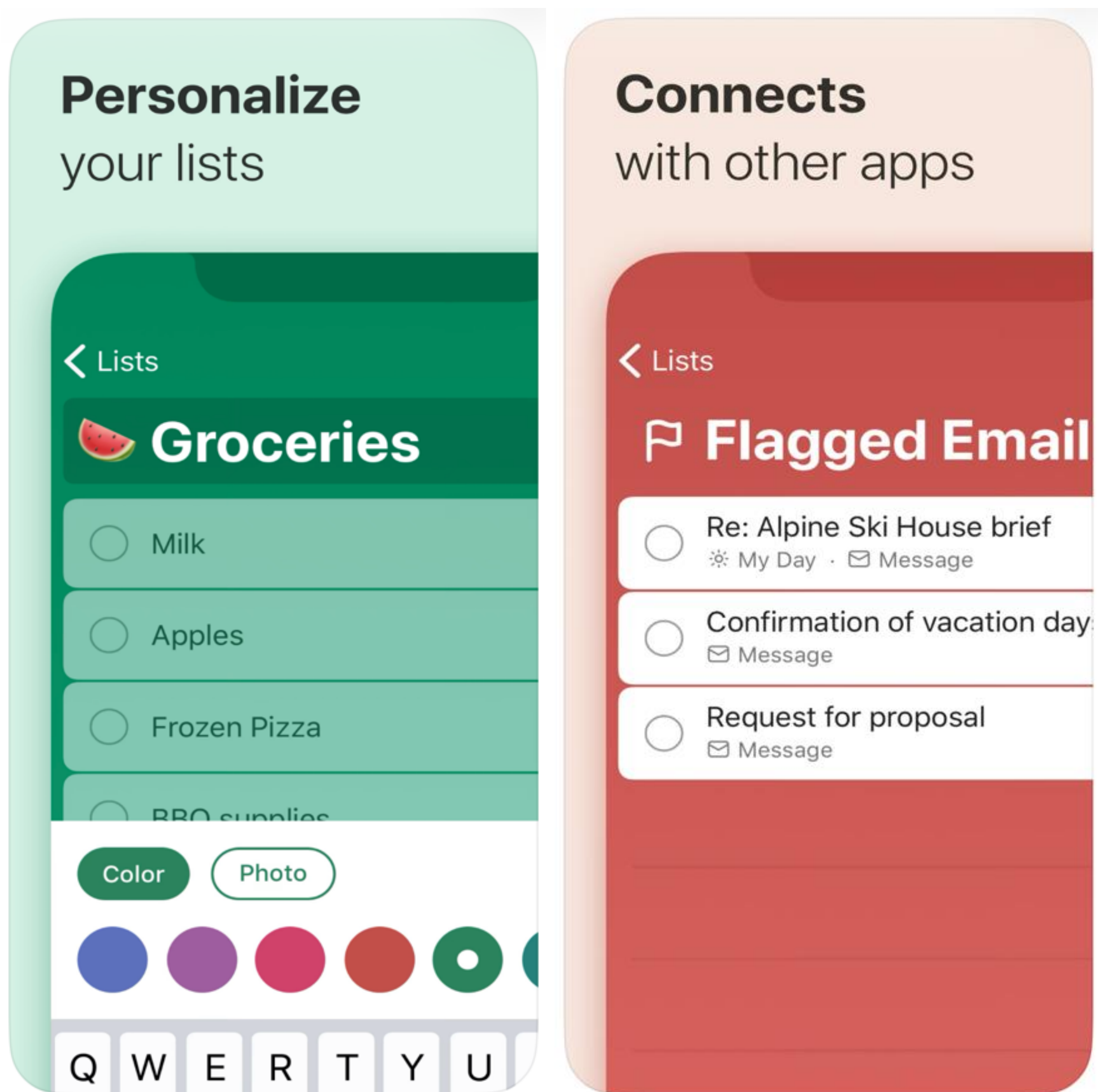

Рисунок 1.3 – Мобільний застосунок Microsoft To Do [9]

1.2.4 Порівняльна таблиця існуючих аналогів з розроблюваним програмним продуктом

Уважно проаналізувавши існуючі аналоги, всі вони дають можливість створювати локальні нагадування з повтором та доступною синхронізацією між пристроями, списки нагадувань та надання пріоритетів для сортування нагадувань.

Проте, найбільш корисний функціонал надає застосунок від Apple. Він має найбільш тісну інтеграцію зі сторонніми додатками, даючи API для створення або редагування нагадувань та списків. Для створення складних нагадувань у сторонньому додатку це найліпший спосіб, тому оптимальним

рішенням було взяти це API для створення та керування нагадуваннями та їх списками.

Також необхідно зробити смарт-списки з фільтрацією нагадувань за різними ознаками, такими як теги нагадування, належність чи неналежність до того чи іншого списку нагадувань, дати та часу нагадування тощо, та AI-списки які за допомогою нейромереж автоматично відфільтровуватимуть лише ті нагадування, які за тематикою збігаються з тематикою цих списків. Ці списки повинні значно полегшити користувачеві організовувати свої нагадування.

Запланований для розробки застосунок буде враховувати наступні деталі:

– простий та зрозумілий інтерфейс;
– синхронізація списків з Apple Reminders;
– синхронізація нагадувань з Apple Reminders;
– звичайні списки нагадувань;
– смарт-списки нагадувань з фільтрацією за ознаками нагадувань;
– AI-списки де нейромережа автоматично відбирає нагадування за змістом;
– сортування нагадувань у списках;
– теги для сортування нагадувань.

У табл. 1.1 проведено порівняння функціональних можливостей розглянутих програмних продуктів та програми «Reminders Plus», яка буде створена в ході виконання роботи.

Таблиця 1.1 – Порівняльна характеристика застосунків

Критерій	Reminders	Google Tasks	Microsoft To Do	Reminders Plus
1	2	3	4	5
Синхронізація між Apple пристроями	+	-	-	+

Продовження таблиці 1.1

1	2	3	4	5
Списки нагадувань	+	+	+	+
Смарт-списки нагадувань з алгоритмами фільтрації	+	-	-	+
Сортування нагадувань	+	+	+	+
Присвоєння нагадуванням тегів	+	-	-	+
AI-списки де нейромережа автоматично відбирає нагадування за змістом	-	-	-	+
Потужна система створення повторюваних нагадувань	+	-	-	+

1.3 Постановка завдання для роботи

Проаналізувавши аналоги застосунків, розроблюваний застосунок має виконувати наступні функції:

– повна синхронізація списків та нагадувань з Apple Reminders;

– створення нагадування, яке повинно мати повний набір параметрів нагадування з Apple Reminders та прикріплені теги;

– можливість видаляти та редагувати створені нагадування;

– створення списку нагадування яке має повний набір параметрів списку Apple Reminders, іконка та збережені фільтри;

– створення смарт-списку з потужним механізмом фільтрації за датою, часом, тегами та списками;

– створення AI-списку, де нейромережа автоматично відбирає нагадування за змістом;

– можливість видаляти та редагувати створені списки;

– можливість сортування нагадувань за назвою, датою створення, пріоритетом, датою нагадування та власноруч;

– можливість налаштування сортування списку.

Застосунок буде розроблено для мобільних девайсів з операційною системою iOS.

1.4 Висновки до розділу

У першому розділі було розглянуто область застосування, аналізуючи існуючі аналоги програм, такі як "Reminders", "Google Tasks", "Microsoft To Do".

У цілому, представлені аналоги мають гарно продумані системи створення та керування нагадуваннями. Вони також дають змогу створювати повторювані нагадування різної складності та списки з ними.

Під час аналізу цих застосунків було виявлено деякі недоліки, такі як перенасичений інтерфейс, що може призвести до плутанини серед користувачів, а також відсутність можливості автоматичного сортування нагадувань за допомогою штучного інтелекту.

Порівнявши аналоги з застосунком "Reminders Plus", було зроблено висновок, що розробка цього мобільного застосунку є актуальною та доцільною. Це обумовлено низкою функцій, які передбачає "Reminders Plus", на відміну від існуючих аналогів.

2 МАТЕРІАЛИ І МЕТОДИ

2.1 Огляд основних алгоритмів для класифікації текстів

У підрозділі 1.2 для класифікації нагадувань за назвою було обрано модель Transfer learning: Bert Embedding або BERT. У даному підрозділі буде виконано розбір основних моделей текстової класифікації доступних у Create ML (MaxEnt, CRF та BERT) та проведено їхнє порівняння для обґрунтування вибору моделі.

2.1.1 MaxEnt

Принцип максимальної ентропії – це метод вибору розподілу ймовірностей, який максимізує ентропію з урахуванням заданих обмежень, гарантуючи, що не передбачається жодної додаткової інформації, крім відомої. У наборі всіх розподілів у скінченному просторі вибірки розподіл, який максимізує ентропію, є рівномірним розподілом, де всі окремі елементи є рівноймовірними. MaxEnt працює за принципом максимальної ентропії, який стверджує, що з усіх моделей, які відповідають нашим навчальним даним, він вибирає ту, яка має найбільшу ентропію. MaxEnt не передбачає, що функції є умовно незалежними одна від одної [10].

Основними перевагами даного методу є його ефективність у роботі з процесором та пам'яттю, а також легкість масштабування. Але він погано працює з нелінійними залежностями та не моделює залежності між послідовними станами, що виливається в гіршу точність роботи методу.

2.1.2 CRF

Conditional Random Fields (CRF) — це клас дискримінаційних імовірнісних графічних моделей, розроблених для задач структурованого

прогнозування, метою яких є моделювання умовної ймовірності послідовності міток, заданої послідовністю спостережуваних даних [11].

Ця модель машинного навчання справді добре позначає послідовності, наприклад, позначає кожне слово в реченні як ІМЕННИК, ДІЄСЛОВО або ПРИКМЕТНИК. Унікальність CRF у тому, що вони не просто позначають кожне слово окремо — вони також враховують зв'язки між словами [12].

Перевагами моделі є гарне моделювання залежностей між словами чи мітками та гнучкість опрацювання ознак. Недоліками є слабка здатність до масштабування та погане опрацювання нелінійних залежностей.

2.1.3 BERT

Bidirectional Encoder Representations from Transformers (BERT) є добре відомою моделлю, яка використовує механізм уваги та досягла помітного успіху в різноманітних завданнях NLP. Його ефективність призвела до широкого впровадження, і багато інших методів глибокого навчання також включили механізм уваги, що призвело до істотного підвищення точності класифікації. BERT надає надійні рішення, які можуть генерувати контекстні представлення слів [13].

Вона використовує трансформерну нейронну мережу для розуміння та створення людської мови. У звичайній моделі трансформера є дві основні частини: одна частина (звана кодувальником) читає наданий текст, а інша частина (декодер) використовує цю інформацію, щоб робити прогнози для будь-якого завдання, над яким вона працює. Проте BERT робить усе трохи інакше. Він використовує лише кодер, оскільки його основна робота полягає в розумінні та моделюванні мови [14].

На відміну від інших моделей, ця модель розуміє глобальний контекст і завдяки цьому створює дуже сильні векторні представлення всього тексту, що робить її найбільш точною в задачах класифікації текстів. Недоліком можна було виділити б високу вартість тренування та роботи.

Порівняльний аналіз даних моделей представлено в табл. 2.1. Після аналізу результатів порівняння прийнято рішення, що BERT є оптимальним вибором через його високу точність класифікації, обумовлену здатністю встановлювати контекст, а високу вартість навчання та роботи під мобільні пристрої було компенсовано оптимізацією моделі у додатку Create ML.

Таблиця 2.1 – Порівняльний аналіз моделей текстової класифікації

Параметр порівняння	MaxEnt	CRF	BERT
Врахування контексту	Ні	Лише лінійну послідовність	Глобальний контекст
Робота з різними мовами	Низька	Низька	Висока
Потреба у великих даних	Низька	Середня	Висока
Швидкість навчання	Швидка	Середня	Повільна
Точність на складних текстах	Низька	Середня	Висока

2.2 Створення датасету для навчання та тестування моделі

Для навчання та тестування моделі було створено власний датасет, який базувався на основі «MS-LaTTE» з GitHub репозиторію [15]. Обраний набір даних містив 10101 екземпляр типових текстів «to do tasks» англійською мовою, поділених на 293 класи, де кожен анотовано для ймовірних місць і часу доби, коли він завершується. Приклад екземпляра з датасету представлено на рис. 2.1.

```
[
    {
        "ID": "3026964",
        "TaskTitle": "rearrange closet",
        "ListTitle": "home",
        "LocJudgements": [
            {
                "Known": "yes",
                "Locations": "home",
                "PublicLocations": ""
            },
            {
                "Known": "yes",
                "Locations": "home",
                "PublicLocations": ""
            },
            {
                "Known": "yes",
                "Locations": "home",
                "PublicLocations": ""
            }
        ],
        "TimeJudgements": [
            {
                "Known": "yes",
                "Times": "WE-morning"
            },
            {
                "Known": "yes",
                "Times": "WE-morning"
            },
            {
                "Known": "yes",
                "Times": "WE-afternoon,WD-evening"
            },
            {
                "Known": "yes",
                "Times": "WE-morning"
            },
            {
                "Known": "yes",
                "Times": "WE-evening"
            }
        ]
    },
```


Рисунок 2.1 – Екземпляр з датасету

Під час створення власного датасету поля «TaskTitle» та «ListTitle» були залишені як найбільш значущі для класифікації. Їх було перейменовано на «Text» та «Label» відповідно, тобто мітка «Text» є вхідним параметром у моделі та буде відповідати за текст нагадування, яке потрібне класифікувати, а мітка «Label» є вихідним параметром, який буде тематичним списком. Також кількість класів було узагальнено з 293 до 5, через те що модель майже не могла розрізнити дуже схожі за значенням класи. Екземпляри вилучених класів або переходили до існуючого класу, який мав той самий контекст, або

були вилучені з датасету, якщо вони не додавали репрезентативності. Після закінчення процесу узагальнення класи мали дуже різну кількість екземплярів, що призводило до перенавчання, тому датасет було збалансовано, створивши власні екземпляри даних у класах, де їх було недостатньо. Результати оптимізації базового датасету представлено на рис. 2.2.

Рисунок 2.2 – Оптимізований датасет

2.3 Висновки до розділу

У даному розділі було виконано огляд основних доступних моделей текстової класифікації у Create ML (MaxEnt, CRF та BERT) та проведено їхнє порівняння для вибору моделі. За результатами порівняння моделлю було обрано BERT через найліпшу точність завдяки розумінню глобального контексту та гарній оптимізації під мобільні пристрої. Для навчання та тестування моделі для класифікації нагадувань за темою було створено власний датасет на основі «MS-LaTTE» з GitHub репозиторію.

3 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

3.1 Функціональні вимоги до програми

Розроблюваний мобільний застосунок має наступні функціональні вимоги:

- повна синхронізація списків та нагадувань з Apple Reminders;
- створення нагадування, яке повинно мати повний набір параметрів нагадування з Apple Reminders та прикріплені теги;
- можливість видаляти та редагувати створені нагадування;
- створення списку нагадувань яке має повний набір параметрів списку Apple Reminders, іконку та збережені фільтри;
- створення смарт-списку з потужним механізмом фільтрації за датою, часом, тегами та списками;
- створення AI-списку де нейромережа автоматично відбирає нагадування за змістом;
- можливість видаляти та редагувати створені списки;
- можливість сортування нагадувань за назвою, датою створення, пріоритетом, датою нагадування та власноруч;
- можливість налаштування сортування списку.

На основі цих функціональних вимог було створено діаграму прецедентів нижче (рис. 3.1).

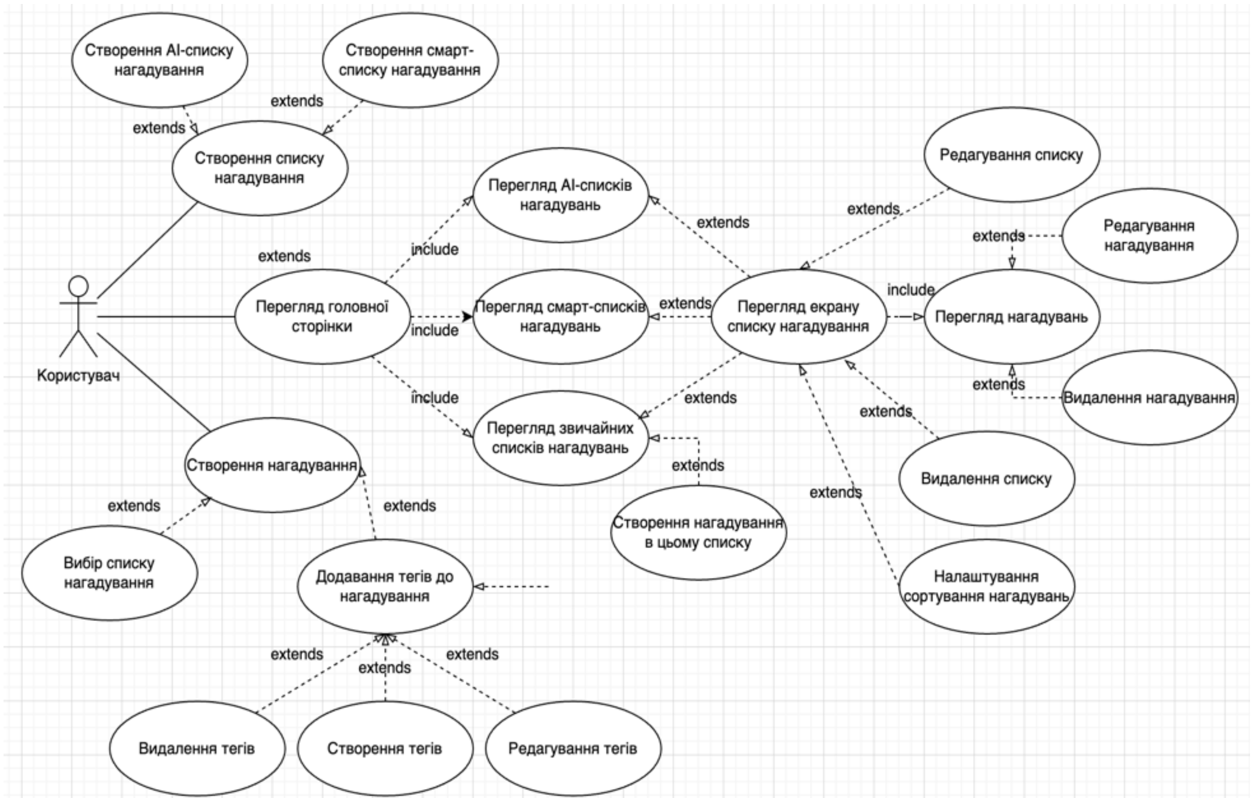

Рисунок 3.1 – Діаграма прецедентів

3.2 Вибір програмних ресурсів

3.2.1 Вибір фреймворку та мови програмування

Для виконання вказаного завдання було проведено аналіз існуючих фреймворків та обрано фреймворк SwiftUI.

SwiftUI — це набір інструментів інтерфейсу користувача, який дозволяє нам розробляти програми декларативним способом. Це дивовижний спосіб сказати, що ми повідомляємо SwiftUI, як ми хочемо, щоб наш інтерфейс користувача виглядав і працював, і він визначає, як це зробити, коли користувач взаємодіє з ним [16].

SwiftUI стає геймчейнджером у сфері розробки програм для iOS. Він пропонує більш інтуїтивно зрозумілий, ефективний і сучасний підхід порівняно з традиційним UIKit. Його декларативний синтаксис, попередній перегляд у реальному часі, сумісність між платформами та автоматична

обробка макета роблять його переконливим вибором для розробників, яким потрібна швидша та приємніша розробка [17].

У табл. 3.1 наведено порівняльний аналіз фреймворку SwiftUI та інших фреймворків для створення мобільних додатків під iOS.

Таблиця 3.1 – Порівняння фреймворків

Критерій	Flutter	SwiftUI	UIKit
Повний інструментарій Apple	-	+	+
Парадигма програмування	Декларативна	Декларативна	Імперативна
Простота використання	Середня	Непогана	Найвища
Розвиток фреймворку	Так	Так	Ні (тільки підтримка оновлень)
Споживання ресурсів	Середнє	Ефективне	Найменше

Найбільш важливими для iOS додатку є відповідність інтерфейсу гайдлайнам Apple та ефективне споживання ресурсів. Обидва цих критерії притаманні обраному фреймворку.

Swift — це універсальна багатопарадигмальна мова програмування, розроблена Apple Inc. Вона була вперше представлена в 2014 році як заміна мови програмування Objective-C. Swift розроблений як швидкий, безпечний і виразний. Він включає сучасні концепції та функції програмування, що полегшує розробникам написання ефективного коду, який зручно підтримувати [18].

Основні переваги обраної мови програмування:

– регулярна підтримка оновлень;
– сучасний синтаксис;
– підтримка асинхронності та багатопоточності;
– автоматичний збір сміття;
– висока продуктивність.

3.2.2 Вибір середовища розробки

В якості IDE для подальшої розробки після аналізу доступних варіантів було обрано Xcode, оскільки він забезпечує розробника всіма потрібними інструментами, спеціально призначеними для мобільної розробки під iOS.

Xcode – це інтегроване середовище розробки (IDE), створене Apple Inc. для розробки програмного забезпечення для iOS, iPadOS, macOS, watchOS, tvOS тощо. Він містить набір інструментів, які розробники можуть використовувати для написання, налагодження та тестування програмного забезпечення, а також інструменти для керування файлами та ресурсами проєкту [19].

Таблиця 3.2 – Порівняння середовищ розробки

Критерій	Xcode	Visual Studio Code	AppCode
1	2	3	4
Підтримка системи контролю версій	+	+	+
Безкоштовний AI агент	+	-	-
Безкоштовна	+	+	-

Продовження таблиці 3.2

1	2	3	4
Підтримка розширень	+	+	+
Підтримка Swift/SwiftUI	Повна	Обмежена	Повна
Вбудований відлагоджувач	+	Через розширення	+
Ресурсоємність	Середня	Середня	Висока
Інтеграція з інструментами Apple	+	-	Частково

3.2.3 Вибір інструментарію для побудови нейромережі

У табл. 3.3 представлено порівняння актуальних інструментів для побудови нейронних мереж для iOS девайсів: Create ML, Core ML tools та TFLite. З-поміж представлених варіантів було обрано варіант Create ML за можливість швидкого навчання моделі та простоту використання.

Create ML — це потужний інструмент, розроблений Apple, який дозволяє легко створювати моделі машинного навчання без написання коду. Він надає графічний інтерфейс користувача та набір інструментів для навчання та тестування моделей машинного навчання з використанням різних типів даних, включаючи текст, зображення та табличні дані [20].

Таблиця 3.3 – Огляд актуальних інструментів побудови нейронних мереж під iOS

Критерій	Create ML	Core ML tools	TFLite
1	2	3	4
Зручність використання	Дуже висока	Висока	Середня

Продовження таблиці 3.3

1	2	3	4
Швидкість виконання	Висока	Висока	Середня
Швидкість навчання	Дуже висока	Висока	Помірна
Споживання ресурсів	Низьке	Низьке	Помірне
Популярність та підтримка	Висока	Висока	Висока

3.3 Вибір типу бази даних та бібліотеки для її побудови

Вибір бази даних дуже важливий у проєкті будь-якого призначення, спочатку потрібно визначити тип БД (реляційний чи нереляційний).

Реляційна база даних — це тип бази даних, яка зберігає дані та забезпечує доступ до них. Ці типи баз даних називаються «реляційними», оскільки елементи даних у них мають заздалегідь визначені зв’язки один з одним. Дані в реляційній базі даних зберігаються в таблицях. Таблиці з’єднані унікальними ідентифікаторами або «ключами». Коли користувачеві потрібно отримати доступ до певної інформації, він може використовувати ключ для доступу до всіх таблиць даних, які були попередньо визначені як пов’язані з цим ключем [21].

Бази даних NoSQL, також відомі як нереляційні бази даних, розроблені для конкретних моделей даних і зберігають дані в гнучких схемах, які легко масштабуються для сучасних програм. Багато робочих навантажень бази даних можуть виграти від економічності та продуктивності баз даних NoSQL [22].

Порівняльна характеристика реляційної та нереляційної баз даних наведена в табл. 3.4.

Таблиця 3.4 – Порівняння реляційної та нереляційної баз даних

Критерії	Реляційні бази даних	Нереляційні бази даних
Модель даних	Таблична	Гнучка схема
Мова запитів	SQL	Відрізняється залежно від конкретної бази даних
Зв'язки між даними	За допомогою ключів	Відсутні або мають інший механізм (наприклад, посилання на документи)
Можливість створювати складні запити	Так	Дуже обмежена
Масштабованість	Складніше масштабувати горизонтально	Більш гнучкі, гарно масштабуються горизонтально

Оскільки в додатку потрібно створити смарт-списки, які матимуть дуже складні динамічні запити до БД, єдиним варіантом залишається реляційна база даних.

Для застосунку потрібно, щоб дані завжди були доступні перш за все локально, але й з можливістю синхронізації з хмарною БД. Для цього потрібно створити локальну БД з можливістю синхронізації з хмарою.

Було віддано перевагу базі даних SwiftData від компанії Apple з наступних причин:

– простота використання з SwiftUI;
– SQL запити;
– безшовна синхронізація між локальними та віддаленими даними.

У табл. 3.5 зображено порівняння SwiftData з аналогами.

Таблиця 3.5 – Порівняння SwiftData з аналогами

Критерій	SwiftData	Core Data	Realm
Тип	ORM на Swift	ORM на Swift/Objective-C	Незалежна БД з власним рушієм
Сумісність з SwiftUI	Повна	Часткова	Добра
Реактивність	Так	Обмежена	Так
Синхронізація	Автоматична між локальною БД та iCloud	Власноруч між локальною БД та iCloud	Тільки віддалено
Підтримка	Так	Так	Так

3.4 Архітектура розроблювального програмного продукту

Для даного застосунку було обрано архітектурний підхід MVVM, який передбачає розділення проєкту на три основні компоненти:

– model (відповідає за дані застосунку, їх збереження та бізнес-логіку);

– viewModel (здійснює зв’язок між даними (model) та інтерфейсом користувача (view), обробляє події та готує дані для відображення);

– view (відповідає за відображення даних на екрані користувача та передачу дій користувача до viewModel).

Таке розділення дозволяє зробити код зрозумілішим, полегшує його тестування, а також забезпечує зручність у масштабуванні.

Важливою особливістю архітектури MVVM є одностороння взаємодія між шарами, що забезпечує передбачуваність та стабільність роботи застосунку. Як показано на рис. 3.2, під час взаємодії користувача з інтерфейсом створюється подія, яка передається до viewModel. ViewModel обробляє цю подію, викликає необхідні методи з model для отримання або

оновлення даних, після чого формує оновлений стан. Потім цей стан автоматично передається до view, яка оновлює інтерфейс і відображає актуальні дані користувачу.

Рисунок 3.2. – Схема роботи шарів MVVM

3.5 Керування станом застосунку

Об’єкти керування станом потрібні для розділення шару бізнес-логіки від презентаційного шару застосунку. Тому до вибору об’єкта керування станом потрібно відноситись дуже серйозно.

У SwiftUI є вбудовані рішення стейт-менеджменту, наприклад ObservableObject, State та EnvironmentObject. Оскільки ці рішення від самої Apple, вони використовуються у всіх її фреймворках.

Також є велика кількість сторонніх пакетів, призначених для стейт-менеджменту, найбільш популярні – це ReSwift та The Composable Architecture.

Порівняння цих пакетів з вбудованими рішеннями наведено у табл. 3.6.

Таблиця 3.6 – Порівняння стейт-менеджерів

Критерії	Вбудовані інструменти	ReSwift	The Composable Architecture
Простота використання	Висока	Середня	Середня
Підхід до керування станом	Локальний та реактивний	унідірекційний потік даних	унідірекційний потік даних
Архітектурна структура	MVVM	Store – Reducer – Action – State	Store – Reducer – Action – Environment
Сумісність з SwiftUI	Повна	Через адаптери	Висока
Масштабованість	Висока	Середня	Непогана
Підтримка реактивності	Так	Через адаптери	Так

Після аналізу порівняльної табл. 3.6 було обрано вбудоване рішення з наступних причин:

– простота та зручність використання;
– підтримка реактивності;
– спрощена архітектурна структура.

3.6 Висновки до розділу

У третьому розділі було виконано опис програми з визначенням функціональних вимог, мови програмування, фреймворку, середовища

розробки, типу бази даних, бібліотеки бази даних, архітектури та стейт-менеджеру.

Ретельно проаналізувавши можливі варіанти фреймворку та мови програмування для iOS застосунків, було обрано SwiftUI та Swift через значні переваги у швидкодії та зручності у використанні.

Для використання SwiftUI було обрано IDE Xcode як найліпшу для iOS-розробки через її тісну інтеграцію з нативними інструментами.

Типом бази даних було обрано реляційну базу даних через можливість робити складні запити, а бібліотекою – SwiftData за її реактивність та простоту використання.

Архітектурою застосунку буде MVVM через зручність реалізації та тестування.

Для керування станом застосунку було обрано вбудовані рішення, які мають гнучку інтеграцію з SwiftUI та простий інтерфейс взаємодії з ними.

4 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

4.1 Схема функціонування програмного продукту

На рис. 4.1 подано функціональну схему програмного забезпечення.

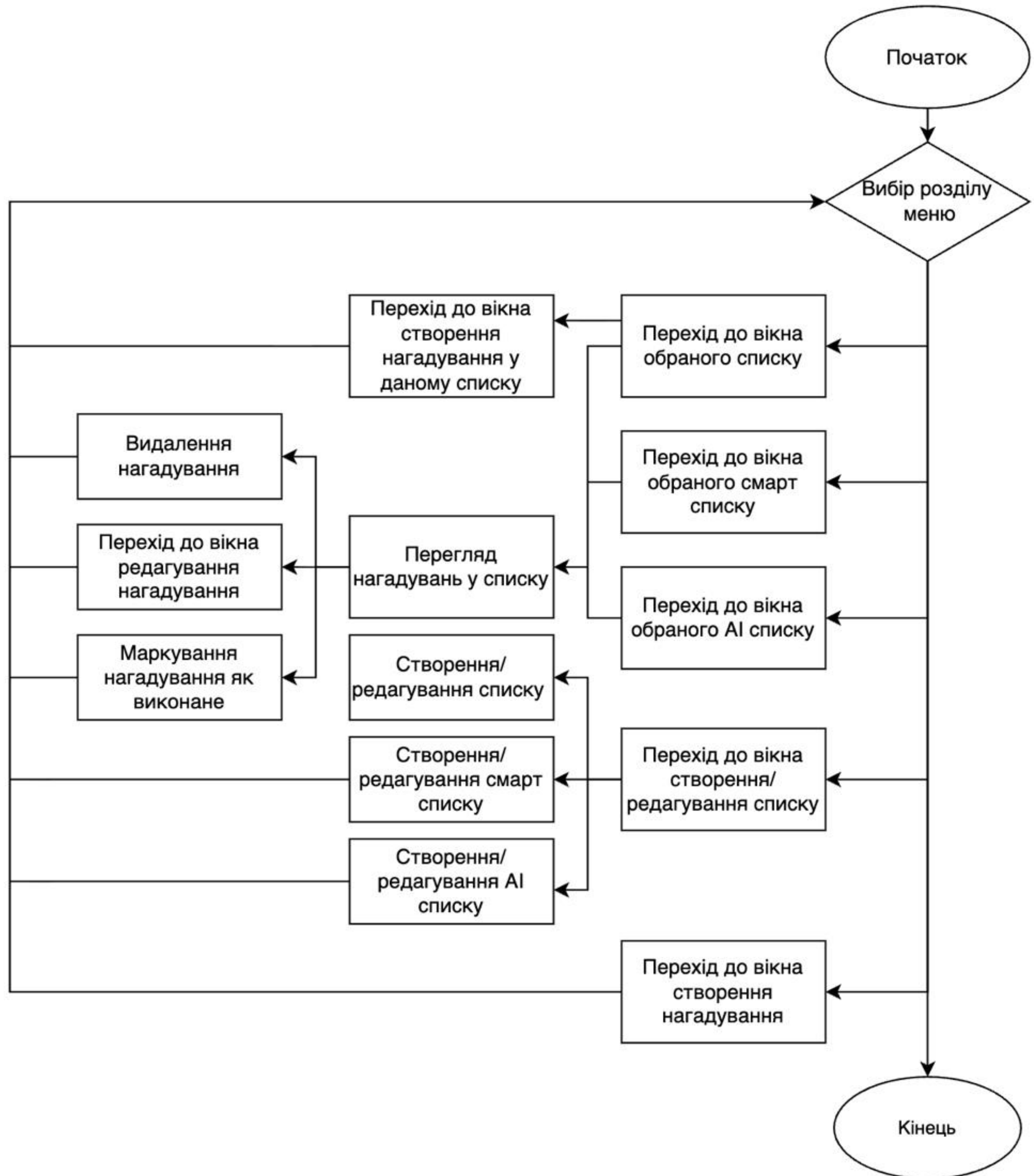

Рисунок 4.1 – Функціональна схема програмного забезпечення

4.2 Структура бази даних

У підрозділі 3.3 було проаналізовано бази даних, та обрано бібліотеку SwiftData для створення реляційної бази даних. До складу створеної БД входять наступні сутності:

- Reminder – зберігає створені нагадування;
- ReminderList – зберігає створені списки;
- ReminderSmartList – зберігає створені смарт-списки;
- ReminderAIList – зберігає створені AI-списки;
- Tags – зберігає створені теги;
- SortSettings – зберігає створені налаштування сортування.

Сутність «Reminder» має наступні поля: persistentIdentifier, calendarItemIdentifier, title, dueDate, dueHour, isCompleted, creationDate, lastModifiedDate, calendarIdentifier, tags. Опис полів сутності міститься в табл. 4.1.

Таблиця 4.1 – Поля сутності «Reminder»

Атрибут	Тип атрибута	Призначення
1	2	3
persistentIdentifier	PersistentIdentifier	Унікальний ідентифікатор
calendarItemIdentifier	String	Унікальний ідентифікатор нагадування у EventKit
title	String	Назва нагадування
dueDate	Date?	Дата спрацювання нагадування
dueHour	Int?	Година спрацювання нагадування
isCompleted	Bool	Статус завершення

Продовження таблиці 4.1

1	2	3
creationDate	Date	Дата створення нагадування
lastModifiedDate	Date?	Дата останньої зміни нагадування
calendarIdentifier	String	Унікальний ідентифікатор календаря у EventKit
tags	Array<Tag>	Список зв’язаних тегів

Сутність «ReminderList» містить наступні поля: persistentIdentifier, calendarIdentifier, title, systemImage, colorHex, sortSettings. Опис полів сутності міститься в табл. 4.2.

Таблиця 4.2 – Поля сутності «ReminderList»

Атрибут	Тип атрибута	Призначення
persistentIdentifier	PersistentIdentifier	Унікальний ідентифікатор
calendarIdentifier	String	Унікальний ідентифікатор календаря у EventKit
title	String	Назва списку
systemImage	String	Ресурс з іконкою списку
colorHex	String	Колір списку
sortSettings	SortSettings	Налаштування сортування списку

Сутність «ReminderSmartList» містить наступні поля: persistentIdentifier, calendarIdentifier, title, systemImage, colorHex, sortSettings, queriesData, matchingRuleData. Опис полів сутності міститься в табл. 4.3.

Таблиця 4.3 – Поля сутності «ReminderSmartList»

Атрибут	Тип атрибута	Призначення
persistentIdentifier	PersistentIdentifier	Унікальний ідентифікатор
title	String	Назва списку
systemImage	String	Ресурс з іконкою списку
colorHex	String	Колір списку
sortSettings	SortSettings	Налаштування сортування списку
queriesData	Data?	Список фільтрів у JSON форматі
matchingRuleData	Data?	Правило логічних зв’язків між фільтрами у JSON форматі

Сутність «ReminderAIList» містить наступні поля: persistentIdentifier, calendarIdentifier, title, systemImage, colorHex, sortSettings. Опис полів сутності міститься в табл. 4.4.

Таблиця 4.4 – Поля сутності «ReminderAIList»

Атрибут	Тип атрибута	Призначення
persistentIdentifier	PersistentIdentifier	Унікальний ідентифікатор
labelData	Data?	Тип AI-списку у JSON форматі
remindersId	Array<String>	Список ідентифікаторів нагадувань
sortSettings	SortSettings	Налаштування сортування календаря

Сутність «Tag» містить наступні поля: persistentIdentifier, calendarIdentifier, title, colorHex, reminders. Опис полів сутності міститься в табл. 4.5.

Таблиця 4.5 – Поля сутності «Tag»

Атрибут	Тип атрибута	Призначення
persistentIdentifier	PersistentIdentifier	Унікальний ідентифікатор
title	String	Назва тегу
colorHex	String	Колір тегу
reminders	Array<Reminder>	Список нагадувань

Сутність «SortSettings» містить наступні поля: persistentIdentifier, showCompleted, showOverdue, sortByData, sortOrderData. Опис полів сутності міститься в табл. 4.6.

Таблиця 4.6 – Поля сутності «SortSettings»

Атрибут	Тип атрибута	Призначення
persistentIdentifier	PersistentIdentifier	Унікальний ідентифікатор
showCompleted	Bool	Статус відображення відмічених нагадувань
showOverdue	Bool	Статус відображення прострочених нагадувань
sortByData	Data?	Правило сортування у JSON форматі
sortOrderData	Data?	Правило порядку сортування у JSON форматі

На рис. 4.2 представлено структуру бази даних з відображенням зв'язків між цими сутностями.

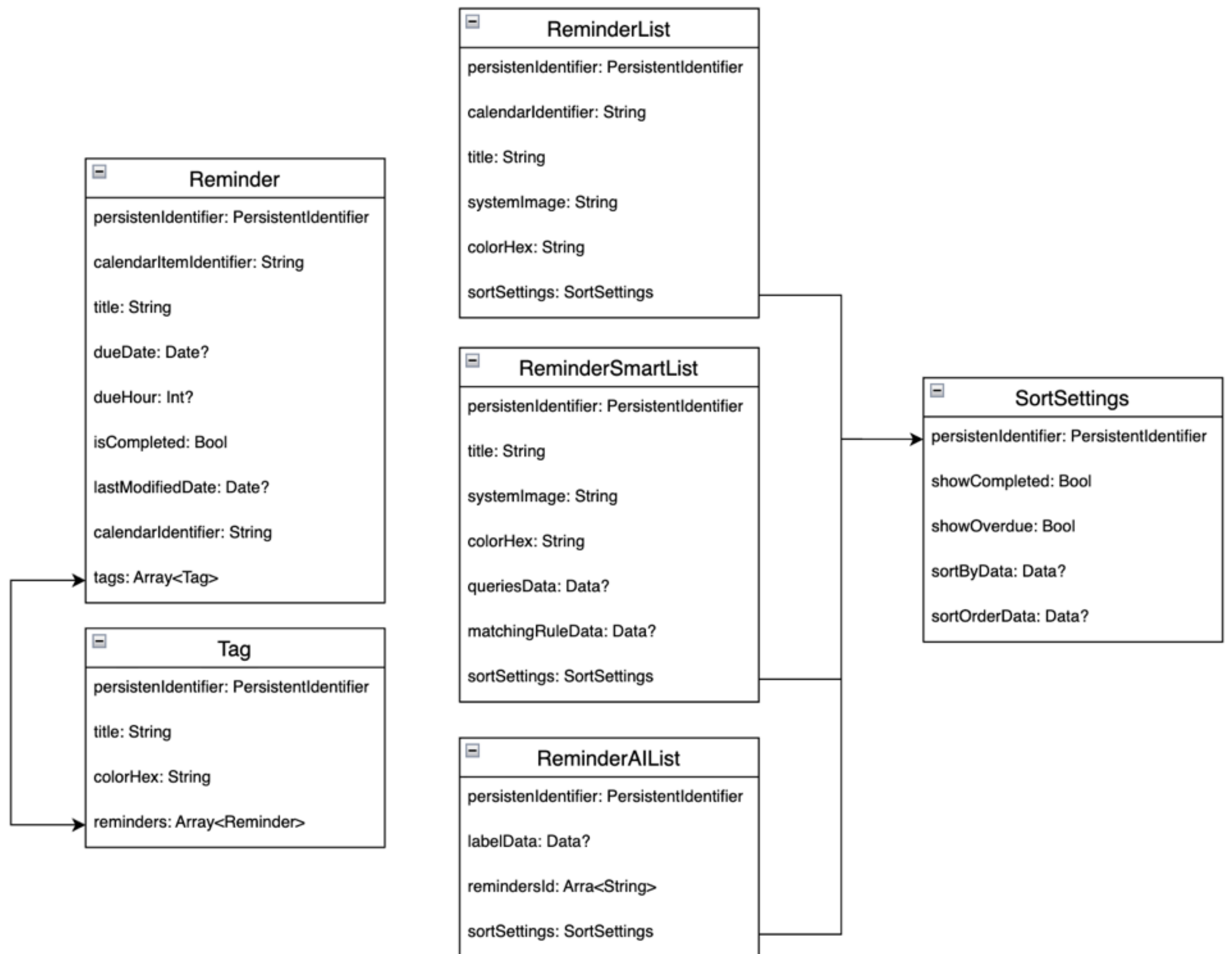

Рисунок 4.2 – Структура бази даних з відображенням зв'язків сутностей

4.3 Опис структури програмного продукту

Проєкт був побудований на основі MVVM архітектури яка була обрана у розділі 3.4. На рис. 4.3 зображено структуру проєкту застосунку.

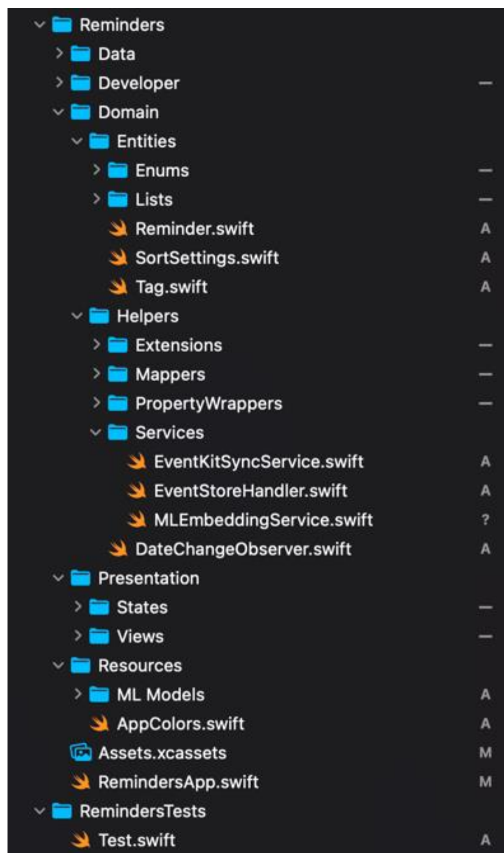

Рисунок 4.3 – Структура проєкту застосунку

4.3.1 Опис пакету Data

Пакет Data зберігає всі створені сутності та зв'язані з ними класи, а також відповідає за роботу з міграціями (рис. 4.4).

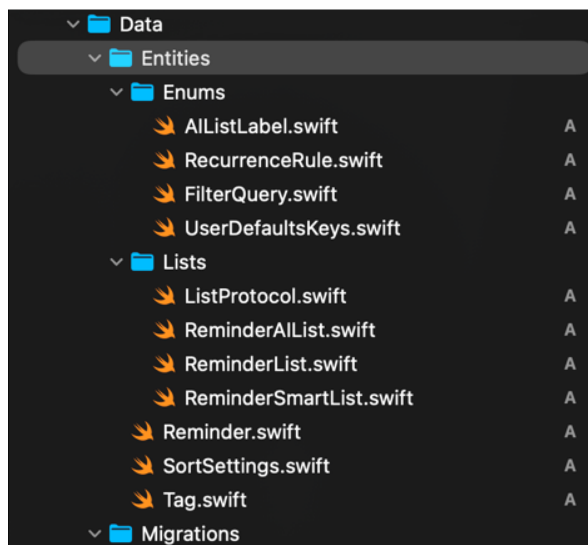

Рисунок 4.4 – Пакет Data

Директорія Entities містить піддиректорії Enums та Lists, а також файли Reminder, SortSettings та Tag. Всі файли в директорії Enums є Enum об’єктами, які конвертуються в тип даних Data?, коли зберігаються в БД. Піддиректорія Lists містить моделі сутностей окрім ListProtocol, оскільки він є протоколом для всіх сутностей.

Директорія Migrations містить міграції бази даних.

4.3.2 Опис пакету Domain

В пакеті Domain (рис. 4.5) знаходяться модулі які відповідальні за бізнес-логіку програми.

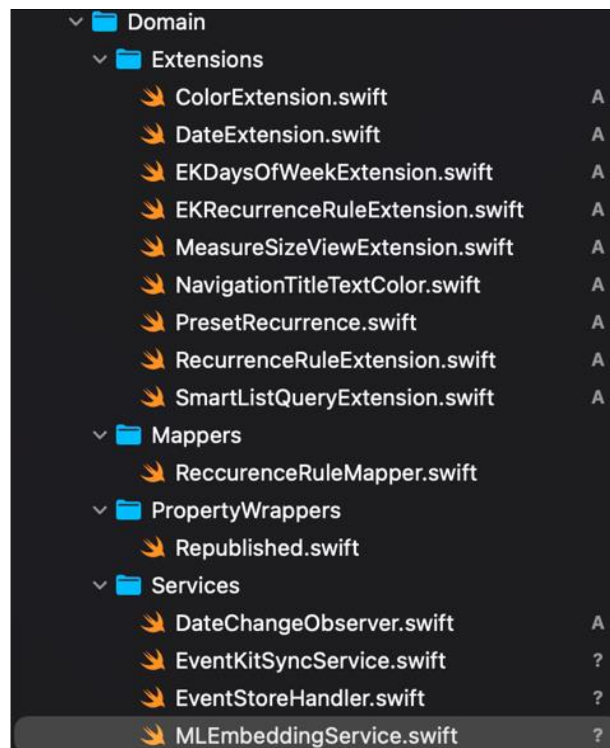

Рисунок 4.5– Пакет Domain

Директорія Extensions відповідає за зберігання файлів розширень класів.

Директорія Mappers відповідає за зберігання класів-мапперів.

Директорія PropertyWrappers зберігає створені обгортки для класів.

Директорія Services відповідає за зберігання всіх сервісів які керують створенням, редагуванням та синхронізацією об’єктів сутностей.

4.3.3 Опис пакету Presentation

В пакеті Presentation (рис. 4.6) знаходяться модулі для створення та відображення користувацького інтерфейсу застосунку.

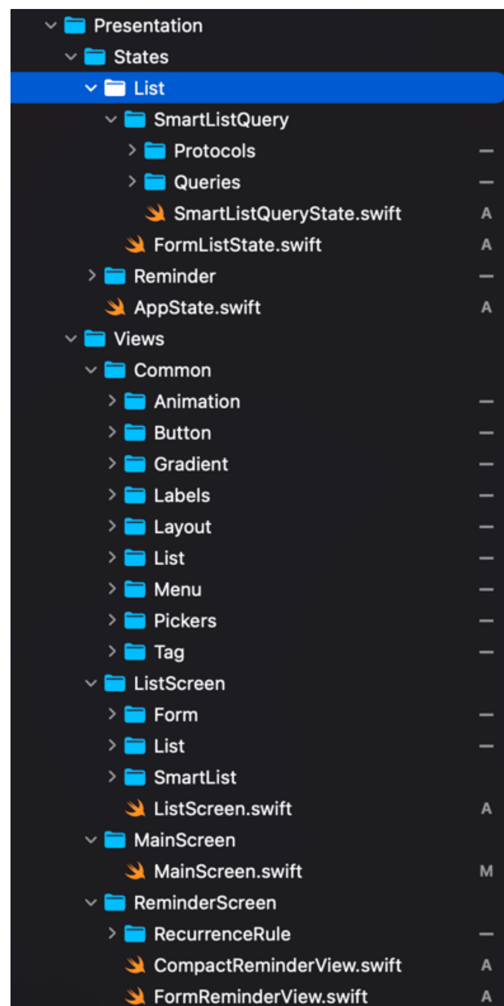

Рисунок 4.6 – Пакет Presentation

Директорія States зберігає створені класи, які віповідають за керування станом View.

Директорія Views зберігає в собі директорії Common, ListScreen, MainScreen та ReminderScreen. Піддиректорія Common зберігає модулі з

створеними View, які можуть бути перевикористані в різних екранах застосунку. Піддиректорія ListScreen зберігає класи View, які відповідають за відображення користувацького інтерфейсу у вікні списків та форми створення списків. Піддиректорія ReminderScreen зберігає класи View, які відповідають за відображення користувацького інтерфейсу у вікні нагадувань та форми створення нагадувань. Піддиректорія MainScreen зберігає класи для відображення користувацького інтерфейсу головної сторінки додатку.

4.3.4 Опис пакету Resources

В пакеті Resources (рис. 4.7) знаходяться модулі з необхідними ресурсами для застосунку.

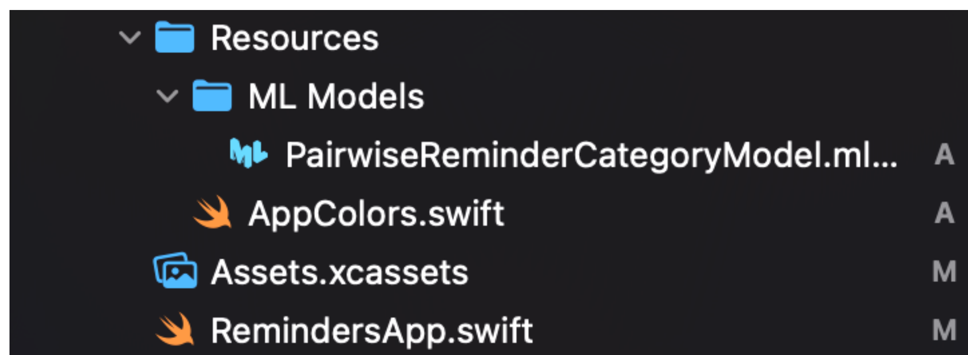

Рисунок 4.7 – Пакет Resources

Директорія ML Models зберігає файли створених моделей нейронних мереж.

Файл AppColors відповідає за зберігання кольорів програми.

Файл Assets зберігає необхідні ресурси різного призначення для застосунку.

Файл RemindersApp є точкою входу в застосунок і зберігає відповідний клас.

4.3.5 Опис пакету ReminderTests

В пакеті ReminderTests (рис. 4.8) знаходяться файли з тестуванням застосунку.

Рисунок 4.8 – Пакет ReminderTests

4.4 Опис алгоритмів роботи програмного продукту

4.4.1 Алгоритм синхронізації з EventKit

Крок 1. Користувач вперше входить у застосунок або приходить сповіщення від EventStore про зміни.

Крок 2. Викликається функція handleEventStoreChange у EventStoreHandler, в якій виконується перевірка паузи синхронізації. Якщо isPaused == true, то метод чекає 200 мс і повторює перевірку. Якщо синхронізація призупинена – повернення до перевірки паузи синхронізації у кроці 2.

Крок 3. Викликається setSyncState(true) на головному акторі, щоб позначити початок процесу синхронізації.

Крок 4. Викликається await handleCalendarChanges(), який звіряє наявні EKCalendar із локальними моделями ReminderList, оновлює назви/кольори для існуючих списків, видаляє локальні списки, яких більше немає в EventKit, додає нові списки, що з’явилися в EventKit.

Крок 5. Викликається await handleReminders(), який отримує всі актуальні ідентифікатори EKReminder, оновлює локальні Reminder, якщо змінилися дані в EventKit, видаляє локальні Reminder, яких немає в EventKit, додає нові Reminder, що з’явилися в EventKit, для змін у заголовках незавершених нагадувань ініціює оновлення AI-списків через mlService.

Крок 6. Якщо modelContext.hasChanges == true, виконується try modelContext.save() для збереження всіх оновлень у SwiftData.

Крок 7. Викликається try modelContext.save(), якщо були якісь зміни в SwiftData, та викликається setSyncState(false), щоб вимкнути індикатор синхронізації.

4.4.2 Алгоритм створення та редагування списку

Крок 1. Користувач або натискає на кнопку створення списку, або натискає на кнопку редагування у вже створеному списку.

Крок 2. Користувач вносить необхідні дані у форму зі списком та натискає на кнопку збереження.

Крок 3. Виконується перевірка, чи listType дорівнює list. Якщо так, то виконується перехід до кроку 4, інакше викликається метод збереження іншого списку та даний алгоритм завершується.

Крок 4. Якщо календар вже існує, то поля змінюються на нові, які введені користувачем, та викликається метод updateCalendar класу EventKitSyncService.

Крок 5. Викликається eventStore.requestFullAccessToReminders(). Якщо доступ не надано або сталася помилка, – кінець алгоритму.

Крок 6. Якщо eventStore.calendar(withIdentifier: list.calendarIdentifier) != nil, тоді calendar = list.ekCalendar, інакше, якщо list.ekCalendar не nil, використовується він як базовий календар для збереження.

Крок 7. Якщо знайдено iCloud (sourceType == .calDAV і title містить "icloud") – використовується як calendar.source. інакше, якщо знайдено локальне джерело (sourceType == .local) — використовується як calendar.source, інакше використовується перше доступне джерело.

Крок 8. Перевірка наявності календаря. Якщо calendar == nil – кінець алгоритму.

Крок 9. Викликається try eventStore.saveCalendar(calendar, commit: true). У разі помилки – вивід помилки та кінець алгоритму.

Крок 10. Якщо list.calendarIdentifier відрізняється від calendar.calendarIdentifier (наприклад, при створенні нового календаря EventKit присвоїв новий ідентифікатор) то оновлюється list.calendarIdentifier = calendar.calendarIdentifier, та викликається modelContext.insert(list) для фіксації зв’язку.

Крок 11. Якщо modelContext.hasChanges == true, викликається try? modelContext.save() для збереження даних у SwiftData.

4.4.3 Алгоритм створення та редагування нагадування

Крок 1. Користувач або натискає на кнопку створення нагадування, або натискає на кнопку редагування у вже створеному нагадуванні.

Крок 2. Користувач вносить необхідні дані у форму нагадування та натискає на кнопку збереження.

Крок 3. Оновлюються поля updatedEKReminder який є об’єктом класу EKReminder та updatedReminder який є об’єктом Reminder на ті, що передані у формі.

Крок 4. Викликається eventStore.requestFullAccessToReminders(). Якщо доступ не надано або сталася помилка — кінець алгоритму.

Крок 5. Викликається eventStoreHandler.pauseSync(true) для призупинення синхронізації з EventStore.

Крок 6. Зберігається об’єкт updatedEKReminder у eventStore.

Крок 7. Перевірка чи updatedReminder.calendarItemIdentifier не дорівнює updatedEKReminder.calendarItemIdentifier, якщо ні, перехід до кроку 9.

Крок 8. Змінній updatedReminder.calendarItemIdentifier присвоюється значення змінної updatedEKReminder.calendarItemIdentifier та updatedReminder вставляється в контекст SwiftData.

Крок 9. Виконується збереження даних у SwiftData та викликається eventStoreHandler.pauseSync(false) для зняття паузи.

4.4.4 Алгоритм створення та редагування AI-списку

Крок 1. Користувач або натискає на кнопку створення списку та обирає AI List, або натискає на кнопку редагування у вже створеному списку.

Крок 2. Користувач вносить необхідні дані у форму зі списком та натискає на кнопку збереження.

Крок 3. Виконується перевірка, чи listType дорівнює aiList. Якщо так, то виконується перехід до кроку 4, інакше викликається метод збереження іншого списку та даний алгоритм завершується.

Крок 4. Присвоюється нове значення label об’єкту newAIList.

Крок 5. Дістаються усі об’єкти створених нагадувань у SwiftData, у яких isCompleted дорівнює false.

Крок 6. Викликається nlModel.predictedLabelHypotheses, куди передаються всі назви об’єктів нагадувань. Він повертає масив словників [[String : Double]], де перше значення у словнику це назва AI-списку, а друга – вірогідність, що він підходить для нагадування.

Крок 7. У даного масиву викликається метод compactMap, який відбирає лише ідентифікатори нагадувань, в яких вірогідність, що він підходить до об’єкту newAIList, більше половини. Отриманий результат записується у newAIList.remindersId.

Крок 8. Виконується перевірка, чи об’єкт newAIList вже існує у БД. Якщо так, то перехід до кроку 10.

Крок 9. Об’єкт newAIList вставляється в контекст SwiftData.

Крок 10. Виконується збереження даних у SwiftData.

4.4.5 Алгоритм створення та редагування смарт-списку

Крок 1. Користувач або натискає на кнопку створення списку та обирає AI List, або натискає на кнопку редагування у вже створеному списку.

Крок 2. Користувач вносить необхідні дані у форму зі списком та натискає на кнопку збереження.

Крок 3. Виконується перевірка, чи listType дорівнює smartList. Якщо так, то виконується перехід до кроку 4, інакше викликається метод збереження іншого списку та даний алгоритм завершується.

Крок 4. Всі дані, введені користувачем, записуються до об’єкту newSmartList.

Крок 5. Виконується перевірка, чи об’єкт newSmartList вже існує у БД. Якщо так, то перехід до кроку 7.

Крок 6. Об’єкт newSmartList вставляється в контекст SwiftData.

Крок 7. Виконується збереження даних у SwiftData.

4.4.6 Алгоритм фільтрації нагадувань у смарт-списку

Крок 1. Користувач натискає на об’єкт смарт-списку на головному екрані.

Крок 2. Об’єкт смартсписку передається у функцію buildPredicate, де створюється та повертається об’єкт предикату завдяки розпакуванню атрибутів SmartListQuery та SortSettings об’єкта смарт-списку.

Крок 3. Створений у попередньому кроці предикат передається до об’єкту reminders структури Query, що реактивно підбирає лише ті нагадування зі SwiftData, що задовольняють предикат.

Крок 4. Всі нагадування, що зібралися в об’єкті reminders, виводяться списком на екран користувача.

4.5 Висновки до розділу

У даному розділі було розроблено схему функціональних можливостей програми, здійснено розбір створених сутностей для БД і спроєктовано схему зв'язків між цими сутностями в БД.

Було проаналізовано структуру програми та розібрано всі пакети створеного застосунку.

Розглянуто основні алгоритми функціонування програми та розібрано кожний алгоритм на кроки.

5 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

5.1 Призначення програмного продукту та вимоги до виконання

Метою створення цього застосунку було надати користувачам можливість легко та зручно створювати й редагувати нагадування з функцією потужної фільтрації цих нагадувань.

Мінімальні вимоги до апаратного забезпечення для роботи застосунку такі:

– смартфон на базі операційної системи iOS версії 17.0 або вищої;
– мінімум 200 Мб вільної пам’яті на пристрої.

5.2 Експлуатація програмного продукту

Перед запуском програми потрібно встановити застосунок на девайс. Для запуску програми на iOS застосунок може бути завантажений лише через App Store чи сертифіковані Apple сторонні магазини Євросоюзу. Для публікації в App Store потрібно мати сертифікат розробника Apple, створити на офіційному порталі проєкт куди завантажити файл .ipa та документацію до нього. Також застосунок може бути завантажений через запуск програми через Xcode та підв’язавши свій реальний девайс. Для запуску програми потрібно натиснути на іконку з назвою «Reminders plus».

При першому відкритті застосунку у користувача запитується дозвіл на доступ до системних нагадувань (рис. 5.1). Якщо користувач надав доступ, то він потрапляє на головний екран застосунку (рис. 5.2). Якщо дозвіл не було отримано, то застосунок автоматично закриється і при спробі його відкрити знову буде запит на дозвіл до системних нагадувань.

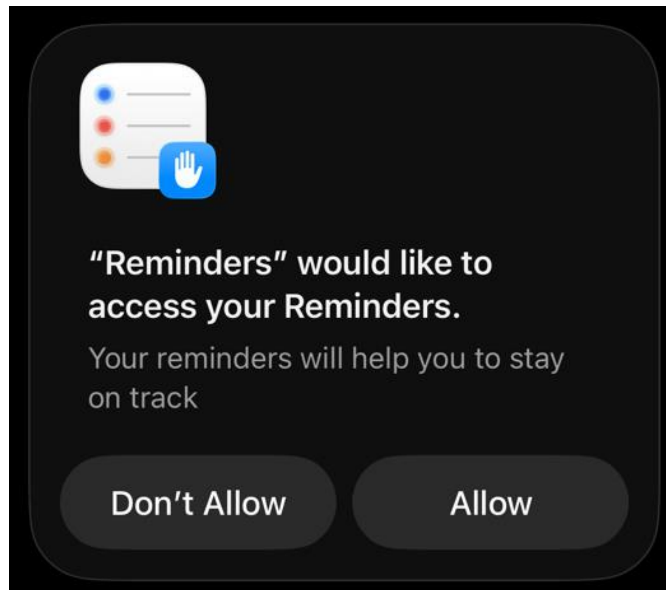

Рисунок 5.1 – Вікно запиту дозволу до системних нагадувань

Рисунок 5.2 – Головний екран додатку

На головному екрані з'являються всі списки (окрім смарт-списків), які користувач створив у додатку Reminders, а також містять усі створені користувачем нагадування. Ці списки відображені знизу в секції «My Lists». Зверху відображені передвстановлені та створені смарт-списки та AI-списки.

Для створення нового списку користувач має натиснути відповідну іконку у верхньому меню і відкрити форму створення списку. Після цього відкриється вікно форми створення нового списку (рис. 5.3).

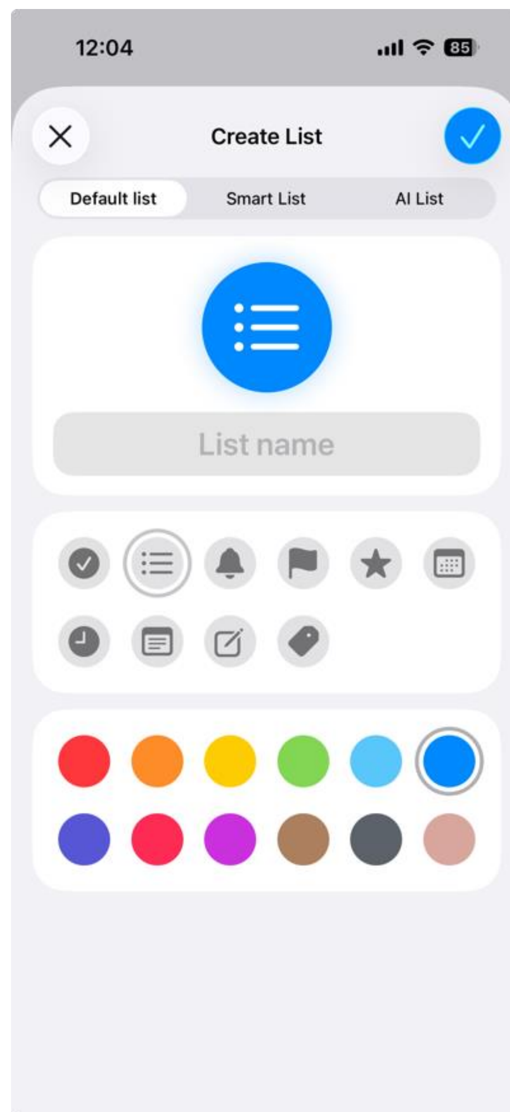

Рисунок 5.3 – Форма створення списку

Користувач може обирати тип списку між звичайним, смарт-списком або AI-списком у верхній панелі форми. У звичайному та смарт-списку

користувач може обирати назву, іконку та колір списку. У смарт-списку є ще можливість вибрати фільтри нагадувань (рис. 5.4). Фільтрувати нагадування можливо за датою, тегами, списками чи часом. Також користувач може змінити правило логічної побудови фільтрів у верхньому пікері, обравши, чи всі фільтри мають бути пройдені, чи достатньо будь-якого, для того, щоб нагадування було у цьому списку.

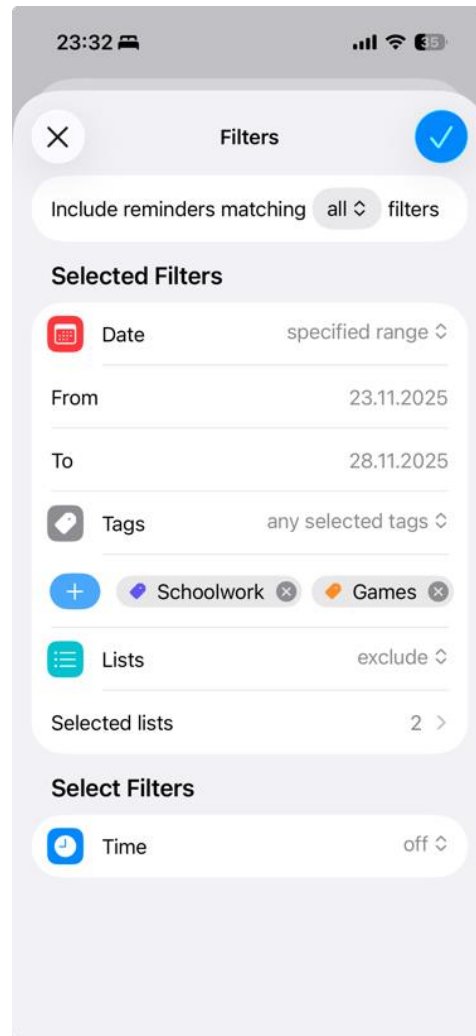

Рисунок 5.4 – Форма вибору фільтрів для смарт-списку

Якщо користувач вирішив створити AI-список, він має обирати з доступних тем для його створення (рис. 5.5).

Після того, як користувач ввів необхідні дані та обрав список, він має натиснути праву верхню кнопку, щоб підтвердити створення списку, після чого він повернеться на головне меню, де буде новостворений список.

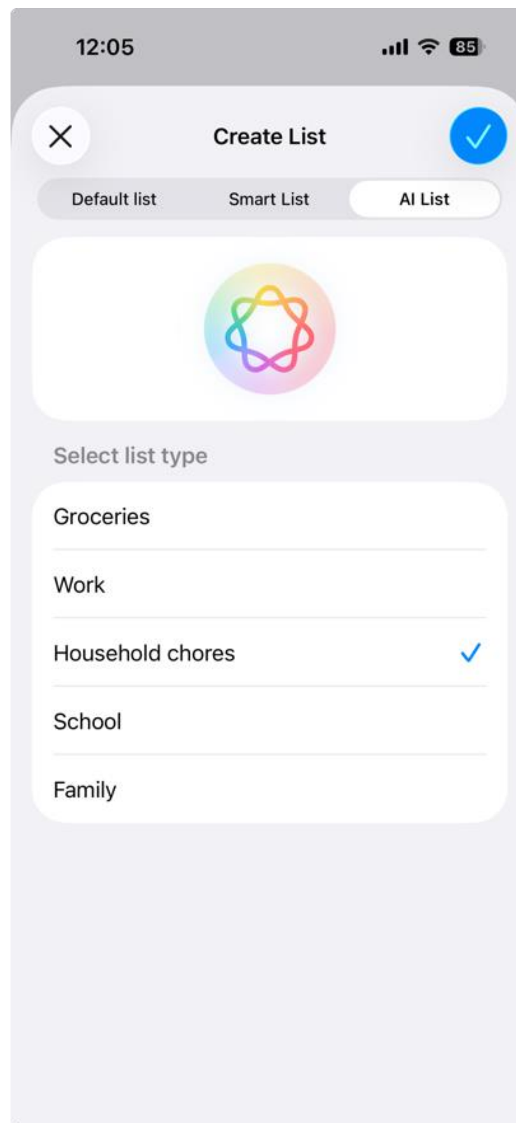

Рисунок 5.5 – Екран створення AI-списку

Для того щоб створити нагадування, користувач має натиснути на нижню праву кнопку, після чого відкриється форма створення нагадування у списку, встановленому в застосунку Reminders за замовчуванням (рис. 5.6). Користувач може ввести назву, нотатки, встановити дату нагадування і змінити список з нагадуваннями.

При натисканні на опцію з повтором користувач має декілька передвстановлених повторів та можливість створити свій, натиснувши варіант «Custom» (рис. 5.7).

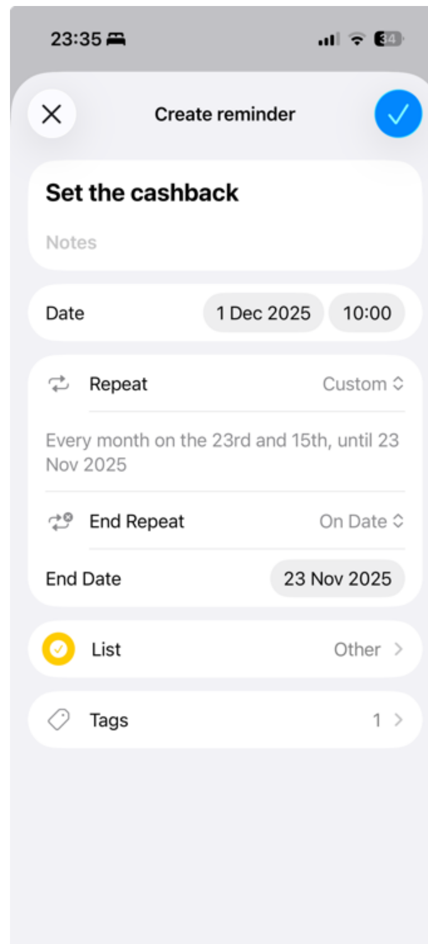

Рисунок 5.6 – Екран створення нагадування

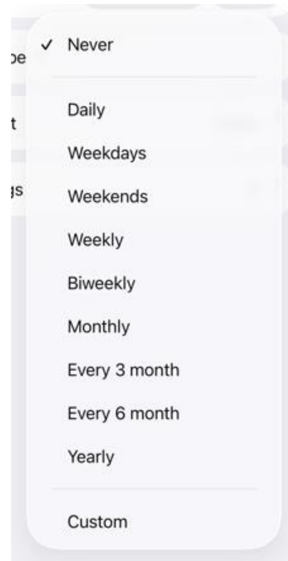

Рисунок 5.7 – Доступні повтори нагадування

Якщо користувач натиснув варіант «Custom», то відкривається форма створення власного повтору (рис. 5.8), де він може дуже детально

налаштувати його, обираючи з-поміж доступних варіантів частоти (рис. 5.9). Якщо обраний повтор не «Never», то користувач може обрати кінець повтору «Never», або після вказаної дати. Також користувач може додати теги до нагадування, обравши вже створені або створивши нові у формі, вказавши назву та колір тегу (рис. 5.10). Після того, як користувач ввів усі необхідні дані, він натискає кнопку підтвердження для збереження нагадування та повертається на головний екран.

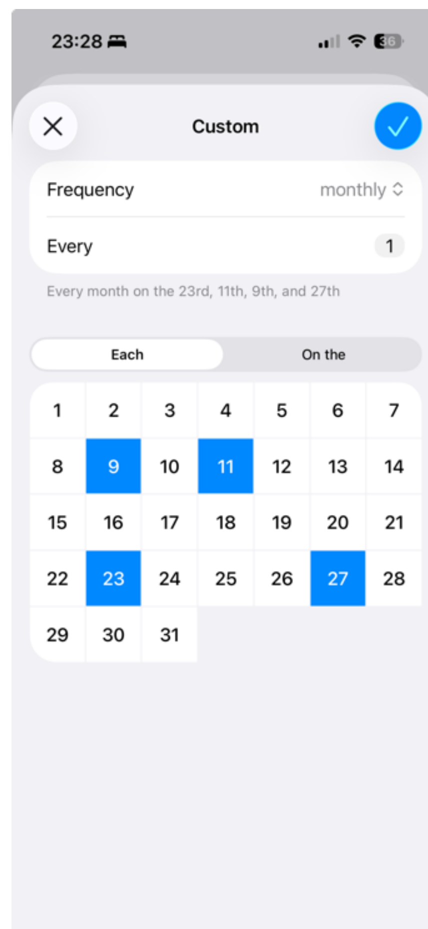

Рисунок 5.8 – Форма створення власного повтору

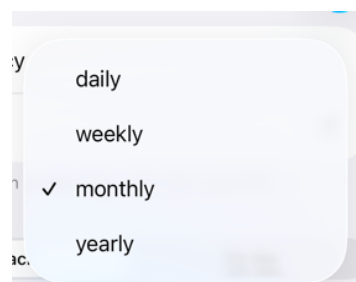

Рисунок 5.9 – Доступні варіанти частоти повтору

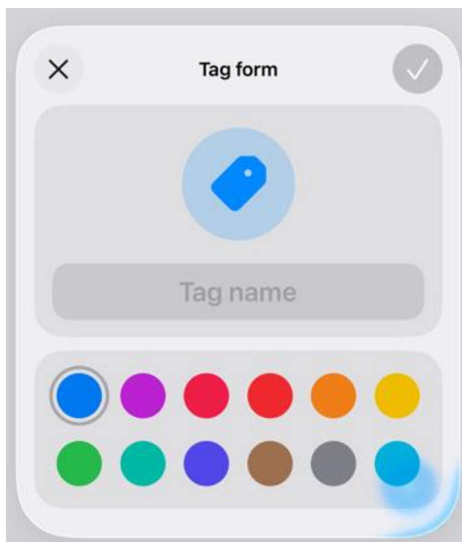

Рисунок 5.10 – Форма створення тегу

Для того, щоб перейти до екрану зі списком нагадувань (рис. 5.11), користувач має натиснути на список будь-якого типу, але кнопка створення нагадування буде лише у звичайному списку.

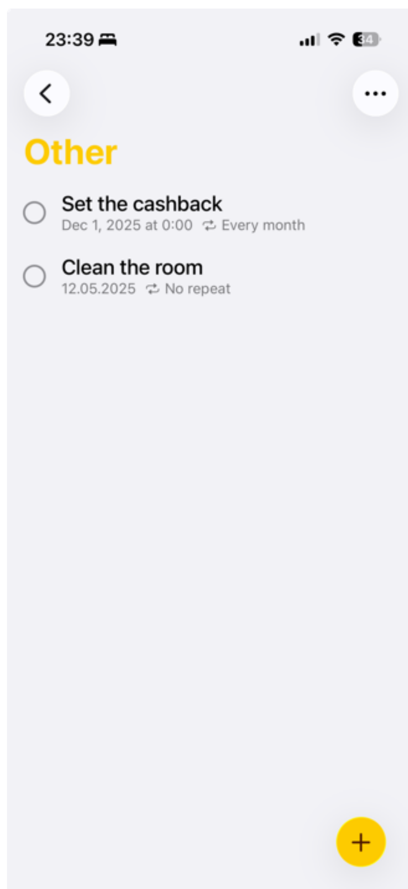

Рисунок 5.11 – Екран з нагадуваннями

Користувач може змінити налаштування відображення та сортування у списку, натиснувши на верхню праву кнопку і потрапивши до відповідного меню (рис. 5.12).

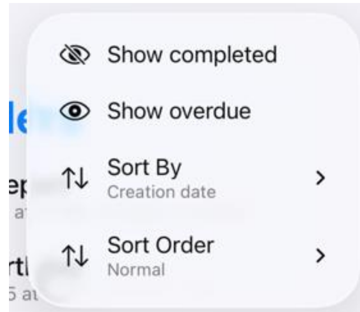

Рисунок 5.12 – Меню налаштування відображення

Для того, щоб відмітити нагадування як виконане, користувач має натиснути на круглу кнопку в об'єкті нагадування зліва та дочекатися відліку (рис. 5.13). Якщо користувач натисне на кнопку ще раз до того, як відлік закінчиться, нагадування не буде вважатися виконаним.

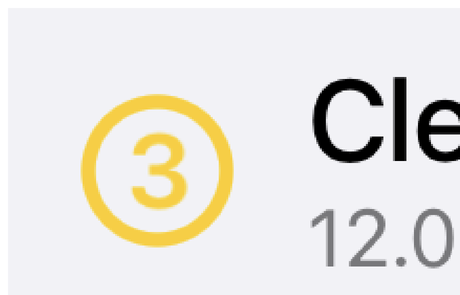

Рисунок 5.13 – Відлік до того як нагадування буде відмічене як виконане

5.3 Проведення тестування програмного продукту та аналіз отриманих результатів

Для перевірки якості створеного застосунку було проведено наступні тестування:

– тестування UI частини застосунку;
– функціональне тестування.

Для кожного тестування було створено чеклісти, які містять інформацію про пристрої, на яких проводилося тестування, перелік перевірок та результати цих перевірок.

Нижче на рис. 5.14 представлений чекліст функціонального тестування додатку.

Перевірка	iPhone 15 Pro (iOS 26.2)	iPhone 17 Pro (iOS 26.1)
Перевірити коректне створення звичайного списку	Пройдено	Пройдено
Перевірити коректне створення смарт списку	Пройдено	Пройдено
Перевірити коректне створення AI списку	Пройдено	Пройдено
Перевірити редагування списку	Пройдено	Пройдено
Перевірити видалення списку	Пройдено	Пройдено
Перевірити переходи між екранами	Пройдено	Пройдено
Перевірити синхронізацію нагадувань і списків з EventKit	Пройдено	Пройдено
Перевірити створення нагадування	Пройдено	Пройдено
Перевірити редагування нагадування	Пройдено	Пройдено
Перевірити роботу відмітки нагадування як виконаного	Пройдено	Пройдено
Перевірити видалення нагадування	Пройдено	Пройдено
Перевірити створення тегів	Пройдено	Пройдено
Перевірити фільтрацію нагадувань	Пройдено	Пройдено
Перевірити сортування нагадувань	Пройдено	Пройдено
Перевірити роботу нейромережі для AI списку	Пройдено	Пройдено

Рисунок 5.14 – Чекліст функціонального тестування

На рис 5.15 зображено чекліст тестування UI частини застосунку.

Перевірка	iPhone 15 Pro (iOS 26.2)	iPhone 17 Pro (iOS 26.1)
Перевірити коректне відображення кнопок, блоків меню, тощо	Пройдено	Пройдено
Перевірити нявність усіх сторінок	Пройдено	Пройдено
Перевірити коректне відображення шрифтів та стилів тексту	Пройдено	Пройдено
Перевірити коректне відображення сторінок згідно дизайну	Пройдено	Пройдено
Перевірити граматику та орфографію застосунку	Пройдено	Пройдено
Перевірити коректне відображення даних у формах	Пройдено	Пройдено
Перевірити коректне відображення анімацій і переходів	Пройдено	Пройдено
Перевірити коректне відображення модальних вікон	Пройдено	Пройдено

Рисунок 5.15 – Чекліст тестування UI частини застосунку

Проаналізувавши рис. 5.14 та 5.15 було зроблено висновок, що створений застосунок пройшов усі тест-кейси та його поведінка стабільна та передбачувана на різних девайсах.

5.4 Висновки до розділу

У даному розділі було наведено призначення програмного продукту та вимоги до виконання. Також було представлено опис експлуатації програмного продукту та проведено його тестування, а саме: верстки та функціональної частини програми. У результаті всі тести були успішно пройдені і жоден із пунктів не позначено як «Не пройдено».

ВИСНОВКИ

Було розібрано найпопулярніші аналоги програм, таких як «Reminders», «Google Tasks» та «Microsoft To Do», та порівняно їхні функціональні можливості з функціональними можливостями «Reminders Plus». Порівняльний аналіз показав, що розроблюваний застосунок є актуальним та доцільним на фоні конкурентів.

Було здійснено огляд основних моделей класифікації тексту (MaxEnt, CRF та BERT) та проведено їхнє порівняння для вибору моделі. В результаті базовою моделлю було вирішено використовувати BERT через її спроможність розуміти контекст та високу точність. Для навчання та тестування моделі для класифікації нагадувань за темою було створено власний датасет на основі датасету «MS-LaTTE» з GitHub репозиторію.

Для розробки програмного продукту було визначено мову програмування Swift і фреймворк SwiftUI, а IDE було обрано Xcode. Для організації збереження даних було обрано базу даних SwiftData SQL-типу.

Було організовано структуру бази даних, описано алгоритми роботи програми та здійснено розробку ключових класів і методів.

Було перевірено коректність роботи верстки та функціональної частини застосунку. У результаті всі тести були успішно пройдені.

Після всіх етапів було створено програмний продукт «Reminders Plus», який надає користувачу потужну категоризацію нагадувань за різними їх ознаками як за допомогою алгоритмів фільтрації, так і штучного інтелекту.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. The Power of Smart Reminders [Electronic resource]. – Access mode: https://hyteams.app/blog/the-power-of-smart-reminders-never-miss-a-deadline.

2. Reminders Eliminate Age-Related Declines in Prospective Memory [Electronic resource]. – Access mode: https://pmc.ncbi.nlm.nih.gov/articles/PMC11781985/?utm_source=chatgpt.com.

3. What is reminder [Electronic resource]. – Access mode: https://uk.wikipedia.org/wiki/Нагадування_(Apple).

4. Description of Reminders features [Electronic resource]. – Access mode: https://support.apple.com/en-us/102484#:~:text=With%20the%20Reminders%20app%20on,based%20on%20time%20and%20location.

5. Reminders [Electronic resource]. – Access mode: https://apps.apple.com/ua/app/reminders/id1108187841.

6. Description of Google tasks features [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/Google_Tasks.

7. Google Tasks [Electronic resource]. – Access mode: https://apps.apple.com/ua/app/google-tasks-get-things-done/id1353634006.

8. Description of Microsoft To Do [Electronic resource]. – Access mode: https://uk.wikipedia.org/wiki/Microsoft_To_Do.

9. Microsoft To Do [Electronic resource]. – Access mode: https://apps.apple.com/ua/app/microsoft-to-do/id1212616790.

10. Introduction to Maximum Entropy in Computer Science [Electronic resource]. – Access mode: https://www.sciencedirect.com/topics/computer-science/maximum-entropy.

11. Conditional Random Field [Electronic resource]. – Access mode: https://www.sciencedirect.com/topics/computer-science/conditional-random-field.

12. Alberto Citron. Understanding How Conditional Random Fields (CRFs) Work for Labelling Text: A Simple Step-by-Step Guide [Electronic

resource]. – Access mode: https://medium.com/@albertocitron88/understanding-how-conditional-random-fields-crfs-work-for-labelling-text-a-simple-step-by-step-a7ddb0974c26.

13. Effective text classification using BERT, MTM LSTM, and DT [Electronic resource]. – Access mode: https://www.sciencedirect.com/science/article/abs/pii/S0169023X24000302.

14. Maleesha De Silva. Text Classification using BERT: A Complete Guide [Electronic resource]. – Access mode: https://medium.com/@maleeshadesilva21/text-classification-using-bert-an-complete-guide-7be30a8285c2.

15. MS-LaTTE Dataset [Electronic resource]. – Access mode: https://github.com/microsoft/MS-LaTTE.

16. Paul Hudson. What is SwiftUI? [Electronic resource]. – Access mode: https://www.hackingwithswift.com/quick-start/swiftui/what-is-swiftui.

17. Voislav Stojkoski. The Power of SwiftUI: A Guide to its Benefits Over UIKit? Electronic resource]. – Access mode: https://nionit.com/the-power-of-swiftui-a-guide-to-its-benefits-over-uikit.

18. Jay Tillu. What is Swift Language? [Electronic resource]. – Access mode: https://jaytillu.medium.com/what-is-swift-language-09a5d56988da.

19. What is Xcode: Features, Installation, Uses, Advantages and Limitations [Electronic resource]. – Access mode: https://www.browserstack.com/guide/what-is-xcode.

20. Surabhi Kumari. CreateML: A Comprehensive Guide to Building Machine Learning Models Without Writing Code [Electronic resource]. – Access mode: https://medium.com/@ksurabhi0502/createml-a-comprehensive-guide-to-building-machine-learning-models-without-writing-code-2ce141468346.

21. Relational Database: Definition, Examples, and More [Electronic resource]. – Access mode: https://www.coursera.org/articles/relational-database.

22. What is NoSQL? [Electronic resource]. – Access mode: https://aws.amazon.com/nosql/.

ДОДАТОК А

Технічне завдання

А.1 Підстави до розробки

Виконання дипломної кваліфікаційної роботи здійснюється за темою «Дослідження та програмна реалізація розумного планувальника для iOS» відповідно до наказу №447 від 30 вересня 2025 р. за Національним університетом «Запорізька політехніка».

А.2 Призначення розробки

Застосунок призначений для того, щоб користувачі могли зручно створювати та редагувати нагадування та мати потужну категоризацію нагадувань за різними ознаками як за допомогою алгоритмів фільтрації, так і штучного інтелекту.

А.3 Вимоги до програмного продукту

А.3.1 Вимоги до функціональних характеристик

Розроблюваний мобільний застосунок має наступні функціональні вимоги:

– повна синхронізація списків та нагадувань з Apple Reminders;

– створення нагадування яке повинно мати повний набір параметрів нагадування з Apple Reminders та прикріплені теги;

– можливість видаляти та редагувати створені нагадування;

– створення списку нагадування яке має повний набір параметрів списку Apple Reminders, іконка та збережені фільтри;

– створення смарт-списку з потужним механізмом фільтрації за датою, часом, тегами та списками;

– створення AI-списку де нейромережа автоматично відбирає нагадування за змістом;

– можливість видаляти та редагувати створені списки;

– можливість сортування нагадувань за назвою, датою створення, пріоритетом, датою нагадування та власноруч;
– можливість налаштування сортування списку.

А.3.2 Вимоги до надійності

Застосунок повинен розпізнавати ситуації, коли користувач не дає дозвіл до системних нагадувань. Також при створенні чи редагуванні нагадування чи списку нагадувань, коли користувач вводить або невалідні дані або не вводить їх зовсім.

А.3.3 Вимоги до складу та параметрів технічних засобів

Для забезпечення нормальної роботи застосунку необхідні такі мінімальні апаратні забезпечення:
– операційна система iOS версією не нижче 17.0;
– об'єм вільної пам'яті на телефоні не менше 100 Мб;
– доступ до системних нагадувань.

А.4 Порядок контролю та приймання

Дипломна робота має узгоджуватися з вимогами, визначеними у пункті А.3, після чого обов’язкового перевіряється керівником. Процедура її прийняття здійснюється відповідно до вимогам, встановлених для робіт магістерського рівня.

ДОДАТОК Б

Текст програми

Б.1 Текст файлу EventKitService.swift

```
//
//  EventKitSyncService.swift
//  Reminders
//
//  Created by Максим Грищенков on 16.10.2025.
//

internal import Combine
@preconcurrency internal import EventKit
import SwiftData
import SwiftUI
import UIKit

class EventKitSyncService: ObservableObject {
    private(set) var eventStore = EKEventStore()
    private let modelContext: ModelContext
    var mlService: ReminderListClassifier?
    private(set) static var shared: EventKitSyncService?
    private(set) var eventStoreHandler: EventStoreHandler?
    @Published var isSyncing = false
    @Published var reminderLists: [EKCalendarType: [ReminderList]] =
[:]

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

        Self.shared = self
    }

    func prepareService() async {
        self.eventStoreHandler = EventStoreHandler(
            eventStore: EKEventStore(),
            modelContainer: modelContext.container,
            setSyncState: {
                self.isSyncing = $0
            }
        )
        self.mlService = ReminderListClassifier(
            container: modelContext.container
        )
        await self.eventStoreHandler?.prepareService()
```

```
        }


        // Создание/обновление ReminderList из EKCalendar
        private func syncCalendar(_ ekCalendar: EKCalendar)
            -> ReminderList
        {
            let calendarId = ekCalendar.calendarIdentifier
            let descriptor = FetchDescriptor<ReminderList>(
                predicate:    #Predicate    {    $0.calendarIdentifier    ==
calendarId }
            )

            if let existing = try? modelContext.fetch(descriptor).first {
                existing.ekCalendar = ekCalendar
                return existing
            }

            let list = ReminderList(
                calendarIdentifier: calendarId,
                title: ekCalendar.title,
                systemImage: "list.bullet",
                cgColor: ekCalendar.cgColor,
                filterSettings: SortSettings()
            )
            modelContext.insert(list)
            list.ekCalendar = ekCalendar
            return list
        }

        func syncCalendar(_ list: ReminderList){
            let calendar = eventStore.calendar(
                withIdentifier: list.calendarIdentifier
            )
            list.ekCalendar = calendar
        }



        func syncReminder(_ reminder: Reminder)  {
            let ekIdentifier = reminder.calendarItemIdentifier
            let ekReminder =
```

```
                eventStore.calendarItem(withIdentifier:  ekIdentifier)  as?
EKReminder
            reminder.ekReminder = ekReminder
        }



        func updateCalendar(_ list: ReminderList) async {
            // Ensure we have access to reminders
            do {
                let          granted          =          try          await
eventStore.requestFullAccessToReminders()
                guard granted else { return }
            } catch {
                return
            }

            // Try to resolve existing EKCalendar by identifier
            let calendarIdentifier = list.calendarIdentifier
            var calendar: EKCalendar?

            if eventStore.calendar(
                withIdentifier: calendarIdentifier
            ) != nil {
                calendar = list.ekCalendar
            } else if let newCalendar = list.ekCalendar {
                calendar = newCalendar
                let sources = eventStore.sources
                if let iCloud = sources.first(where: {
                    $0.sourceType == .calDAV
                        && $0.title.lowercased().contains("icloud")
                }) {
                    calendar?.source = iCloud
                } else if let local = sources.first(where: {
                    $0.sourceType == .local
                }) {
                    calendar?.source = local
                } else if let first = sources.first {
                    calendar?.source = first
                }
            }

            guard let calendar else { return }
            do {
```

```
                try eventStore.saveCalendar(calendar, commit: true)
            } catch {
                return
            }

            // Update the ReminderList linkage and persist SwiftData
            if list.calendarIdentifier != calendar.calendarIdentifier {
                // Keep our model in sync with the new identifier if it
changed on create
                list.calendarIdentifier = calendar.calendarIdentifier
                modelContext.insert(list)
            }

            if modelContext.hasChanges {
                try? modelContext.save()
            }
        }

        func deleteCalendar(_ calendarIdentifier: String) async throws {

            if let calendar = eventStore.calendar(
                withIdentifier: calendarIdentifier
            ) {
                try eventStore.removeCalendar(calendar, commit: true)
            }

        }

        fileprivate func checkPermission() async -> Bool {
            do {
                let granted = try await
eventStore.requestFullAccessToReminders()
                return granted
            } catch {
                return false
            }
        }


        func updateReminder(_ reminder: Reminder, _ ekValue: EKReminder,
_ tags: [Tag]) async throws {
            guard await checkPermission() else { return }
```

```
                reminder.ekReminder = ekValue
                let calendarItemIdentifier = reminder.calendarItemIdentifier

                // Попробуем получить существующий EKReminder
                var ekReminder =
                    eventStore.calendarItem(withIdentifier:
calendarItemIdentifier)
                    as? EKReminder

                if ekReminder == nil {
                    // Создаём новый EKReminder и устанавливаем календарь
                    let newEK = EKReminder(eventStore: eventStore)

                    // Подбираем календарь по идентификатору списка
                    if let ekCalendar = eventStore.calendar(
                        withIdentifier: reminder.calendarIdentifier
                    ) {
                        newEK.calendar = ekCalendar
                    } else {
                        //   Вариант:   выбрать   дефолтный   календарь   для
напоминаний
                        if let defaultCalendar =
                            eventStore.defaultCalendarForNewReminders()
                        {
                            newEK.calendar = defaultCalendar
                        }
                        else {
                            // Если календаря нет — дальше сохранить нельзя
                            throw NSError(
                                domain: "EventKitSyncService",
                                code: 1,
                                userInfo: [
                                    NSLocalizedDescriptionKey:
                                        "No  reminder  calendar  available  to
save EKReminder"
                                ]
                            )
                        }
                    }

                    ekReminder = newEK
                }
```

```
            guard let ekReminder else { return }

            // Переносим поля из вашей модели (reminder.ekReminder может
быть источником значений)
            let source = reminder.ekReminder

            ekReminder.title = source?.title ?? ekReminder.title
            ekReminder.notes = source?.notes ?? ekReminder.notes
            ekReminder.dueDateComponents = source?.dueDateComponents
            ekReminder.priority = source?.priority ?? 0
            ekReminder.isCompleted = source?.isCompleted ?? false
            ekReminder.completionDate = source?.completionDate
            ekReminder.startDateComponents = source?.startDateComponents
            ekReminder.alarms = source?.alarms ?? []

            // ВАЖНО: заменяем правила повторения целиком, чтобы не
дублировать
            if let rules = source?.recurrenceRules, !rules.isEmpty {
                ekReminder.recurrenceRules = rules
            } else {
                ekReminder.recurrenceRules = nil
            }
            await eventStoreHandler?.pauseSync(true)
            try eventStore.save(ekReminder, commit: true)
            // При необходимости синхронизируем локальную модель
            // Например: обновить lastModifiedDate локально
            reminder.lastModifiedDate = ekReminder.lastModifiedDate
            reminder.calendarIdentifier =
ekReminder.calendar.calendarIdentifier
            if reminder.calendarItemIdentifier !=
ekReminder.calendarItemIdentifier {
                reminder.calendarItemIdentifier =
ekReminder.calendarItemIdentifier
                modelContext.insert(reminder)
            }

            if modelContext.hasChanges {
                try modelContext.save()
            }
            await eventStoreHandler?.pauseSync(false)
            return
        }
```

```
        func completeReminder(_ reminder: Reminder) async throws{
            guard await checkPermission() else { return }
            guard let ekReminder = eventStore.calendarItem(withIdentifier:
reminder.calendarItemIdentifier) as? EKReminder else {
                return
            }
            ekReminder.isCompleted = true
            ekReminder.completionDate = Date()

            await eventStoreHandler?.pauseSync(true)
            try eventStore.save(ekReminder, commit: true)

            reminder.isCompleted = true
            reminder.lastModifiedDate = ekReminder.lastModifiedDate

            if modelContext.hasChanges {
                try modelContext.save()
            }
            await eventStoreHandler?.pauseSync(false)
        }

        private func cleanupOrphanedReminders() {
            guard
                let allReminders = try? modelContext.fetch(
                    FetchDescriptor<Reminder>()
                )
            else {
                return
            }

            for reminder in allReminders where
!reminder.calendarItemIdentifier.isEmpty {
                if eventStore.calendarItem(withIdentifier:
reminder.calendarItemIdentifier)
                    == nil
                {
                    modelContext.delete(reminder)
                }
            }
        }

        func getDefaultList() -> ReminderList? {
            let calendar = eventStore.defaultCalendarForNewReminders()
```

```
            do {

                let list = syncCalendar(calendar!)

                if modelContext.hasChanges {
                    try modelContext.save()
                }
                return list
            } catch {
                return nil
            }
        }

        func getListByIdentifier(_ identifier: String) -> ReminderList? {
            if let ekCalendar = eventStore.calendar(
                withIdentifier: identifier
            ) {
                return syncCalendar(ekCalendar)
            }
            return nil
        }

        func requestAccessAndSync() async -> Bool {
            do {
                let          granted          =          try          await
eventStore.requestFullAccessToReminders()
                if granted {

                    await prepareService()
                }
                return granted
            } catch {
                return false
            }
        }
    }
```

Б.2 Текст файлу EventStoreHandler.swift

```
//
//  EventStoreHandler.swift
```

```
//  Reminders
//
//  Created by Максим Грищенков on 17.11.2025.
//
internal import EventKit
import SwiftData
import SwiftUI


public actor EventStoreHandler : ModelActor{
    public let modelContainer: ModelContainer

    public let modelExecutor: any ModelExecutor

    var eventStore: EKEventStore
    private var currentPauses: Int = 0
    private var isPaused: Bool {
        currentPauses > 0
    }
    var mlService: ReminderListClassifier?
    private var context: ModelContext { modelExecutor.modelContext }
    private var observers: [NSObjectProtocol] = []
    private let setSyncState: @MainActor (Bool) -> Void


    init(
        eventStore: EKEventStore,
        modelContainer: ModelContainer,
        setSyncState: @escaping @MainActor (Bool) -> Void
    ) {
        self.eventStore = eventStore
        self.setSyncState = setSyncState
        self.modelContainer = modelContainer
        let context = ModelContext(modelContainer)
        modelExecutor = DefaultSerialModelExecutor(
            modelContext: context
        )

    }

    func pauseSync(_ isPaused: Bool) {
        if(isPaused){
            currentPauses += 1
```

```
        }
        else{
            if(currentPauses > 0 ){
                currentPauses -= 1
            }
        }
    }

    func prepareService() async {
        self.mlService = ReminderListClassifier(
            container: modelContainer
        )
        setupNotificationObservers()
        await handleEventStoreChange()
    }

    func requestAccessAndSync() async -> Bool {
        do {
            let granted = try await
eventStore.requestFullAccessToReminders()
            if granted {

                await prepareService()
            }
            return granted
        } catch {
            return false
        }
    }

    func syncAllReminders() async {

        await setSyncState(true)
    }

    private func setupNotificationObservers() {
        // Отслеживание изменений в EventKit
        let observer = NotificationCenter.default.addObserver(
            forName: .EKEventStoreChanged,
            object: eventStore,
            queue: .main
        ) { [weak self] notification in
            Task { [weak self] in
```

```
                await self?.handleEventStoreChange()
            }
        }
        observers.append(observer)
    }

    private func handleEventStoreChange() async {
        while isPaused {
            do{
                try await Task.sleep(nanoseconds: 200_000_000)
            }
            catch{}
        }
        await setSyncState(true)

        await handleCalendarChanges()
        await handleReminders()
        do{
            if modelContext.hasChanges{
                try modelContext.save()}
        }
        catch{
        }
        await setSyncState(false)
    }

    func handleCalendarChanges() async {
        var newReminderLists: [EKCalendarType: [ReminderList]] = [:]
        let currentEKCalendars = eventStore.calendars(for: .reminder)
        let currentCalendarIDs = Set(
            currentEKCalendars.map {
                return $0.calendarIdentifier
            }
        )

        guard
            let swiftDataLists = try? modelContext.fetch(
                FetchDescriptor<ReminderList>()
            )

        else {
            await setSyncState(false)
            return
```

```
                }

                let existingCalendarIDs = Set(
                    swiftDataLists.map { $0.calendarIdentifier }
                )

                for list in swiftDataLists {
                    if let ekCalendar = currentEKCalendars.first(where: {
                        $0.calendarIdentifier == list.calendarIdentifier
                    }) {

                        let titleChanged =
                            list.ekCalendar?.title != ekCalendar.title
                        let colorChanged =
                            list.ekCalendar?.cgColor != ekCalendar.cgColor

                        if titleChanged || colorChanged {
                            list.ekCalendar = ekCalendar
                            if newReminderLists[ekCalendar.type] == nil {
                                newReminderLists[ekCalendar.type] = [list]
                            } else {

newReminderLists[ekCalendar.type]?.append(list)
                            }

                        }

                    } else {

                        modelContext.delete(list)

                    }
                }

                let newCalendarIDs = await Task.detached {
                    currentCalendarIDs.subtracting(existingCalendarIDs)
                }.value
                for id in newCalendarIDs {
                    if let ekCalendar = currentEKCalendars.first(where: {
                        $0.calendarIdentifier == id
                    }) {
                        if newReminderLists[ekCalendar.type] == nil {
                            newReminderLists[ekCalendar.type] = [
```

```
                        syncCalendar(ekCalendar)
                    ]
                } else {
                    newReminderLists[ekCalendar.type]?.append(
                        syncCalendar(ekCalendar)
                    )
                }

            }
        }

    }

    private func handleReminders() async {
        let currentEKReminderIDs =
            await fetchAllEKReminderIDs()

        Task {
            guard
                let swiftDataReminders = try? modelContext.fetch(
                    FetchDescriptor<Reminder>()
                )
            else {
                await setSyncState(false)
                return
            }
            let existingIDs = Set(
                swiftDataReminders.map {
                    $0.calendarItemIdentifier
                }
            )

            for reminder in swiftDataReminders {

                let ekIdentifier = reminder.calendarItemIdentifier
                let lastModifiedDate = reminder.lastModifiedDate

                if let ekReminder = eventStore.calendarItem(
                    withIdentifier: ekIdentifier
                ) as? EKReminder {

                    if lastModifiedDate != ekReminder.lastModifiedDate
{
```

```
                Task {
                    _ = self.syncReminder(
                        ekReminder
                    )
                }

            }
        } else {

            do {

                try modelContext
                    .delete(
                        model: Reminder.self,
                        where: #Predicate {
                            $0.calendarItemIdentifier         ==
ekIdentifier

                        }
                    )

            } catch {

            }

        }

    }
    let newIDs =
        Set(currentEKReminderIDs).subtracting(
            existingIDs
        )

    for id in newIDs {
        if let ekReminder = self.eventStore.calendarItem(
            withIdentifier: id
        )
            as? EKReminder
        {
            _ = self.syncReminder(ekReminder)

        }

    }
```

```
        }

    }

    func fetchAllEKReminderIDs() async -> [String] {
        let calendars = eventStore.calendars(for: .reminder)
        let    predicate    =    eventStore.predicateForReminders(in:
calendars)
        return await withCheckedContinuation { continuation in
            eventStore.fetchReminders(matching: predicate) { reminders
in
                let ids = (reminders ?? []).compactMap {
                    reminder -> String? in
                    let id = reminder.calendarItemIdentifier
                    guard !id.isEmpty else {
                        return nil
                    }
                    return id
                }
                continuation.resume(returning: ids)
            }
        }
    }

    func syncCalendar(_ ekCalendar: EKCalendar)
        -> ReminderList
    {
        let calendarId = ekCalendar.calendarIdentifier
        let descriptor = FetchDescriptor<ReminderList>(
            predicate:   #Predicate   {   $0.calendarIdentifier   ==
calendarId }
        )

        if let existing = try? modelContext.fetch(descriptor).first {
            existing.ekCalendar = ekCalendar
            return existing
        }

        let list = ReminderList(
            calendarIdentifier: calendarId,
            title: ekCalendar.title,
            systemImage: "list.bullet",
            cgColor: ekCalendar.cgColor,
```

```
                filterSettings: SortSettings.init(
                    sortByData: nil,
                    sortOrderData: nil
                )
            )
            modelContext.insert(list)
            list.ekCalendar = ekCalendar
            return list
        }

        func syncReminder(_ ekReminder: EKReminder) -> Reminder {
            let id = ekReminder.calendarItemIdentifier

            let descriptor = FetchDescriptor<Reminder>(
                predicate: #Predicate { $0.calendarItemIdentifier == id }
            )

            if let existing = try? modelContext.fetch(descriptor).first {
                existing.ekReminder = ekReminder
                existing.lastModifiedDate = ekReminder.lastModifiedDate
                let newTitle = ekReminder.title ?? ""
                if existing.title != newTitle && !existing.isCompleted {
                    existing.title = newTitle

                    Task {
                        do {
                            try await mlService?.updateAILists(
                                ekReminder: ekReminder
                            )
                        } catch {
                        }
                    }
                }

                return existing
            }
            Task {
                do {
                    try     await     mlService?.updateAILists(ekReminder:
ekReminder)
                } catch {
                }
            }
```

```
            let reminder = Reminder(
                ekIdentifier: id,
                listIdentifier: ekReminder.calendar.calendarIdentifier,
                ekReminder: ekReminder,
                lastModifiedDate: ekReminder.lastModifiedDate
            )

            modelContext.insert(reminder)
            return reminder
        }

        func getDefaultList() -> ReminderList? {
            let calendar = eventStore.defaultCalendarForNewReminders()
            do {

                let list = syncCalendar(calendar!)

                if modelContext.hasChanges {
                    try modelContext.save()
                }
                return list
            } catch {
                return nil
            }
        }

        func getEKReminderByIDentifier(_ identifier: String) async throws
->  EKReminder? {

            let granted = try await eventStore.requestFullAccessToEvents()
            if !granted {
                return nil
            }
            return eventStore.calendarItem(withIdentifier: identifier) as?
EKReminder
        }

        func getListByIdentifier(_ identifier: String) async throws ->
ReminderList? {
            let granted = try await eventStore.requestFullAccessToEvents()
            if !granted {
                return nil
```

```
            }
            if let ekCalendar = eventStore.calendar(
                withIdentifier: identifier
            ) {
                return syncCalendar(ekCalendar)
            }
            return nil
        }

    }
```

Б.3 Текст файлу MlEmbeddingService.swift

```
//  MLEmbeddingService.swift
//  Reminders
//
//  Created by Максим Грищенков on 16.10.2025.
//

import Foundation
import NaturalLanguage
import CoreML
import SwiftData
internal import EventKit


actor ReminderListClassifier {
    private let context: ModelContext
    private let nlModel: NLModel

    init?(
        modelName: String = "PairwiseReminderCategoryModel",
        in bundle: Bundle = .main,
        container: ModelContainer
    ) {
        guard let url = bundle.url(forResource: modelName, withExtension: "mlmodelc") else {
            assertionFailure("Resource \(modelName).mlmodelc not found in bundle")
            return nil
        }
```

```
            do {
                let model = try NLModel(contentsOf: url)
                self.nlModel = model
                self.context = ModelContext(container)
            } catch {
                assertionFailure("Failed to load NLModel: \(error)")
                return nil
            }
        }


        func updateAILists(ekReminder: EKReminder) async throws {
            let                 aiLists                 =                 try
context.fetch(FetchDescriptor<ReminderAIList>())
            let probsForCategories = nlModel.predictedLabelHypotheses(
                for: ekReminder.title ?? "",
                maximumCount: 3
            )
            for list in aiLists{
                let labelRawValue = list.label.rawValue
                if let probability =  probsForCategories[labelRawValue],
probability >= 0.5{
                    if
!list.remindersId.contains(ekReminder.calendarItemIdentifier){

list.remindersId.append(ekReminder.calendarItemIdentifier)
                    }
                    break
                }
                    let index = list.remindersId
                        .firstIndex(of: ekReminder.calendarItemIdentifier)
                if index != nil {
                    list.remindersId.remove(at: index!)
                }

            }

        }

        func updateAIList(_ label: AIListLabel) async throws -> [String] {
            let reminders = try context.fetch(FetchDescriptor<Reminder>(
                predicate: #Predicate<Reminder> { $0.isCompleted == false
}
```

```
            ))
            let remindersTitles: [String] = reminders.map { $0.title }
            var iterator : Int = 0
            let probs: [[String : Double]] = nlModel
                .predictedLabelHypotheses(
                    forTokens: remindersTitles,
                    maximumCount: 3
                )
            let res = probs.compactMap{
                    dict in
                    let probability: Double? = dict[label.rawValue]
                    if probability != nil && probability! >= 0.5 {
                        iterator += 1
                        return             reminders[iterator              -
1].calendarItemIdentifier
                    }
                    iterator += 1
                    return nil
                }
            return res
        }
    }
```

Б.4 Текст файлу FilterQueryExtension.swift

```
    //
    //  SmartListQueryExtension.swift
    //  Reminders
    //
    //  Created by Максим Грищенков on 12.11.2025.
    //

    import Foundation
    import SwiftData

    extension FilterQuery {
        ///  Builds  a  combined  Predicate<Reminder>  from  the  smart  list
queries.
        /// Returns `nil` when there are no effective filters so the query
fetches all reminders.
```

```
        static func buildPredicate(from smartList: ReminderSmartList) ->
Predicate<
            Reminder
        >? {
            // Collect individual predicates (skipping any `.off` rules)
            var parts: [Predicate<Reminder>] = []
            let notCompleted = #Predicate<Reminder> { !$0.isCompleted }
            let containsCompletedRule: Bool = smartList.queries.contains {
q in
                if case .special(let s) = q, s == .completed { return true
}
                return false
            }

            for query in smartList.queries {
                if let p = predicate(for: query) {
                    parts.append(p)
                }
            }
            guard let first = parts.first else { return notCompleted }

            if smartList.matchingRule == .all {
                return parts.dropFirst().reduce(first) { acc, next in
                    #Predicate<Reminder> { r in
                        acc
                            .evaluate(r) && next
                            .evaluate(r) && notCompleted
                            .evaluate(r)
                    }
                }
            } else {
                return parts.dropFirst().reduce(first) { acc, next in
                    #Predicate<Reminder> { r in
                        acc.evaluate(r) || next.evaluate(r) &&
notCompleted.evaluate(r)
                    }
                }
            }
        }

        /// Builds an optional predicate for a single smart list query
item.
```

```
            private   static   func   predicate(for   query:   FilterQuery)   ->
Predicate<
                Reminder
            >? {

                switch query {
                case .tags(let tagRule):
                    switch tagRule {
                    case .off:
                        return nil
                    case .anyTag:
                        return #Predicate<Reminder> { !$0.tags.isEmpty }
                    case .allSelectedTags(let selTags):
                        return #Predicate<Reminder> { v in
                            v.tags.allSatisfy         {          tag          in
selTags.contains(tag.id) }
                        }
                    case .anySelectedTags(let selTags):
                        return #Predicate<Reminder> { v in
                            v.tags.contains  {  tag  in  selTags.contains(tag.id)
}

                        }
                    case .noTags:
                        return #Predicate<Reminder> { v in v.tags.isEmpty }
                    }


                case .date(let dateRule):
                    let defaultValue = Date()
                    switch dateRule {
                    case .off:
                        return nil
                    case .any:
                        return #Predicate<Reminder> { $0.dueDate != nil }
                    case .today:
                        let calendar = Calendar.current
                        let startOfDay = calendar.startOfDay(for: Date())
                        let endOfDay = calendar.date(
                            byAdding: .day,
                            value: 1,
                            to: startOfDay
                        )!
```

```
                return #Predicate<Reminder> {
                    $0.dueDate != nil
                    && ($0.dueDate ?? defaultValue) >= startOfDay
                    && ($0.dueDate ?? defaultValue) < endOfDay
                }
            case .onDate(let date):
                return #Predicate<Reminder> {
                    $0.dueDate != nil && $0.dueDate == date
                }
            case .beforeDate(let date):
                return #Predicate<Reminder> {
                    $0.dueDate != nil && ($0.dueDate ?? defaultValue)
< date
                }
            case .afterDate(let date):
                return #Predicate<Reminder> {
                    $0.dueDate != nil && ($0.dueDate ?? defaultValue)
> date
                }
            case .specifiedRange(let date, let date2):
                return #Predicate<Reminder> {
                    $0.dueDate != nil && ($0.dueDate ?? defaultValue)
< date
                    && ($0.dueDate ?? defaultValue) > date2
                }
            case .relativeRange(
                let relativeRange,
                let amount,
                let timeSpaceRange
            ):
                let now = Date()
                let direction: Int = {
                    switch relativeRange {
                    case .inTheNext: return 1
                    case .inThePast: return -1
                    }
                }()

                let boundary: Date = {
                    let value = direction * amount
                    switch timeSpaceRange {
                    case .hours:
```

```
                                return Calendar.current.date(byAdding: .hour,
value: value, to: now) ?? now
                            case .days:
                                return Calendar.current.date(byAdding: .day,
value: value, to: now) ?? now
                            case .weeks:
                                return Calendar.current.date(byAdding:
.weekOfYear, value: value, to: now) ?? now
                            case .months:
                                return Calendar.current.date(byAdding: .month,
value: value, to: now) ?? now
                            case .years:
                                return Calendar.current.date(byAdding: .year,
value: value, to: now) ?? now
                            }
                        }()
                        let defaultValue = now
                        switch relativeRange {
                        case .inTheNext:
                            return #Predicate<Reminder> {
                                $0.dueDate != nil && ($0.dueDate ??
defaultValue) >= now && ($0.dueDate ?? defaultValue) < boundary
                            }
                        case .inThePast:
                            return #Predicate<Reminder> {
                                $0.dueDate != nil && ($0.dueDate ??
defaultValue) > boundary && ($0.dueDate ?? defaultValue) <= now
                            }
                        }

                    }

                case .time(let timeRule):
                    let defaultValue = -1
                    switch timeRule {
                    case .off:
                        return nil
                    case .any:
                        return #Predicate<Reminder> {
                            $0.dueHour != nil
                        }
                    case .morning:
                        return #Predicate<Reminder> {
```

```
                        $0.dueHour != nil && ($0.dueHour ?? defaultValue)
>= 5
                        && ($0.dueHour ?? defaultValue) <= 11
                    }
                case .afternoon:
                    return #Predicate<Reminder> {
                        $0.dueHour != nil && ($0.dueHour ?? defaultValue)
>= 12
                        && ($0.dueHour ?? defaultValue) <= 16
                    }
                case .evening:
                    return #Predicate<Reminder> {
                        $0.dueHour != nil && ($0.dueHour ?? defaultValue)
>= 17
                        && ($0.dueHour ?? defaultValue) <= 21
                    }
                case .night:
                    return #Predicate<Reminder> {
                        $0.dueHour != nil && ($0.dueHour ?? defaultValue)
>= 22
                    }
                case .noTime:
                    return #Predicate<Reminder> {
                        $0.dueHour == nil
                    }
                }

            case .lists(let listRule):
                switch listRule {
                case .off:
                    return nil
                case .include(let array):
                    return #Predicate<Reminder> {
                        array.contains($0.calendarIdentifier)
                    }
                case .exclude(let array):
                    return #Predicate<Reminder> {
                        !array.contains($0.calendarIdentifier)
                    }
                }
            case .special(let specialRule):
                switch specialRule {
                case .completed:
```

```
                return #Predicate<Reminder> {
                    $0.isCompleted
                }
            }

        }
    }
}
```

Б.5 Текст файлу MainScreen.swift

```
//
//  ContentView.swift
//  Reminders
//
//  Created by Максим Грищенков on 02.07.2025.
//

internal import EventKit
import SwiftData
import SwiftUI

struct MainScreen: View {
    @Environment(\.modelContext) var modelContext
    @EnvironmentObject private var syncService: EventKitSyncService
    //var defaultLists: [ReminderList]
    @State var isShowingListForm = false
    @State var isShowingReminderForm = false

    var body: some View {
        ScrollView {
            ReminderSmartLists().padding(.horizontal)
            HStack {
                Text("My Lists")
                    .font(.title2.bold())
                    .fontDesign(.rounded)
                Spacer()
            }.padding(.horizontal, 40).padding(.top, 16)
            RemindersList().padding(.horizontal)
        }
        .background(AppColors.background)
```

```
                .contentMargins(
                    .top,
                    0
                )
                .listSectionSpacing(.compact)

                .toolbar {
                    ToolbarSpacer(.flexible, placement: .bottomBar)
                    ToolbarItem(placement: .bottomBar) {
                        Button("Add", systemImage: "plus", role: .confirm) {
                            isShowingReminderForm.toggle()
                        }.sheet(
                            isPresented: $isShowingReminderForm,
                        ) {
                            if          let          existDefaultList          =
syncService.getDefaultList() {
                                FormReminderView(
                                    reminder: Reminder.getDefaultReminder(),
                                    ekreminder: EKReminder(
                                        eventStore: syncService.eventStore
                                    ),
                                    list: existDefaultList
                                )
                            }

                        }
                    }
                    ToolbarItem {
                        Button(
                            "Add list",
                            systemImage: "folder.badge.plus",
                        ) {
                            isShowingListForm.toggle()
                        }.sheet(
                            isPresented: $isShowingListForm,
                        ) {
                            FormListScreen(list: nil)
                        }
                    }
                   // ToolbarSpacer(.fixed)
                    ToolbarItem {
                        Button("Edit",                              systemImage:
"line.3.horizontal.decrease") {
```

```
                // действие добавления
            }
        }
    }

}
}

#Preview {
    NavigationStack {
        MainScreen()
    }.environmentObject(AppState())
}
```

Б.6 Текст файлу FormReminderView.swift

```
//
//  ReminderFormView.swift
//  Reminders
//
//  Created by Максим Грищенков on 28.08.2025.
//

internal import EventKit
import SwiftUI
import _SwiftData_SwiftUI

struct FormReminderView: View {
    @Environment(\.dismiss) private var dismiss
    @Environment(\.modelContext) private var context
    @State private var error: String? = nil
    @StateObject private var formReminderState: FormReminderState
    @State private var isShowingListForm: Bool = false
    @State private var isShowingRecurrenceRuleForm: Bool = false
    @Query private var lists: [ReminderList]

    init(reminder: Reminder, ekreminder: EKReminder, list: ReminderList) {
        _formReminderState = StateObject(
            wrappedValue: FormReminderState(
                reminder: reminder,
```

```
                    ekReminder: ekreminder,
                    list: list
                )
            )
        }

        var body: some View {
            NavigationStack {
                Form {
                    Section {
                        VStack(spacing: 20) {
                            TextField("Title",                          text:
$formReminderState.title)
                                .font(.title2).bold()
                            TextField(
                                "Notes",
                                text: $formReminderState.notes
                            )
                            .font(.default)
                            .bold()
                        }
                    }
                    Section {
                        DatePicker(
                            "Date",
                            selection: $formReminderState.date,
                            displayedComponents: [.date, .hourAndMinute]
                        )
                    }.listRowInsets(.vertical, 0)
                    Section {
                        Picker(

                            selection: $formReminderState.presetRecurrence
                        ) {
                            ForEach(PresetsRecurrence.allCases,         id:
\.self) {
                                preset in
                                Text(preset.rawValue)
                                    .tag(
                                        preset
                                            .EKValue(
                                                formReminderState
```

```
.getReccurrenceEndDate()
                                                )
                                        )
                                    if preset == .never || preset == .yearly {
                                        Divider()
                                    }
                                }
                            } label: {
                                ListLabel(
                                    title: "Repeat",
                                    systemImageName: "repeat"
                                )
                            }

                            if formReminderState.presetRecurrence == .custom {
                                Button {
                                    isShowingListForm.toggle()
                                } label: {
                                    Text(

formReminderState.ekRecurrenceRule?.stringValue
                                            ?? "Never"
                                    )
                                }.foregroundStyle(.secondary)
                            }
                            if formReminderState.presetRecurrence != .never {
                                Picker(
                                    selection: $formReminderState.endOption
                                ) {
                                    ForEach(
                                        EndRepeatOption.allCases,
                                        id: \.self
                                    ) { option in
                                        Text(option.rawValue).tag(option)
                                    }
                                } label: {

                                    ListLabel(
                                        title: "End Repeat",
                                        systemImageName: "repeat.badge.xmark"
                                    )
```

```
                                }
                            }
                            if formReminderState.endOption == .onDate {
                                DatePicker(
                                    "End Date",
                                    selection:
$formReminderState.recurrenceEndDate,
                                    displayedComponents: [.date]
                                ).listRowInsets(.vertical, 0)
                            }

                        }
                        Section {
                            NavigationLink(
                                destination: {
                                    NavigationStack {
                                        List {
                                            Picker(
                                                "List",
                                                selection:
$formReminderState.list
                                            ) {
                                                ForEach(lists,  id:  \.self)  {
list in
                                                    ReminderListLabel(
                                                        title: list.title,
                                                        imageName:
list.systemImage,

                                                        color: list.color,
                                                        isCircled: true
                                                    )
                                                }
                                            }

.pickerStyle(.inline).labelsHidden()

                                        }.navigationTitle(
                                            "Select list for the reminder"
                                        ).navigationBarTitleDisplayMode(
                                            .inline
                                        )
                                    }
                                }) {
```

```
                            HStack {
                                ReminderListLabel(
                                    title: "List",
                                    imageName: formReminderState.list
                                        .systemImage,
                                    color:
formReminderState.list.color,
                                    isCircled: true
                                )
                                Spacer()
                                Text(formReminderState.list.title)
                                    .foregroundStyle(.secondary)
                            }
                        }
                    }
                    Section {
                        NavigationLink(
                            destination:
                                TagPicker(
                                    selectedTags: $formReminderState.tags,
                                    context
                                )
                        ) {

                            HStack {
                                ListLabel(
                                    title: "Tags",
                                    systemImageName: "tag"
                                )

                                Spacer()

Text(formReminderState.tags.count.description)
                                    .foregroundStyle(.secondary)
                            }
                        }
                    }

                }
                .animation(
                    .default,
                    value: formReminderState.presetRecurrence
                )
```

```
                    .animation(
                        .default,
                        value: formReminderState.endOption
                    )
                    .contentMargins(.top, 10, for: .scrollContent)
                    .listSectionSpacing(.compact)
                    .navigationTitle("Create reminder")
                    .navigationBarTitleDisplayMode(.inline)
                    .toolbar {
                        ToolbarItem(placement: .navigationBarLeading) {
                            Button("Cancel",   systemImage:   "xmark",   role:
.cancel) {
                                dismiss()
                            }
                        }
                        ToolbarItem(placement: .navigationBarTrailing) {
                            Button("Save",   systemImage:   "checkmark",   role:
.confirm) {
                                do {
                                    Task {
                                        do {
                                            try await formReminderState
                                                .updateReminder(modelContext:
context)
                                            dismiss()
                                        } catch {

                                        }
                                    }
                                } catch {
                                    self.error = error.localizedDescription
                                }
                            }

                        }
                    }
                    .sheet(
                        isPresented: $isShowingListForm,
                    ) {

                        CustomRecurrenceRuleForm(
                            formState: formReminderState,
                        )
```

```
                }
                .sheet(isPresented: $isShowingRecurrenceRuleForm) {
                    Text("Test")
                }
                .alert(
                    "\(error?.description ?? "")",
                    isPresented: .constant(error != nil)
                ) {

                    Button("OK") {
                        error = nil
                    }

                }
            }

        }
    }

    #Preview {
        var list =
            ReminderList
            .getDefaultList()
        FormReminderView(
            reminder: Reminder.getDefaultReminder(),
            ekreminder: EKReminder(eventStore: EKEventStore()),
            list: list

        )
    }
```

Б.7 Текст файлу FormListScreen.swift

```
//
//  AddListScreen.swift
//  Reminders
//
//  Created by Максим Грищенков on 17.08.2025.
//
```

```
import SwiftUI

struct FormListScreen: View {
    @Environment(\.dismiss) private var dismiss
    @Environment(\.modelContext) private var context
    @StateObject private var state: FormListState

    @State private var error: String? = nil
    @State private var showFilterEditor: Bool = false
    private let colors: [Color] = [
        .red, .orange, .yellow, AppColors.listGreen, AppColors.listLightCyan,
        AppColors.listBlue,
        AppColors.listIndigo, AppColors.listPink, AppColors.listAccentPurple,
        AppColors.listWarmBrown,
        AppColors.listGray, AppColors.listBeige,
    ]

    private let reminderIcons: [String] = [
        "checkmark.circle.fill",
        "list.bullet",
        "bell.fill",
        "flag.fill",
        "star.fill",
        "calendar",
        "clock.fill",
        "note.text",
        "square.and.pencil",
        "tag.fill",
    ]

    init(list: ReminderList? = nil) {
        if let list {
            _state = StateObject(wrappedValue: FormListState(list: list))
        } else {
            _state = StateObject(wrappedValue: FormListState())
        }
    }

    init(smartList: ReminderSmartList? = nil) {
        if let smartList {
            _state = StateObject(
                wrappedValue: FormListState(smartList: smartList)
            )
```

```
    } else {
        _state = StateObject(wrappedValue: FormListState())
    }
}

init(aiList: ReminderAIList? = nil) {
    if let aiList {
        _state = StateObject(wrappedValue: FormListState(aiList: aiList))
    } else {
        _state = StateObject(wrappedValue: FormListState())
    }
}

private let adaptiveColumn = [
    GridItem(.adaptive(minimum: 45))
]

var body: some View {
    NavigationStack {

        ZStack(alignment: .top) {
            Form {
                if state.listType != .aiList {
                    Section {
                        VStack(spacing: 0) {
                            ZStack(alignment: .center) {
                                Circle().fill(state.color)
                                    .frame(width: 100, height: 100)
                                    .shadow(
                                        color: state.color.opacity(0.3),
                                        radius: 10
                                    )
                                Image(systemName: state.systemImage)
                                    .resizable()
                                    .fontWeight(.semibold)
                                    .scaledToFit()
                                    .frame(width: 50, height: 50)
                                    .foregroundStyle(.white)
                            }.frame(maxWidth: .infinity).padding(5)
                            TextField("List name", text: $state.title)
                                .padding(10)

                                .multilineTextAlignment(.center)
```

```
                    .font(
                        .system(size: 24, weight: .semibold)
                    )
                    .background(.secondary.opacity(0.2))
                    .clipShape(
                        RoundedRectangle(cornerRadius: 16)
                    ).padding(.top, 10)

            }
        }
    }
    if state.listType == .aiList {
        Section {

            ZStack(alignment: .center) {
                Circle()
                    .frame(width: 100, height: 100)
                    .foregroundStyle(getAIGradient())
                    .mask {
                        Circle().stroke(lineWidth: 8).blur(
                            radius: 15
                        )
                    }
                    .shadow(
                        color: state.color.opacity(0.3),
                        radius: 10
                    )
                Image(systemName: "apple.intelligence")
                    .resizable()
                    .fontWeight(.regular)
                    .scaledToFit()
                    .frame(width: 70, height: 70)
                    .foregroundStyle(
                        getAIGradient().opacity(0.7)
                    )
            }.frame(maxWidth: .infinity).padding(5)

        }
    }
    if state.listType == .aiList {
        Picker(
            "Select list type",
            selection: $state.aiListLabel
```

```
    ) {
        ForEach(
            AIListLabel.allCases,
            id: \.self
        ) { label in
            Text(label.stringValue).tag(label)
        }
    }.pickerStyle(
        .inline
    )
}
if state.listType == .smartList {
    Button(action: { showFilterEditor.toggle() }) {
        HStack {
            Text("Edit Filters")
            Spacer()
            Image(systemName: "chevron.right")
                .fontWeight(.bold)
                .foregroundStyle(.tertiary)
        }
    }.tint(.primary)
        .sheet(isPresented: $showFilterEditor) {
            NavigationStack {
                SmartQueryPickerView(
                    state: state.smartListQueryState
                ).toolbar {
                    ToolbarItem(
                        placement: .navigationBarLeading
                    ) {
                        Button(
                            "Cancel",
                            systemImage: "xmark",
                            role: .cancel
                        ) {
                            showFilterEditor = false
                        }
                    }

                    ToolbarItem(
                        placement:
                            .navigationBarTrailing
                    ) {
                        Button(
```

```
                                "Done",
                                systemImage: "checkmark",
                                role: .confirm
                            ) {
                                showFilterEditor = false
                            }
                        }
                    }.navigationTitle("Filters")
                        .navigationBarTitleDisplayMode(
                            .inline
                        )
                }
            }
        }
        if state.listType != .aiList {
            Section {
                LazyVGrid(columns: adaptiveColumn, spacing: 10) {
                    ForEach(reminderIcons, id: \.self) { item in
                        SystemImagePick(
                            systemImage: item,
                            isSelected: item == state.systemImage
                        ) {
                            state.systemImage = item
                        }
                    }
                }
            }

            Section {
                LazyVGrid(columns: adaptiveColumn, spacing: 10) {
                    ForEach(colors, id: \.self) { item in
                        CirclePick(
                            item: item,
                            isSelected: item.hexString
                                == state.color.hexString
                        ) {
                            state.color = item
                        }
                    }
                }
            }
        }
```

```
                }.animation(.linear, value: state.listType)

                Picker("List type", selection: $state.listType) {
                    Text("Default list").tag(ListType.list)
                    Text("Smart List").tag(ListType.smartList)
                    Text("AI List").tag(ListType.aiList)
                }.pickerStyle(.palette)
                    .padding(.horizontal, 20)

            }
            .listSectionSpacing(.compact)
            .contentMargins(.top, 45)
            .navigationTitle("Create List")
            .navigationBarTitleDisplayMode(.inline)
            .toolbar {
                ToolbarItem(placement: .navigationBarLeading) {
                    Button("Cancel", systemImage: "xmark", role: .cancel) {
                        dismiss()
                    }
                }
                ToolbarItem(placement: .navigationBarTrailing) {
                    Button("Save", systemImage: "checkmark", role: .confirm)
{
                        Task {
                            do {
                                try await state.saveList(context: context)
                                dismiss()
                            } catch {
                                self.error = error.localizedDescription
                            }
                        }
                    }

                }
            }
            .alert("Creation error", isPresented: .constant(error != nil)) {
                VStack {
                    Text(error ?? "Unknown error")

                    Button("OK") {
                        error = nil
                    }
                }
```

```
            }

        }
    }
}

#Preview {
    NavigationStack {
        EmptyView()
    }.sheet(isPresented: .constant(true)) {
        FormListScreen(list: ReminderList.getDefaultList())
    }
}
```

ДОДАТОК В

Слайди презентації

Національний університет «Запорізька політехніка»
Кафедра програмних засобів

Дипломна кваліфікаційна робота магістра

Дослідження та програмна реалізація розумного планувальника для iOS
Research and software implementation of smart scheduler for iOS

Виконав
Студент групи КНТ-214м Максим ГРИЩЕНКОВ

Керівник роботи
к.т.н., доцент Тетяна ФЕДОРОНЧАК

Рисунок В.1 – Слайд 1

Мета роботи, об’єкт та предмет дослідження

Мета роботи – реалізація мобільного застосунку для створення та керування нагадуваннями за допомогою алгоритмів фільтрації та штучного інтелекту для платформи iOS.

Об'єкт дослідження – алгоритми для реалізації розумних планувальників задач.

Предмет дослідження – алгоритми фільтрації та штучного інтелекту для створення та керування нагадуваннями в мобільному телефоні.

2

Рисунок В.2 – Слайд 2

Порівняння розроблюваного застосунку з аналогами

Критерій	Reminders	Google Tasks	Microsoft To Do	Reminders Plus
Синхронізація між Apple пристроями	+	-	-	+
Списки нагадувань	+	+	+	+
Смарт-списки нагадувань з алгоритмами фільтрації	+	-	-	+
Сортування нагадувань	-	+	+	+
Присвоєння нагадуванням тегів	+	-	-	+
AI-списки з класифікацією нагадувань за допомогою нейронних мереж	-	-	-	+
Потужна система створення повторюваних нагадувань	+	-	-	+

3

Рисунок В.3 – Слайд 3

Порівняльний аналіз моделей текстової класифікації

Параметр порівняння	MaxEnt	CRF	BERT
Врахування контексту	Ні	Лише лінійну послідовність	Глобальний контекст
Робота з різними мовами	Низька	Низька	Висока
Потреба у великих даних	Низька	Середня	Висока
Швидкість навчання	Швидка	Середня	Повільна
Точність на складних текстах	Низька	Середня	Висока

4

Рисунок В.4 – Слайд 4

Діаграма прецедентів

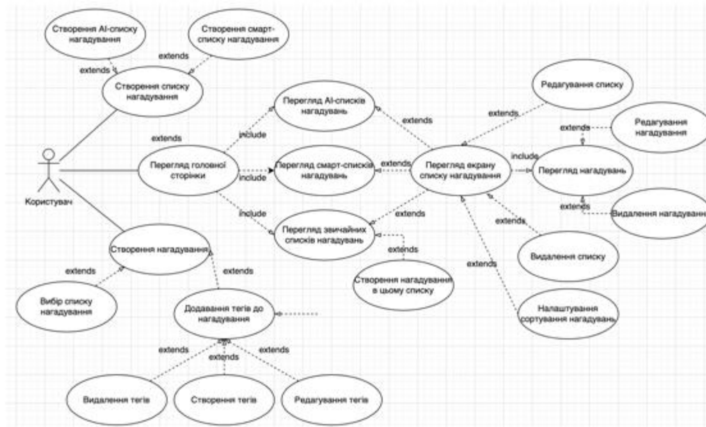

Рисунок 1 – Діаграма прецедентів

5

Рисунок В.5 – Слайд 5

Порівняння фреймворків

Критерій	Flutter	SwiftUI	UIKit
Повний інструментарій Apple	-	+	+
Парадигма програмування	Декларативна	Декларативна	Імперативна
Простота кастомізації	Середня	Непогана	Найвища
Розвиток фреймворку	Так	Так	Ні (тільки підтримка оновлень)
Споживання ресурсів	Середнє	Ефективне	Найменше

6

Рисунок В.6 – Слайд 6

Порівняння середовищ розробки

Критерій	Xcode	Visual Studio Code	AppCode
Підтримка системи контролю версій	+	+	+
Безкоштовний AI агент	+	-	-
Безкоштовна	+	+	-
Підтримка розширень	+	+	+
Підтримка Swift/SwiftUI	Повна	Обмежена	Повна
Вбудований відлагоджувач	+	Через розширення	+
Ресурсоємність	Середня	Середня	Висока
Інтеграція з інструментами Apple	+	-	Частково

7

Рисунок В.7 – Слайд 7

Порівняння інструментів побудови нейронних мереж

Критерій	Create ML	Core ML tools	TFLite
Зручність використання	Дуже висока	Висока	Середня
Швидкість виконання	Висока	Висока	Середня
Швидкість навчання	Дуже висока	Висока	Помірна
Споживання ресурсів	Низьке	Низьке	Помірне
Популярність та підтримка	Висока	Висока	Висока

8

Рисунок В.8 – Слайд 8

Порівняння баз даних

Критерій	SwiftData	Core Data	Realm
Тип	ORM на Swift	ORM на Swift/Objective-C	Незалежна БД з власним рушієм
Сумісність з SwiftUI	Повна	Часткова	Добра
Реактивність	Так	Обмежена	Так
Синхронізація	Автоматична між локальною БД та iCloud	Власноруч між локальною БД та iCloud	Тільки віддалено
Підтримка	Так	Так	Так

9

Рисунок В.9 – Слайд 9

Архітектура застосунку

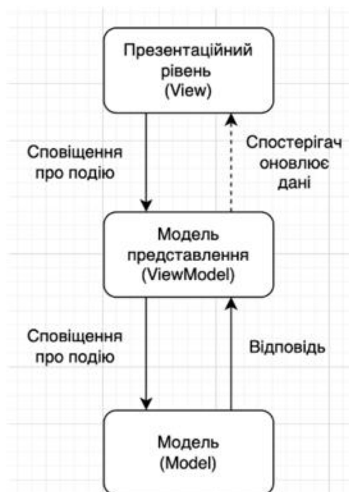

Рисунок 2 – Архітектура застосунку

10

Рисунок В.10 – Слайд 10

Схема функціонування застосунку

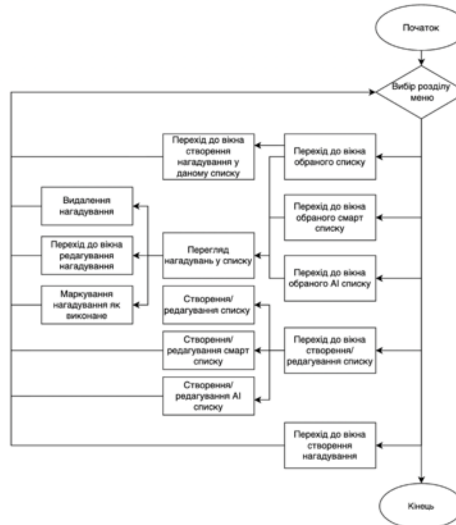

Рисунок 3 – Схема функціонування застосунку

11

Рисунок В.11 – Слайд 11

Схема зв’язків між сутностями у БД

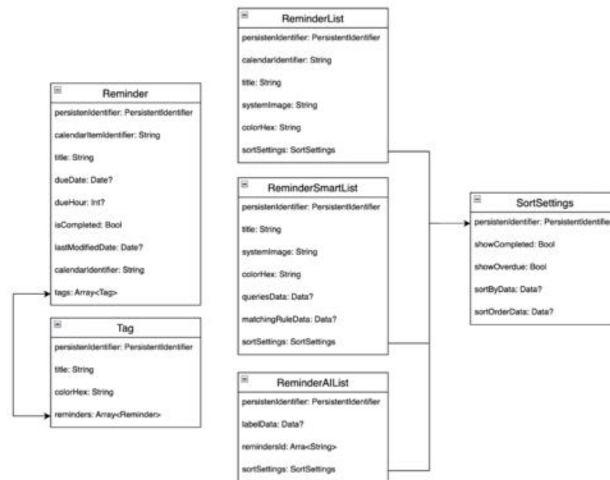

Рисунок 4 – Схема зв'язків між сутностями у базі даних

12

Рисунок В.12 – Слайд 12

Структура проєкту

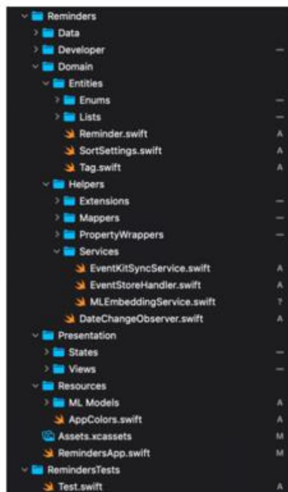

Рисунок 5 – Схема зв'язків між сутностями у базі даних

13

Рисунок В.13 – Слайд 13

Інтерфейси застосунку

Рисунок 6 – Головний екран застосунку

Рисунок 7 – Форма створення списку

Рисунок 8 – Форма вибору фільтрів для смарт списку

Рисунок 9 – Форма створення AI списку

Рисунок 10 – Форма створення нагадування

Рисунок 11 – Екран з нагадуваннями

14

Рисунок В.14 – Слайд 14

Тестування застосунку

Перевірка	iPhone 15 Pro (iOS 26.2)	iPhone 17 Pro (iOS 26.1)
Перевірити коректне створення звичайного списку	Пройдено	Пройдено
Перевірити коректне створення смарт списку	Пройдено	Пройдено
Перевірити коректне створення AI списку	Пройдено	Пройдено
Перевірити редагування списку	Пройдено	Пройдено
Перевірити видалення списку	Пройдено	Пройдено
Перевірити переходи між екранами	Пройдено	Пройдено
Перевірити синхронізацію нагадувань і списків з EventKit	Пройдено	Пройдено
Перевірити створення нагадування	Пройдено	Пройдено
Перевірити редагування нагадування	Пройдено	Пройдено
Перевірити роботу відмітки нагадування як виконаного	Пройдено	Пройдено
Перевірити видалення нагадування	Пройдено	Пройдено
Перевірити створення тегів	Пройдено	Пройдено
Перевірити фільтрацію нагадувань	Пройдено	Пройдено
Перевірити сортування нагадувань	Пройдено	Пройдено
Перевірити роботу нейромережі для AI списку	Пройдено	Пройдено

Рисунок 12 – Чекліст функціонального тестування

Перевірка	iPhone 15 Pro (iOS 26.2)	iPhone 17 Pro (iOS 26.1)
Перевірити коректне відображення кнопок, блоків меню, тощо	Пройдено	Пройдено
Перевірити наявність усіх сторінок	Пройдено	Пройдено
Перевірити коректне відображення шрифтів та стилів тексту	Пройдено	Пройдено
Перевірити коректне відображення сторінок згідно дизайну	Пройдено	Пройдено
Перевірити граматику та орфографію застосунку	Пройдено	Пройдено
Перевірити коректне відображення даних у формах	Пройдено	Пройдено
Перевірити коректне відображення анімацій і переходів	Пройдено	Пройдено
Перевірити коректне відображення модальних вікон	Пройдено	Пройдено

Рисунок 13 – Чекліст тестування верстки

15

Рисунок В.15 – Слайд 15

Висновки

Було розібрано найпопулярніші аналоги програм, таких як «Reminders», «Google Tasks» та «Microsoft To Do», та порівняно їхні функціональні можливості з функціональними можливостями «Reminders Plus». Порівняльний аналіз показав, що розроблюваний застосунок є актуальним та доцільним на фоні конкурентів.

Було здійснено огляд основних моделей класифікації тексту (MaxEnt, CRF та BERT) та проведено їхнє порівняння для вибору моделі. В результаті базовою моделлю було вирішено використовувати BERT через її спроможність розуміти контекст та високу точність. Для навчання та тестування моделі для класифікації нагадувань за темою було створено власний датасет на основі датасету «MS-LaTTE» з GitHub репозиторію.

Для розробки програмного продукту було визначено мову програмування Swift і фреймворк SwiftUI, а IDE було обрано Xcode. Для організації збереження даних було обрано базу даних SwiftData SQL-типу.

Було організовано структуру бази даних, описано алгоритми роботи програми та здійснено розробку ключових класів і методів.

Було перевірено коректність роботи верстки та функціональної частини застосунку. У результаті всі тести були успішно пройдені.

Після всіх етапів було створено програмний продукт «Reminders Plus», який надає користувачу потужну категоризацію нагадувань за різними їх ознаками як за допомогою алгоритмів фільтрації, так і штучного інтелекту.

16

Рисунок В.16 – Слайд 16