Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти перший (бакалаврський)

Кросплатформний мобільний застосунок для менеджменту та аналізу задач

(тема)

Виконав:
здобувач 4 року навчання,
групи КІУКІ-21-5
Данил МИРГОРОДСЬКИЙ
(власне ім’я, прізвище)

Спеціальність
123 «Комп’ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма
Комп’ютерна інженерія
(повна назва освітньої програми)

Керівник: ас. Юлія АНДРУСЕНКО
(посада, власне ім’я, прізвище)

Допускається до захисту

Завідувач кафедри ЕОМ _____ Андрій КОВАЛЕНКО
(підпис) (власне ім’я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти перший (бакалаврський)

Спеціальність 123 «Комп’ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп’ютерна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
“____” _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Миргородському Данилу Борисовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Кросплатформний мобільний застосунок для менеджменту та аналізу задач.

затверджена наказом по університету від “ 26 ” травня 2025 р. № 424 Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 17 червня 2025 р.

3. Вхідні дані до роботи 1) інтегроване середовище розробки Microsoft Visual Studio
2) компоненти мови програмування C# та .NET
3) фреймворк Xamarin.Forms
4) засіб розробки компонентів програмного забезпечення Android SDK
5) інтегроване середовище розробки Apple Xcode
6) Android та iOS пристої або їх емулятори

4. Перелік питань, що потрібно опрацювати у роботі
1) аналіз предметної області та постановка задачі
2) огляд та вибір програмного забезпечення
3) опис програмної реалізації застосунку
4) інструкція користувача
5) висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій

Слайд-презентація – 13 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Аналіз предметної області та постановка задачі	27.05.25 - 29.05.25	
2	Огляд та вибір програмного забезпечення	30.05.25 - 31.06.25	
3	Програмна реалізація застосунку	01.06.25 - 04.06.25	
4	Інструкція користувача	05.06.25 - 06.06.25	
5	Оформлення матеріалів кваліфікаційної роботи	07.06.25 - 11.06.25	
6	Подання кваліфікаційної роботи керівникові та	12.06.25 – 13.06.25	
	її попередній захист		
7	Подання кваліфікаційної роботи на рецензу-	14.06.25 – 16.06.25	
	вання		

Дата видачі завдання " 26 " травня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ ас. Юлія АНДРУСЕНКО
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 77 с., 19 рис., 2 табл., 2 дод., 16 джерел.

МОБІЛЬНИЙ ЗАСТОСУНОК, КЕРУВАННЯ ЧАСОМ, ANDROID, IOS, TASKMANAGER, КРОСПЛАТФОРМЕНІСТЬ, ЗАДАЧІ, СПОВІЩЕННЯ, XAMARIN.FORMS.

Метою кваліфікаційної роботи є створення кросплатформного мобільного застосунку для організації та управління задачами.

У ході виконання кваліфікаційної роботи було вивчено теоретичний матеріал, обрано загальну концепцію застосунку, проаналізовано існуючі аналоги та їх можливості, і, на основі цього, розроблено власний дизайн і функціональний алгоритм для реалізації програми.

Кінцевим результатом роботи є працездатний мобільний кросплатформний застосунок, який надає користувачам можливість створювати нові задачі різних типів, групувати їх за категоріями, додавати опис, редагувати кінцеву дату та час виконання, змінювати назву та опис задачі, переглядати детальну інформацію та за потреби видаляти її.

Також користувач має змогу переглядати статистику задач у різних форматах, що дає можливість аналізувати успішність їх виконання, завантаженість робочого дня, та отримувати поради щодо більш ефективного менеджменту часу. Крім цього, передбачено гнучкі налаштування самого додатку та системи сповіщень.

Програму складено мовою C# .NET з використанням Xamarin.Forms у середовищі програмування Visual Studio.

ABSTRACT

Bachelor’s thesis: 77 pages, 19 figures, 2 tables, 2 appendices, 16 sources.

MOBILE APPLICATION, TIME MANAGEMENT, ANDROID, IOS, TASKMANAGER, CROSS-PLATFORM, TASKS, NOTIFICATIONS, XAMARIN.FORMS.

The major goal of this qualification work is to create a cross-platform mobile application for organizing and managing tasks. During the qualification work, theoretical material was studied, the general concept of the application was selected, existing analogues and their functionality were analyzed, and based on this, an own design and functional algorithm for implementing the program was developed.

The final result of the work is a functional mobile cross-platform application that allows users to create new tasks of different types, group them by categories, add a description, edit the due date and time, change the name and description of the task, view detailed information and, if necessary, delete it.

The user is also able to view task statistics in various formats, which makes it possible to analyze the success of their implementation, the workload of the working day and receive advice on more effective time management. In addition, flexible settings of the application itself and the notification system are provided.

The program is written in C# .NET using Xamarin.Forms in the Visual Studio programming environment.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ЗБД – зовнішня база даних

КПЗ – кросплатформне програмне забезпечення

API – інтерфейс програмування застосунків (англ., Application Programming Interface)

APK – формат пакета встановлення для Android (англ. Android Package)

CRUD – базові операції з даними: створення, читання, оновлення, видалення (англ., Create, Read, Update, Delete)

CV – елемент інтерфейсу, що використовується для відображення списку задач (англ., CollectionView)

MVVM – шаблон проєктування архітектури застосунків (англ., Model-View-ViewModel)

NuGet – менеджер пакетів для бібліотек .NET

Popap / Попап – різновид спливаючого вікна (англ., Pop-Up)

UI – інтерфейс користувача (англ., User Interface)

UX – користувацький досвід (англ., User Experience)

XAML – мова розмітки, що використовується для опису інтерфейсу в Xamarin.Forms

ВСТУП

У реаліях сучасного світу дедалі гостріше постає необхідність у вмінні швидко й ефективно виконувати численні завдання в обмежені часові рамки. Активний темп життя, висока конкуренція на ринку праці, стрімкий розвиток інформаційних технологій та постійне зростання обсягів інформації суттєво ускладнили процес організації власного часу. Людям доводиться постійно балансувати між професійними обов'язками, особистими справами та соціальним життям. До цього додається потреба вирішувати рутинні побутові завдання та різні незначні справи, які залишаються важливою частиною повсякденного життя кожної людини, але зазвичай залишаються поза увагою, і тому часто забуваються. В умовах такої багатозадачності критично важливим стає вміння правильно планувати свій час та ефективно розподіляти ресурси. Усі ці фактори формують потребу в розумному підході до організації особистого часу, що передбачає наявність надійного та зручного інструменту для управління завданнями. Саме тому тематика цифрового тайм менеджменту набуває дедалі більшої актуальності в сучасному суспільстві.

Управління завданнями стало критично важливим як у сфері особистої продуктивності, так і в професійному середовищі. Люди, які вміють грамотно планувати свій день, відстежувати прогрес виконання справ та адаптуватися до змін обставин, частіше досягають вищої результативності та мають менше стресу. З розвитком цифрових технологій з'явився цілий сектор програмних рішень, завдання яких – полегшити процес планування. Це можуть бути як звичайні додатки для ведення списків справ, так і системи управління проєктами або різного роду медичні, спортивні та інші трекери, ціль яких – вчасно вказати на правильний проміжок часу. Більшість з них дозволяють формувати завдання, призначати дедлайни, контролювати процес виконання, використовувати нагадування, інтегруватися з календарями. Однак, попри

розмаїття таких інструментів, користувачі часто зіштовхуються з певними труднощами. Більшість популярних рішень мають складний або перевантажений інтерфейс, що відштовхує менш досвідчених користувачів і потребує тривалого часу для освоєння. Ці аспекти роблять розробку нового, більш гнучкого й доступного таскменеджера актуальною та перспективною.

Запропонований у межах цієї роботи застосунок орієнтований на задоволення потреб користувачів, які прагнуть не лише швидко, за будь-яких умов та без особливих зусиль фіксувати справи, а й активно аналізувати та управляти своїм часом. Основна мета проєкту – створення простого, але водночас інтуїтивно зрозумілого та функціонального таскменеджера, який дозволить легко адаптувати його під конкретні запити, незалежно від рівня володіння мобільним пристроєм. Запропоноване рішення покликане стати надійним помічником для широкого кола користувачів – від студентів і фрілансерів до працівників різних сфер. Його використання сприятиме кращому плануванню, підвищенню особистої продуктивності, зниженню ризику забути про справи, а також загальному покращенню балансу між роботою й особистим життям.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної області

Управління завданнями є важливою складовою ефективної організації робочого часу та ресурсів, особливо в сучасному швидкоплинному світі. Нинішній ринок програмного забезпечення для управління завданнями включає численні інструменти, які допомагають користувачам планувати, організовувати, відслідковувати виконання завдань та підвищувати продуктивність. Ці рішення активно використовуються як у професійному, так і в особистому житті для покращення ефективності роботи [1].

Сучасні, так звані, менеджери задач дозволяють користувачам створювати завдання з детальними параметрами, такими як назва, опис, терміни виконання, нагадування, додавати підзадачі, категорії та мітки, та виконувати різноманітні операції над цими задачами. Незважаючи на різноманіття існуючих рішень, багато з них мають певні обмеження, або підлаштовані лише під певні категорії користувачів. Надмірна складність інтерфейсу, обмеження безкоштовного функціоналу, відсутність аналітики або підтримки офлайн режиму, це створює нішу для нових рішень, які б поєднували інтуїтивний інтерфейс, функціональну гнучкість програми і аналітичні можливості для відстеження та аналізу прогресу, а також виявлення неефективних практик при роботі з завданнями.

Говорячи про цільову аудиторію даних застосунків, варто зазначити, що вона охоплює широке коло користувачів, об’єднаних спільною потребою в ефективному управлінні та моніторингу завданнями. Насамперед це люди, які прагнуть структурувати свій день, зменшити рівень стресу через забуті справи та покращити особисту продуктивність. До таких користувачів належать як студенти та фрілансери, які планують навчальні активності або робочі процеси міксуючи їх з повсякденними справами, так і професіонали,

що координують свій графік в умовах обмежених ресурсів і дедлайнів.

Водночас іншою частиною цільової аудиторії є люди, не зацікавлені в детальному відстеженні всіх своїх активностей, проте вони мають у власному графіку події, які потребують особливої уваги до часу та періодичності виконання. Це, в основному, різного роду трекери, які допомагають спортсменам відстежувати час між підходами у вправах, хворим людям – час між прийомом ліків та залишок до закінчення курсу лікування, а також інші типи активностей для різних категорій користувачів. Для цих людей важливою є можливість гнучкого налаштування часу між циклами завдань, налаштування кінцевої мети завдання, зручне відстеження та аналіз прогресу.

Таким чином, основна аудиторія очікує від застосунку не лише базового функціоналу, а й адаптивності до особистих потреб, можливості налаштування тем оформлення, швидкої навігації та гнучких опцій перегляду й редагування задач. Це користувачі, які хочуть мати у своєму розпорядженні інструмент, що допомагає не лише виконувати справи, а й розуміти власну продуктивність і те, що можна змінити для досягнення кращих результатів.

Особливу увагу також заслуговує впровадження аналітичного модуля для представлення користувачам статистики щодо задач. Це дозволяє оцінювати кількість виконаних завдань, час їх виконання, найпродуктивніші або найбільш навантажені дні, що дає змогу користувачеві приймати більш обґрунтовані рішення щодо планування та оптимізації своєї роботи.

Таким чином, предметна область кросплатформного мобільного застосунку для менеджменту та аналізу задач охоплює як технічні, так і поведінкові аспекти, спрямовані на покращення ефективності роботи користувача. Ключем до успішної реалізації такого рішення є глибоке розуміння потреб кінцевих користувачів, об'єднання їх в єдину програму з можливістю задовольнити їхні основні потреби та загальна зручність користування додатком.

1.2 Огляд існуючих програмних рішень

На ринку застосунків для менеджменту та аналізу задач існує багато готових програмних рішень. Проводячи їх аналіз, доцільно розпочати з розгляду їх ключових функцій, характерних для сучасних менеджерів та трекерів задач.

Такі системи зазвичай надають користувачеві можливість створювати завдання з широким набором атрибутів, включаючи дату виконання, рівень пріоритетності, нагадування та розподіл за категоріями [2]. Окрім базового планування, багато додатків орієнтовані на спільну роботу з можливістю додавати вкладення й відстежувати хід виконання в режимі реального часу. Сучасні рішення також прагнуть до інтеграції в повсякденну цифрову екосистему. Деякі інструменти додатково пропонують візуальні способи організації задач, інтерактивні таймлайни, а також вбудовані аналітичні інструменти для оцінки ефективності користувача чи команди. Загальна тенденція даних програмних рішень – це потреба в підключенні до мережі, що з одного боку приносить плюси, такі як можливість інтеграції з іншими сервісами або синхронізація користувачів в додатку, а з іншого боку не дає використовувати весь функціонал при відсутності мережі або взагалі обмежує доступ в застосунок [3].

Таким чином основна особливість розробленого програмного рішення буде його повна автономність. Скористуватися додатком можна буде в будь-яких умовах, і при цьому він збереже потрібний функціонал представлених з застосунках аналогах.

1.2.1 Todoist

Одним із найпопулярніших рішень є Todoist, представлений на рисунку 1.1, який відзначається широкими можливостями налаштування, включаючи використання міток, фільтрів і кольорового кодування. Його інтерфейс

базується на списках і проєктах, де користувачі можуть створювати завдання, встановлювати дедлайни, пріоритети, використовувати мітки, теги та фільтри. Завдяки синхронізації між пристроями, Todoist зручно використовувати як на смартфоні, так і на комп'ютері. Крім того, система сповіщень і нагадувань сприяє кращій організації часу та завдань. Основна особливість застосунку полягає в тому, що користувач може швидко додавати завдання за допомогою розумного вводу, який автоматично встановлює час, категорію та мітку за введеною назвою. Програмне рішення дозволяє створювати повторювані завдання з гнучкими параметрами циклічності та умов виконання. Todoist також підтримує інтеграцію з популярними сервісами, такими як Google Calendar, Slack і Outlook.

До переваг можна віднести зрозумілий інтерфейс, гнучку систему нагадувань, аналітику продуктивності. До недоліків можна віднести те, що для декого спектр функціоналу може здатися занадто обширним, і спочатку досить складно в ньому розібратися. Також недоліком є те, що такі функції, як нагадування або фільтри, доступні лише в платній версії. Крім того, у базовій версії відсутні візуальні представлення, які інші сервіси пропонують безкоштовно.

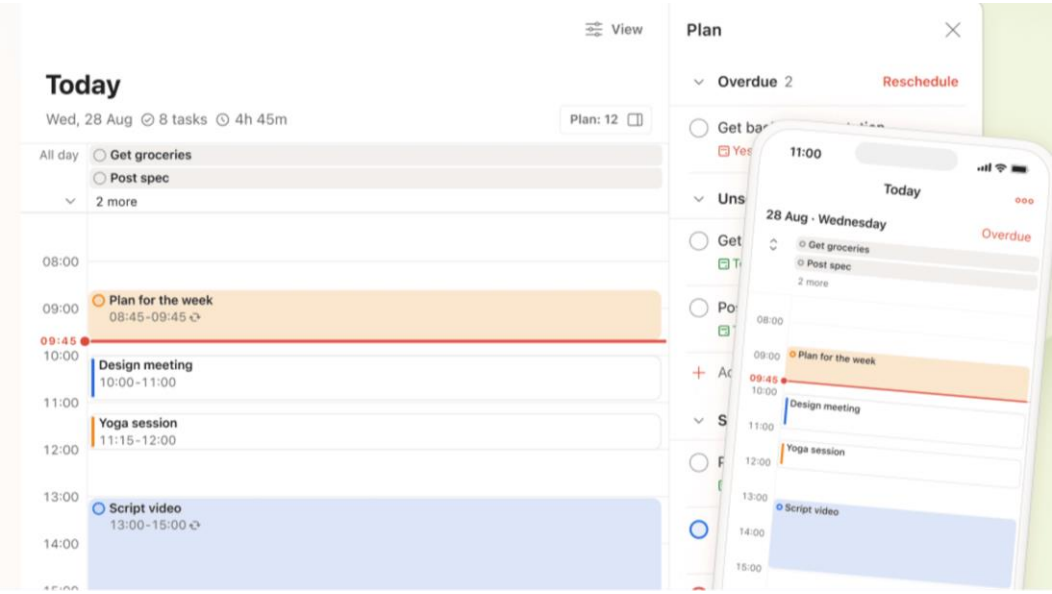


Рисунок 1.1 – Інтерфейс застосунку Todoist

1.2.2 Any.do

Інше популярне рішення – це Any.do (рисунок 1.2). Це застосунок робить ставку на інтуїтивність, мінімалізм і голосове введення. Головна мета даного програмного рішення – зробити додавання та обробку задач якомога простішими. Завдання організовані по днях, що дає змогу планувати справи на тиждень наперед. Є можливість додавати до задач підзадачі, замітки та нагадування. Існує функція планувальника дня, де щоранку програма пропонує переглянути всі заплановані задачі й розподілити їх. Any.do підтримує синхронізацію між платформами, інтегрується з різними календарями, такими як Google, iCloud та Outlook.

Головними перевагами є естетичний інтерфейс, зручність використання і розумна інтеграція з голосовими помічниками, повна інтеграція та синхронізація з усіма платформами. Однак недоліками можна назвати обмеженість у структуризації задач. Немає складної ієрархії, не підтримується теги або фільтри. Крім того, розширені можливості, такі як повторювані нагадування або робота з іншими користувачами, доступні лише в преміум версії, що може бути незручним для користувачів, які шукають більше свободи в базовому функціоналі.

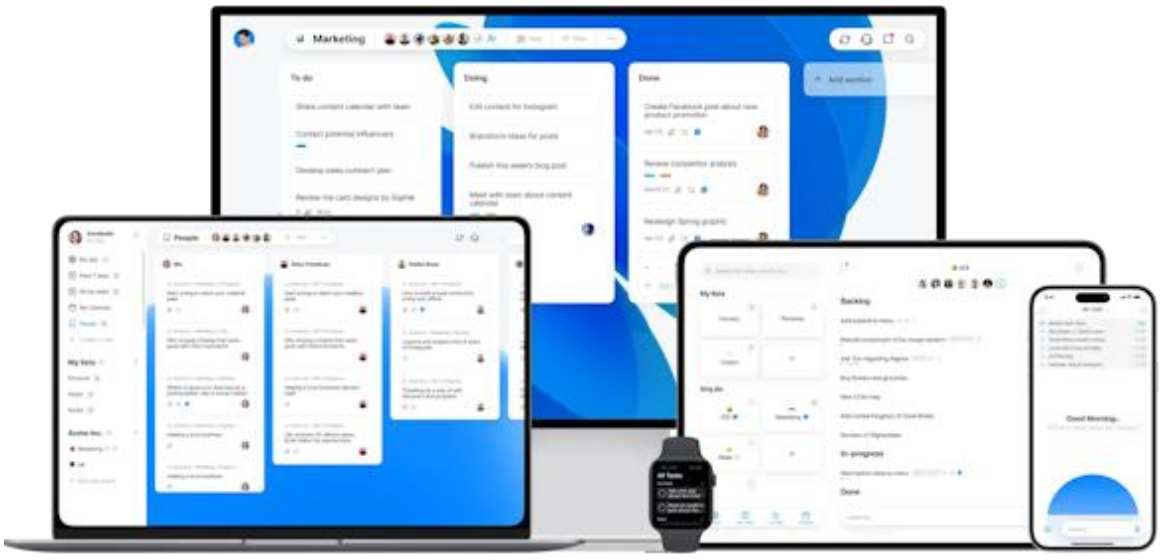


Рисунок 1.2 – Інтерфейс застосунку Any.do

1.2.3 Microsoft To Do

Також до популярних рішень відносять є Microsoft To Do (рисунок 1.3) – це офіційний таск-менеджер Microsoft, який прийшов на заміну Wunderlist і став частиною екосистеми Microsoft 365. Його головна особливість – це інтеграції з іншими сервісами. Microsoft. To Do працює на основі списків, в які можна додавати задачі та підзадачі, встановлювати для них нагадування, дедлайни та нотатками з описом. Іншою особливістю є планувальник дня, що дозволяє фокусуватись лише на важливому. Завдяки синхронізації між пристроями Microsoft To Do забезпечує безперервний доступ до задач з будь-якої платформи. Також користувачі можуть спільно працювати над списками, надаючи доступ іншим для перегляду чи редагування. Інтерфейс максимально простий і мінімалістичний, що робить його ідеальним для людей без потреби в складних функціях.

Перевагою є повна безкоштовність, хороша інтеграція з офісним середовищем, автоматичне перенесення задач з інших сервісів. Недоліком може бути обмежена кількість розширених функцій під час пошуку або створення задачі: немає тегів, фільтрів, аналітики чи повноцінної командної роботи.

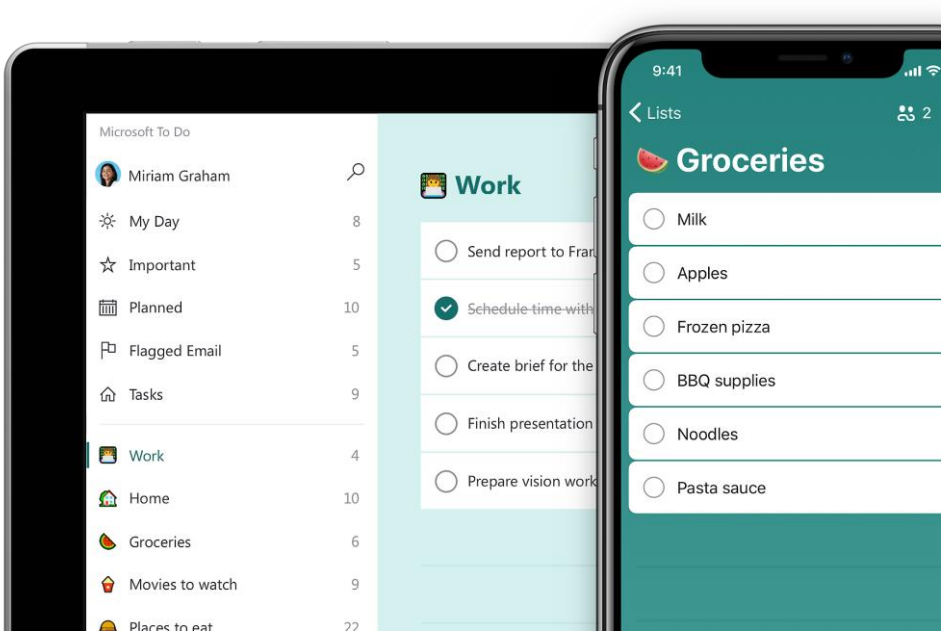


Рисунок 1.3 – Інтерфейс застосунку Microsoft To Do

1.2.4 TickTick

Наступним програмним рішенням є TickTick (рисунок 1.4) – це мультифункціональний менеджер задач, який намагається поєднати особисте планування з інструментами продуктивності. Він підтримує не лише списки задач, а й календар, Pomodoro-таймер, трекер звичок, можливість встановлення пріоритетності задач, мітки та повторювані події. Особливістю є можливість планування тижня в інтерактивному календарі з перетягуванням задач між днями. TickTick дозволяє створювати розділені секції в рамках одного списку, підтримує вкладені завдання, додавання файлів та коментарів.

Його перевагами є велика кількість функцій навіть у безкоштовній версії, висока швидкість роботи додатку та широкий спектр можливостей кастомізації. Особливо цінується функція Pomodoro з підрахунком фокус-часу для найдетальнішого ведення менеджменту часу та завдань. Проте, як і багато інших програмних рішень, TickTick має обмеження у безкоштовному варіанті, зокрема щодо кількості списків, міток, а також функцій нагадувань, фільтрів та аналітики, які доступні лише у платній підписці. Дизайн, хоч і зручний, але може здатися перевантаженим через велику кількість функцій.

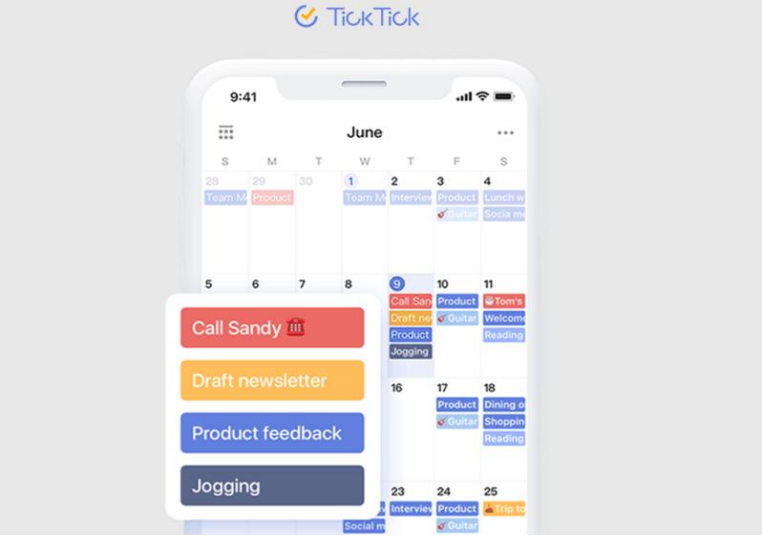


Рисунок 1.4 – Інтерфейс застосунку TickTick

1.3 Постановка задачі

Даний проєкт, розроблений в рамках кваліфікаційної роботи, орієнтований на створення кросплатформного мобільного застосунку для ефективного менеджменту та аналізу завдань. Задля реалізації поставленої цілі було обрано популярний підхід до розробки – створення застосунку з використанням кросплатформених засобів, у моєму випадку – технології Xamarin.Forms. Такий підхід забезпечує достатню швидкість розробки, гнучкість у масштабуванні застосунку, можливість легкого розгортання його на різних платформах, а також спрощення процесу тестування та підтримки проєкту. Детально проаналізувавши існуючі програмні рішення, було визначено основний функціонал майбутнього застосунку, який включає:

- реалізація стандартних CRUD операцій;
- робота за задачами різних типів;
- побудову статистики виконання у різних форматах;
- зміну локалізації та теми інтерфейсу;
- збереження даних локально;
- підтримку гнучких сповіщень в автономному режимі.

Реалізація такого функціоналу потребує проходження низки взаємопов’язаних етапів розробки.

Початковим етапом є вибір відповідного програмного забезпечення, середовища розробки та апаратного забезпечення, необхідного для реалізації повного циклу створення додатку. На цьому ж етапі здійснюється ознайомлення з теоретичними аспектами кросплатформної розробки в середовищі Xamarin.Forms, що включає вивчення мови програмування C#, архітектурного шаблону MVVM, принципів написання платформозалежного коду та інтеграції сторонніх бібліотек і компонентів. Окремо потрібно визначити, які додаткові компоненти необхідні для обраного функціоналу, будь то графіки, бази даних, сповіщення чи фонові процеси, та обрати найкращий і найбільш підходящий варіант серед аналогів.

Важливим кроком є планування загальної структури майбутнього застосунку. На цьому етапі необхідно створити детальну модель логіки та інтерфейсу користувача, продумати схему взаємодії між окремими її частинами. Зокрема, планується створення головного екрана з навігаційними елементами, який буде виводити всю необхідну інформацію по задачах та забезпечить швидкий доступ до основних функцій програми. Також проєкт передбачає окремі вікна для реалізації функціоналу створення нових задач, їх редагування, вікна для налаштувань зміни зовнішнього вигляду та локалізації застосунку, а також вікна перегляду статистики продуктивності користувача у вигляді різних графіків з подальшим їх аналізом.

Наступним етапом є безпосередня реалізація коду – написання логіки для запланованого функціоналу застосунку за допомогою C# та Xamarin.Forms. Тут здійснюється побудова інтерфейсів, реалізація моделі представлення даних, обробка взаємодій користувача, інтеграція з локальною базою даних, а також підключення сторонніх компонентів, які допомагають пришвидшити розробку або реалізувати складні функції. Також доведеться реалізовувати платформозалежний код для специфічного функціоналу, такого як локальні повідомлення або кастомізація платформних елементів.

Передостаннім етапом є тестування, налагодження та оптимізація застосунку. Він включає перевірку стабільності роботи застосунку на різних пристроях і платформах, тестування на виявлення помилок і коректності роботи при різних сценаріях. У разі виявлення проблем або помилок здійснюється оптимізація структури або виправлення технічних помилок.

Завершальним етапом розробки є оформлення технічної та користувацької документації. У технічному описі детально викладається загальна концепція застосунку, його структура, архітектурні принципи побудови, основні компоненти програми та їх взаємодія. Користувацька документація містить покрокові інструкції щодо налаштування та використання додатку, а також опис усіх функцій і можливостей даного програмного рішення.

2 ОГЛЯД ТА ВИБІР ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Методи розробки мобільних застосунків

Перед безпосереднім початком роботи у розробників завжди постає питання щодо вибору методу розробки застосунку. Для мобільних застосунків це питання є особливо важливим, оскільки мобільні платформи, такі як Android чи iOS, однаково популярні та зовсім не схожі одна на одну. Це рішення є одним із ключових архітектурних виборів, що закладає фундамент усього проєкту та впливає на його життєвий цикл у довгостроковій перспективі [4]. Тому, щоб не втратити половину потенційних користувачів, варто розробляти програмне рішення під обидві платформи одночасно.

Вибір методу розробки визначає технологічні обмеження, продуктивність, витрати на підтримку, а також часові та фінансові ресурси, необхідні для створення та підтримки програмного рішення. Залежно від архітектурного підходу, застосунки поділяються на три основні типи: нативні, кросплатформні та гібридні [5]. Кожен із цих підходів має власну специфіку реалізації, набір технологій, а також переваги й недоліки, що впливають на вибір при конкретних умовах розробки [6].

2.1.1 Нативна розробка

Нативна розробка – напевне, найпопулярніший і найбільш традиційний метод розробки. Він передбачає створення програмного забезпечення безпосередньо під певну мобільну платформу, Android або iOS, із використанням офіційно підтримуваних мов програмування та Software Development Kit. Для платформи Android розробники зазвичай застосовують мови Java або Kotlin, та працюють у середовищі Android Studio, тоді як для

iOS використовується Swift або Objective-C в середовищі Xcode. Цей підхід забезпечує найвищий рівень оптимізації та повний, безперешкодний доступ до всіх функціональних можливостей пристрою, як-от GPS, камера, акселерометр та інші, які іноді недоступні або обмежені в застосунках, розроблених іншим підходом.

Основною перевагою нативної розробки є висока продуктивність, стабільність та ідеальна інтеграція з операційною системою. Завдяки цьому UX стає максимально плавним і природним, адже інтерфейс відповідає стандартним тайлайнам платформи, що особливо важливо для застосунків із великим навантаженням або складною анімацією, наприклад мобільних ігор [7]. З іншого боку, головним недоліком є потреба у паралельній розробці для кожної платформи, що значно збільшує витрати часу та бюджету й ускладнює подальшу підтримку. Команда розробників повинна мати окремих фахівців із досвідом розробки саме для Android та iOS, що також суттєво впливає на загальний бюджет проєкту.

Прикладами є майже всі застосунки від великих компаній, такі як Instagram, WhatsApp, сервіси Google, Uber та інші. Враховуючи ресурси компаній, які займаються розробкою та підтримкою подібних рішень, їх створюють нативним методом задля досягнення найвищого рівні UX, та досягнення максимально можливої ефективності, надійності та конкурентоспроможності на ринку.

2.1.2 Кросплатформна розробка

Кросплатформний підхід кардинально відрізняється від нативного, оскільки він дає змогу створювати застосунки для кількох операційних систем одночасно, використовуючи єдину кодову базу [8]. Серед найпоширеніших інструментів для кросплатформної розробки можна відзначити Flutter, React Native, а також Xamarin. Ці фреймворки дозволяють створювати застосунки, рівень яких майже не поступається нативним

рішенням, забезпечуючи зручне, майже повторне використання коду та значне скорочення витрат на розробку [9]. Це досягається завдяки тому, що фреймворк містить міст, який транслює універсальний код у нативні команди для кожної ОС. Хоча цей механізм є ефективним, сам процес трансляції може створювати невеликі затримки у виконанні коду, що є однією з причин можливого зниження продуктивності.

Головною перевагою є економія часу та коштів, адже одна команда розробляє застосунок одночасно для Android та iOS. Також оновлення або виправлення помилок відбуваються синхронно для обох платформ, що значно покращує керованість продуктом. Утім, такі застосунки можуть поступатися нативним у плані продуктивності, особливо якщо мають високу складність або потребують тісної взаємодії з апаратними функціями. У деяких випадках розробникам доводиться реалізовувати окремі платформозалежні функції нативно, що частково нівелює переваги підходу. Це вимагає додаткових знань у нативній розробці та ускладнює структуру проєкту. Крім того, оновлення операційних систем можуть порушувати роботу деяких функцій до виходу відповідних оновлень фреймворку.

Детально розглянувши платформи розробки, можна сказати що, Flutter використовує мову Dart та рендерить UI за допомогою власного графічного рушія Skia, забезпечуючи однаковий вигляд інтерфейсу на всіх платформах.

React Native, навпаки, використовує JavaScript і напряму взаємодіє з нативними UI-елементами платформи, що дає змогу досягати більш природного вигляду інтерфейсу.

Xamarin, заснований на C# та .NET, дозволяє з легкістю розробляти застосунки для iOS, Android та Windows.

Прикладами кросплатформних застосунків можна назвати продукти компаній-розробників відповідних фреймворків, такі як Facebook, написаний на React Native, чи Alibaba, написаний на Flutter. Також кросплатформні додатки часто створюють стартапи та невеликі компанії, щоб заощадити час та людські ресурси під час їх розробки.

2.1.3 Гібридна розробка

Гібридні застосунки створюються з використанням стандартних вебтехнологій, таких як HTML, CSS та JavaScript. По суті, це вебсайт, який обгортається в нативну оболонку, яка називається контейнер, і запускається всередині компонента WebView [10]. Ця оболонка забезпечує доступ до нативного API за допомогою таких фреймворків, як Apache Cordova, Ionic або PhoneGap, які надають спеціальні плагіни для взаємодії з функціями пристрою, такими як камера, або контакти. Цей підхід дозволяє створювати застосунки максимально швидко, з використанням мінімуму ресурсів, без потреби глибоко занурюватись у специфіку мобільних ОС.

Серед переваг можна відмітити велику простоту реалізації, низькі вимоги до технічної бази та найшвидший запуск готового рішення. Такий підхід ідеально підходить для створення простих застосунків, прототипів або програм, де основний функціонал дублює вже наявний вебсайт. Проте в даних рішеннях зазвичай не найкраща продуктивність, оскільки вона напряму залежить від потужності системного компонента WebView. Будь-які складні обчислення або анімації можуть призводити до помітних затримок. Користувацький інтерфейс може виглядати менш природно, ніж у нативних або кросплатформних аналогів, а доступ до розширених можливостей ОС може бути ускладненим або неможливим.

З прикладів подібних застосунків можна назвати деякі програми для онлайн-банкінгу, де значна частина інтерфейсу є вебсторінкою, або сервіси, що запускалися в обмежені терміни, наприклад, бета-версії Instagram. Також це популярний вибір для застосунків, основний функціонал яких представлений у вебверсіях, наприклад, сервіси з доставки або онлайн-магазини.

2.2 Огляд використаного програмного забезпечення

2.2.1 Платформа Xamarin.Forms

Xamarin.Forms – це високорівнева бібліотека та фреймворк для створення кросплатформних мобільних застосунків. Вона дозволяє розробляти інтерфейси користувача, які працюють на платформах Android, iOS і Windows із використанням єдиного базового коду мовою C#. Однією з найважливіших особливостей Xamarin.Forms є використання декларативної мови розмітки XAML для створення UI-компонентів, що дозволяє реалізувати потужний механізм прив'язки даних [11]. Цей підхід забезпечує чітке відокремлення логіки від представлення, значно спрощуючи процес розробки, тестування й подальшої підтримки додатка.

Xamarin.Forms суттєво знижує витрати часу та ресурсів, адже розробники можуть писати спільний код, адаптивний до різних платформ. Це забезпечує швидкість розробки та знижує ризики, пов’язані з дублюванням роботи. Крім того, Xamarin.Forms дозволяє легко інтегрувати нативні функції пристроїв через бібліотеку Xamarin.Essentials, яка надає єдиний API для доступу до камери, геолокації, датчиків руху, файлової системи та інших апаратних можливостей [12]. Ще однією вагомою перевагою є активна підтримка від Microsoft і велика спільнота розробників, що надає доступ до широкого спектра ресурсів.

Функціоналу Xamarin.Forms зазвичай вистачає для створення безлічі різнорідних додатків, що покривають найрізноманітніші потреби користувачів. Це можуть бути як прості ігри, так і структурно складні застосунки, наприклад, корпоративні системи, банківські клієнти або онлайн-магазини (рисунок 2.1).

Однією з ключових переваг Xamarin.Forms є вбудована підтримка архітектурного патерну MVVM. Цей патерн допомагає організувати код, де Model представляє дані, View – користувацький інтерфейс, а ViewModel

виступає посередником, що містить логіку відображення. Такий підхід робить код чистішим, модульним і зрозумілішим, що значно спрощує тестування та підтримку великих проєктів. Платформа підтримує широку бібліотеку попередньо налаштованих елементів інтерфейсу, таких як різні кнопки та перемикачі, текстові поля, списки та багато іншого, що прискорює створення прототипів та дозволяє виконувати базовий функціонал. Для складніших вимог розробники можуть створювати власні кастомні елементи за допомогою механізму Custom Renderers, який дозволяє перевизначити відображення елемента для кожної платформи окремо.

Порівнюючи Xamarin.Forms з іншими популярними кросплатформними рішеннями, такими як Flutter чи React Native, варто відзначити низку переваг. На відміну від Flutter або React Native, Xamarin.Forms побудовано на базі C# та тісно інтегровано з екосистемою .NET. Це дозволяє не тільки використовувати знайомі інструменти, а й спільно використовувати бізнес-логіку між мобільним застосунком, вебсервером ASP.NET та настільними програмами. Платформа забезпечує повний доступ до нативних API без складних “мостів”. Саме поєднання ефективної розробки, гнучкості, стабільності та глибокої інтеграції з .NET стало вирішальним фактором на користь вибору цієї технології.

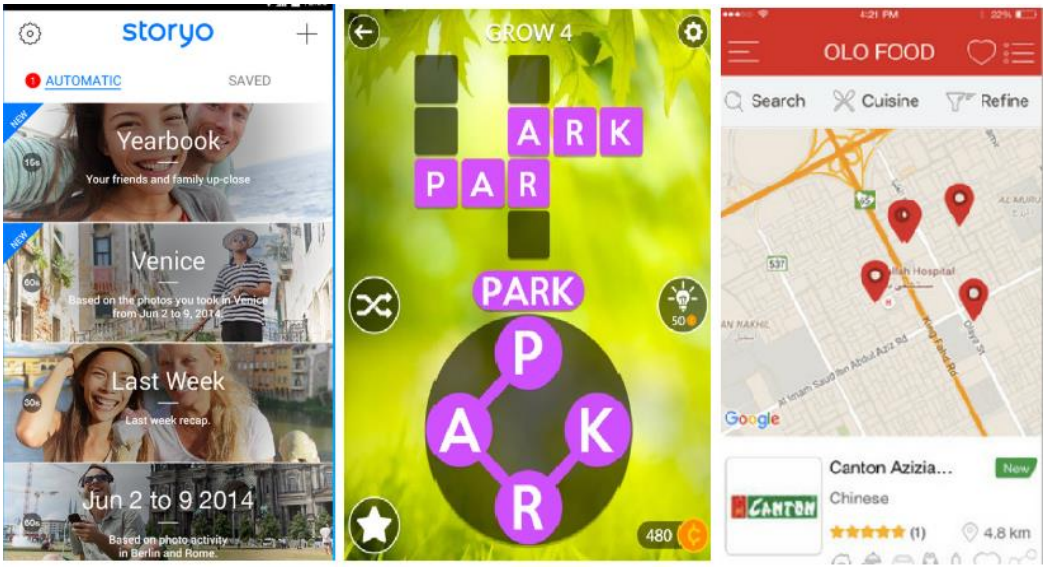


Рисунок 2.1 – Приклади застосунків розробених на Xamarin.Forms

2.2.2 Мова програмування C#

Мова програмування C# стала основою для розробки застосунку завдяки своїм численним перевагам у сфері об’єктно-орієнтованого програмування та широкому набору функціональних можливостей. Розроблена корпорацією Microsoft, ця мова зарекомендувала себе як сучасна, безпечна, досить ефективна і продуктивна платформа для створення додатків будь-якого рівня складності, від простих мобільних і десктопних утиліт до великих корпоративних систем і високонавантажених сервісів [13].

Однією з головних переваг C# є повна підтримка об’єктно-орієнтованої парадигми, що дозволяє створювати добре структуровані та масштабовані програмні рішення, що забезпечує можливість повторного використання коду, полегшує його розширення, що важливо для довготривалих і складних проєктів. Завдяки статичній типізації C# вдається виявляти багато логічних помилок на етапі компіляції, що підвищує надійність і стабільність застосунку. Підтримка інкапсуляції, наслідування та поліморфізму спрощує розробку складних бізнес-логік та забезпечує гнучкість у масштабуванні проєкту.

Особливу увагу можна приділити інтеграції C# з екосистемою .NET, що надає доступ до широкого спектра інструментів розробки, зокрема Visual Studio. Це середовище значно полегшує процес кодування, налагодження, тестування та розгортання додатків, завдяки інтелектуальним підказкам, емуляторам мобільних пристроїв і багатьом іншим можливостям. Також C# підтримує сучасні парадигми програмування: лямбда-вирази, делегати, анонімні методи, асинхронне програмування, а також інтегровані запити LINQ, що дозволяє ефективно працювати з колекціями та базами даних.

У контексті використання Xamarin.Forms як основного фреймворку для розробки власного додатку, C# виступає як невід’ємна частина платформи. Xamarin.Forms побудований на основі .NET і оптимізований саме для роботи з C#, що забезпечує природну інтеграцію з усіма його можливостями.

Завдяки цьому розробники мають змогу створювати єдиний код користувацького інтерфейсу та логіки, який працює на Android, iOS та інших платформах, що скорочує час і витрати на розробку, тестування й підтримку застосунку. Крім того, Xamarin.Forms підтримує MVVM-архітектуру, що дає змогу чітко розмежувати логіку програми, інтерфейс користувача та модель даних.

C# також має багату екосистему сторонніх бібліотек і інструментів, доступних через менеджер пакетів NuGet, що дозволяє легко інтегрувати нові функції та розширяти можливості застосунку без написання складного коду з нуля, а використовувати вже перевірені готові рішення.

Крім того, C# має широку та активну спільноту розробників, що значно спрощує розробку, забезпечує доступ до різних безкоштовних навчальних матеріалів та прикладів готових рішень. У результаті, можна сказати, що застосунок, створений на C# у поєднанні з Xamarin.Forms, є функціональним, надійним, легко підтримуваним та готовим до масштабування або розширення відповідно до майбутніх потреб і чудово підходить під технічні вимоги застосунку.

2.2.3 Бібліотека SQLite

Бібліотека SQLite – це вбудована реляційна база даних, яка не потребує окремих налаштувань для роботи. Вся інформація зберігається у вигляді одного локального файлу на пристрої користувача, тому вона є важливим компонентом у розробці невеликих додатків, оскільки забезпечує ефективне, надійне та просте зберігання даних без необхідності в зовнішньому сервері [14]. Така архітектура є особливо корисною для мобільних додатків, де важлива автономність, швидкодія та мінімальне споживання ресурсів. Завдяки повній підтримці SQL-стандарту, SQLite дозволяє працювати з базою даних за допомогою знайомих запитів: SELECT, INSERT, UPDATE, DELETE, які спрощують розробку й обслуговування застосунку.

Великим плюсом є легка інтеграція SQLite у Xamarin.Forms через бібліотеку SQLite-net-pcl. Це дозволяє описувати дані, наприклад задачі, у вигляді класів і виконувати операції з ними через прості асинхронні методи, такі як додавання, редагування або видалення елементів, а також фільтрацію за допомогою LINQ-запитів. Усі дані автоматично зберігаються локально, що дає змогу працювати з додатком навіть без підключення до Інтернету.

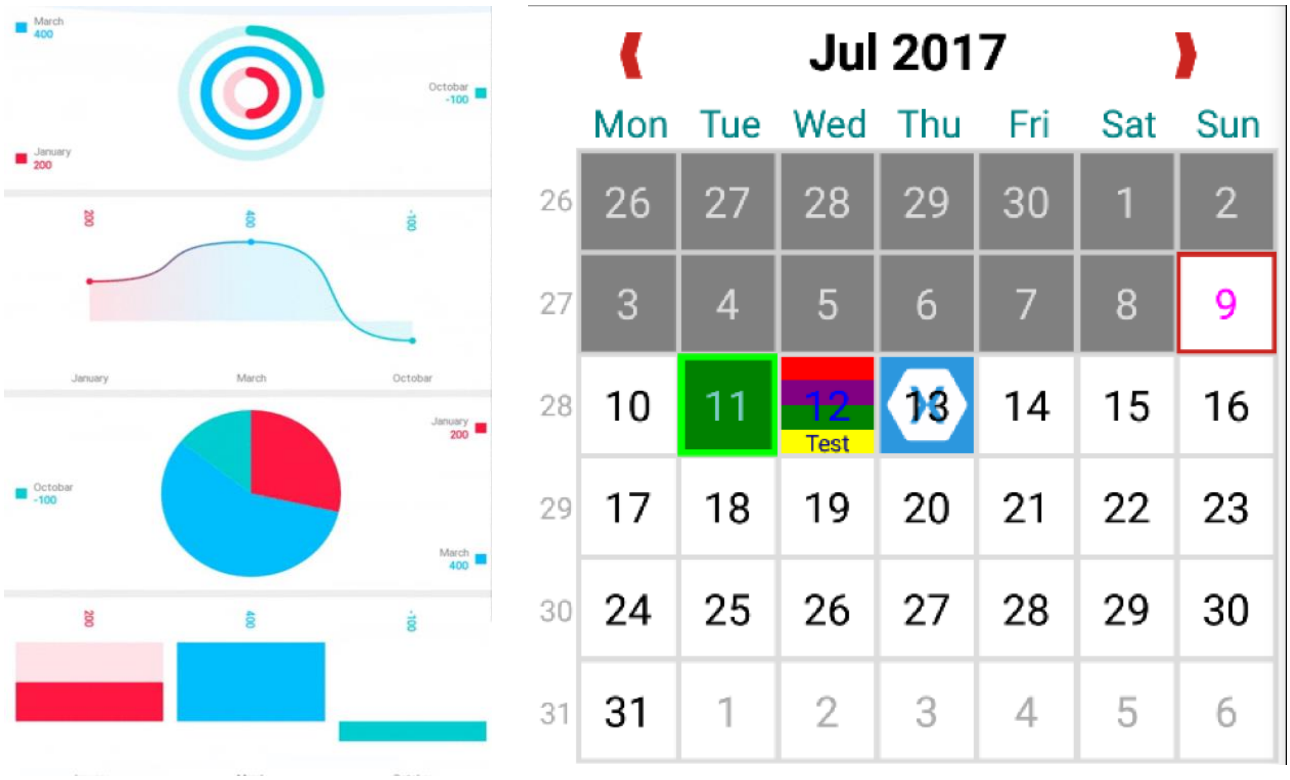
SQLite має низку переваг над іншими популярними рішеннями для локального зберігання, такими як Realm, LiteDB, Room або CoreData. На відміну від інших рішень, які мають власний формат зберігання, SQLite використовує універсальну SQL-мову і сумісний формат, що значно спрощує міграцію, діагностику та аналіз даних. Більшість варіантів-аналогів обмежені лише платформою, будь то Android або iOS, тоді як SQLite є повністю кросплатформним. Інші рішення складно інтегруються з Xamarin. Саме тому SQLite вирізняється своєю універсальністю, компактністю, простотою інтеграції та повною відповідністю вимогам сучасних мобільних рішень.

У контексті мого застосунку SQLite використовується як основне сховище для задач. При запуску застосунку створюється або відкривається файл бази даних. Далі всі операції, будь то додавання, редагування, видалення чи перегляд задач, відбуваються через взаємодію з цим файлом. Даний підхід є значно кращим, ніж збереження в файлі за допомогою серіалізації, та забезпечує високу швидкодію, збереження даних навіть при закритті застосунку, а також просту реалізацію інтерфейсу з фільтрацією та групуванням задач.

Загалом, можна сказати, що використання SQLite разом із Xamarin.Forms дозволить реалізувати стабільну й зручну систему зберігання задач, яка поєднує кросплатформність, продуктивність і простоту підтримки. Це оптимальний вибір для мобільного середовища, де важливо мати контроль над даними, можливість працювати офлайн, що знадобиться для системи сповіщень, та забезпечити швидкий відгук інтерфейсу без складної архітектури.

2.2.4 Бібліотека Microcharts та Xamarin.Plugin.Calendar

Бібліотеки Microcharts та Xamarin.Plugin.Calendar – це дві популярні, повністю безкоштовні бібліотеки, які добре підходять для мобільних застосунків, створених у Xamarin.Forms. Головна їх задача – це створювати візуальні елементи для застосунку, що значно покращує візуальне сприйняття користувача (рисунок 2.2). Кожна з них виконує окрему функцію, і їх можна вдало комбінувати в одному проєкті, забезпечуючи як візуалізацію даних в різних форматах, так і гнучку інтерактивну роботу з календарем. Комбінація цих структурних елементів особливо влучна для застосунків, орієнтованих на роботу із задачами, оскільки вони майже повністю закривають потребу у візуалізації завдань та розподіленні їх за категоріями.



а) б)

Рисунок 2.2 – Приклад реалізації компонентів в бібліотеці: а) Microcharts; б) Xamarin.Plugin.Calendar

Перша розглянута бібліотека, Microcharts, це легка кросплатформна бібліотека для побудови різних типів діаграм та графіків. У бібліотеці реалізовано підтримку різних типів діаграм: кругових, стовпчикових, лінійних, прогресивних тощо. Є налаштування кольорів, стилів шрифтів і розмірів графіків, що дозволяє розробникам адаптувати їхній вигляд під стиль програми. Бібліотека базується на SkiaSharp, яка забезпечує якісне рендерення на рівні нативної графіки [15]. Крім того, бібліотека підтримує анімацію, що дозволяє робити графіки більш інтерактивними й візуально привабливими.

Microcharts чудово підходить для візуалізації статистики, пов’язаної з продуктивністю, кількістю виконаних задач, розподілом часу та іншими параметрами. Завдяки простоті інтеграції її можна легко зв’язати із збереженими даними в локальній базі даних SQLite.

Серед причин вибору саме Microcharts – її простота, мінімалістичний стиль та нативна підтримка Xamarin.Forms без додаткових платформозалежних налаштувань. Вона не має зайвої складності, ідеально підходить для швидкої реалізації візуального компонента без перевантаження додатку. Також бібліотека на відміну від аналогів повністю безкоштовна та з відкритим вихідним кодом.

Інша бібліотека, Xamarin.Plugin.Calendar, це простий плагін для відображення календаря з можливістю взаємодії з подіями в межах місяця. Серед безкоштовних рішень немає варіантів з підтримкою погодинного планування, проте як аналог, Xamarin.Plugin.Calendar зручний для створення календаря де є можливість робити позначки задач або підписів у комірках.

У випадку сворованного застосунку, Xamarin.Plugin.Calendar використовується для перегляду задач у форматі календаря на місяць. Такий підхід в поєднанні з засобами візуалізації та методами кастомізації самого календаря дозволить зручно та швидко надавати поденну й потижневу оцінку задачам, та за результатами відображення формувати поради щодо оптимізації та перепланування робочого тижня або місяця.

2.2.5 Бібліотека Shiny

Shiny – потужна та популярна кросплатформна бібліотека для Xamarin.Forms, створена для спрощення роботи з низкою складних системних функцій, які зазвичай потребують платформозалежної реалізації. Вона спеціально розроблена для того, щоб звільнити розробника від необхідності писати громіздкий, дублюючий і важко підтримуваний код окремо для Android та iOS. Завдяки уніфікованому API, Shiny дозволяє використовувати сучасні системні сервіси в мобільних додатках через зручний і зрозумілий інтерфейс [16]. Це робить її особливо актуальною для створення продуктивних застосунків, зокрема тих, що потребують надійної логіки фонової роботи та нагадувань.

Shiny включає широкий спектр функціональних можливостей, серед яких ключовими є:

- фонові задачі;
- локальні повідомлення;
- таймери та планування завдань;
- геолокація та геофенсінг;
- Bluetooth LE.

Якщо розглядати більш детально, то Shiny дозволяє створювати процеси, які можуть виконуватися навіть у фоні, коли застосунок закритий. Це важлива можливість для застосунків, де потрібно виконувати періодичні дії, наприклад оновлення сповіщень у режимі реального часу. Для менеджера задач ця функція дає змогу забезпечити стабільну систему нагадувань без залежності від активності користувача в додатку.

Також Shiny надає можливість надсилати локальні повідомлення з розширеними налаштуваннями, включаючи час сповіщення з можливістю його відкладання, канали сповіщень, кнопки дій, групування та інші. Такі сповіщення можуть легко запускатися в фонових задачах, створюючи інтегровану систему нагадувань. У випадку із застосунком для менеджменту

та аналізу задач це дозволяє повідомляти користувача про наближення дедлайнів чи запланованих подій, або про його розпорядок на день. Також можна налаштувати циклічне повторення цих подій, наприклад щогодини, щодня або через вказаний інтервал, що дає змогу реалізувати задачі, які потребують періодичної обробки. Для задачника – це ідеальне рішення для циклічного оновлення даних або повторних нагадувань.

Іншими функціями бібліотеки є геолокація та геофенсінг, що значно спрощують роботу з координатами користувача, та дозволяють не лише отримувати поточне місце розташування, а й запускати певні дії при вході або виході з заданої зони. Bluetooth LE дає змогу сканувати пристрої поблизу, з’єднуватися з ними та обмінюватися данами. Хоч ці функції й не використовуються в контексті розробки даного програмного рішення, проте є дуже потужними та важливими інструментами мобільної розробки загалом.

Крім того, Shiny може реагувати на події, пов’язані з запуском, згортанням або зупинкою застосунку. Це дає змогу зберігати стан задач, запускати автоматичні збереження або поновлення під час зміни стану.

У контексті застосунків, що стосуються управління задачами, Shiny найчастіше використовується для реалізації системи нагадувань. Завдяки комбінації фонових задача, сповіщень та можливості циклічного повторення, дана бібліотека забезпечує високу надійність і безперервність функціональності, що є критично важливим для додатків, які мають допомагати людям не пропускати важливі справи.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАСТОСУНКУ

3.1 Опис структури програми

Для вирішення поставлених задач в даному проєкті був використаний фреймворк Xamarin.Forms. У загальному вигляді, структура проєкту передбачає наявність чотирьох основних XAML-файлів, кожен з яких відповідає за окреме вікно користувацького інтерфейсу. До цих файлів підключено відповідні обробники подій, у яких реалізовано основну логіку програми, та відбувається виконання ключового функціоналу застосунку. Крім основних вікон, проєкт включає в себе набір із восьми спливаючих вікон. Призначення даних вікон є надзвичайно важливим, адже саме через них реалізується розширення можливостей основних інтерфейсів, вони забезпечують зручне відображення структурних елементів, які, з міркувань зручності чи організації коду, були винесені за межі головних вікон.

Особливу роль у проєкті відіграють файли, що містять моделі для двох локальних баз даних, а також сервіси, які реалізують логіку взаємодії програми з цими базами. Саме через ці сервіси забезпечується обмін даними між інтерфейсом користувача та внутрішнім сховищем, що дозволяє зберігати, змінювати та отримувати інформацію в автономному режимі.

Не менш важливою складовою проєкту є ресурсні файли, у яких описано визначення світлої та темної схем оформлення інтерфейсу, які дозволяють динамічно адаптувати вигляд застосунку. Крім цього, проєкт містить .resx-файли, які використовуються для зберігання текстів кількома мовами. Для керування цим процесом у проєкті передбачена спеціальна модель, яка відповідає за динамічну зміну мови застосунку відповідно до вибору користувача.

Іншими важливими файлами в застосунку є:

- App.xaml та App.xaml.cs які визначають глобальні стилі, початкові

налаштування, та проводять ініціалізацію початкового вінка MainPage;

- модель для встановлення текстових іконок;
- файл AssemblyInfo.cs що використовується для зберігання метаданих про збірку;
- модель з нотифікаціями та створення фонових процесів.

Крім основних складових, структура проєкту передбачає використання окремого XAML-компонента для реалізації кастомного рендеру елемента вибору часу, що підвищує зручність взаємодії з додатком. Також у проєкті реалізовано механізм конвертації дат у зручний для користувача формат, що дозволяє модифікувати відображення для різних типів задач. Візуальні ресурси, як-от шрифти та іконки, організовані у відповідні файли для централізованого доступу.

В загальному, проєкт було організовано таким чином, щоб максимізувати зручність навігації по ньому, а також щоб код був гнучким та легко модифікувався і розширявся (рисунок 3.1).

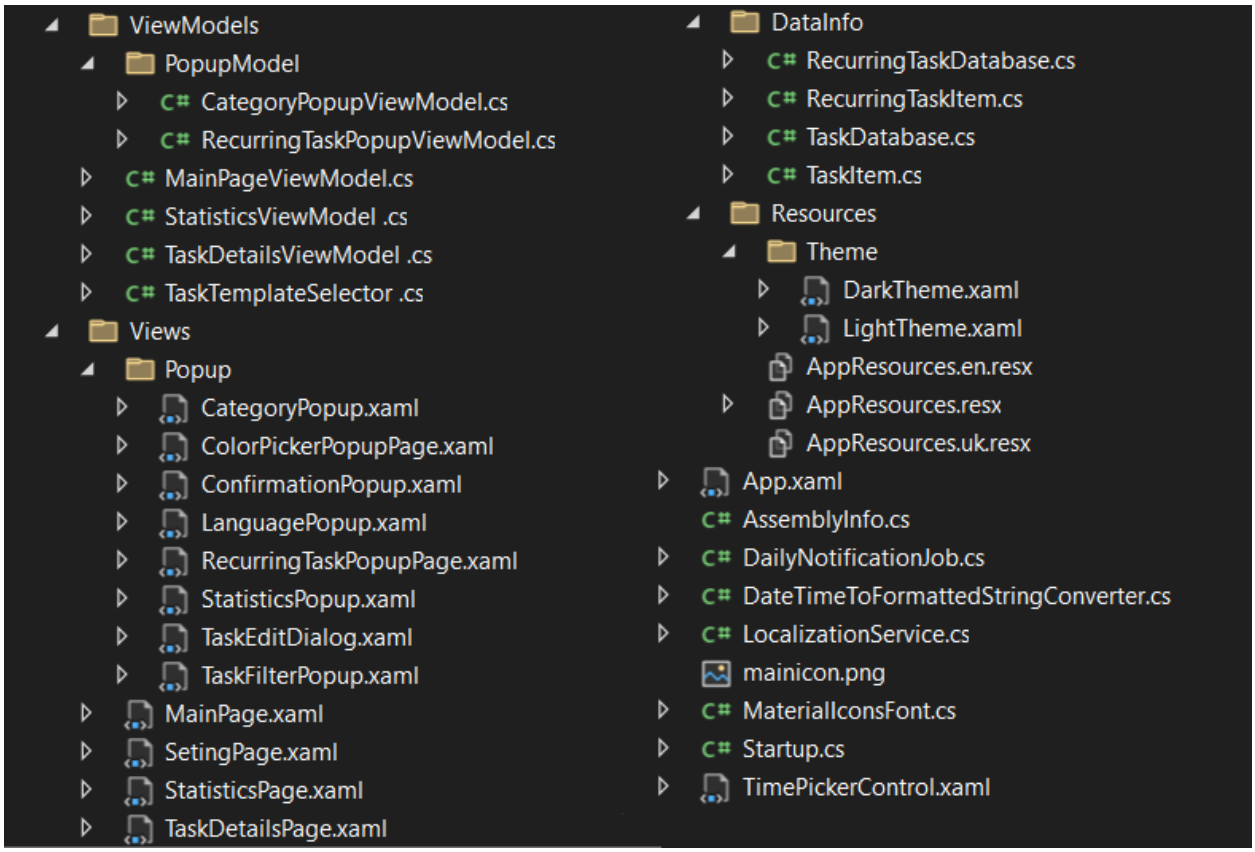


Рисунок 3.1 – Загальна структура проєкту

3.2 Аналіз функціоналу головних вікон

3.2.1 Вікно MainPage

Вікно MainPage виконує роль стартового вікна додатка, яке організовує і відображає завдання користувача. Його функціональність сфокусована на управлінні списком завдань, їх сортуванні, редагуванні, взаємодії та переключенні між типами завдань. Вікно складається з кількох основних елементів, які забезпечують ефективність роботи.

У верхній частині вікна знаходиться Toolbar із кнопкою пошуку та фільтрації за параметрами. Пошук відкриває текстове поле для введення запиту, дозволяючи користувачеві шукати завдання за назвою, тегами або ключовими словами. Кнопка фільтрації ж викликає відповідний попап, який надає можливість відфільтрувати задані задачі за їх статусом виконання, а також по категорії самої задачі. При цьому можна застосовувати одразу декілька фільтрів різних типів для пошуку.

Основна частина вікна – це область заокругленої прямокутної форми, де знаходиться перемикач між списками звичайних та циклічних задані задач, відображення яких реалізовано через компонент CollectionView (лістинг 3.1).

Лістинг 3.1 – Сортування задач відповідно до обраного типу

```
private async void SortTasks()
{
    if (!isRecurringMode){
  var list = await Task.Run(() => App.Database.GetItemsAsync());
  GroupedItems = new ObservableCollection<Grouping<string,
TaskItem>>(grouped);});
    }
    else{  var list = await Task.Run(() =>App.RecurringDatabase.
GetItemsAsync());
     GroupedItems = new ObservableCollection<Grouping
<string, RecurringTaskItem>>(grouped);});
    }
}
```

Самі задачі представлені у вигляді карток, оформлених за допомогою Frame, та розподілених сортуванням по за категоріями задач. Поле задач і загальному включають інформацію про назву завдання, умову виконання, стан виконання задачі та доступні з нею операції. Також картка задачі включає в себе кілька інтерактивних елементів таких як:

- кнопка «Змінити опис» що викликає діалогове вікно яке дозволяє користувачу змінювати опис та назву завдання;
- кнопка «Видалити» пропонує видаленням вибраного завдання;
- перемикач який дозволяє змінювати статус задачі (виконано/не виконано) для звичайних, і кнопка про відмітку для циклічних задач.

У нижній частині вікна знаходяться кнопки навігації для відкриття вікон StatisticPage, TaskDetailPage або SettingPage.

Головною особливістю інтерфейсу стартового вікна є динамічне переключення між типами задач та їх динамічне сортування всередині одного типу при зміні характеристик або властивостей якоїсь задачі, будь то зміна дати, опису, статусу, видалення завдання або створення нового завдання. Всередині типу, задачі розподіляються за групами: «Задачі на сьогодні», «Невиконані задачі», «Виконані задачі», «Прострочені задачі» для звичайних задач, та на «Активні задачі» і «Неактивні задачі» для циклічних типів задач. Завдяки цьому користувач завжди має структурований і впорядкований список завдань, у яких легко орієнтуватися.

У разі відсутності задач у активному типі в списку буде відображатися повідомлення: «Список задач пустий», яке має спонукати користувача до створення нової активності.

Додатково, вікно інтегрує функціональність пошуку, що дозволяє швидко знаходити потрібні завдання навіть у великому списку. Всі ці функції роблять інтерфейс інтуїтивно зрозумілим і орієнтованим на продуктивність, допомагаючи користувачу ефективно організовувати свою роботу.

3.2.2 Вікно SetingPage

Вікно SetingPage у мобільному застосунку створено для гнучких налаштувань. Його основна мета полягає у наданні можливості змінювати ключові параметри зовнішнього вигляду й поведінки застосунку безпосередньо через інтерфейс. Це вікно є важливою складовою зручності використання застосунку, оскільки забезпечує користувачам доступ до керування темою інтерфейсу, мови локалізації та параметрами сповіщень.

Візуально сторінка оформлена у вигляді згрупованих горизонтальних блоків, що містять іконки, підписи та інтерактивні елементи взаємодії з параметрами налаштування.

Один із таких рядків дозволяє користувачеві змінити кольорову тему програми. Натисканням на відповідний елемент відкривається попап вибору теми та кольорового рішення застосунку. Якщо колір змінюється, застосовується нове значення у словнику ресурсів і оновлюється зовнішній вигляд інтерфейсу поточного вікна для передогляду нового встановлених змін.

Наступний блок налаштувань відповідає за вибір мови інтерфейсу. Після натискання на відповідний елемент викликається метод, що ініціює зміну мови через сервіс локалізації. Завдяки реалізації прив’язки даних і використанню сервісу локалізації, вікно забезпечує динамічне оновлення текстів при зміні мови та підтримку мультимовності. Усі текстові ресурси автоматично оновлюються, відображаючи нову мову, але зміни набувають чинності лише після підтвердження користувачем.

Ще одна важлива функція вікна налаштувань – керування сповіщеннями. Користувач може активувати або деактивувати цю функцію за допомогою перемикача Switch або через GestureRecognizers шару, де він знаходиться. Якщо увімкнути сповіщення, з’являється прихований за замовчуванням елемент TimePickerControl, який дозволяє вибрати точний час щоденного сповіщення.

Зміна будь-якого з встановлених параметрів активує панель підтвердження збереження змін, вона з’являється в нижній частині екрана й містить кнопки «Зберегти» та «Скасувати». Збереження застосовує зміни для всього додатку та зберігає нові налаштування, скасування, відповідно, повертає попередньо встановлені значення (лістинг 3.2).

Лістинг 3.2 – Методи зберігання та скасування налаштувань

```
private async void OnSaveChanges(object sender, EventArgs e)
{
    Preferences.Set("NotificationTime",
NotificationTime.ToString(@"hh\:mm"));
    Preferences.Set("ThemeColor",
((Color)Application.Current.Resources["BarColor"]).ToHex());
    if (TimePicker.IsVisible == true)
    {OnToggleTimePicker(null, null);
    }
    LocalizationService.SaveLanguage(selectedLanguage);
    await Navigation.PopAsync();
}
private void OnCancelChanges(object sender, EventArgs e)
{
    Application.Current.Resources["BarColor"] = _previousColor;
    UpdateTheme();
    LocalizationService.InitLanguage();
    TimePicker.ResetTime();
    if (TimePicker.IsVisible == true)
    {OnToggleTimePicker(null, null);
    }
    NotificationTime = _previousTime;
    TimePicker.DeselectAll();;
    HideSavePanel();
}
```

Важливим аспектом реалізації є збереження даних у локальній пам’яті пристрою за допомогою Preferences. Це дозволяє при відкритті сторінки автоматично відновити раніше обрані параметри: кольорову тему, мову та час сповіщення. Також вікно підтримує логіку скасування змін. Якщо користувач змінив будь-яке налаштування, але вийшов зі сторінки без підтвердження, йому буде запропоновано підтвердити або скасувати зміни через окремий popup. Таким чином, уникнуто втрати важливих налаштувань через неуважність.

Програмна логіка вікна реалізована з дотриманням принципів MVVM, з прив’язкою до властивостей у BindingContext і обробкою подій у коді сторінки. Крім того, вікно містить анімації появи та зникнення нижньої панелі логічних кнопок при змінах існуючих властивостей, які реалізовано через кастомну функцію AnimateHeight з плавною зміною висоти.

3.2.3 Вікно StatisticPage

StatisticsPage – це сторінка візуалізації статистичних даних, яка надає користувачам узагальнену інформацію про виконання задач у вигляді інтерактивного календаря та динамічних графіків. Це вікно дає змогу аналізувати продуктивність, кількість виконаних задач та розподіл активності по днях у вибраному часовому інтервалі. Його основна ціль – надати зручний інструмент для аналізу особистої ефективності, допомагаючи виявити, в які дні було найбільше навантаження або коли задачі систематично не виконувалися.

Інтерфейс складається з двох основних сторінок – діаграмної секції виводу статистики та календаря. Переключення між сторінками відбувається шляхом вертикального свайпу в робочій області, і містить плавну анімацію зміни відображення вікна, де попередня сторінка пропадає і починає відображатися нова (лістинг 3.3).

Лістинг 3.3 – Метод переключення сторінок вікна шляхом свайпу

```
private void OnPanUpdated(object sender, PanUpdatedEventArgs e)
{
    switch (e.StatusType)
    {
        case GestureStatus.Running:
            totalY = e.TotalY;
            break;
        case GestureStatus.Completed:
            if (Math.Abs(totalY) > 40)
            {
                if (totalY < 0 && Page1View.IsVisible)
                {
```

```
                    AnimateTransition(Page1View, Page2View, true);
                }
                else if (totalY > 0 && Page2View.IsVisible)
                {
                AnimateTransition(Page2View, Page1View, false);
                }
            }
                totalY = 0;
                break;
    }
}

private async Task AnimateTransition(View fromView, View toView,
bool swipeUp)
{
    const uint duration = 300;
    toView.TranslationY = swipeUp ? 100 : -100;
    toView.Opacity = 0;
    toView.IsVisible = true;
    await Task.WhenAll(fromView.FadeTo(0, duration),
fromView.TranslateTo(0, swipeUp ? -100 : 100, duration));
    fromView.IsVisible = false;
    fromView.TranslationY = 0;
    fromView.Opacity = 1;
    await Task.WhenAll(toView.FadeTo(1,
duration),toView.TranslateTo(0, 0, duration));
}
```

На першій сторінці для аналізу задач використовуються кругова та стовпцева діаграми, створені за допомогою бібліотеки Microcharts. Вони наочно відображають співвідношення задач у різних статусах. Кольори діаграм інтуїтивно зрозумілі: зелений позначає виконані задачі, червоний – прострочені, жовтий – ті, термін виконання яких наближається, а блакитний – задачі у процесі виконання. Водночас під діаграмою, в текстовому форматі виводиться кількість задач, що дозволяє швидко орієнтуватися в загальному стані справ. Дані для побудови графіків також беруться з локальної бази даних, і фільтруються відповідно до обраного діапазону. Сторінка автоматично реагує на зміни у діапазоні дат, миттєво оновлюючи статистику.

Іншою сторінкою є календар, реалізований за допомогою сторонньої бібліотеки XamForms.Controls.Calendar, що дозволяє налаштовувати візуальний вигляд окремих днів. Щоб відобразити завантаженість кожного дня, у вибраному діапазоні підраховується кількість задач для кожної дати.

Залежно від цього значення обчислюється прозорість кольору відображення – від 10 % для одного завдання до 90% при 5+ задачах на обраний день. Така реалізація забезпечує плавний градієнт, що дозволяє візуально швидко оцінити активність. Колір фону задається динамічно через властивість SpecialDates, яка передається до календаря у вигляді колекції об’єктів із відповідними датами та стилями.

При натисканні на певну дату виділяється тиждень, до якого вона належить, що реалізується через визначення першого дня тижня та генерацію списку з семи дат. Ці дати додатково підсвічуються, аби користувач міг бачити загальну активність за тиждень. Паралельно нижче, в CollectionView, відображається перелік задач на обрану дату, якщо вона є. Список отримується із SQLite-бази на основі вибраної дати. При цьому вибір дати автоматично викликає оновлення прив’язаної колекції задач у ViewModel. Для обраного тижня також формуються поради, як оптимізувати свій час порівняно з іншими днями, якщо в цьому є необхідність.

3.2.4 Вікно TaskDetailPage

TaskDetailsPage – це сторінка створення нових задач, яка забезпечує зручний і простий інтерфейс для користувачів, що дозволяє додавати задачі різного типу – будь то звичайні або циклічні – до списку. Вікно надає користувачеві загальні структурні поля для зміни параметрів задачі. Перше поле визначає назву задачі та є ключовим, адже воно визначає ідентифікатор задачі у списку. Другий елемент – це редактор для введення опису задачі. Він є опціональним, і користувач може залишити його порожнім, якщо задача не потребує додаткових коментарів. Третій елемент – це механізм вибору категорії задачі, яка визначається в спеціальному попапі. За замовчуванням вона створюється без категорії, проте за потреби її можна змінити на одну з переліку, який охоплює різні області активностей, що зустрічаються в повсякденному житті. Для пришвидшення процесу створення нової задачі,

від користувача потребується ввести лише її назву. При створенні такої задачі всі інші параметри встановлюються за шаблоном, і можуть бути змінені в будь-який момент у меню налаштувань задачі.

Інші параметри задачі залежать від типу самої задачі та динамічно перемикаються за допомогою кастомного слайдера. Так, при активному типі звичайних задач видимим стає DatePicker з вибором дедлайну задачі, де за замовчуванням встановлена сьогоднішня дата, а також поле TimePicker з кастомно реалізованим вибором часу, який є необов'язковим.

Дещо складніша логіка при виборі циклічної задачі. Перш за все буде показано поле вибору інтервалу, де за замовчуванням вказано повторення щодня. Для зміни параметра викликається попап, у якому реалізований кастомний рендер TimePicker, але додатково з можливістю вибору кількості днів. Іншим полем є вибір умови виконання задачі з двома варіантами вибору, реалізованими через CheckBox. При виборі одного з варіантів автоматично стають видимими поля налаштувань умов виконання вибраного типу, а інший CheckBox стає неактивним (лістинг 3.4).

При перемиканні між типами задач їх параметри зберігаються, проте всі багаторівневі вкладені елементи приховуються. Для створення самої задачі використовується модель з ViewModel вікна, там же вказані налаштування за замовчуванням для параметрів.

Лістинг 3.4 – Метод вибору параметрів завершення задачі

```
private void OnCheckedChanged(object sender,
CheckedChangedEventArgs e)
{
    if (!(sender is CheckBox selectedCheckBox))
        return;
    if (e.Value)
    {
        foreach (var view in CheckOptions.Children)
        {
            if (view is StackLayout layout && layout.Children[0]
is CheckBox cb)
            {
               bool isSelected = cb == selectedCheckBox;
               cb.IsEnabled = isSelected;
```

```
                cb.Color = isSelected ? Color.Green : Color.Gray;
                 if (isSelected)
                 {   if (cb == Option1)
                     {
                         viewModel.EndDate =
DateTime.Today.AddDays(2);
                         DateS.IsVisible = true;
                         B2.IsVisible = true;
                         date1.IsVisible = false;
                         date2.IsVisible = true;
                     }
                      else if (cb == Option2)
                      {
                         NumberS.IsVisible = true;
                         B2.IsVisible = true;

    } } } } }
    else
    {   foreach (var view in CheckOptions.Children)
        {
            if (view is StackLayout layout && layout.Children[0]
is CheckBox cb)
            {
                cb.IsEnabled = true;
                cb.IsChecked = false;
                cb.Color = Color.Default;
}    }    }    }
```

Для забезпечення коректності введених даних реалізовано механізм перевірки на клієнтській стороні. Після введення нових даних, якщо користувач не заповнив поле назви задачі, спробував встановити дату завершення у минулому або ввів у текстове поле дані, що не відповідають умовам формату, у вікні автоматично приховується кнопка створення задачі, а система генерує відповідні повідомлення про помилку, які відображаються у спеціальному ErrorLabel у вигляді червоного тексту під формою. Ці перевірки допомагають уникнути введення некоректних даних і забезпечують відповідність параметрів створюваної задачі.

3.3 Розгляд функціоналу спливаючих вікон

Не менш важливими за головні вікна є попапи. Їх використання є важливим елементом користувацького інтерфейсу в мобільному застосунку,

оскільки вони дозволяють реалізувати зручну взаємодію з користувачем без потреби переходу на окремі сторінки. Вони викликаються з головних сторінок та забезпечують швидкий доступ до нескладних об’єктів. Це може бути додаткова інформація, параметри чи дії, такі як редагування, вибір категорій, підтвердження дій або фільтрація – і все це без порушення основного потоку використання застосунку. Завдяки цьому покращується загальна зручність користування, зменшується навантаження на навігацію і підвищується ефективність роботи з задачами.

Майже в кожному з головних вікон є декілька попапів, які виконують свою специфічну роль для покращення взаємодії користувача з задачами.

CategoryPopup – попап, що відображає список доступних категорій, зв’язаних із CategoryPopupViewModel, та містить 12 категорій на вибір, розташованих в табличному форматі з вказаною назвою та іконкою. Він дозволяє вибрати одну з них (рисунок 3.2), після чого викликається подія CategorySelected із відповідним об'єктом. Попап закривається, а обрана категорія передається в основне вікно.

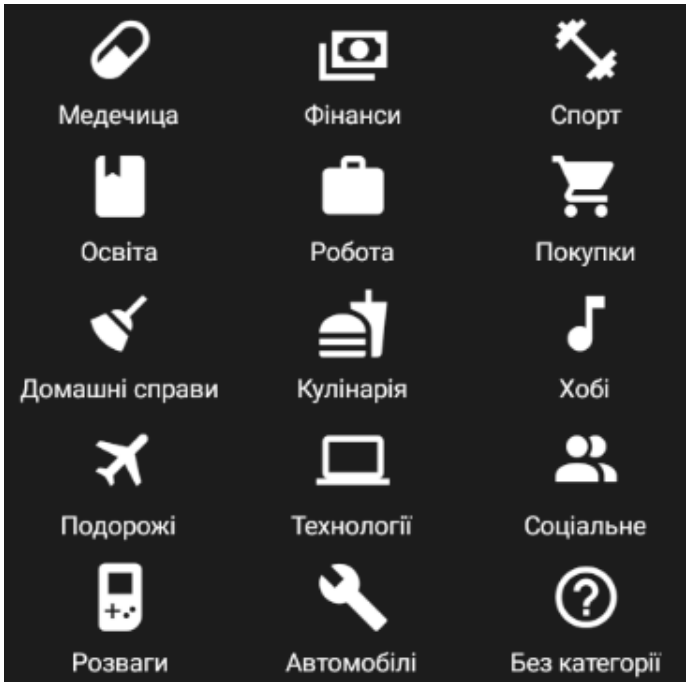


Рисунок 3.2 – Доступні для вибору категорії задач

ConfirmationPopup – простий попап підтвердження дії, наприклад, при видаленні або виході з застосунку. Він показує користувачу повідомлення з двома кнопками варіантами вибору: «Так» і «Скасувати». Після вибору відповідь повертається в логічній змінній bool, що дозволяє основному коду асинхронно дочекатися рішення користувача.

TaskFilterPopup – попап для фільтрації задач за обраними критеріями. Користувач може активувати фільтрацію за статусом задач, включаючи виконані, невиконані, на сьогодні та прострочені, або фільтрувати за категоріями. Усі обрані параметри зберігаються у відповідних списках і після натискання кнопки «Застосувати» передаються назад в основний інтерфейс, де за ними оновлюється список задач. Попап дозволяє вибирати одночасно кілька категорій та статусів одночасно.

TaskEditDialog потрібен для редагування існуючої задачі. Як параметр він отримує екземпляр класу задачі, на основі якого заповнюються значення полів. У ньому можна змінити заголовок, опис, дату та час виконання, а також вибрати категорію через виклик CategoryPopup. Обрана категорія відображається у вигляді іконки та тексту. У разі спроби збереження задачі без заголовка виводиться повідомлення про помилку. Після збереження зміни передаються через подію OnSave, і попап закривається.

RecurringTaskPopupPage – попап для перегляду історії виконання циклічного завдання. Дане рішення дозволяє користувачу швидко оцінити прогрес по завданню без додаткової навігації, та показує загальну кількість ітерацій, а також окремо – кількість виконаних і невиконаних. Крім того статистика відображається у вигляді індикаторів, які світіться зеленим або червоним залежно від статусу вказаного в історії виконання задачі. Під кожним індикатором вказано дату ітерації задачі, що дозволяє детально проаналізувати власний прогрес у виконанні певної цілі.

ColorPickerPopupPage використовується для вибору теми оформлення між світлою, системною або темною за допомогою інтерактивного слайдера, який можна перетягувати або натискати на відповідні зони. Зміна теми

зберігається в Preferences і застосовується в інтерфейсі. Крім того, попап дозволяє вибрати кольорову схему застосунку з набору кольорових плиток, після чого викликається подія вибраним значенням та закривається попап.

LanguagePopup – попап для вибору мови інтерфейсу між українською та англійською. Активна на моменту виклику попапу мова підсвічується в UI, а результат повертається через TaskCompletionSource<string> після натискання на поле з відповідною мовою та закриває попап.

CyclingTimePicerPopup – останній представлений попап, який використовується для вибору часу, що відкриває кастомний компонент TimePicker у спеціальному режимі з підтримкою вибору днів. Після встановлення значення користувач підтверджує його натиснувши кнопку, і попап повертає обране значення TimeSpan закриваючись автоматично.

3.4 Опис реалізованої моделі зберігання даних

У реалізованому застосунку для менеджменту та аналізу задач використано локальну модель зберігання даних, побудовану на основі SQLite та реалізовану засобами ORM-бібліотеки SQLite.Net-PCL для C#. Такий підхід забезпечує ефективне збереження, доступ, оновлення та видалення даних без необхідності підключення до мережі, що є критично важливим для мобільних додатків, орієнтованих на автономне використання.

Основу структури зберігання складають дві ключові моделі: TaskItem – для одноразових задач, і RecurringTaskItem – для задач, що повторюються. Кожна з моделей представляє відповідну таблицю у базі даних, зі своїм унікальним набором полів та логікою обробки. Збереження даних у таблицях відбувається через класи TaskDatabase та RecurringTaskDatabase, які інкапсулюють виклики до SQLite-запитів і надають асинхронний інтерфейс доступу до даних. Вони також містять функціонал створення екземпляру бази даних, створення, видалення або заміни поля, а також отримання значення всієї бази даних або окремого елемента.

Модель TaskItem реалізує базову задачу, яка містить у собі такі властивості, як заголовок, опис, дата дедлайну, іконка категорії та прапорець виконання. Для кожної задачі передбачено автоматичне оновлення кольору рамки залежно від її статусу, що реалізовано в методі UpdateFrameColor. Крім того, для зручності відображення реалізовано властивості ShortTitle, GroupName, IsOverdue, а також подія IsCompletedChanged, яка дозволяє реагувати на зміну стану виконання (таблиця 3.1).

Таблиця 3.1 – Структура бази даних звичайних задач

Поле	Тип даних	Опис
Id	int	Ідентифікатор задачі
Title	string	Заголовок задачі
Desctiption	string	Опис задачі
Categoryicon	string	Іконка категорії у вигляді юнікод шляху
DueDate	DateTime	Дата дедлайну
IsCompleted	bool	Статус виконання задачі
ShortTitle	string	Скорочений заголовок для відображення
GroupName	string	Назва групи для сортування
FrameColor	Color	Колір рамки задачі
ThumbColor	Color	Колір перемикача статусу
IsOverdue	bool	Показник прострочення задачі

Окрім звичайних задач, у застосунку реалізовано підтримку повторюваних задач через модель RecurringTaskItem. Вона дозволяє задавати, крім звичайного імені та опису, дату початку, інтервал повторення, обмеження за кількістю ітерацій або дати завершення. Кожне виконання такої задачі фіксується у властивості CompletionStates, яка серіалізується у форматі JSON у поле CompletionStatesSerialized для зберігання в базі.

До даної моделі була реалізована властивість NextExecuting, яка динамічно обчислює наступний момент виконання задачі, а також

властивість IsInExecutionWindow, що визначає, чи перебуває задача у межах допустимого вікна виконання (таблиця 3.2).

Таблиця 3.2 – Структура бази даних циклічних задач

Поле	Тип даних	Опис
Id	int	Ідентифікатор задачі
Title	string	Заголовок задачі
Desctiption	string	Опис задачі
Categoryicon	string	Іконка категорії у вигляді юнікод шляху
StartDate	DateTime	Дата початку повторення
RepeatInterval	TimeSpan	Інтервал повторення задачі
RepeatNumber	int	Кількість виконань задачі
RepeatCount	int?	Кількісна умова виконання
IsRepeatCount	bool?	Ознака врахування провалених ітерацій до загальної кількості
EndDate	DateTime?	Ознака виконання в формі дати
IsActive	bool	Ознака активності задачі
CompletionStates Serialized	string	JSON-представлення історії виконання
CompletionStates	List <TaskCompletion>	Історія виконання в формі List
NextExecuting	DateTime	Час наступної ітерації задачі
ShortTitle	string	Скорочений заголовок
IsInExecutionWindow	bool	Ознака яка вказує чи задача в межах часу її виконання
FrameColor	Color	Колір рамки задачі
ButColor	Color	Колір кнопки задачі

3.5 Опис кастомних рендерів елементів

3.5.1 Реалізований TimePicker

У рамках розробки застосунку було створено кастомний компонент TimePickerControl, як заміну стандартному TimePicker на Android, де при 24-годинному форматі дня вибір годин та хвилин, а також перемикання між ними є досить складним через тісне розташування елементів один до одного.

Було прийнято рішення щодо кастомного варіанту за прикладом реалізації на iOS, але з можливістю гнучкого налаштування параметрів та дизайну. Також було вирішено розширити стандартну функціональність компонента, додавши до звичайного вибору часу в форматі годин та хвилин опціональний параметр вибору кількості днів (рисунок 3.3). Стандартні елементи не дозволяють реалізувати подібну поведінку у візуально привабливому форматі або не надають можливості розширити функціональність, наприклад, на вибір кількох одиниць часу, таких як дні, години, хвилини окремо. Саме тому й було створено цей кастомний елемент.

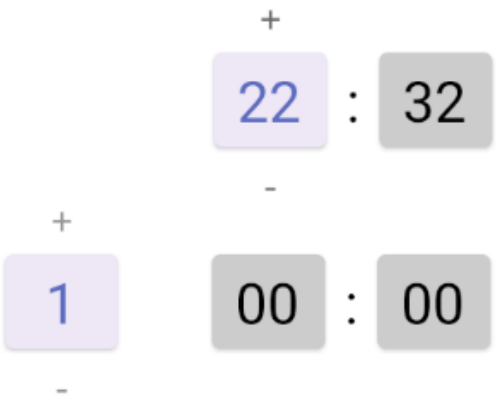


Рисунок 3.3 – Варіанти вигляду кастомного TimePicker

TimePickerControl підтримує два режими роботи – Mode 2 та Mode 3. У режимі 2 дозволяється змінювати лише години та хвилини, що типово для встановлення часу протягом одного дня. Режим 3 додає ще й можливість

вибору кількості днів, що робить його особливо корисним у випадках, коли потрібно щоб дія відбулася через кілька днів у заданий час. Наприклад, це може бути актуальним для повторюваних або відкладених задач.

У даному компоненті було реалізовано логіку керування станами залежно від того, який елемент був натиснутий – окремо можна виділити або години, або хвилини, або дні. Для кожного з них з’являються відповідні стрілки для збільшення або зменшення значення, а також візуальні ефекти, зокрема, зміна кольору та анімація мигання стрілок.

Саме збільшення або зменшення значення відбувається декількома методами. Основним з них є натиснення на відповідну стрілку, що збільшить або зменшить значення вибраного поля. Також, аби часто не натискати на стрілки, передбачена можливість затиснення на вибраному елементі, що буде збільшувати значення до моменту відтискання. Крім того, стрілки не лише є елементами інкременту або декременту, а й вказують напрям свайпу вгору або вниз по всьому контролю. Значення часу змінюється відповідно до того, який елемент було активовано, що є зручно як для налаштування хвилин, яких може бути 60, так і для годин в яких усього 24, і при цьому враховано затримку між свайпами щоб уникнути надто частих змін. Також реалізовано тактильний зворотний зв’язок – коротку вібрація при кожній зміні значення, що дає чітке відчуття того, що параметр було збільшено або зменшено.

Основною особливістю TimePickerControl є його адаптивність та збереження стану. Якщо користувач вже раніше встановлював певний час, то під час наступного відкриття компонента він автоматично підтягується з Preferences і відображається. Окрім цього, компонент дозволяє програмно встановлювати час, скидати його до початкового значення, отримувати його в залежності від режиму роботи а також зовнішні компоненти можуть реагувати на зміну часу через подію TimeChanged.

3.5.2 Реалізований Slider

Іншим створеним у Xamarin.Forms кастомним елементом є слайдер, який використовується для перемикання між двома або більше режимами, наприклад, для переключення між звичайними задачами та циклічними. Стандартні засоби, як-от Switch або Picker надто обмежені в функціоналі та не підходять для масштабних переключень властивостей, а сторонні бібліотеки не надають необхідний для реалізації функціонал, тому було прийнято рішення розробки кастомного рішення даного слайдеру.

Даний елемент забезпечує інтуїтивну взаємодію: користувач може або тапнути на відповідну мітку, або перетягнути повзунок вручну. Крім того, даний контейнер слайдеру легко розширити, додавши інші варіанти вибору, та досить просто кастомізувати, замінивши кольори елементів (рисунок 3.4).

Рисунок 3.4 – Варіанти вигляду кастомного слайдеру

Структура цього слайдера побудована на основі Grid контейнеру, в якому розміщується візуальний контейнер Frame з округлими кутами, що задає межі перемикача, має фоновий колір і певну висоту. Усередині контейнеру основними елементами є:

- заокруглений прямокутник BoxView з ім’ям Slider, що виконує роль самого повзунка, що переміщується вліво або вправо;
- прозорий BoxView, щоб реагувати на жести перетягування;
- певна кількість текстових полів, що відображають назви режимів і мають прив’язані TapGestureRecognizer для перемикання по натисканню.

Логіка зміни положення повзунка реалізується в методі MoveSlider, який із допомогою анімації TranslateTo переміщує повзунок ліворуч або праворуч залежно від вибору. При цьому оновлюється логічна змінна, яка сигналізує, який режим активний (лістинг 3.5).

Лістинг 3.5 – Метод перемикання слайдера при певній умові

```
private async void MoveSlider(bool moveRight)
{
    double targetX = moveRight ? (SliderContainer.Width -
Slider.Width - 8) : 2;
    await Slider.TranslateTo(targetX, 0, 200, Easing.SinInOut);
    isRecurringMode = moveRight;
    SortTasks();
}
```

Обробка жесту перетягування відбувається в методі OnSliderPanUpdated, де в залежності від статусу жесту (Started, Running, Completed) зберігається початкова позиція, змінюється TranslationX, і в кінці визначається, в який бік перемістити повзунок залежно від того, на якій половині контейнера він зупинився (лістинг 3.6).

Лістинг 3.6 – Метод здвигу слайдеру вручну

```
private void OnSliderPanUpdated(object sender,
PanUpdatedEventArgs e)
{
    switch (e.StatusType)
    {
        case GestureStatus.Started:
          isDragging = true; startX = e.TotalX;
          sliderStartX = Slider.TranslationX; break;

        case GestureStatus.Running:
          double newX = sliderStartX + e.TotalX;
          newX = Math.Max(2, Math.Min(newX,
SliderContainer.Width - Slider.Width - 2));Slider.TranslationX =
newX;break;

        case GestureStatus.Completed:
          isDragging = false; double middle =
(SliderContainer.Width - Slider.Width) / 2; bool moveRight =
Slider.TranslationX >= middle; MoveSlider(moveRight); break;
    }}
```

3.6 Платформозалежна реалізація застосунку

Кросплатформна розробка в Xamarin.Forms дозволяє створювати додатки для Android, iOS, Windows із використанням єдиного коду на C#. Це досягається завдяки унікальній архітектурі, що поєднує спільну кодову базу та платформозалежну інтеграцію. Проте деякий функціонал у проєкті потребує реалізації, специфічної для певної операційної системи. Такий функціонал реалізується через окремі проєкти .Droid та .iOS у структурі рішення, які відповідають за платформозалежну реалізацію застосунку для Android та iOS відповідно.

Ці проєкти є частинами багаторівневої архітектури, де загальний проєкт містить спільний код та логіку, а платформи Android та iOS забезпечує платформозалежні функції, інтеграцію з нативними API а адаптацію під специфічні вимоги операційних систем. Ця структура дозволяє одночасно розробляти кросплтформний функціонал і глибоко інтегрувати застосунок у кожну платформу.

На прикладі Android-проєкту було реалізовано декілька важливих функцій які залежать від платформи. Насамперед, у класі MainActivity, який є головною точкою входу в застосунок на Android, було налаштовано початкові налаштування застосунку, такі як відображення головної іконки через ресурс drawable/mainicon, який попередньо було додано до ресурсних файлів разом з іншими іконками, необхідними для проєкту, так і інші візуальні елементи, включаючи екран завантаження. Крім цього, у папку Resources/font були додані та налаштовані власні шрифти.

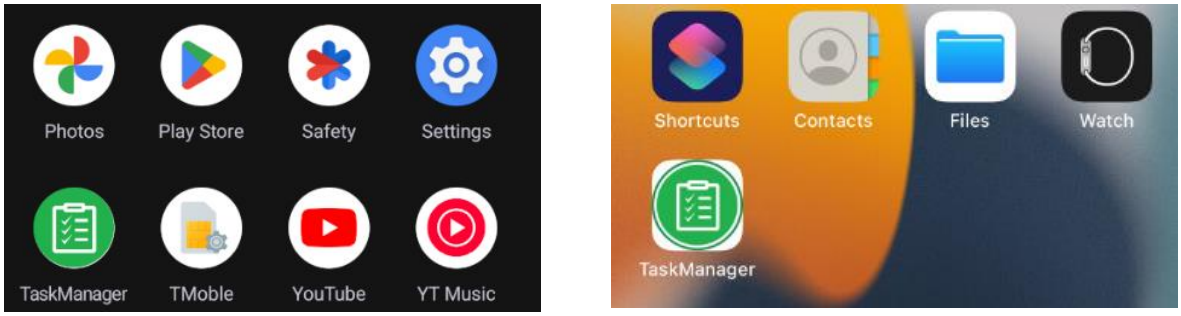
Для адаптації інтерфейсу до теми, обраної користувачем, було реалізовано власний рендер для DatePicker. У файлі DarkDatePickerRenderer.cs перевіряється поточне значення теми, збережене в Preferences, і на основі цього застосовується відповідний стиль: AppTheme_Light_DatePicker для світлої, або AppTheme_Dark_DatePicker для темної теми відповідно.

Ці стилі описані у файлі styles.xml, що розміщується в директорії Resources. Вони визначають фон, кольори тексту та кольори акцентів, та інші параметри відображення для різних елементів інтерфейсу, які залежать виключно від платформи.

Іншою важливою функцією платформозалежної реалізації є налаштування сторонніх бібліотек у стартових файлах платформи. Такі компоненти, як Rg.Plugins.Popup або Shiny вимагають додаткових платформних налаштувань, щоб їхній функціонал був доступний у загальному проєкті.

Для iOS-проєкту налаштування виконується аналогічно, з урахуванням відмінностей у назвах файлів конфігурацій, типах та форматах даних.

Загалом, реалізація платформозалежного коду фактично і дозволяє створити застосунки для декількох платформ (рисунок 3.5), та суттєво розширити функціональність кросплатформного рішення, забезпечуючи при цьому глибшу інтеграцію з можливостями пристрою. Такий підхід гарантує гнучкість реалізації та адаптивність інтерфейсу, що важливо для покращення користувацького досвіду.

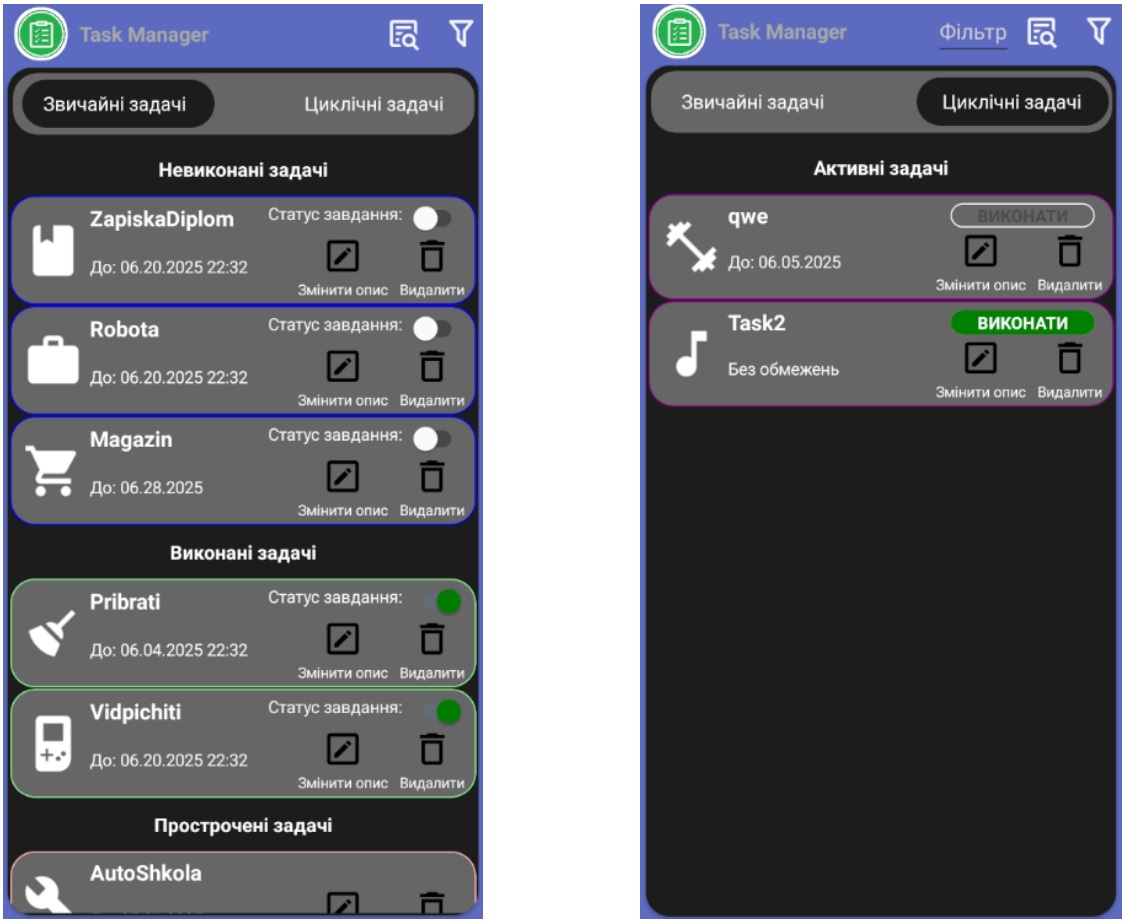


а) б)

Рисунок 3.5 – Іконка розробленого застосунку: а) на Android пристрої; б) на iOS пристрої

4 ІНСТРУКЦІЯ КОРИСТУВАЧА

Щоб скористатися програмою, користувачеві потрібно спочатку запустити застосунок, натиснувши на його іконку. Після запуску відкриється основна сторінка з переліком структурованих задач, розподілених за категоріями відповідно до вибраного активного типу задач на слайдері (рисунок 4.1). На головній сторінці користувач може взаємодіяти із задачами: переглядати їх опис, редагувати або видаляти, а також змінювати статус виконання. У верхній частині інтерфейсу знаходиться тапбар, де розташовані кнопка пошуку задачі за назвою та кнопка фільтрації.



а) б)

Рисунок 4.1 – Вигляд початкової сторінки з переліком завдань: а) при відображенні звичайних задачх; б) при відображенні циклічних задачах

При натисканні кнопки пошуку з’являється поле для введення тексту, що дозволяє швидко фільтрувати задачі за ключовими словами. Повторне натискання цієї кнопки приховує поле пошуку. Кнопка фільтрації відкриває додаткове вікно з можливістю вибору кількох фільтрів за статусом виконання та категоріями задач. Після встановлення фільтрів користувач натискає кнопку «Застосувати» (рисунок 4.2).

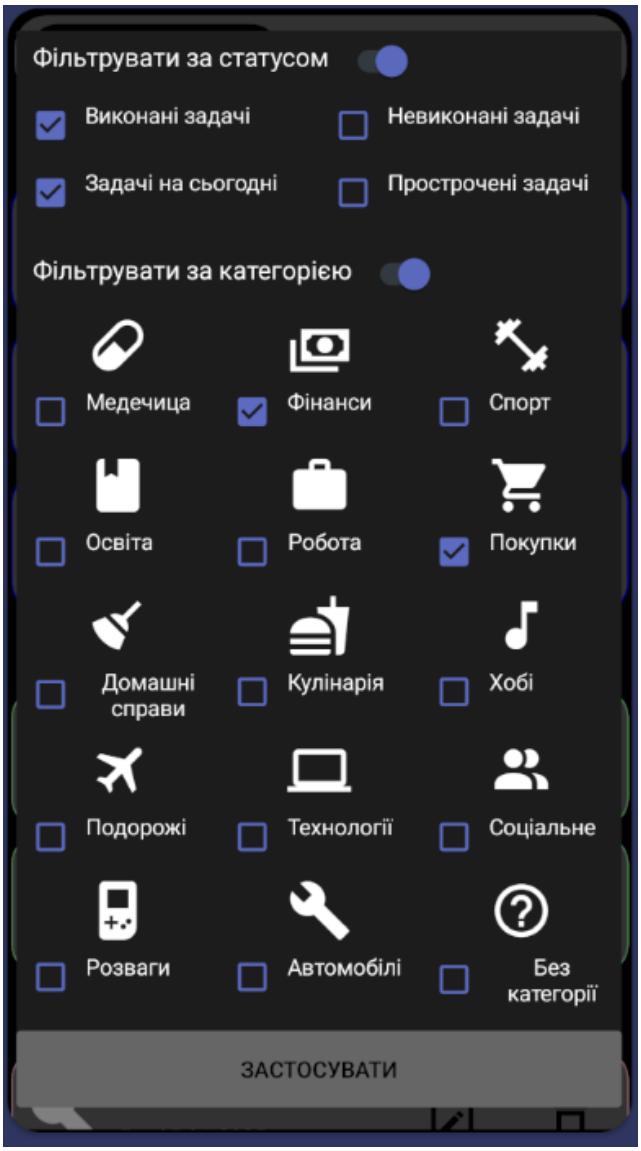


Рисунок 4.2 – Вигляд меню фільтрації задач за критеріями

На кожній задачі передбачено кнопки «Змінити опис» та «Видалити», які відкривають відповідні вікна редагування або підтвердження видалення задачі (рисунок 4.3).

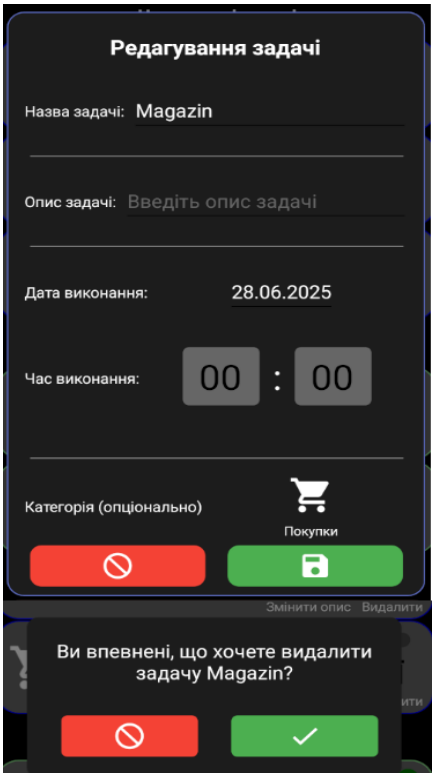


Рисунок 4.3 – Вигляд меню зміни опису та видалення задачі

У нижній частині екрана знаходяться три навігаційні кнопки (рисунок 4.4), що відповідають за створення нової задачі, перегляд статистики по існуючим задачам та перехід до налаштувань застосунку.

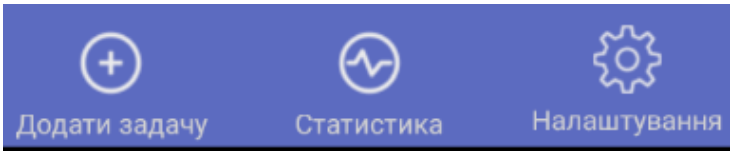
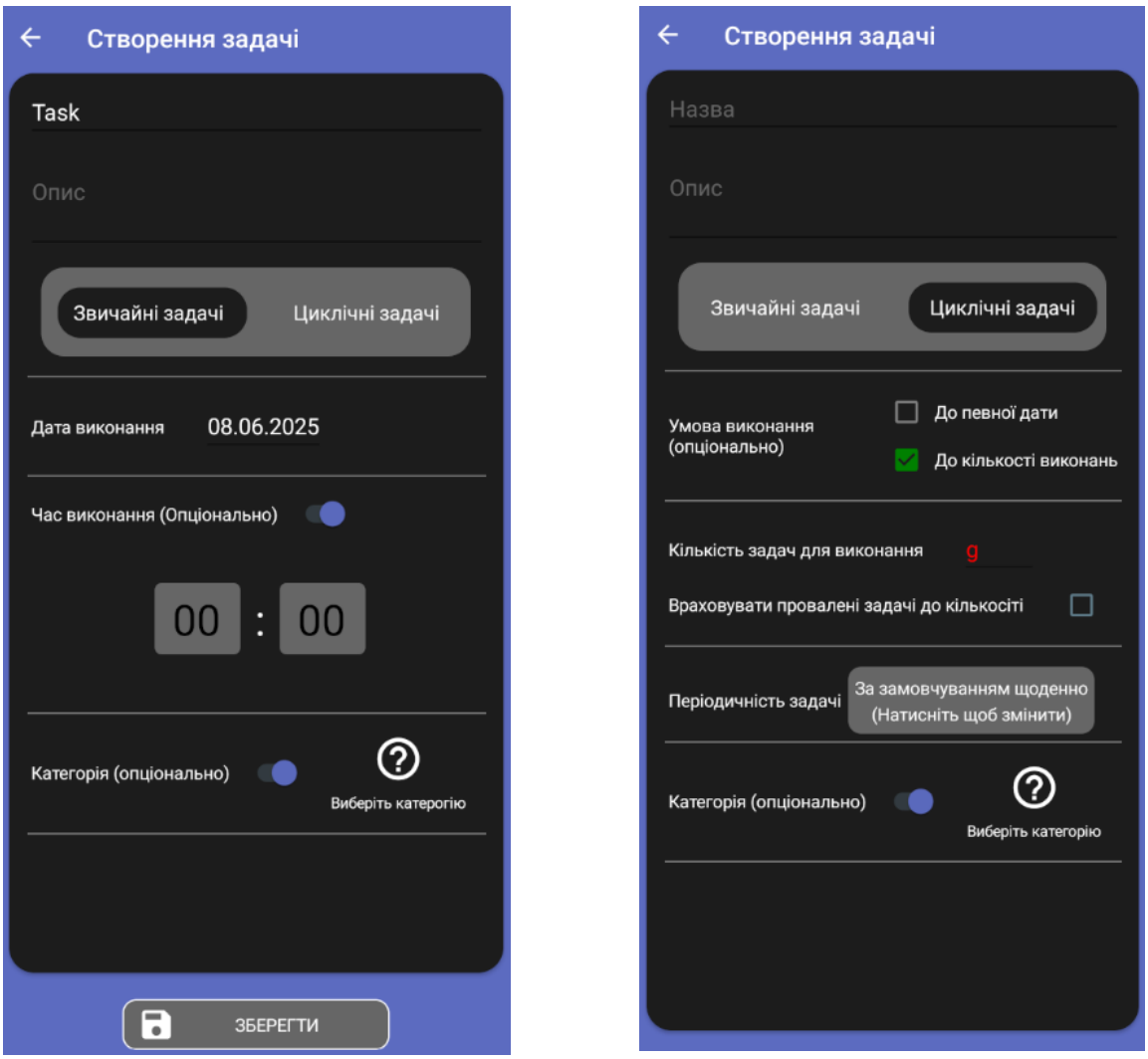


Рисунок 4.4 – Кнопки навігації по застосунку

При натисканні кнопки створення нової задачі відкривається відповідне вікно, де користувач може створити звичайну або циклічну задачу, перемикаючи режим за допомогою слайдера. Для створення задачі знадобиться ввести її назву, а також за бажанням опис та інші параметри, які залежать від типу вибраної задачі (рисунок 4.5). Задача не може бути створена, якщо не вказано назву або допущено помилки при заповненні інших полів.

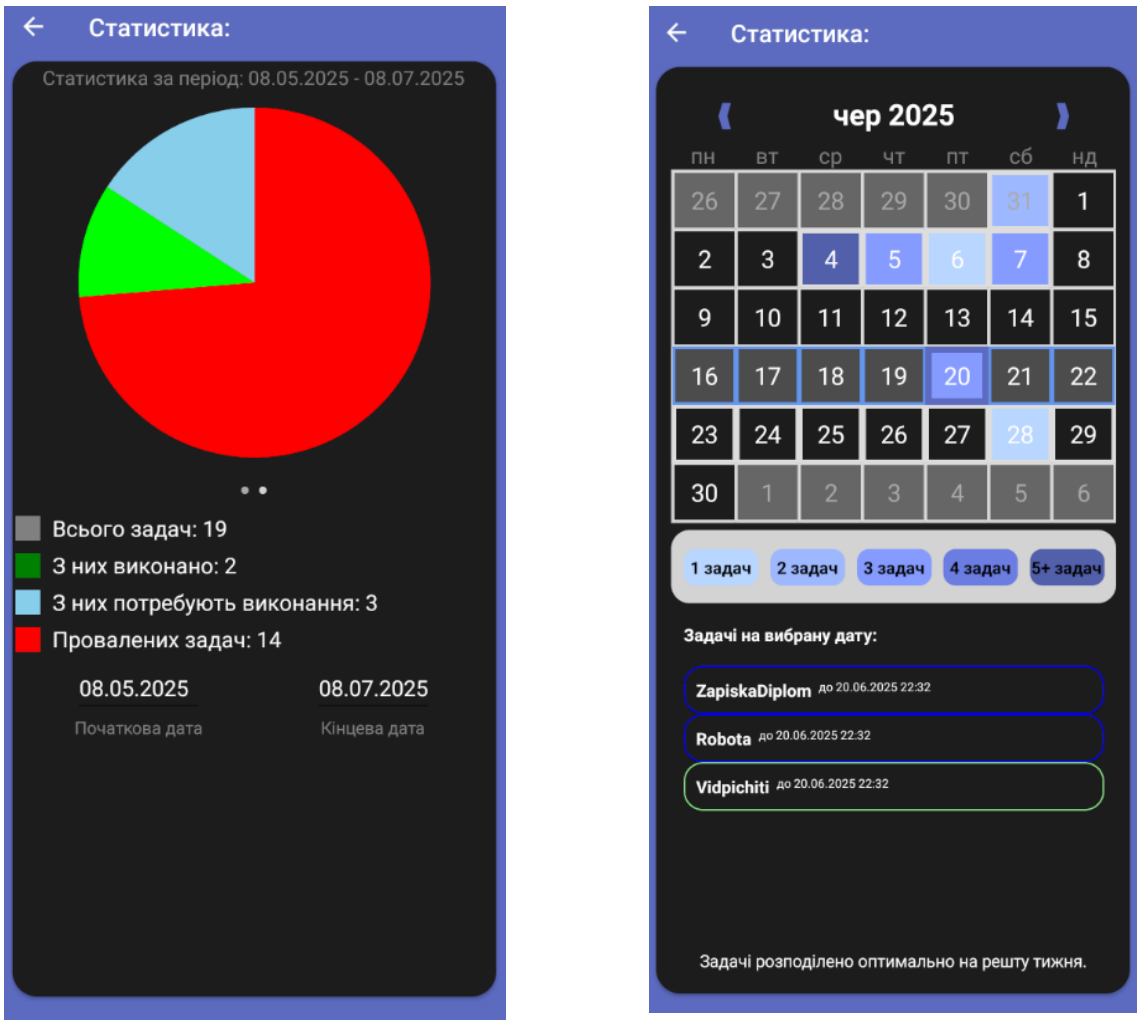


а) б)

Рисунок 4.5 – Вигляд сторінки створення нової задачі:
а) при створенні звичайних задач; б) при створені циклічних задач

При натисненні кнопки перегляду статистики програма перейде в аналітичне вікно, де користувач в зручному форматі зможе ознайомитися з існуючими задачами та їх статусом в декількох діаграмних варіантах, а також відсортувати статистику задач за датою їх виконання.

Іншою можливістю є переключити сторінку вертикальним свайпом, та отримати доступ до структурного календаря, з яким можна взаємодіяти вибираючи певні дні, та переглядаючи задачі на них в зручному форматі (рисунок 4.6).

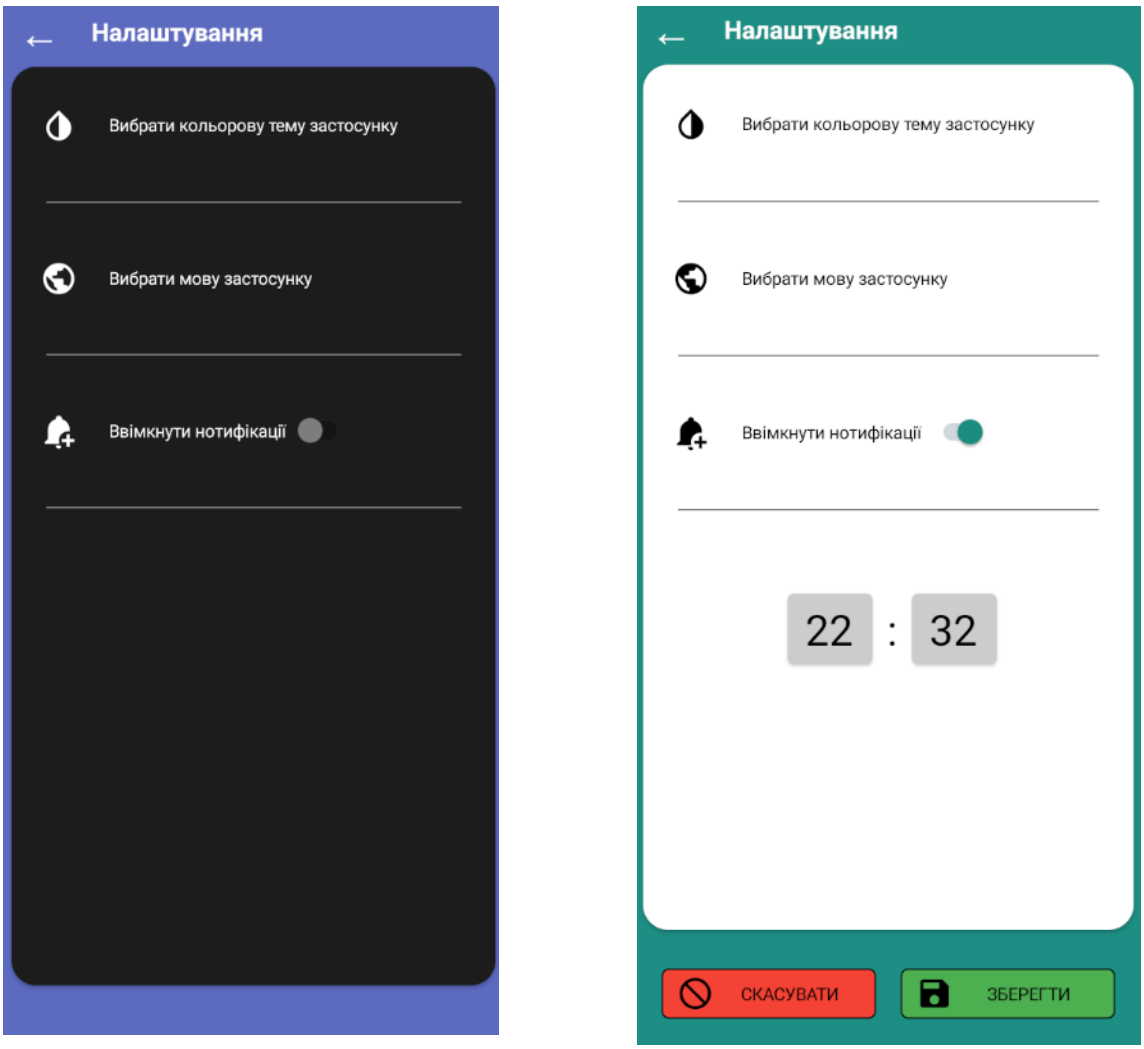


а) б)

Рисунок 4.6 – Вигляд сторінки перегляду статистики виконання задач:
а) з відображенням діаграм; б) з відображення функціонального календаря

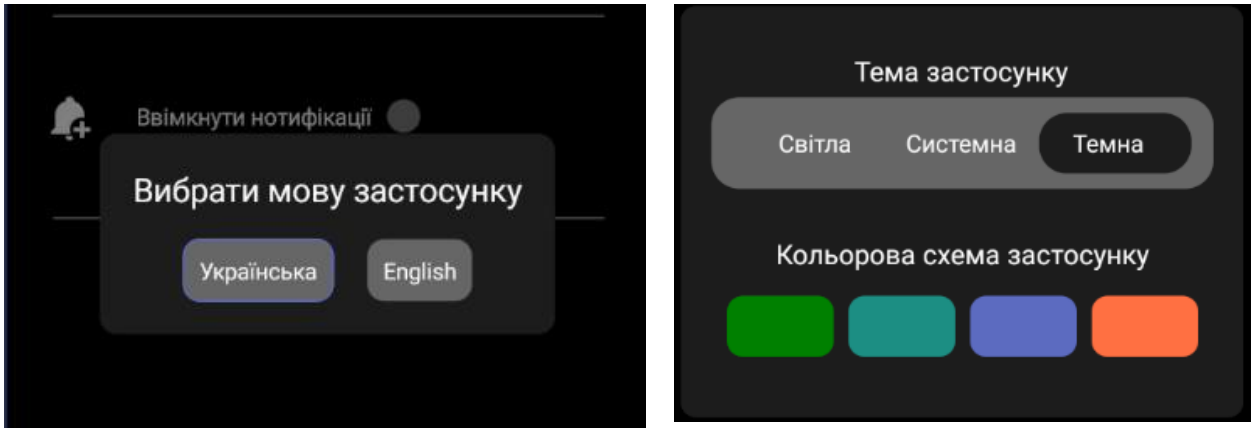
При натисненні кнопки налаштувань програма перейде в вікно з налаштуваннями, де в інтуїтивно зрозумілому інтерфейсі користувач зможе змінити параметри теми та кольорової схеми застосунку, локалізації (рисунок 4.8), а також увімкнути систему сповіщень для задач та встановити час щоденного оповіщення про загальний план дня.

При зміні одного з параметру користувач зможе або зберегти нові параметри, або скинути внесені зміни (рисунок 4.7). При активних незбережених параметрах користувач не може просто вийти з вікна – при спробі це зробити, буде виведено повідомлення пропозицією вийти без збережень або повернутися назад.



а) б)

Рисунок 4.7 – Вигляд сторінки налаштувань: а) при відкритті сторінки б) при зміні параметрів



а) б)

Рисунок 4.8 – Інтерфейси налаштувань параметрів:

а) для зміни локалізації; б) для зміни теми та кольороові схеми

ВИСНОВКИ

В ході виконання кваліфікаційної роботи було проведено детальне, обґрунтоване вивчення та аналіз теоретичного матеріалу, необхідного для успішного створення кросплатформного мобільного застосунку для менеджменту та аналізу задач.

Було детально досліджено предметну область на актуальність теми та аналізу зацікавленої аудиторії. Також розглянуто існуючі програмні рішення мобільних застосунків для роботи з задачами, що дозволило виявити їх сильні та слабкі сторони, та сформувати бачення власного продукту з урахуванням їх переваг, недоліків та особливостей, які будуть виділяти створений проєкт на фоні інших.

На основі проведеного аналізу сформовано чітку поетапну постановку задачі, яка включає всі основні етапи: від формування ідеї, розробки концепції та проєктування інтерфейсу до безпосередньої реалізації застосунку та підготовки технічної документації. Було визначено з основний функціоналом застосунку, серед якого: створення, редагування та групування задач, побудова статистики виконання, зміна локалізації та теми, збереження даних та робота зі сповіщеннями в автономному режимі. Також було розглянуто основні підходи до програмної реалізації мобільних застосунків, після чого обґрунтовано вибір відповідного програмного забезпечення.

Реалізований даний проєкт був у середовищі розробки Microsoft Visual Studio з використання фреймворку Xamarin.Forms. Використання Xamarin.Forms, як основного інструменту розробки, дозволило створити застосунок, який працює на різних платформах, зокрема Android та iOS. Під час виконання роботи пройдено повний цикл розробки застосунків, та значною мірою удосконалено теоретичні знання та практичні навички в сфері розробки програмного забезпечення для мобільних пристроїв.

У розробленому застосунку було реалізовано весь запланований

функціонал що повністю відтворюється на обох цільових платформах. Під час розробки були враховані сучасні вимоги до мобільних застосунків, зокрема адаптивний дизайн, зручність використання та ефективне розподілення ресурсів пристрою. Для застосунку було проведене тестування роботоздатності як на Android, так і на iOS пристроях з різними версіями операційної системи платформи, яке показало стабільну роботу додатку при різних сценаріях, а також підтвердило відповідність функціоналу поставленим вимогам.

В загальному, розглядаючи реалізований в проєкті функціонал, можна сказати що застосунок вийшов конкурентоспроможним, має низку унікальних особливостей і відмінностей в порівняні з представленими на ринку мобільних застосунків готових рішень, що підвищує його привабливість для цільової аудиторії.

Що до подальшого розвитку та підтримки застосунку, то його функціонал в майбутньому можна значною мірою розширити, додавши декілька нових можливостей для більш гнучких налаштувань задач, інтеграції з іншими сервісами та можливості синхронізації задач між пристроями на різних платформах. Окрім підтримки мобільних платформ Android та iOS, перспективним є створення десктопної версії для Windows і веб-версії застосунку, що дозволить охопити ширшу аудиторію користувачів і підвищити конкурентоспроможність продукту на ринку.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Аллен Д. Як упорядкувати справи. Ефективність без стресу / пер. з англ. Харків: Клуб сімейного дозвілля, 2019. 320 с.

2. Косенко Ю. В., Урсатьєв А. А. Аналіз та порівняння сучасних систем управління проєктами та завданнями. Вісник Національного технічного університету "ХПІ". 2022. № 2 (8).

3. Вивчення популярних таск-менеджерів та порівняння їх функціональних можливостей та інтерфейсів. URL: https://www.softwareadvice.com/project-management/task-management-comparison/

4. Соммервіль І. Інженерія програмного забезпечення / пер. з англ. Київ : вид. група BHV, 2022. 656 с.

5. Тітов І. О., Лисенко С. М. Продуктивність нативних та кросплатформних мобільних застосунків: порівняльний аналіз. Проблеми програмування. 2023. № 2

6. Мартін Робертс, "Мобільна розробка: огляд технологій", Kyiv: IT-Press, 2021. 256 с.

7. Grepix Infotech. The Benefits of Native Mobile App Development / Grepix Infotech. URL: https://www.grepixit.com/blog/the-benefits-of-native-mobile-app-development.html.

8. Bryan Harris, "Cross-platform Mobile Development", New York: TechBooks, 2019. 280 p.

9. Aziz. Cross-Platform Mobile App Development Tools Comparison: React Native vs Flutter vs Xamarin (Medium). – 16.02.2024. URL: https://medium.com/huawei-developers/cross-platform-mobile-app-development-tools-comparison-react-native-vs-flutter-vs-xamarin-bfd3b1236081

10. Ali, R. Hybrid mobile app development tools / R. Ali // International Journal of Computer Science and Network Security. 2017. Vol. 17, No. 5.

11. James M. Learning Xamarin.Forms by Example: Create Modern, Cross-Platform Mobile Apps with C#. Publisher: Packt Publishing, 2021. 384 p.

12. Charles P., Nathan H., Stellman A. Programming Xamarin.Forms: Cross-Platform Apps for iOS, Android, and Windows with Xamarin.Forms. Publisher: O'Reilly Media; 2nd edition, 2020. 416 p

13. Skeet J. C# in Depth. Publisher: Manning; 4th edition, 2019. 528 p.

14. О'Ніл П. SQLite. Практичне керівництво / П. О'Ніл. К.: Видавництво "Наука і Техніка", 2020. 256 с.

15. Microcharts: Elegant Cross-Platform Charts for Every App. URL: https://devblogs.microsoft.com/xamarin/microcharts-elegant-cross-platform-charts-for-any-app/

16. Shiny Library for Xamarin.Forms. Official Documentation. URL: https://shinylib.net