

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

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

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАСТОСУНКУ ДЛЯ
ПЛАНУВАННЯ РОБОЧОГО ПРОЦЕСУ.

RESEARCH AND SOFTWARE IMPLEMENTATION OF A WORKFLOW
PLANNING APPLICATION

Виконав: студент(ка) 2 курсу, групи КНТ-223М

Спеціальності 122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Системи штучного інтелекту

ТАРАН В.О.

(прізвище та ініціали)

Керівник ЗАЙКО Т.А.

(прізвище та ініціали)

Рецензент СКРУПСЬКИЙ С.Ю.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

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

ЗАТВЕРДЖУЮ

Завідувач кафедри _____

_____ **Сергій СУББОТІН**
“ _____ ” _____ 2024 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТА(КИ)

ТАРАН Владислава Олеговича

(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проекту (роботи) Дослідження та програмна реалізація застосунку для планування робочого процесу. Research and Software Implementation of a Workflow Planning Application

керівник проекту (роботи) к.т.н., доцент ЗАЙКО Тетяна Анатоліївна

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “ 7 ” жовтня 2024 року № 398

2. Строк подання студентом проекту (роботи) 02 грудня 2024 року

3. Вихідні дані до проекту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Матеріали і методи. 3. Опис програми. 4. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ЗАЙКО Т.А., доцент		
Нормоконтроль	КАМІНСЬКА Ж.К., асистент		

7. Дата видачі завдання «9» вересня 2024 року

КАЛЕНДАРНИЙ ПЛАН

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

Студент (ка)

(підпис) Владислав ТАРАН
(Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Тетяна ЗАЙКО
(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
76 с., 3 табл., 44 рис., 3 дод., 13 джерел.

VISUAL STUDIO, XAMARIN, MAUI, BINDING, CONTENTPAGE, .NET,
TODO-LIST, TASK.

Об'єкт дослідження – процес розробки програмного забезпечення для планування робочого процесу.

Предмет дослідження – програмні засоби для планування робочого процесу.

Мета роботи – дослідження та розробка програмного забезпечення для вирішення проблеми з плануванням робочого процесу.

Матеріали, методи та технічні засоби: Visual Studio, мова програмування C#, мова програмування XAML, .Net Maui.

Результати. Розроблено програмне забезпечення для планування робочого процесу.

Практична цінність роботи полягає у розробці програмного забезпечення для вирішення проблеми з плануванням робочого часу та процесу.

Висновки. Виконано проєктування програмного забезпечення для планування робочого процесу для вирішення проблем з розподілом часу. Розроблено програмне забезпечення для планування робочого процесу.

Здійснено тестування розробленого програмного забезпечення.

Галузь використання – створення завдань для планування робочого процесу в галузі роботи, освіти та побутових моментів.

ABSTRACT

Explanatory note to the master's thesis: 76 pages, 44 figures, 3 tables, 3 appendices, 13 sources.

VISUAL STUDIO, XAMARIN, MAUI, BINDING, CONTENTPAGE, .NET, TODO-LIST, TASK.

Object of the research- software tools for implementing a workflow planning application.

Subject of the research - a workflow planning application.

The purpose of the work is to develop a website with a current topic.

Goal of the work - development of an application to address the problem of workflow planning. The results. The site was created using modern tools.

Materials, methods, and technical tools- Visual Studio, the C# programming language XAML, and .NET MAUI.

Results: A workflow planning application was developed.

Conclusions - The application for workflow planning was created to solve issues related to time management.

The area of use – task creation for planning workflows in the fields of work, education, and household activities.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок	8
Вступ.....	9
1 Аналіз предметної області	10
1.1 Поняття застосунку для планування робочого процесу.....	10
1.2 Порівняння з аналогами.....	11
1.3 Основні пункти для створення застосунку	17
1.4 Висновки за розділом 1	18
2 Матеріали та методи	19
2.1 Вибір середовища розробки для створення програмного забезпечення .	19
2.2 Вибір мови програмування.....	23
2.3 Фреймворк .NET MAUI.....	26
2.4 Порівняння мов програмування та фреймворків.	27
2.5 Висновки за розділом 2	29
3 Опис програми	30
3.1 Розробка проекту у IDE	30
3.2 Файл MainPage.xaml.cs.....	35
3.3 Файл MainPage.xaml	37
3.4 Файл TodoItem.cs	38
3.5 Файл TodoViewModel.cs	39
3.6 Файл App.xaml	41
3.7 Файл App.xaml.cs	42
3.8 База даних SQLite та шаблон MVVM.....	43
3.9 Послідовність виконання програми.....	44
3.10 Висновки за розділом 3	45
4. Експлуатація, тестування та експериментальне дослідження програми	46
4.1 Інструмент для тестування застосунку	46
4.2 Тест програмного забезпечення.....	48

4.3 Висновки за розділом 4	51
Висновки	52
Перелік джерел посилання	53
Додаток А Технічне завдання	55
Додаток Б Фрагмент тексту програми	59
Додаток В Слайди презентації	71

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

APP	– Application;
API	– Application Programming Interface;
ASP	– Active Server Pages;
IDE	– Integrated Development Environment;
ID	– Identifier;
iOS	– Operating system for Apple's mobile devices
MAUI	– Multi-platform App UI;
MVVM	– Model-View-ViewModel;
UI	– User interface;
XAML	– Extensible Application Markup Language;

ВСТУП

Кожен день додає людям нові завдання, виклики, труднощі та потреби, з якими доводиться справлятися, використовуючи свої навички та таланти. Кількість завдань зростає, що вимагає великих витрат часу та зусиль. Ще більше занепокоєнь викликає те, наскільки важко їх виконувати. Крім того, не можна забувати, що окрім вашої професії, яка часто забирає велику частину дня людини, є ще й сімейні обставини, які також потребують уваги. На перший погляд, головна та єдина труднощі полягає в тому, як саме вирішувати ці проблеми. Але насправді нюанси можуть виникнути на самому початку шляху.

У вирішенні завдань, які входять у спектр ваших обов'язків: робота, навчання, сімейні справи, хобі, важливо вміти організовувати та планувати робочий день. В іншому випадку людина ризикує зіткнутися з тим, що її ефективність протягом дня буде набагато нижчою, ніж могла б бути, адже про багато завдань можна просто забути або не встигнути зробити через те, що немає чіткого графіка.

В таких умовах застосунок для управління завданнями набуває особливої актуальності, пропонуючи користувачам структурувати свої обов'язки, розставляти пріоритети та контролювати виконання завдань. Такі застосунки не лише допомагають зберігати концентрацію та фокус на важливих справах, але й сприяють більш раціональному використанню часу, дозволяючи ефективно балансувати між різними сферами життя.

На завершення можна сказати, що застосунок для планування робочого процесу повинен стати невід'ємною частиною життя будь-якої людини, яка хоче успішно розвиватися як на роботі, так і в вільний час, тому що вирішує найпершу проблему, яка виникає у кожної людини.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Поняття застосунку для планування робочого процесу

Типовий мобільний застосунок складається з кількох основних елементів, які забезпечують його функціональність та зручність використання. На головному екрані зазвичай розташовується список завдань, представлених у вигляді таблиці або списку рядків. Кожен рядок відповідає окремому завданню і містить такі компоненти, як назва завдання, блок з анімацією для відмітки його виконання, термін виконання та іконки для редагування або видалення завдання.

Верхня частина інтерфейсу часто включає панель навігації з кнопкою додавання нового завдання, що дозволяє швидко додати нову позицію до списку. Додаткові фільтри та категорії можуть бути розташовані в боковому меню або у вигляді вкладок, що дозволяє користувачам сортувати завдання за пріоритетом, датою чи іншими критеріями. Інтерфейс може також містити графічні елементи, такі як прогрес-бари або діаграми, що відображають виконання завдань та досягнення цілей.

Основні задачі, які повинен виконувати застосунок, включають створення, редагування та видалення завдань. Користувачі повинні мати змогу легко додавати нові завдання з описом, встановлювати для них терміни виконання та пріоритети. Інтеграція з календарем дозволяє відображати завдання у контексті щоденних подій, що сприяє кращому плануванню часу.

Застосунок також має підтримувати функції нагадувань та сповіщень, які допомагають користувачам не забувати про важливі задачі. Можливість створення підзадач та встановлення залежностей між ними дозволяє структурувати складні проекти та забезпечити їх послідовне виконання. Для командної роботи застосунок може пропонувати функції спільного доступу до

списків завдань, коментування та призначення відповідальностей окремим учасникам.

1.2 Порівняння з аналогами

Оскільки планування завдань і створення нагадувань є необхідністю, яка стоїть перед кожною людиною, то в наш час існує безліч застосунків, які надають такі функції. Ці застосунки бувають платними і безкоштовними. Їх створюють як великі компанії, такі як Google, Microsoft, Xiaomi, так і невеликі команди розробників, які роблять акцент виключно на одному застосунку. У кожного застосунку свої особливості, плюси і мінуси. На рис. 1.1 зображено застосунок "Xiaomi Notes" [1].

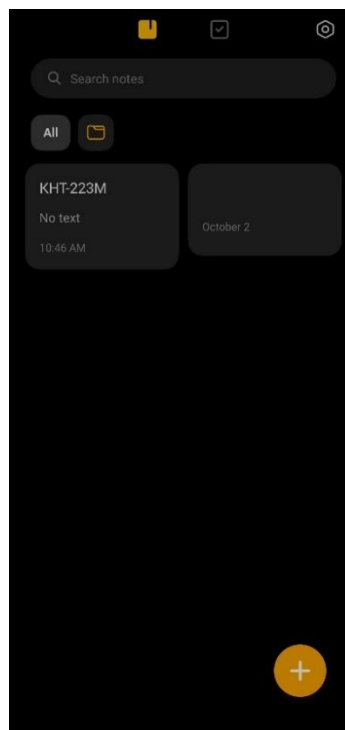


Рисунок 1.1- Головна сторінка застосунку " Xiaomi Notes "

Він за замовчуванням встановлений на телефонах компанії Xiaomi. Маємо можливість спостерігати такі елементи інтерфейсу, як рядок для пошуку нотаток, куди ми можемо ввести назву або її частину, щоб знайти потрібну нотатку. Самі нотатки займають невеликий простір. Зараз бачимо їх назву і частину тексту, якщо він є. В самому низу є кнопка для створення нових нотаток. Крім того, є кнопки для створення категорій нотаток, щоб упорядковувати їх. У верхній частині розташовані три кнопки: перша з них вказує на головну сторінку з нотатками, друга кнопка — на розділ із завданнями, а третя кнопка призначена для налаштувань застосунку.

На рис. 1.2 зображено інтерфейс другої сторінки, яка дозволяє користувачеві створювати прості завдання. У ці завдання не можна додати жодних зображень чи інших елементів. Є лише кнопка для додавання тексту та рядок пошуку завдань на випадок, якщо їх стане занадто багато. Суть завдань полягає в тому, щоб швидко та просто створювати невеликі задачі, які можна швидко видалити після їх виконання.

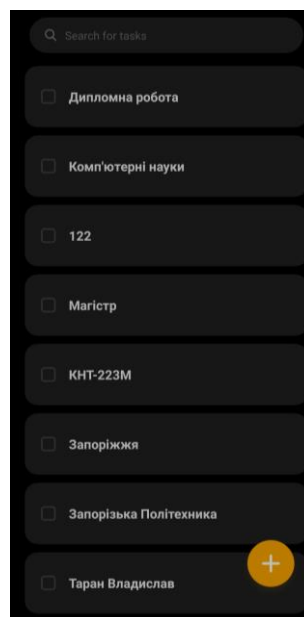


Рисунок 1.2 - Друга сторінка застосунку "Xiaomi Notes"

На рис. 1.3 зображено інтерфейс створення нотаток. Нотатки відрізняються від швидких завдань тим, що мають багато функцій для редагування. Крім того, можемо форматувати текст, додавати зображення, створювати графіки, діаграми, пункти та навіть додавати голосові повідомлення. Також можна переглядати детальну інформацію про нотатку та саму форму, в якій пишеться текст або додаються додаткові елементи. Нотатки необхідні для створення великих записів із можливістю додавати різні елементи, що робить цю функцію набагато зручнішою.

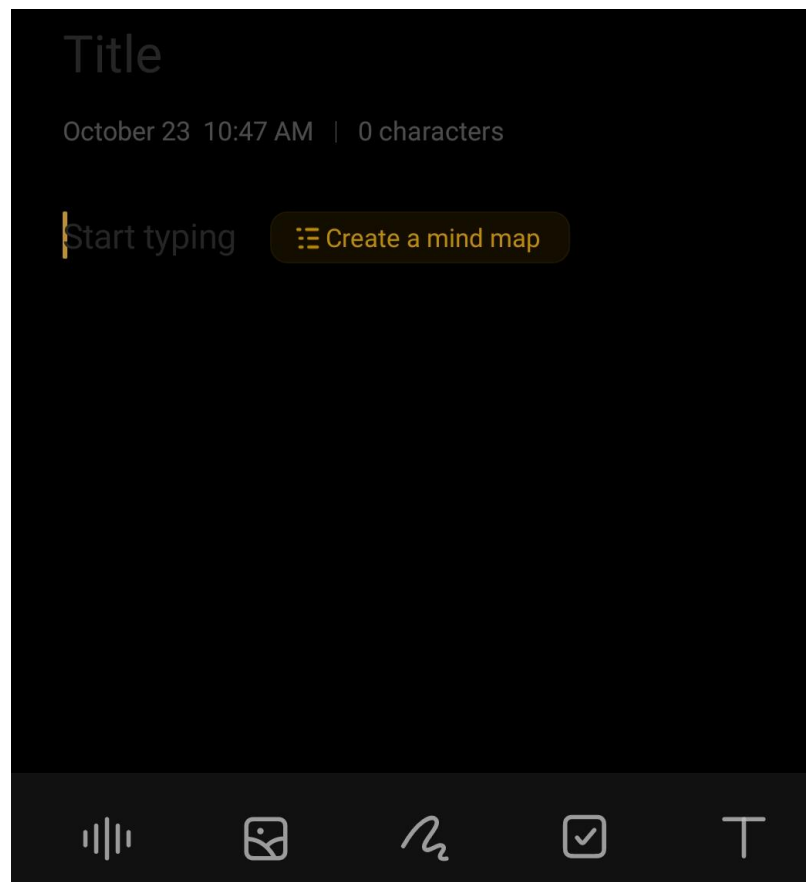


Рисунок 1.3 – Створення нотатку у "Xiaomi Notes"

Наступний застосунок від компанії Google "Google Tasks" зображений на рис. 1.4. Це легендарна компанія, яка не потребує представлення. Всі її платформи і застосунки користуються великою популярністю. Однак, застосунок

для нотаток від Google не досяг такого результату. Причиною цього стало те, що застосунок не встановлювався за замовчуванням, і в Google були інші інструменти для планування завдань [2].

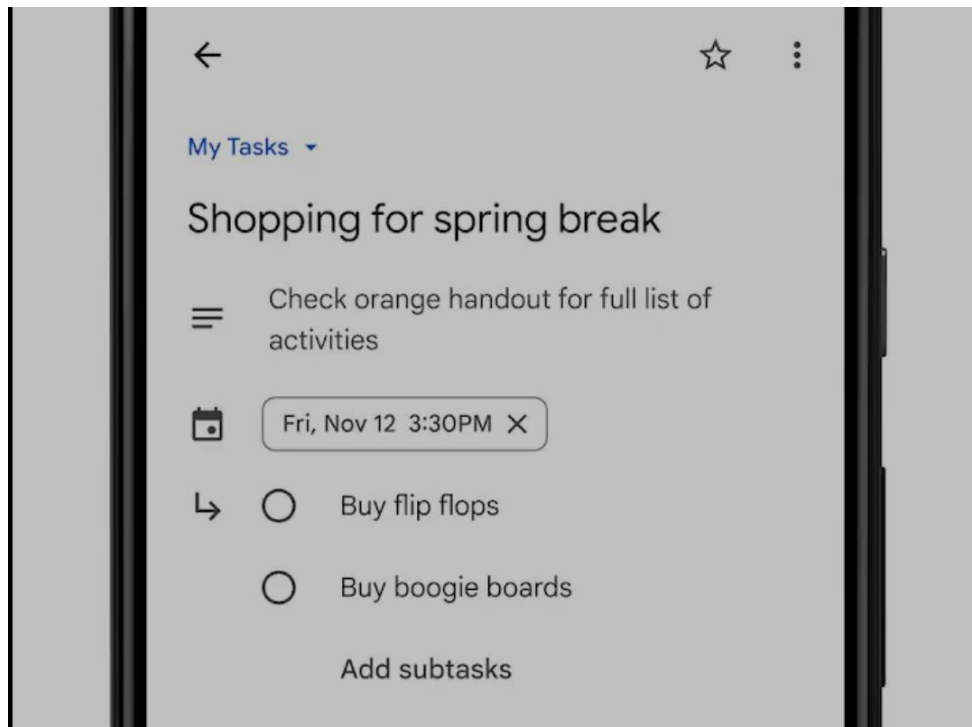


Рисунок 1.4 – Створення нотатку у "Google Tasks"

Цей застосунок взаємодіє з сервісами Google, що робить роботу з ним приємнішою та зручнішою. У користувачів є можливість створювати завдання з правильною структурою, де у пунктів можуть бути підпункти. Також присутня можливість кастомізації завдань.

На рис. 1.5 зображений процес взаємодії застосунку для нотаток із сервісом календаря Google. Це залишається найбільш зручною частиною застосунку.

Функціонально він поступається аналогам, що може бути однією з причин його низької популярності та наочно показує, що продукт великої компанії не завжди гарантує популярність.

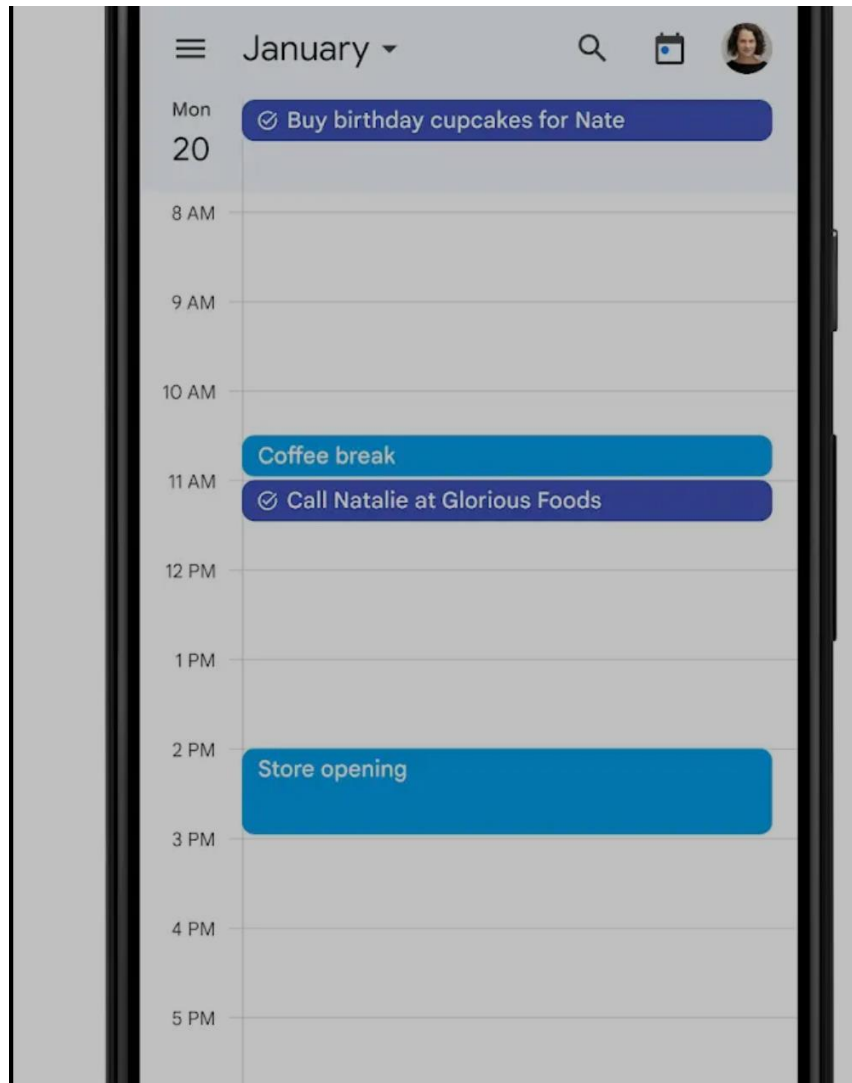


Рисунок 1.5 – Взаємодія з "Google Calendar"

На рис. 1.6 зображено застосунок "Todoist: Planner & Calendar" від компанії "Doist Inc.". Це застосунок від невеликої компанії, і для того, щоб завоювати популярність, їм довелося створити продукт, який був би набагато якіснішим за аналоги від відомих корпорацій [3].

Застосунок пропонує один із найбільш зручних інтерфейсів для створення нотаток і завдань. Він сучасний і максимально дружній до користувача. Є можливість створювати власні нотатки за допомогою кнопки, що спрощує процес додавання нового контенту.

Застосунок також надає доступ до редагування нотаток. Анімація роботи виконана на найвищому рівні та значно перевершує аналоги від більших і відомих компаній. Крім того, застосунок має вбудований календар, що дозволяє зручно і приємно розміщувати завдання.

Як і в попередніх прикладах, передбачена функція нагадувань про завдання, що робить управління часом ще ефективнішим.

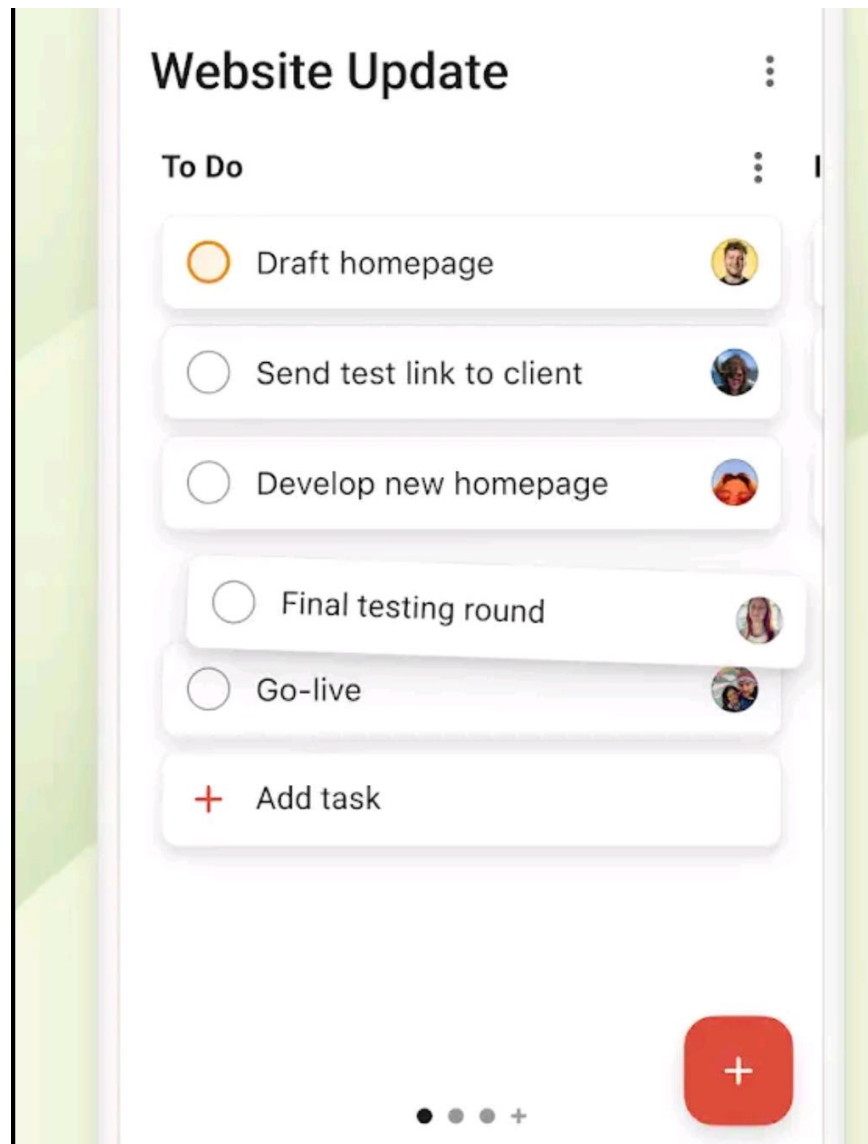


Рисунок 1.6 – Інтерфейс "Todoist: Planner & Calendar"

У таблиці 1.1 наведено порівняння застосунків.

Таблиця 1.1 – Порівняння застосунків

Атрибут	Xiaomi Notes	Google Tasks	Todoist: Planner & Calendar
Дизайн	-	-	+
Можливості	-	+	+
Оригінальність	-	-	+
Зручність у використанні	+	-	+
Ефективність	+	+	+

1.3 Основні пункти для створення застосунку

Для створення застосунку нам необхідно дотримуватися певних пунктів:

- на початковому етапі важливо чітко визначити цілі і функціональні можливості застосунку. Необхідно зрозуміти, які завдання має вирішувати застосунок, хто є його цільовою аудиторією і які ключові функції необхідно реалізувати;
- наступним кроком є розробка інтерфейсу користувача. Важливо створити зручний, інтуїтивно зрозумілий та привабливий дизайн, який допоможе користувачам легко взаємодіяти з застосунком;
- вибір технологій та архітектури. На цьому етапі необхідно визначитися з технологіями, які будуть використовуватись для розробки застосунку;
- цей етап включає безпосередню реалізацію застосунку. Розробники створюють інтерфейс користувача, реалізують логіку взаємодії з користувачем, а також розробляють серверну частину для обробки даних, зберігання завдань, синхронізації між пристроями і підтримки багатокористувацького режиму, якщо

він передбачений. Важливим аспектом є реалізація основних функцій — створення, редагування, видалення завдань;

— після завершення розробки застосунків необхідно протестувати. Тестування включає перевірку всіх функціональних можливостей, виявлення та усунення багів.

1.4 Висновки за розділом 1

У цьому розділі було розглянуто базові поняття та основи створення структури застосунку для планування робочого процесу. Також було розглянуто застосунки з аналогічною тематикою.

2 МАТЕРІАЛИ ТА МЕТОДИ

2.1 Вибір середовища розробки для створення програмного забезпечення

Інтегроване середовище розробки або "Integrated Development Environment" – це програмний інструмент, що забезпечує розробників усіма необхідними засобами для створення програмного забезпечення в одному середовищі. IDE поєднує редактор коду та компілятор, інструменти для налагодження та часто додаткові функції, такі як інтеграція з системами контролю версій або підтримка шаблонів коду.

Інтегровані середовища розробки необхідні для:

- розробки програмного забезпечення для написання, тестування та налагодження коду;
- навчання для того, щоб студенти могли писати та експериментувати з кодом в зручному середовищі;
- командної розробки завдяки інтеграції з інструментами керування версіями (Git, SVN);
- різних мов програмування для спрощення роботи з мовами, що мають складний синтаксис чи процес компіляції.

Найвідоміші IDE:

- Visual Studio. Багатофункціональне середовище від Microsoft для розробки застосунків під .NET, C++, Python тощо;
- IntelliJ IDEA. IDE для розробки на Java, Kotlin, а також багатьох інших мов;
- Eclipse. Популярне середовище для Java, але також підтримує C, C++, Python;
- Android Studio. офіційне IDE для розробки Android-застосунків;

- Xcode – IDE від Apple для розробки програм під macOS та iOS;
- NetBeans – інструмент з підтримкою Java, PHP, C++;
- Vs Code – легкий текстовий редактор із функціями IDE, що здобув свою популярність за допомогою розширень.

IDE сприяють підвищенню продуктивності розробників завдяки автоматизації багатьох рутинних завдань, зменшенню кількості помилок у коді та спрощенню процесу тестування й розгортання різних застосунків, створених за допомогою різних ресурсів.

Visual Studio 2022 – це потужне та універсальне інтегроване середовище розробки (IDE), яке чудово підходить для створення складних, великих та сучасних застосунків. Воно є одним із найкращих інструментів для розробників, особливо для тих, хто працює з технологіями Microsoft. Воно підходить для того, щоб максимально ефективно використовувати функціонал цього середовища для написання якісного й оптимізованого коду [4].

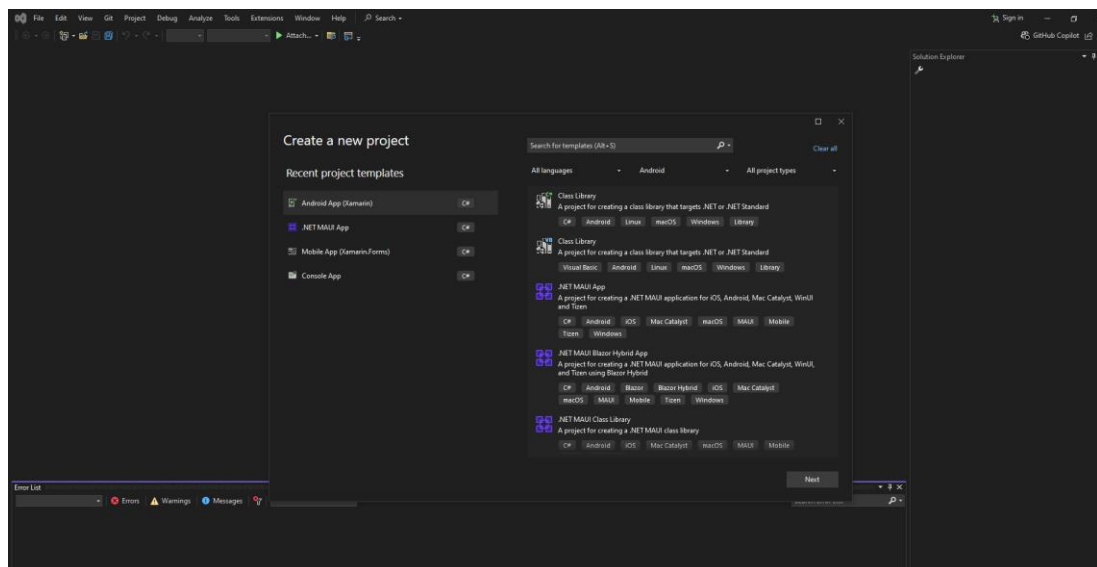


Рисунок 2.1 – Visual Studio

Visual Studio надає розробникам широкий спектр можливостей для ефективної та спільної роботи над написанням високоякісного коду [5]:

- налаштування на основі робочих навантажень – встановлюйте лише ті компоненти, які потрібні для конкретних завдань;
- потужні засоби розробки – усе необхідне для створення застосунків доступно в одному місці;
- підтримка багатьох мов програмування – C++, C#, JavaScript, TypeScript, Python та інші;
- кросплатформена розробка – створення застосунків для будь-якої платформи, включно з Windows, macOS, Android, iOS;
- інтеграція з системами контролю версій – зручна співпраця над кодом з колегами;
- розробка з підтримкою штучного інтелекту – підвищення ефективності завдяки функціям для генерації та вдосконалення коду;
- ця середовище розробки забезпечує інструменти, що допомагають створювати продуктивні програми з мінімальними зусиллями, зберігаючи високу якість і адаптивність.

Для того, щоб встановити Visual Studio, необхідно встановити Visual Studio Installer, за допомогою якого можливо налаштувати середовище так, як потрібно користувачу. Microsoft пропонує різні платформи для розробки, компоненти для можливості обрати мову (рис 2.2). У табл. 2.1 наведено порівняння сайтів.

Серед платформ для розробки є:

- ASP.Net;
- Azure;
- Python;
- .Net Multi-Platform App Ui;

- класичні застосунки .Net;
- мобільні застосунки.

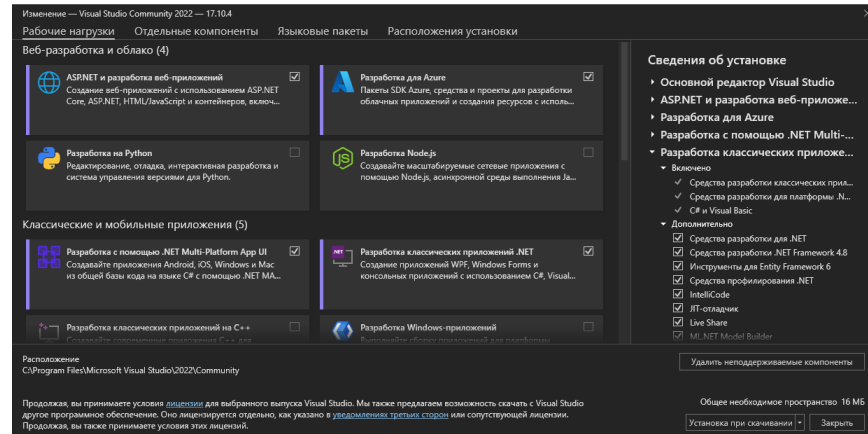


Рисунок 2.2 – Visual Studio Installer

Таблиця 2.1- Порівняння різних IDE [6]

Назва IDE	Переваги	Недоліки
Android Studio	Це офіційна IDE для Android від Google. Підтримує Kotlin і Java, пропонує вбудовані інструменти для налагодження, емуляції та перевірки застосунків. Глибока інтеграція з Android SDK, швидке завантаження емулятора	Висока вимога до ресурсів системи.
Visual Studio	Ідеально підходить для кросплатформної розробки. Дозволяє створювати застосунки для iOS, Android і Windows з використанням C#. Спільний код.	Деякі функції потребують додаткової оптимізації для окремих платформ.

Продовження таблиці 2.1

Назва IDE	Переваги	Недоліки
Rider	Підтримка кількох платформ. Rider працює на Windows, macOS і Linux. Інтеграція з популярними фреймворками: Rider має вбудовану підтримку таких фреймворків, як ASP.NET, Xamarin, Unity та інших	Rider є платним продуктом, що може бути проблемою для деяких користувачів. Rider, як і багато інших IDE, може бути досить вимогливим до ресурсів комп'ютера, особливо при роботі з великими проектами. Відсутність повної інтеграції з усіма інструментами Microsoft, такими як Azure DevOps, може бути недоліком для деяких команд.
IntelliJ IDEA	Підтримує розробку на багатьох мовах програмування, включаючи Kotlin і Java, підходить для мобільних застосунків. Зрозумілий інтерфейс та величезна кількість плагінів	Платні функції в Ultimate Edition

2.2 Вибір мови програмування

Платформа .NET підтримує декілька мов програмування. Оскільки всі ці мови працюють на одній платформі та використовують однакові бібліотеки класів, вибір мови для розробки залежить переважно від особистих уподобань програміста. Іншими словами, ви можете виконувати ті самі завдання, незалежно від того, якою мовою працюєте. Завдяки .NET ви отримуєте свободу вибору мови програмування, зберігаючи при цьому всі переваги, які надає платформа [7].

Мова програмування C# - це об'єктно-орієнтована мова програмування, розроблена компанією Microsoft у 2000 році як частина платформи .NET. Мова призначена для створення широкого спектра застосунків, включаючи веб-застосунки, десктопні програми, мобільні застосунки, ігри та системне програмне забезпечення (рис 2.3).

Переваги мови програмування C#:

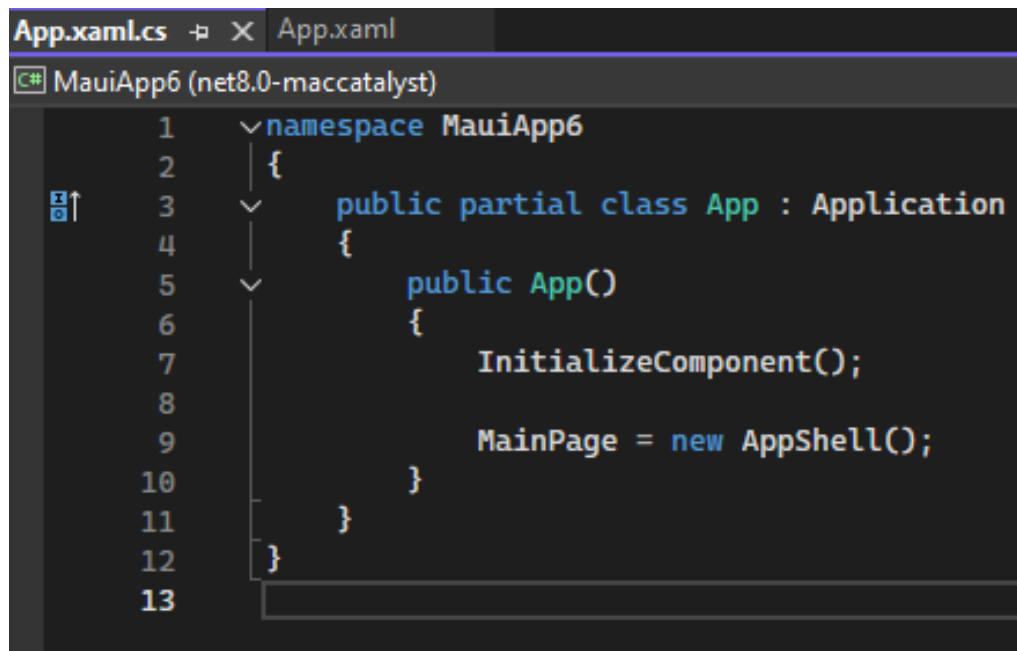
- об'єктно-орієнтоване програмування. Полегшує управління великими проєктами завдяки інкапсуляції, наслідування та поліморфізму;
- кросплатформність. Завдяки .NET Core і .NET 6+ розробка можлива для Windows, macOS, Linux і навіть мобільних платформ через Xamarin або MAUI;
- велика стандартна бібліотека. Платформа .NET пропонує багато готових рішень для роботи з файлами, базами даних, мережею та іншими завданнями;
- інтеграція з іншими інструментами Microsoft. Тісно пов'язаний із Visual Studio, Azure, SQL Server тощо.

Ця мова програмування має певну структуру [8]:

- простори імен (Namespaces) надають порядок і структуру там, де інакше виникнув би хаос. Бібліотеки середовища виконання .NET містять тисячі типів, а ще більше їх можна знайти в NuGet-пакетах від Microsoft і сторонніх розробників. Додайте до цього класи, які ви створите самі, — і виникає дві ключові проблеми. По-перше, складно гарантувати унікальність імен. По-друге, важко знайти потрібний API: без відповідного контексту або знання правильного імені орієнтуватися в неструктурованому списку з десятків тисяч елементів стає майже неможливо. Простори імен вирішують обидві ці проблеми. Вони дозволяють розділяти класи на логічні групи та забезпечують унікальність назв, значно полегшуючи навігацію та пошук потрібного функціоналу;

– local Variable. Як і більшість мов, C# дозволяє визначати локальні змінні, які є іменованими елементами всередині методу, що зберігають певну інформацію. У специфікації C# термін "variable" може стосуватися як локальних змінних, так і полів об'єктів та елементів масивів. C# — це статично типізована мова програмування. Це означає, що кожен елемент коду, який представляє або створює інформацію, наприклад змінна чи вираз, має заздалегідь визначений тип даних під час компіляції. Це відрізняє її від динамічно типізованих мов, таких як JavaScript, де типи визначаються під час виконання програми;

– більшість типів, з якими треба працювати в C#, будуть класами. Клас може містити як код, так і дані, а також може надавати доступ до деяких своїх функцій публічно, залишаючи інші доступними лише для внутрішнього використання в межах самого класу. Таким чином, класи забезпечують механізм інкапсуляції — вони дозволяють визначити чіткий публічний інтерфейс для взаємодії з іншими частинами програми.



The image shows a code editor window with two tabs: 'App.xaml.cs' and 'App.xaml'. The 'App.xaml.cs' tab is active, displaying C# code for a class named 'App' within the 'MauiApp6' namespace. The code is as follows:

```
1 namespace MauiApp6
2 {
3     public partial class App : Application
4     {
5         public App()
6         {
7             InitializeComponent();
8
9             MainPage = new AppShell();
10        }
11    }
12 }
13
```

Рисунок 2.3 – Клас C#

2.3 Фреймворк .NET MAUI

.NET MAUI (Multi-platform App UI) — це фреймворк, створений для спрощення розробки кросплатформних застосунків. Він дозволяє розробникам створювати програми, які працюють на різних платформах, таких як Android, iOS, macOS і Windows, використовуючи єдину кодову базу. Розуміння структури .NET MAUI є важливим для ефективного використання його можливостей [9].

Основою .NET MAUI є уніфікований API, який абстрагує специфічні для платформ деталі. Цей API дозволяє писати спільний код, що працює на різних платформах. Зазвичай структура застосунку на основі .NET MAUI включає:

Головний проєкт: Містить спільний код і ресурси програми, включаючи логіку застосунку, визначення інтерфейсу користувача (UI) та спільні активи.

Проекти для конкретних платформ: Містять код і конфігурації, специфічні для певної платформи (Android, iOS, macOS, Windows). Вони дозволяють використовувати функції, характерні для кожної платформи.

.NET MAUI використовує XAML (Extensible Application Markup Language) для визначення інтерфейсу користувача. XAML — це декларативна мова, що дозволяє описувати елементи інтерфейсу та їх властивості у зрозумілій і зручній формі. Інтерфейс, створений за допомогою XAML, є спільним для всіх платформ, забезпечуючи узгоджений зовнішній вигляд [9].

XAML — це мова розмітки, яка використовується для опису візуального інтерфейсу застосунку. Вона дозволяє відокремити дизайн інтерфейсу від коду, що значно полегшує управління та підтримку проєкту. У .NET MAUI XAML активно використовується для створення кросплатформених застосунків з єдиним стилем і виглядом (рис 2.4).

XAML-файл зазвичай складається з кореневого елемента і вкладених елементів, що описують компоненти інтерфейсу користувача. Кореневим

елементом найчастіше виступає контейнер для макета, наприклад, `ContentPage` або `StackLayout`, які можуть включати інші елементи, такі як текст, зображення чи елементи управління. Прив'язка даних (data binding) є потужною функцією XAML, яка дозволяє зв'язувати елементи інтерфейсу користувача з джерелами даних. Це забезпечує динамічне оновлення та чітке розмежування між інтерфейсом і бізнес-логікою. Визначення прив'язок здійснюється за допомогою синтаксису `{Binding}` [9].

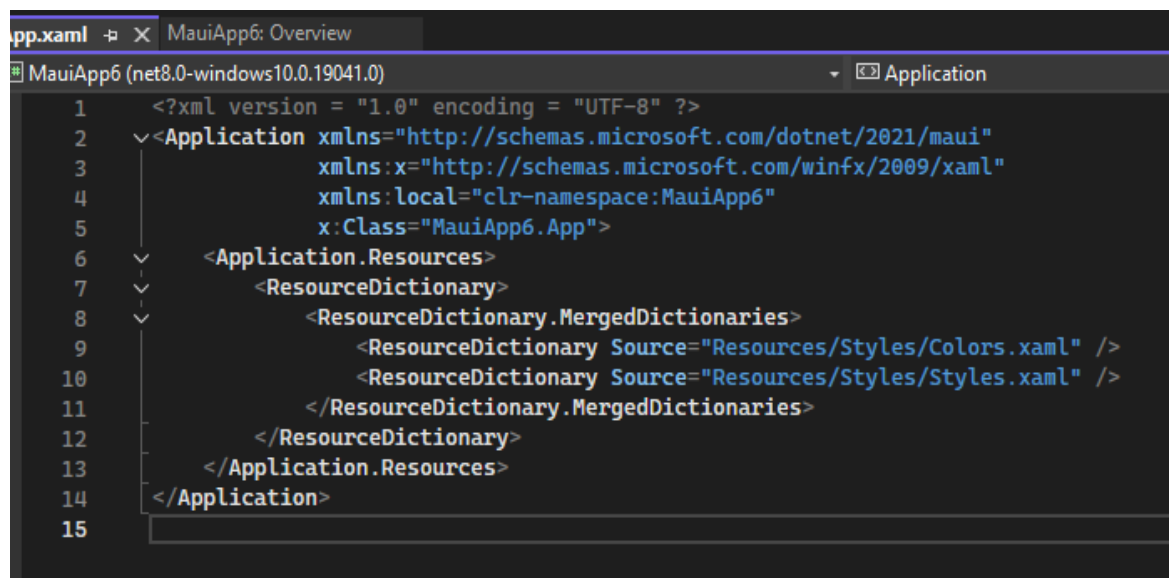


Рисунок 2.4 – Файл App.xaml

2.4 Порівняння мов програмування та фреймворків.

Серед усіх варіантів C#, Xaml та .Net MAUI є оптимальними для розробки TODO-List. Вибір між цими технологіями залежить від потреб проекту та специфікацій платформи. Якщо ваш застосунок потребує складних нативних функцій або ви хочете використовувати найновіші інструменти, .NET MAUI може бути кращим вибором. Xamarin підходить для більш простих проектів, де

критично важлива швидка розробка для кількох платформ. У табл. 2.2 порівняючи можемо дійти до таких висновків:

Таблиця 2.2- Порівняння різних мов програмування та фреймворків

Назва	Переваги	Недоліки
C#,Xaml, .Net MAUI, Xamarin.	Використання C# для бізнес-логіки та XAML для інтерфейсу дозволяє розділяти відповідальність між частинами програми, що робить код чистим та зручним для підтримки. C# є частиною платформи .NET, що надає доступ до багатьох бібліотек, інструментів і технологій для розробки.Обидва фреймворки (Xamarin та .NET MAUI) дозволяють створювати застосунки для кількох платформ (Android, iOS, Windows, macOS) з однією кодовою базою. Xamarin.Forms дозволяє швидко створювати мобільні застосунки за допомогою єдиної кодової бази для всіх платформ. Велика спільнота та підтримка: Xamarin має велику кількість бібліотек[10].	Хоча .NET MAUI обіцяє покращену продуктивність, але в деяких випадках буває навпаки. Хоча .NET MAUI значно покращує доступ до нативних API, в Xamarin все ще можна зустріти ситуації, коли потрібно використовувати платформозалежний код для деяких специфічних функцій.З новим фреймворком .NET MAUI можна стикнутися з деякими труднощами при переході з Xamarin. Хоча Xamarin підтримує основні функції для мобільних платформ, доступ до деяких складних нативних функцій може бути обмежений і потребує додаткової роботи.З Xamarin можуть виникати проблеми, якщо потрібно.

Продовження таблиці 2.2

Назва	Переваги	Недоліки
React Native	React Native дозволяє створювати застосунки для Android та iOS з єдиною кодовою базою, використовуючи JavaScript та React. Завдяки використанню нативних компонентів і можливості працювати з JavaScript, застосунки можуть мати хорошу продуктивність.	Для більш складних або специфічних функцій може знадобитися використання додаткових бібліотек або нативного коду. Розробка кроссплатформних застосунків вимагає певного рівня знань у JavaScript та React.
Flutter (Dart)	Завдяки компіляції в нативний код, застосунки, створені за допомогою Flutter, можуть працювати дуже швидко. Flutter дозволяє створювати застосунки для Android, iOS та навіть веб-платформ з однією кодовою базою.	Оскільки Flutter є відносно новим фреймворком, йому ще не вистачає деяких бібліотек і інструментів, доступних для більш зрілих фреймворків. Для розробників, що не мають досвіду з Dart, може бути складно почати роботу з Flutter.

2.5 Висновки за розділом 2

У другому розділі розглянуто різні варіанти за допомогою яких можна створити мобільний застосунок "TODO-List". Зроблено висновок, що серед усього найкращим варіантом буде створення застосунку за допомогою .Net MAUI та C# з Xaml. Цей фреймворк є новітньою технологією від Microsoft, який позбавився недоліків Xamarin та отримав нові можливості.

3 ОПИС ПРОГРАМИ

3.1 Розробка проекту у IDE

Після того, як було вирішено, які особливості будуть у нашого застосунку, які функції він буде мати та які ресурси будуть залучені для його створення, потрібно переходили до структури самого проекту.

.Net MAUI дозволяє нам створити деякі шаблони, за допомогою яких будемо мати фундамент для подальшої роботи (рис 3.1). Серед них маємо такі варіанти:

- .Net MAUI App - Проєкт для створення .NET MAUI застосунку для iOS, Android, Mac Catalyst, WinUI та Tizen;
- .Net MAUI Blazor Hybrid App - Проєкт для створення .NET MAUI застосунку для iOS, Android, Mac Catalyst, WinUI та Tizen з використанням Blazor Hybrid. Цей варіант дозволяє створювати кросплатформенні застосунки з використанням Blazor Hybrid, що означає можливість інтеграції веб-технологій (HTML, CSS і компоненти Blazor). В цьому випадку можна використовувати веб-сторінки та веб-ресурси всередині мобільного або десктопного застосунку, що дозволяє створювати гібридні програми з мінімальними зусиллями по коду. Застосунок працюватиме на всіх платформах, підтримуваних .NET MAUI, але при цьому більша частина інтерфейсу та логіки буде використовувати веб-технології;
- .NET MAUI Class Library - Проєкт для створення бібліотеки класів .NET MAUI. Цей варіант дозволяє створити бібліотеку класів, яка може бути використана в застосунках .NET MAUI, але сама по собі не є самостійним застосунком. Бібліотека класів використовується для структуризації та розподілу логіки коду, щоб він був зручніший для повторного використання та тестування. Вона може містити спільні компоненти, моделі даних, бізнес-логіку чи інші

допоміжні класи, які можуть бути використані як у .NET MAUI застосунку, так і в інших проєктах.

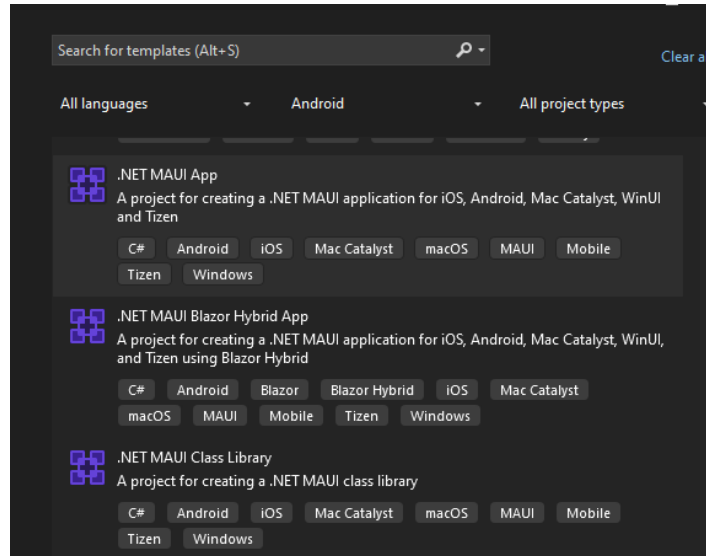


Рисунок 3.1 – Шаблони .Net MAUI

Зараз обираємо .Net MAUI App, тому що це є оптимальним варіантом для створення нашого застосунку, поєднуючим усі необхідні для нас функції.

Після того, як було обрано шаблон, необхідно визначити – на яких платформах може працювати. .NET MAUI (Multi-platform App UI) дозволяє створювати кросплатформенні застосунки для мобільних пристроїв (iOS, Android), а також для десктопів (macOS, Windows) з використанням єдиної кодової бази.

В .NET MAUI реалізовано уніфіковану API для роботи з усіма цими платформами, що дає змогу взаємодіяти з кожною платформою через спільний інтерфейс. Зараз потрібно обрати лише Android, але маємо можливість одразу використати на інших платформах, хоча іноді все одно доведеться налаштовувати для роботи на інших платформах, якщо застосунок має специфічні функції (рис. 3.2).

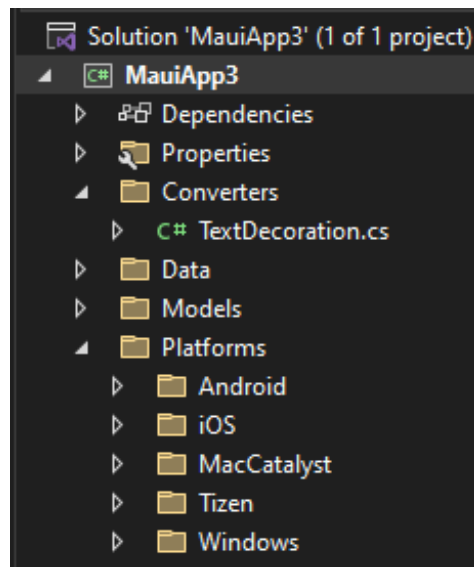


Рисунок 3.2 – Можливість працювати з різними платформами

Потім з'являється робоче місце, де є структура (дерево) класів та інших ресурсів, які є проекту для створення шаблону застосунку. Окрім класів, з якими маємо працювати, важливо налаштувати середовище для тестування застосунку. Visual Studio пропонує робити це за допомогою емуляторів (рис. 3.3).

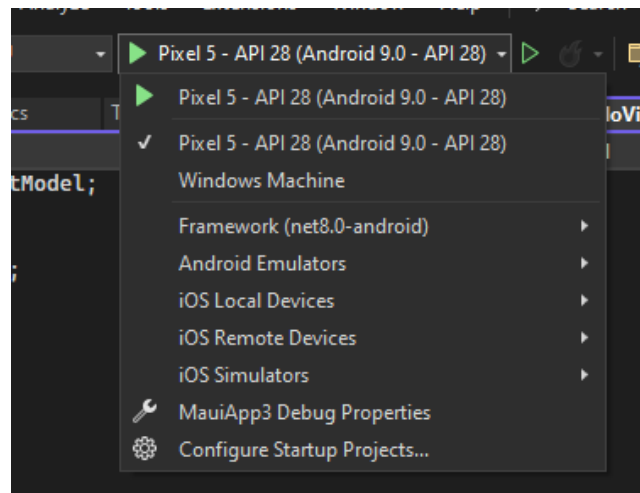


Рисунок 3.3 – Різні емулятори

Емулятор - інструмент, що дає змогу імітувати ОС на іншій платформі, називають емулятором. Віртуальні Android екосистеми створюються на ПК або ноутбуці за допомогою Visual Studio, розгортаючи Android середовище з доступом до магазину Google Play і набором інших функцій (рис.3.4). Для отримання Root-доступу, потрібен акаунт Google [10].

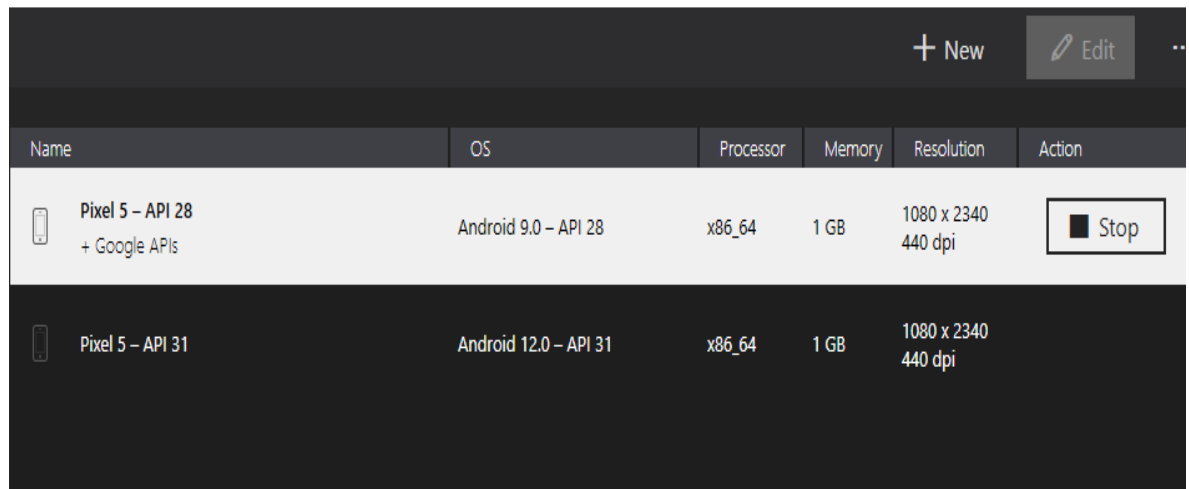


Рисунок 3.4 – Вікно створення емуляторів

При створенні емулятора є можливість налаштувати його більш цілеспрямовано для розробки застосунку (рис. 3.5). У Android SDK Manager є можливість встановити:

- ім'я;
- модель пристрою;
- процесор;
- OS;
- Google APIs;
- Google Play Store;
- різні властивості пристрою;

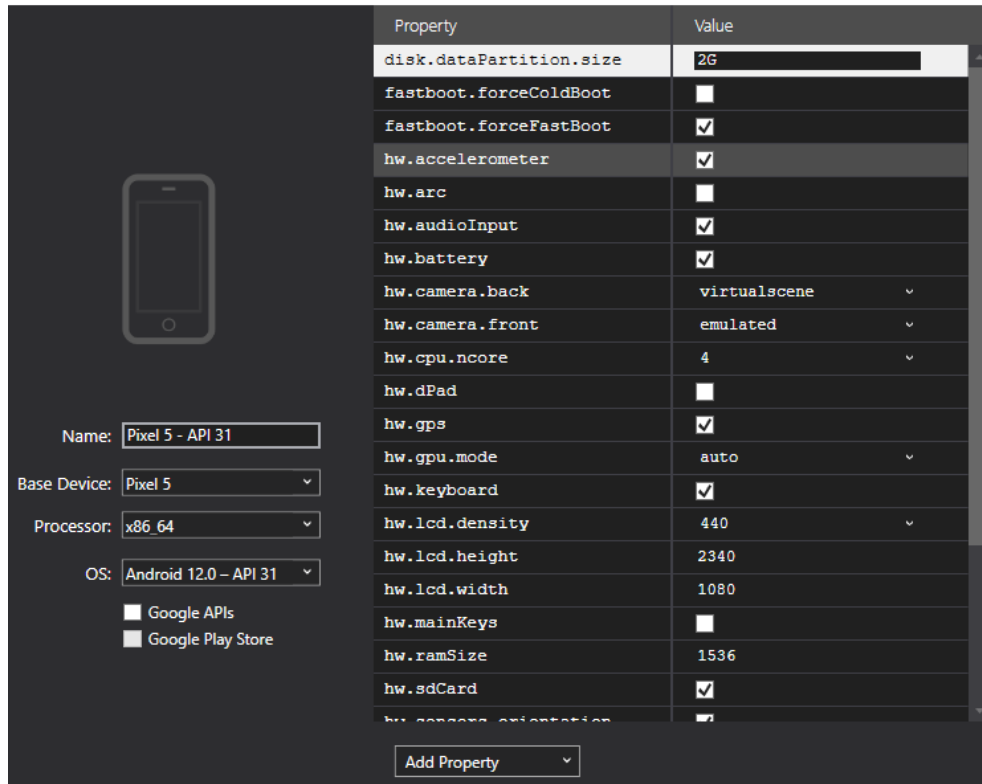


Рисунок 3.5 – Налаштування емулятора

Так виглядає сам емулятор (рис. 3.6). Це Pixel 5- api 28. Його можливо запустити окремо від розроблюємого застосунку. Коли емулятор запускається разом з застосунком, то відразу після завершення компіляції за допомогою Android запускається застосунок. З'являється можливість тестувати функції застосунку та проаналізувати якість та правильну роботу інтерфесу [11].

Якщо включити емулятор без застосунку, то він також буде працювати. В обох випадках є можливість налаштувати систему Android:

- налаштувати яскравість та інші системні налаштування;
- увімкнути Wi-Fi або мобільний інтернет;
- отримати повідомлення;
- використовувати сервіси Google;
- користуватися ішними застосунками.

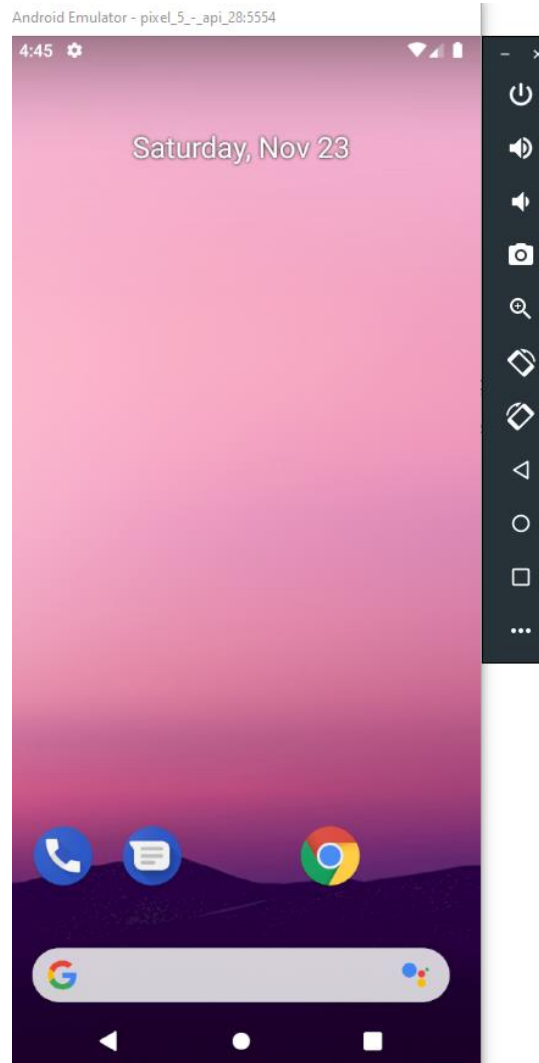


Рисунок 3.6 - Емулятор

3.2 Файл MainPage.xaml.cs

Клас "MainPage.xaml.cs" в застосунку відповідає за взаємодію з елементами інтерфейсу користувача та обробку подій (рис. 3.7).

— конструктор "MainPage()" ініціалізує компоненти сторінки, тобто завантаження всіх елементів інтерфейсу, визначених у файлі Xaml, таких як кнопки, текстові поля та інші.

```

using MauiApp3.ViewModels;
using MauiApp3.Models;
namespace MauiApp3
{
    public partial class MainPage : ContentPage
    {

        public MainPage()
        {
            InitializeComponent();
        }
    }
}

```

Рисунок 3.7. - Конструктор MainPage()

- функція "OnTitleChanged" викликається, коли змінюється назва завдання у полі введення. Вона перевіряє, чи є об'єкт, до якого прив'язаний цей елемент, є завданням "TodoItem". Потім, якщо це так, викликається команда для оновлення цього завдання у ViewModel (рис. 3.8);
- функція "OnIsCompletedChanged" викликається, коли змінюється стан прапорця, наприклад, завершено" чи незавершено. Вона перевіряє, до якого об'єкта прив'язано це завдання. Якщо об'єкт завдання знайдений, виконується команда для оновлення статусу цього завдання у ViewModel (рис. 3.8).

```

private void OnTitleChanged(object sender, TextChangedEventArgs e)
{
    var entry = sender as Entry;
    var todo = entry.BindingContext as TodoItem;

    if (todo != null)
    {
        var viewModel = BindingContext as TodoViewModel;
        viewModel.UpdateTodoCommand.Execute(todo);
    }
}

private void OnIsCompletedChanged(object sender, CheckedChangedEventArgs e)
{
    var checkBox = sender as CheckBox;
    var todo = checkBox.BindingContext as TodoItem;

    if (todo != null)
    {
        var viewModel = BindingContext as TodoViewModel;
        viewModel.UpdateTodoCommand.Execute(todo);
    }
}

```

Рисунок 3.8 - Методи

Цей клас зв'язує елементи інтерфейсу з даними через ViewModel, і забезпечує їх оновлення за допомогою команд, коли змінюються значення в інтерфейсі. Це стандартна практика в MVVM архітектурі, де UI і логіка обробки розділені, а взаємодія відбувається через команди та зв'язування даних.

3.3 Файл MainPage.xaml

MainPage.xaml у застосунку відповідає за створення інтерфейсу користувача. У ньому визначаються всі елементи сторінки: поля вводу, кнопки, списки та взаємодія з даними через механізм прив'язки. Це дозволяє ефективно відокремити логіку програми від її інтерфейсу (рис 3.9).

```
<StackLayout Padding="20" Spacing="10">

    <Label Text="Мої завдання"
           Style="{StaticResource HeaderLabelStyle}" />

    <Grid ColumnSpacing="10">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="Auto" />
        </Grid.ColumnDefinitions>

        <Entry Placeholder="Введіть нове завдання"
                Text="{Binding NewTodoText}"
                Grid.Column="0"
                BackgroundColor="White"
                TextColor="{StaticResource TextColor}"
                HeightRequest="50"
                VerticalOptions="Center" />

        <Button Text="Додати"
                Command="{Binding AddTodoCommand}"
                Style="{StaticResource PrimaryButtonStyle}"
                Grid.Column="1" />
    </Grid>

    <CollectionView ItemsSource="{Binding Todos}"
                    SelectionMode="None">
        <CollectionView.ItemTemplate>
            <DataTemplate>
                <SwipeView>
```

Рисунок 3.9 – MainPage.xaml

Серед елементів інтерфейсу та функцій:

- прив'язка до моделі даних BindingContext;

- відображає заголовок "Мої завдання" у верхній частині. Поле вводу "Entry" прив'язане до властивості "NewTodoText" у "ViewModel". Кнопка викликає команду "AddTodoCommand" для додавання нової задачі;
- відображає всі задачі, прив'язані до колекції. Для кожного елемента в списку створюється окремий шаблон.
- назва завдання. Якщо вона виконана, текст перекреслюється.

3.4 Файл TodoItem.cs

У файлі "TodoItem.cs" визначається модель даних для елементів списку задач у застосунку. Цей клас представляє окрему задачу і містить властивості, які описують її стан, а також реалізує механізм сповіщення про зміни для забезпечення інтерактивності в інтерфейсі (рис. 3.10):

```
namespace MauiApp3.Models
{
    public class TodoItem : INotifyPropertyChanged
    {
        private string taskTitle;
        private bool isCompleted;
        public int Id { get; set; }

        public string Title
        {
            get => taskTitle;
            set
            {
                if (taskTitle != value)
                {
                    taskTitle = value;
                    OnPropertyChanged(nameof(Title));
                }
            }
        }

        public bool IsCompleted
        {
            get => isCompleted;
            set
            {
                if (isCompleted != value)
                {
                    isCompleted = value;
                    OnPropertyChanged(nameof(IsCompleted));
                }
            }
        }
    }
}
```

Рисунок 3.10 – TODO-Item

- властивість "Id" використовується для унікальної ідентифікації кожної задачі;

- властивість "Title" зберігає назву завдання. Використовує метод "OnPropertyChanged", щоб повідомити інтерфейс про зміну назви завдання;
- властивість "IsCompleted" вказує, чи виконана задача. Також через "OnPropertyChanged" повідомляє інтерфейс про зміну стану задачі;
- реалізація інтерфейсу "INotifyPropertyChanged": - інтерфейс "INotifyPropertyChanged" дозволяє автоматично оновлювати інтерфейс користувача, коли створені властивості змінюються;
- метод "OnPropertyChanged" викликає подію "PropertyChanged" для вказаної властивості.

3.5 Файл **TodoViewModel.cs**

У файлі "TodoViewModel.cs" реалізовано модель подання "ViewModel" для списку задач у застосунку. Вона відповідає за зв'язок між даними з моделі та інтерфейсом "Views", забезпечуючи логіку роботи програми і спрощуючи прив'язку даних:

- властивість "Todos" - містить список задач. "ObservableCollection" автоматично оновлює інтерфейс при додаванні, видаленні або зміні елементів;
- властивість "NewTodoText" - використовується для зберігання тексту нової задачі, яку користувач вводить у відповідне поле інтерфейсу;
- "TodoContext" використовується для доступу до локальної бази даних, яка містить задачі;
- ініціалізація даних - при створенні об'єкта "TodoViewModel" ініціалізується контекст бази даних і завантажуються всі задачі.

```

namespace MauiApp3.ViewModels
{
    public partial class TodoViewModel : ObservableObject
    {
        [ObservableProperty]
        private string newTodoText;

        public ObservableCollection<TodoItem> Todos { get; } = new ObservableCollection<TodoItem>();

        private readonly TodoContext _context;

        public TodoViewModel()
        {
            _context = new TodoContext();
            LoadTodosCommand.Execute(null);
        }
    }
}

```

Рисунок 3.11 - Файл TodoViewModel.cs

- команди використання атрибуту [RelayCommand] автоматично генерує команди для виконання дій. Це дозволяє зв'язувати методи ViewModel із подіями в інтерфейсі;
- "LoadTodosAsync". Завантажує всі задачі з бази даних у колекцію (рис 3.12);
- "AddTodoAsync". Додає нове завдання до бази даних та оновлює колекцію (3.12).

```

[RelayCommand]
private async Task LoadTodosAsync()
{
    var todos = await _context.Todos.ToListAsync();
    Todos.Clear();
    foreach (var todo in todos)
    {
        Todos.Add(todo);
    }
}

[RelayCommand]
private async Task AddTodoAsync()
{
    if (!string.IsNullOrWhiteSpace(NewTodoText))
    {
        var todo = new TodoItem { Title = NewTodoText };
        _context.Todos.Add(todo);
        await _context.SaveChangesAsync();

        Todos.Add(todo);
        NewTodoText = string.Empty;
    }
}

```

Рисунок 3.12 – Функції для завантаження та додавання даних

- "DeleteTodoAsync". Видаляє обране завдання з бази даних та колекції;
- "UpdateTodoAsync". Оновлює стан або назву задачі в базі даних.

```

        NewTodoText = string.Empty;
    }
}

[RelayCommand]
private async Task DeleteTodoAsync(TodoItem todo)
{
    if (todo != null)
    {
        _context.Todos.Remove(todo);
        await _context.SaveChangesAsync();

        Todos.Remove(todo);
    }
}

[RelayCommand]
private async Task UpdateTodoAsync(TodoItem todo)
{
    if (todo != null)
    {
        _context.Todos.Update(todo);
        await _context.SaveChangesAsync();
    }
}
}

```

Рисунок 3.13 – функції видалення та оновлення даних

3.6 Файл App.xaml

У файлі "App.xaml" задаються глобальні налаштування, стилі та ресурси для всього застосунку. Це головний файл щодо теми застосунку, де визначаються стилі оформлення, кольорові палітри та інші ресурси, які будуть використовуватися на всіх сторінках програми (рис. 3.14).

```

<?xml version="1.0" encoding="UTF-8" ?>
<Application xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:local="clr-namespace:MauiApp3"
    x:Class="MauiApp3.App">
    <Application.Resources>
        <ResourceDictionary>
            <ResourceDictionary.MergedDictionaries>
                <ResourceDictionary Source="Resources/Styles/Colors.xaml" />
                <ResourceDictionary Source="Resources/Styles/Styles.xaml" />
            </ResourceDictionary.MergedDictionaries>
            <ResourceDictionary Source="Resources/Styles/Dictionary1.xaml" />
            </ResourceDictionary>
            <converters:TextDecoration x:Key="BooleanToTextDecorationsConverter" xmlns:converters="clr-namespace:MauiApp3.Converters" />
            </ResourceDictionary>
        </Application.Resources>
    </Application>

```

Рисунок 3.14 - файл "App.xaml"

3.7 Файл App.xaml.cs

У файлі "App.xaml.cs" описується основна логіка запуску програми, ініціалізація бази даних і налаштування головної сторінки. Ось що він виконує в застосунку:

- "InitializeComponent()" - Завантажує ресурси програми з App.xaml;
- база даних "TodoContext". Метод "EnsureCreated()" перевіряє, чи існує база даних, і створює її, якщо вона ще не була створена. Це гарантує, що структура бази даних готова до роботи з задачами (рис. 3.15);
- головна сторінка "MainPage". Задається головною сторінкою програми, використовуючи обгортку "NavigationPage", що дозволяє здійснювати навігацію між сторінкам;
- подія "OnStart". Виконується під час запуску застосунку. Завантажує всі задачі з бази даних і виводить їх у консоль для діагностики.

```

1  using MauiApp3.Data;
2  namespace MauiApp3
3  {
4      public partial class App : Application
5      {
6          public App()
7          {
8              InitializeComponent();
9              using (var context = new TodoContext())
10             {
11                 context.Database.EnsureCreated();
12             }
13
14             MainPage = new NavigationPage(new MainPage());
15         }
16
17         protected override void OnStart()
18         {
19             using (var context = new TodoContext())
20             {
21                 var tasks = context.Todos.ToList();
22                 foreach (var task in tasks)
23                 {
24                     Console.WriteLine($"ID: {task.Id}, Title: {task.Title}, IsCompleted: {task.IsCompleted}");
25                 }
26             }
27         }
28     }
29 }

```

Рисунок 3.15 – Клас App

3.8 База даних SQLite та шаблон MVVM

SQLite — це компактна вбудована реляційна база даних з відкритим кодом. Вона є однією з найпопулярніших у світі, відзначена нагородою Google-O'Reilly Open Source Awards і широко використовується для організації зберігання даних у застосунках та системах [12]. SQLite підтримується практично на всіх основних платформах, включаючи iOS, Android, Windows і macOS. Це робить її сумісною з .NET MAUI, який створений для кросплатформених застосунків. База даних SQLite зберігається у вигляді одного файлу на пристрої. Це спрощує зберігання даних і забезпечує автономну роботу застосунку без необхідності підключення до зовнішнього сервера.

Шаблон MVVM (Model-View-ViewModel) — це архітектурний підхід до розробки програмного забезпечення, який особливо добре підходить для сучасних платформ розробки інтерфейсу користувача, таких як .NET MAUI. Щоб краще зрозуміти принципи MVVM, необхідно розібратися в трьох його основних компонентах: модель, подання та модель подання [9].

Модель (Model) представляє дані та бізнес-логіку застосунку. Вона відповідає за управління даними, станом і бізнес-правилами програми. Зазвичай модель складається з РОСО-об'єктів (Plain Old CLR Objects) або сутностей даних, які визначають структуру та властивості даних. Модель взаємодіє з джерелами даних, такими як бази даних, вебслужби або інші API, для отримання, обробки та збереження даних.

Подання (View) відповідає за визначення інтерфейсу користувача (UI) програми. Воно відображає дані для користувача та обробляє взаємодії з ним, такі як натискання, жести та інші події введення. Зазвичай представлення визначається за допомогою XAML (Extensible Application Markup Language) у

.NET MAUI, що забезпечує декларативний спосіб створення інтерфейсу користувача.

ViewModel служить посередником між Model (моделлю) і View (представленням). Він інкапсулює логіку відображення та стан інтерфейсу користувача, надаючи дані та команди представленням, одночасно зберігаючи чітке розмежування з моделлю. ViewModel відповідає за перетворення і підготовку даних для відображення, обробку введення користувача та керування взаємодією між представленням та моделлю.

3.9 Послідовність виконання програми

Алгоритм для блок-схеми (рис. 3.16):

- початок - процес починається з файлів App.xaml та App.xaml.cs, які ініціалізують застосунок;
- MauiProgram.cs - після ініціалізації створюється налаштування сервісів через MauiProgram.cs;
- підключення стилів і кольорів - далі застосунок завантажує стилі та кольори з файлів Colors.xaml, Style.xaml, Styles.xaml;
- MainPage.xaml та MainPage.xaml.cs - на основі цих налаштувань відображається MainPage.xaml з функціональністю, що залежить від TodoViewModel;
- робота з даними - TodoViewModel зв'язується з TodoContext, який керує збереженням даних у базі даних;
- обробка задач -взаємодія з моделями даних (TodoItem) для відображення, зміни та видалення задач;
- кінець - процес завершується, коли дані оновлюються або зберігаються в базі даних, і користувач бачить зміни в інтерфейсі.

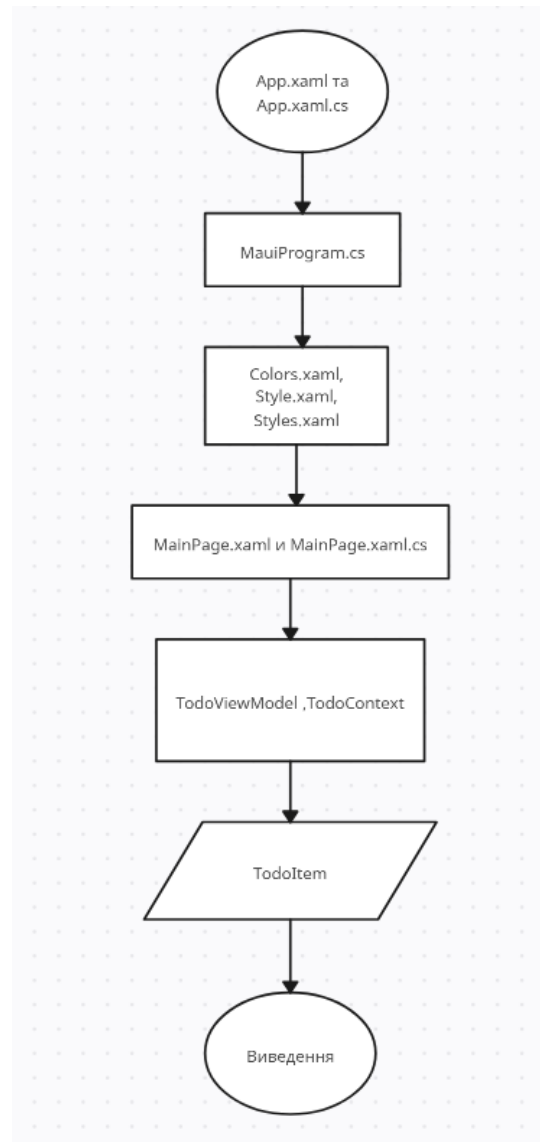


Рисунок 3.16 – Алгоритм виконання програми

3.10 Висновки за розділом 3

Розглянуто процес створення застосунку для планування робочого процесу. Описано структуру файлів та функції, які виконуються в класах. Детально розглянуто та пояснено код проекту.

4. ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Інструмент для тестування застосунку

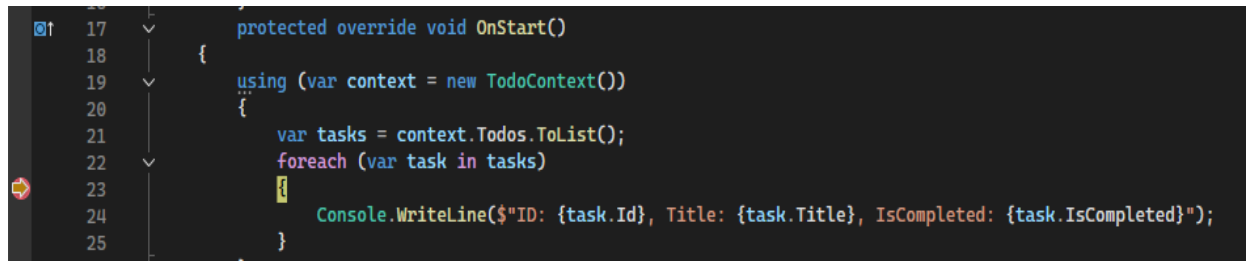
До того як перевіряти роботу застосунку та плавильність виконання функції, треба спочатку скористатися функцією `debug` за допомогою встановлених `breakpoint`.

Debugger - це інструмент відладки, який надає Visual Studio. У контексті Visual Studio, коли ви відлагоджуєте своє застосунок, це зазвичай означає, що запускається застосунок з підключеним відлагоджувачем (тобто в режимі відладки). Коли ви це робите, відлагоджувач пропонує безліч способів побачити, що ваш код робить під час виконання. Є можливість крок за кроком проходити через свій код і перевіряти значення, що зберігаються в змінних, ставити спостереження за змінними, щоб побачити, коли їх значення змінюється, а також аналізувати шлях виконання вашого коду та інше [13].

Breakpoint є одним із найважливіших методів налагодження в наборі інструментів розробника. Необхідно встановити точки зупину скрізь, де потрібно призупинити виконання налагоджувача. Наприклад, можна побачити стан змінних коду або переглянути стек викликів у певній точці зупину.

Ставимо точку зупинки на рядку, де описана функція передачі елементів колекції у консоль. Також ставимо їх у функціях обробки даних, наприклад завантаження, додавання, оновлення та видалення. Запускаємо нашу програму в режимі налагодження (рис. 4.1).

У вікні нижче бачимо дані, які з'являються, коли проходимо від однієї функції до іншої. Там відображається багато важливих даних.



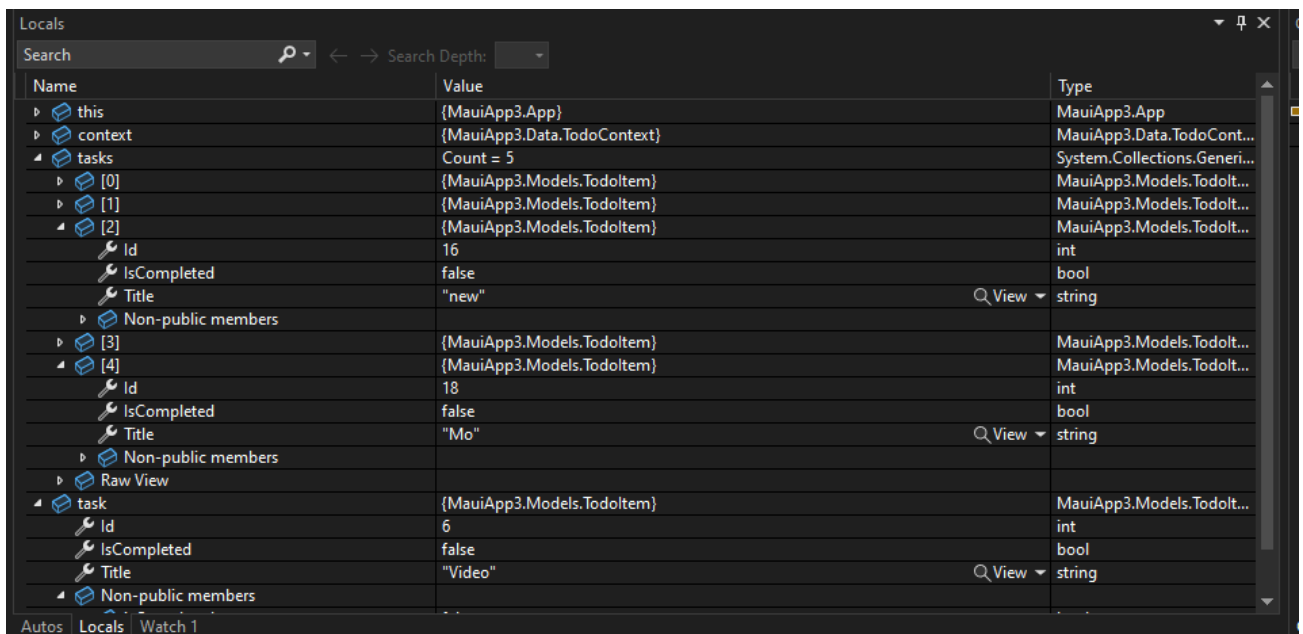
```

17 protected override void OnStart()
18 {
19     using (var context = new TodoContext())
20     {
21         var tasks = context.Todos.ToList();
22         foreach (var task in tasks)
23         {
24             Console.WriteLine($"ID: {task.Id}, Title: {task.Title}, IsCompleted: {task.IsCompleted}");
25         }
26     }
27 }

```

Рисунок 4.1 - Debug

Серед даних, які можна побачити у вікні debug – є дані під назвою "tasks". Це назва нашої змінної, яку описано у функції OnStart. Ця змінна зберігає дані з нашої колекції. Тобто, у неї зберігаються наші завдання, які створюються у застосунку(рис. 4.2).



Name	Value	Type
this	{MauiApp3.App}	MauiApp3.App
context	{MauiApp3.Data.TodoContext}	MauiApp3.Data.TodoCont...
tasks	Count = 5	System.Collections.Generi...
[0]	{MauiApp3.Models.TodoItem}	MauiApp3.Models.TodoIt...
[1]	{MauiApp3.Models.TodoItem}	MauiApp3.Models.TodoIt...
[2]	{MauiApp3.Models.TodoItem}	MauiApp3.Models.TodoIt...
Id	16	int
IsCompleted	false	bool
Title	"new"	string
Non-public members		
[3]	{MauiApp3.Models.TodoItem}	MauiApp3.Models.TodoIt...
[4]	{MauiApp3.Models.TodoItem}	MauiApp3.Models.TodoIt...
Id	18	int
IsCompleted	false	bool
Title	"Mo"	string
Non-public members		
Raw View		
task	{MauiApp3.Models.TodoItem}	MauiApp3.Models.TodoIt...
Id	6	int
IsCompleted	false	bool
Title	"Video"	string
Non-public members		

Рисунок 4.2 – Вікно даних

Можна підтвердити, що дані існують та передаються у функцію через властивість, яка є нашою колекцією. Такий же результат отримаємо, коли перевіряємо функції роботи з даними (рис. 4.3).

При додаванні завдання за допомогою функції `AddTodoAsync` бачимо у вікні змінну `todo`, яку створили у цій функції. Вона містить завдання, які були створені та дані моделі нашого класу, наприклад `Id` та `Title` (рис 4.3).

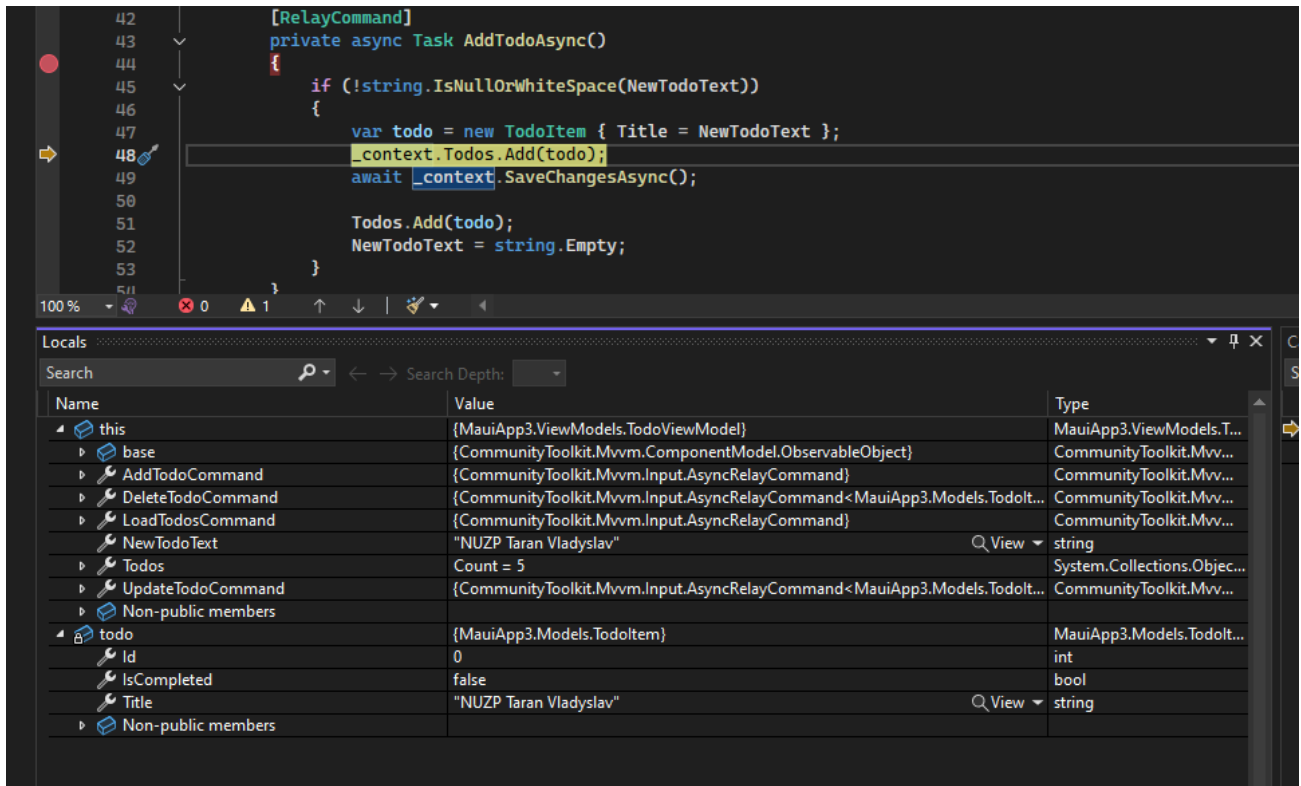


Рисунок 4.3 – Додавання нового завдання

4.2 Тест програмного забезпечення

Після перевірки роботи проекту через `debugger`, коли запускається застосунок, перше що можемо зробити – це подивитись на те, як відображається UI нашого застосунку (рис. 4.4).

Маємо застосунок у мінімалістичному стилі з назвою "Мої завдання". Також маємо кнопку додати та сам список завдань. З попереднього тесту бачимо, що є нове завдання, де вказано ім'я студента та університет.

Мої завдання

☐ New	Відреагувати
☐ Name	Відреагувати
☐ NUZP Taran Vladyslav KNT-223M	Відреагувати
☐ Sam	Відреагувати
☐ CLoSe	Відреагувати

Рисунок 4.4 – UI застосунку TODO-List

Напроти усіх завдань є кнопка з можливістю відзначити завдання, як виконане (рис. 4.5). Після цього завдання закреслюється.

☒ Name	Відреагувати
☐ NUZP Taran Vladyslav KNT-223M	Відреагувати
☒ Sam	Відреагувати

Рисунок 4.5 – Закреслення виконаного завдання

Зараз, коли завдання виконано, є сенс його видалити. Для цього реалізовано функцію видалення, якою можна скористуватися, якщо перетягнути завдання праворуч (рис. 4.6).

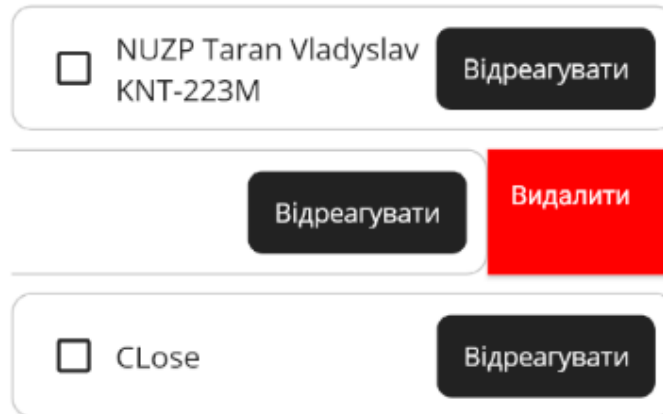


Рисунок 4.6 – Функція видалення завдання

Після видалення додамо ще одне завдання (рис 4.7). Також можемо знову зробити завдання активним. Так бачимо, що завдання "Name" знову активне.

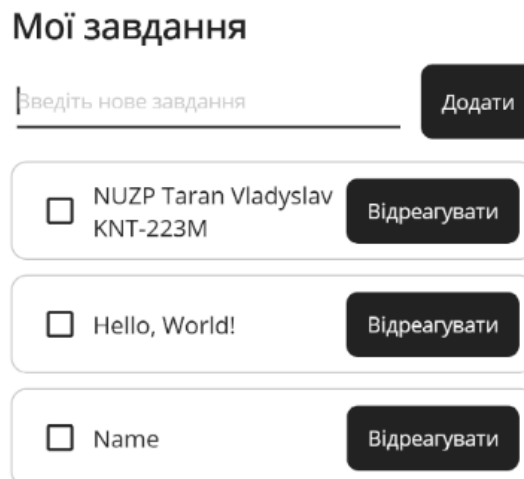


Рисунок 4.7 – Активація завдання

Також є функція редагування або оновлення існуючих завдань. Для того, щоб це зробити, необхідно натиснути на завдання та написати щось інше. Змінюємо завдання "Name" на "main" (рис. 4.8).

Мої завдання

☐ NUZP Taran Vladyslav
KNT-223M

☐ Hello, World!

☐

Рисунок 4.8 - Редагування

4.3 Висновки за розділом 4

Тестування застосунку є необхідною частиною для того, щоб дізнатися, чи правильно усі функції виконують свої завдання та виправити помилки, якщо вони є. Були протестовані усі основні функції.

ВИСНОВКИ

В ході виконання дипломної кваліфікаційної роботи магістра було проаналізовано та досліджено процес розробки програмного забезпечення для планування робочого процесу. Розглянуто різні етапи створення програмного забезпечення.

У першому розділі розглянуто різні застосунки у Google Play Market та зроблено їх аналіз за певними ознаками.

У другому розділі обрано матеріали та методи: платформи, фреймворки та мови програмування для створення програмного забезпечення для планування робочого процесу.

У третьому розділі детально показано процес створення програмного забезпечення: код файлів, які використовуються для цього та встановлені залежності та структура проекту.

У четвертому розділі описано тестування функцій програмного забезпечення з використанням debugger.

Програмне забезпечення для планування робочого процесу дозволить людині зробити план свого дня та заощадити свій час.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Notes [Electronic resource]. - Access mode: Google Play Market.
2. Google Tasks [Electronic resource]. - Access mode: <https://play.google.com/store/search?q=Google%20Tasks&c=apps&hl=uk>.
3. Todoist: Planner & Calendar [Electronic resource]. - Access mode: <https://play.google.com/store/search?q=Todoist%3A%20Planner%20%26%20Calendar&c=apps&hl=uk>.
4. Packt, Hands-On Visual Studio 2022 [Текст] / Packt. 2022. – 4 с.
5. Learn Microsoft [Electronic resource]. - Access mode: <https://learn.microsoft.com/uk-ua/visualstudio/get-started/visual-studio-ide?view=vs-2022>.
6. Procoding [Electronic resource]. - Access mode: <https://www.procoding.com.ua/rozrobka-mobilnykh-dodatkov/>.
7. Майо, Microsoft Visual Studio 2010 [Текст] / Джо Майо. 2011. – 39 с.
8. Griffiths, Programming C# 12 [Текст] / Ian Griffiths. 2024. – 35-36,119 с.
9. Wells, Building Cross-Platform Apps with .Net MAUI and MVVM [Текст] / Frank Wells. 2024. – 25 с.
10. Для чого використовується React Native [Electronic resource]. - Access mode: <https://uk.goodteam.dev/blog/dlya-chego-ispolzuetsya-react-native>.
11. Емулятор Android для Windows 2024 року — поради щодо вибору та 10 найкращих платформ [Електрон. ресурс]. -Режим доступу: https://www.moyo.ua/ua/news/emulyator_android_dlya_windows_2024_goda_sovety_po_vyboru_i_10_luchshikh_platform.html.
12. Що таке SQLite? [Електрон. ресурс]. -Режим доступу: <https://freehost.com.ua/ukr/faq/wiki/chto-takoe-sqlite/>.

13. First look at the Visual Studio Debugger [Электрон. ресурс]. -
Режим доступа: <https://learn.microsoft.com/en-us/visualstudio/debugger/debugger-feature-tour?view=vs-2022>.

ДОДАТОК А
Технічне завдання

Вступ

Програмне забезпечення може використовуватися для підтримки функціонування застосунку для планування робочого процесу.

A.1 Підстава для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему «Дослідження та програмна реалізація застосунку для планування рбочого процесу», затверджене наказом Національного університету «Запорізька політехніка» № 398 від 07 жовтня 2024 р.

A.2 Призначення розробки

Програмний продукт необхідний для підтримки процесу функціонування застосунку для планування робочого процесу.

A.3 Основні вимоги до програми, що розробляється

A.3.1 Вимоги до функціональних характеристик

Програмне забезпечення застосунку забезпечує виконання таких функцій:

- можливість застосувати цей застосунок на телефоні з операційною системою Android;
- можливість відображення UI;
- можливість створення тасків;
- можливість редагування тасків;
- можливість виділення тасків;

- можливість зберігання тасків.

A.3.2 Вимоги до інтерфейсу програми

Інтерфейс програмного забезпечення застосунку для планування робочого процесу повинен бути зрозумілим та зручним та користувача. Налаштування повинно підходити для роботи з телефона. Користувач повин мати змогу створювати завдання.

A.3.3 Вимоги до надійності

Програмне забезпечення застосунку для планування робочого процесу забезпечить надійне функціонування.

A.3.4 Умови експлуатації

Для експлуатації програмного забезпечення для підтримки процесу функціонування застосунку для планування робочого процесу необхідна наявність ноутбука, середовища для розробки, емулятора або реального телефону.

A.3.5 Вимоги до складу та параметрів технічних засобів

Для експлуатації програмного забезпечення застосунку для планування робочого процесу необхідно таке апаратне та програмне забезпечення:

- операційна система Android 9;
- процесор 2 x 2.05 ГГц;

- оперативна пам'ять 6 гб;
- внутрішня пам'ять 64 гб;
- Wi-fi.

A.3.6 Вимоги до маркування і пакування

Програма для підтримки функціонування застосунку для планування робочого процесу може бути записана на різних носіях інформації.

A.4 Вхідні дані до роботи

Вхідними даними до програмного забезпечення застосунку для планування робочого процесу є назва завдань, які створюються за допомогою застосунку.

Вихідними даними програмного забезпечення застосунку для планування робочого процесу є завдання, створені за допомогою застосунку.

ДОДАТОК Б
Фрагмент тексту програми

MainPage.xaml

```

<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:viewModels="clr-namespace:MauiApp3.ViewModels"
    x:Class="MauiApp3.MainPage">

    <ContentPage.BindingContext>
        <viewModels:TodoViewModel />
    </ContentPage.BindingContext>

    <StackLayout Padding="20" Spacing="10">

        <Label Text="Мої завдання"
            Style="{StaticResource HeaderLabelStyle}" />

        <Grid ColumnSpacing="10">
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="*" />
                <ColumnDefinition Width="Auto" />
            </Grid.ColumnDefinitions>

            <Entry Placeholder="Введіть нове завдання"
                Text="{Binding NewTodoText}"
                Grid.Column="0"
                BackgroundColor="White"
                TextColor="{StaticResource TextColor}"
                HeightRequest="50"
                VerticalOptions="Center" />

            <Button Text="Додати"
                Command="{Binding AddTodoCommand}"
                Style="{StaticResource PrimaryButtonStyle}"
                Grid.Column="1" />
        </Grid>
    </StackLayout>
</ContentPage>

```

```

<CollectionView ItemsSource="{Binding Todos}"
    SelectionMode="None">
    <CollectionView.ItemTemplate>
        <DataTemplate>
            <SwipeView>

                <SwipeView.RightItems>
                    <SwipeItems Mode="Reveal">
                        <SwipeItem Text="Видалити"
                            BackgroundColor="Red"
                            IconImageSource="delete.png"
                            Command="{Binding Source={RelativeSource
AncestorType={x:Type viewModels:TodoViewModel}}},
Path=DeleteTodoCommand}"
                            CommandParameter="{Binding .}" />
                    </SwipeItems>
                </SwipeView.RightItems>

                <Frame CornerRadius="10"
                    Padding="10"
                    Margin="0,5"
                    BackgroundColor="White">
                    <Grid>
                        <Grid.ColumnDefinitions>
                            <ColumnDefinition Width="Auto" />
                            <ColumnDefinition Width="*" />
                            <ColumnDefinition Width="Auto" />
                        </Grid.ColumnDefinitions>

                        <CheckBox IsChecked="{Binding IsCompleted}"
                            CheckedChanged="OnIsCompletedChanged"
                            VerticalOptions="Center"
                            Grid.Column="0" />

                        <Entry Text="{Binding Title}"
                            Style="{StaticResource BodyLabelStyle}"
                            VerticalOptions="Center"

```

```

        Grid.Column="1"
        IsVisible="{Binding IsEditing}" />

<Label Text="{Binding Title}"
        Style="{StaticResource BodyLabelStyle}"
        VerticalOptions="Center"
        Grid.Column="1"
        TextDecorations="{Binding IsCompleted, Converter={StaticResource
TextDecoration}}"
        IsVisible="{Binding IsEditing, Converter={StaticResource
InverseBooleanConverter}}" />

        <Button Text="Відреагувати"
                Command="{Binding Source={RelativeSource
AncestorType={x:Type viewModels:TodoViewModel}},
Path=ToggleEditingCommand}"
                CommandParameter="{Binding .}"
                Grid.Column="2"
                VerticalOptions="Center" />

    </Grid>
</Frame>

</SwipeView>
</DataTemplate>
</CollectionView.ItemTemplate>
</CollectionView>

</StackLayout>
</ContentPage>

```

MainPage.cs

```

using MauiApp3.ViewModels;
using MauiApp3.Models;
namespace MauiApp3
{
    public partial class MainPage : ContentPage

```

```

{
    public MainPage()
    {
        InitializeComponent();
    }

    private void OnTitleChanged(object sender,
TextChangedEventArgs e)
    {
        var entry = sender as Entry;
        var todo = entry.BindingContext as TodoItem;

        if (todo != null)
        {
            var viewModel = BindingContext as
TodoViewModel;
            viewModel.UpdateTodoCommand.Execute(todo);
        }

        private void OnIsCompletedChanged(object sender,
CheckedChangedEventArgs e)
        {
            var checkBox = sender as CheckBox;
            var todo = checkBox.BindingContext as TodoItem;

            if (todo != null)
            {
                var viewModel = BindingContext as
TodoViewModel;
                viewModel.UpdateTodoCommand.Execute(todo);
            }
        }
    }
}

```

TodoViewModel.cs

```

using CommunityToolkit.Mvvm.ComponentModel;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using CommunityToolkit.Mvvm.Input;
using MauiApp3.Models;
using MauiApp3.Data;
using Microsoft.EntityFrameworkCore;
namespace MauiApp3.ViewModels
{
    public partial class TodoViewModel : ObservableObject
    {
        [ObservableProperty]
        private string newTodoText;

        public ObservableCollection<TodoItem> Todos { get;
} = new ObservableCollection<TodoItem>();

        private readonly TodoContext _context;

        public TodoViewModel()
        {
            _context = new TodoContext();
            LoadTodosCommand.Execute(null);
        }

        [RelayCommand]
        private async Task LoadTodosAsync()
        {
            var todos = await _context.Todos.ToListAsync();
            Todos.Clear();
            foreach (var todo in todos)
            {
                Todos.Add(todo);
            }
        }
    }
}

```

```

        }
    }

    [RelayCommand]
    private async Task AddTodoAsync()
    {
        if (!string.IsNullOrEmpty(NewTodoText))
    {
        var todo = new TodoItem
        {
            Title = NewTodoText,
            IsCompleted = false,
            IsEditing = false
        };

        _context.Todos.Add(todo);
        await _context.SaveChangesAsync();

        Todos.Add(todo);
        NewTodoText = string.Empty;
    }

    [RelayCommand]
    private async Task DeleteTodoAsync(TodoItem todo)
    {
        if (todo != null)
        {
            _context.Todos.Remove(todo);
            await _context.SaveChangesAsync();

            Todos.Remove(todo);
        }
    }

    [RelayCommand]
    private async Task UpdateTodoAsync(TodoItem todo)
    {
        if (todo != null)
        {
            _context.Todos.Update(todo);

```

```

        await _context.SaveChangesAsync();
    }
}
[RelayCommand]
private void ToggleEditing(TodoItem todo)
{
    todo.IsEditing = !todo.IsEditing;
}
}
}

```

Dictionary1.xaml

```

<?xaml-comp compile="true" ?>
<ResourceDictionary
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">
    <Color x:Key="PrimaryColor">#242424</Color>
    <Color x:Key="AccentColor">#242424</Color>
    <Color x:Key="BackgroundColor">#242424</Color>
    <Color x:Key="TextColor">#212121</Color>
    <Color x:Key="SecondaryTextColor">#242424</Color>

    <Style x:Key="HeaderLabelStyle" TargetType="Label">
        <Setter Property="FontSize" Value="24" />
        <Setter Property="FontAttributes" Value="Bold" />
        <Setter Property="TextColor" Value="{StaticResource TextColor}" />
    </Style>

```

```

<Style x:Key="BodyLabelStyle" TargetType="Label">
    <Setter Property="FontSize" Value="16" />
    <Setter Property="TextColor" Value="{StaticResource TextColor}" />
</Style>

```

```

<Style x:Key="PrimaryButtonStyle" TargetType="Button">
    <Setter Property="BackgroundColor" Value="{StaticResource
PrimaryColor}" />
    <Setter Property="TextColor" Value="White" />
    <Setter Property="CornerRadius" Value="8" />
    <Setter Property="HeightRequest" Value="50" />
</Style>

```

```

</ResourceDictionary>

```

```

using System;
using System.Collections.Generic;
using System.Globalization;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MauiApp3.Converters
{
    public class InverseBooleanConverter : IValueConverter
    {

```

```

        public object Convert(object value, Type targetType,
object parameter, CultureInfo culture)
        {
            return !(bool)value;
        }

        public object ConvertBack(object value, Type
targetType, object parameter, CultureInfo culture)
        {
            return !(bool)value;
        }
    }
}

```

TodoItem.cs

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MauiApp3.Models
{
    public class TodoItem : INotifyPropertyChanged
    {
        private string taskTitle;
        private bool isCompleted;
        public int Id { get; set; }
        private bool isEditing;

        public TodoItem()
        {
            IsEditing = false;
        }
        public string Title
        {
            get => taskTitle;
            set

```

```

        {
            if (taskTitle != value)
            {
                taskTitle = value;
                OnPropertyChanged(nameof(Title));
            }
        }
    }

    public bool IsCompleted
    {
        get => isCompleted;
        set
        {
            if (isCompleted != value)
            {
                isCompleted = value;
                OnPropertyChanged(nameof(IsCompleted));
            }
        }
    }

    public bool IsEditing
    {
        get => isEditing;
        set
        {
            if (isEditing != value)
            {
                isEditing = value;
                OnPropertyChanged(nameof(IsEditing));
            }
        }
    }

    public event PropertyChangedEventHandler
    PropertyChanged;

    protected void OnPropertyChanged(string propertyName)
    {
        PropertyChanged?.Invoke(this, new
        PropertyChangedEventArgs(propertyName));
    }

```

```

    }
}

```

TodoContext.cs

```

using System.Text;
using System.Threading.Tasks;

namespace MauiApp3.Data
{
    public class TodoContext : DbContext
    {
        public DbSet<TodoItem> Todos { get; set; }

        public TodoContext()
        {
            Database.EnsureCreated();
        }

        protected override void
        OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            string dbPath =
            Path.Combine(FileSystem.AppDataDirectory, "todo_new.db");
            optionsBuilder.UseSqlite($"Filename={dbPath}");
        }
    }
}

```

ДОДАТОК В
Слайди презентації

Міністерство освіти і науки України Національний університет «Запорізька політехніка» Кафедра програмних засобів Дослідження та програмна реалізація додатку для планування робочого процесу		1
Виконав: Студент групи КНТ-223М	Владислав Таран	
Керівник: к.т.н., доцент	Тетяна Зайко	

Рисунок В1 – Слайд № 1

		2
	ОБ’ЄКТ І ПРЕДМЕТ ДОСЛІДЖЕННЯ. МЕТА РОБОТИ Об’єкт дослідження – процес розробки програмного забезпечення для планування робочого процесу. Предмет дослідження – програмні засоби для планування робочого процесу. Мета роботи – дослідження та розробка програмного забезпечення для вирішення проблеми з плануванням робочого процесу.	

Рисунок В2 – Слайд № 2

3			
АНАЛОГИ TODO-LIST ДОДАТКІВ			
Атрибут	Xiaomi Notes	Google Calendar	Todoist: Planner & Calendar
Дизайн	-	-	+
Можливості	-	+	+
Оригінальність	-	-	+
Зручність у використанні	+	-	+
Ефективність	+	+	+

Рисунок В3 – Слайд № 3

4		
Вибір IDE		
Назва IDE	Переваги	Недоліки
Android Studio	Це офіційна IDE для Android від Google. Підтримує Kotlin і Java, пропонує вбудовані інструменти для налагодження, емуляції та перевірки додатків. Глибока інтеграція з Android SDK, швидке завантаження емулятора	Висока вимога до ресурсів системи.
Visual Studio	Ідеально підходить для кросплатформної розробки. Дозволяє створювати додатки для iOS, Android і Windows з використанням C#. Спільний код для кількох платформ, інтеграція з іншими продуктами Microsoft.	Деякі функції потребують додаткової оптимізації для окремих платформ.
Rider	Підтримка кількох платформ. Rider працює на Windows, macOS і Linux. Інтеграція з популярними фреймворками: Rider має вбудовану підтримку таких фреймворків, як ASP.NET, Xamarin, Unity та інших	Rider – це платним продуктом. Rider, як і багато інших IDE, може бути досить вимогливим до ресурсів комп'ютера. Відсутність повної інтеграції з усіма інструментами Microsoft, такими як Azure DevOps, може бути недоліком для деяких команд.
Intelli J IDEA	Підтримує розробку на багатьох мовах програмування, включаючи Kotlin і Java, підходить для мобільних додатків. Зрозумілий інтерфейс та величезна кількість плагінів	Платні функції в Ultimate Edition

Рисунок В4 – Слайд № 4

5

Вибір мови програмування

Назва	Переваги	Недоліки	Назва	Переваги	Недоліки
C#,Xaml, .Net MAUI, Xamarin.	Використання C# для бізнес-логіки та XAML для інтерфейсу дозволяє розділяти відповідальність між частинами програми, що робить код чистим та зручним для підтримки. C# є частиною платформи .NET, що надає доступ до багатьох бібліотек, інструментів і технологій для розробки.Обидва фреймворки (Xamarin та .NET MAUI) дозволяють створювати додатки для кількох платформ (Android, iOS, Windows, macOS) з однією кодовою базою.	Хоча .NET MAUI обіцяє покращену продуктивність, але в деяких випадках буває навпаки. Хоча .NET MAUI значно покращує доступ до <u>нативних API</u> , в <u>Xamarin</u> все ще можна зустріти ситуації, коли потрібно використовувати <u>платформозалежний код</u> для деяких специфічних функцій.З новим фреймворком .NET MAUI можна стикнутися з деякими труднощами при переході з <u>Xamarin</u> .	React Native	React Native дозволяє створювати додатки для Android та iOS з єдиною кодовою базою, використовуючи JavaScript та React. Завдяки використанню нативних компонентів і можливості працювати з JavaScript, додатки можуть мати хорошу продуктивність.	Для більш складних або специфічних функцій може знадобитися використання додаткових бібліотек або <u>нативного коду</u> . Розробка <u>кроссплатформних</u> додатків вимагає певного рівня знань у <u>JavaScript</u> та <u>React</u> .
			Flutter (Dart)	Завдяки компіляції в <u>нативний код</u> , додатки, створені за допомогою <u>Flutter</u> , можуть працювати дуже швидко. <u>Flutter</u> дозволяє створювати додатки для <u>Android</u> , <u>iOS</u> та навіть веб-платформ з однієї кодовою базою.	Оскільки <u>Flutter</u> є відносно новим <u>фреймворком</u> , йому ще не вистачає деяких бібліотек і інструментів, доступних для більш зрілих <u>фреймворків</u> . Для розробників, що не мають досвіду з <u>Dart</u> , може бути складно почати роботу з <u>Flutter</u> .

Рисунок B5 – слайд № 5

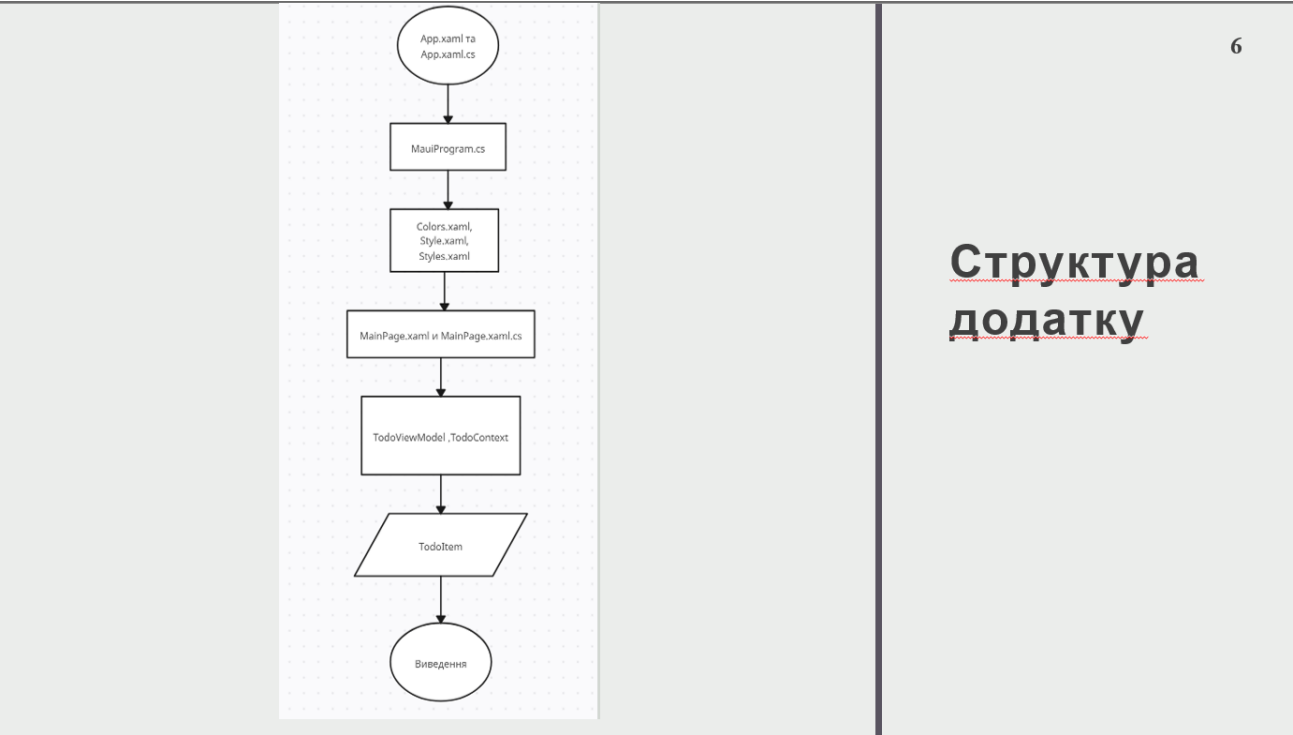


Рисунок B6 – слайд № 6

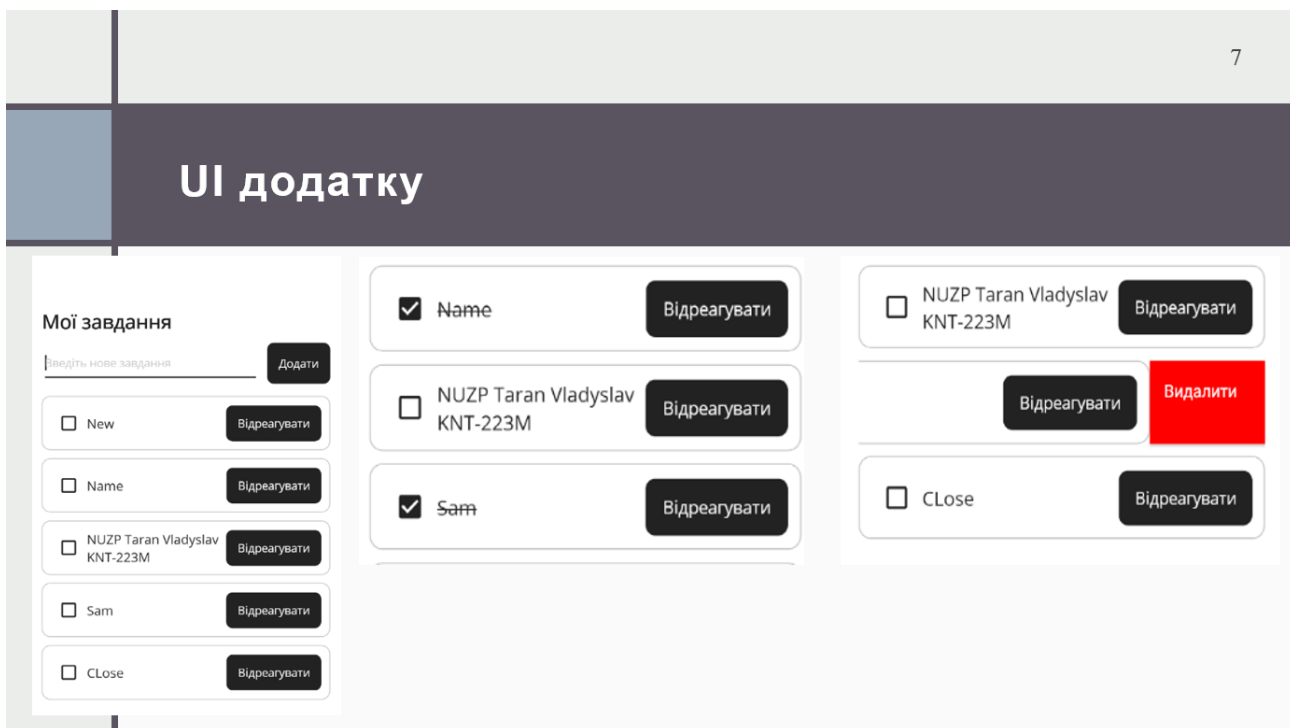


Рисунок В7 – слайд № 7

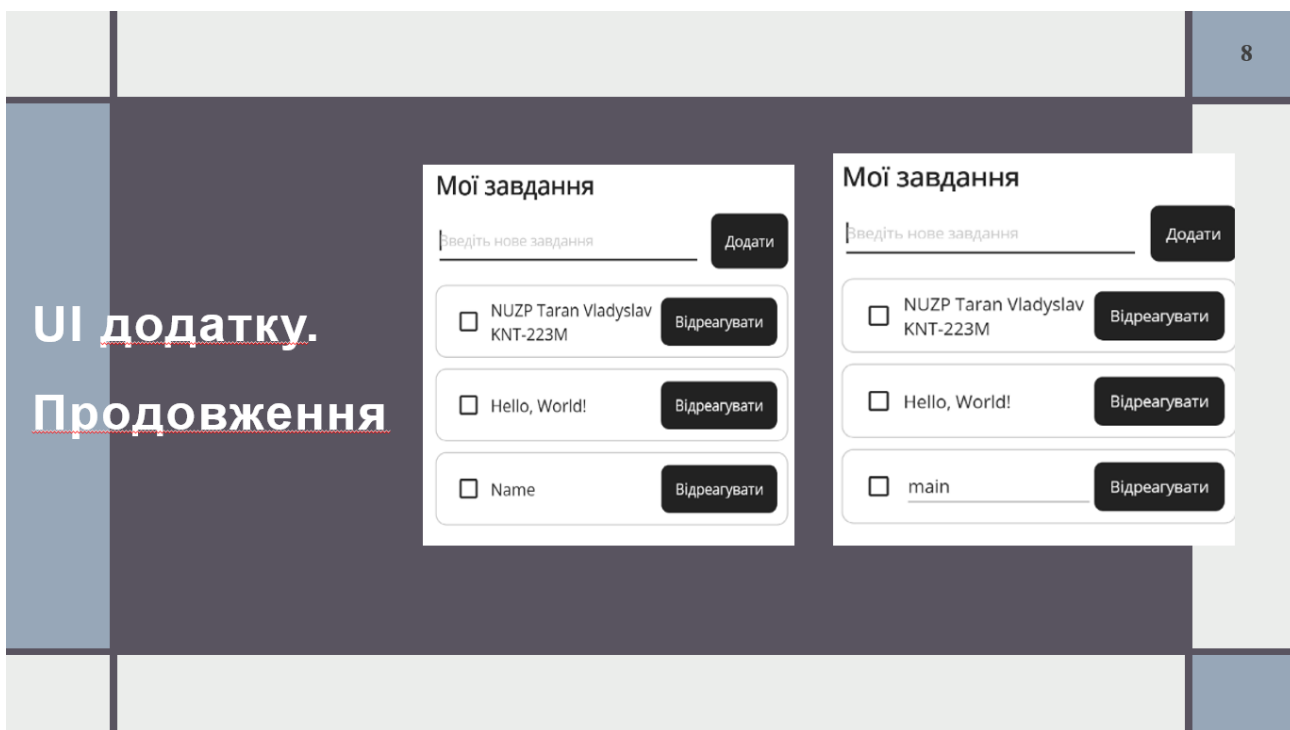


Рисунок В8 – слайд № 8

ВИСНОВКИ

За підсумками виконання дипломної кваліфікаційної роботи було створено додаток для планування робочого процесу. Це рішення має допомогти користувачам із плануванням робочого часу. Під час роботи та створення звіту були розглянуті основні функції сайту. Він був протестован та готовий до виконання завдань.

Рисунок В9 – слайд № 9