Міністерство освіти і науки України
Університет митної справи та фінансів

Факультет інноваційних технологій
Кафедра комп'ютерних наук та інженерії програмного забезпечення

Кваліфікаційна робота магістра

на тему: «Розробка крос-платформного Task-менеджеру на платформі .NET MAUI з використанням технологій експертних систем»

Виконав: студент групи К23-2М

Спеціальність 122 Комп'ютерні науки

Островський В.Д.
(прізвище та ініціали)

Керівник д.е.н., проф. Корнєєв М.В.
(науковий ступінь, вчене звання, прізвище та ініціали)

Рецензент Університет митної справи та фінансів
(місце роботи)

Доцент кафедри кібербезпеки та інфомармаційних технологій
(посада)

к.т.н., доц. Савченко Ю.В.
(науковий ступінь, вчене звання, прізвище та ініціали)

Дніпро – 2025

АНОТАЦІЯ

Островський В. Д. Розробка крос-платформного Task-менеджеру на платформі .NET MAUI з використанням технологій експертних систем.

Кваліфікаційна робота на здобуття освітнього ступеня магістр за спеціальністю 122 «Комп'ютерні науки». – Університет митної справи та фінансів, Дніпро, 2025.

Об'єктом дослідження є програмні засоби для управління завданнями.

Предметом дослідження є процес розробки кросплатформних додатків та експертних систем для управління завданнями.

Мета роботи – розробка та програмна реалізація експертної системи Task-менеджера на платформі .NET MAUI із використанням архітектурного патерну MVVM.

Робота включає аналіз сучасних методів управління завданнями та програмних рішень, таких як Any.DO, Trello, Remember the Milk і Google Calendar, а також дослідження концепції експертних систем, що автоматизують процес прийняття рішень на основі аналізу даних. Обґрунтовано вибір платформи .NET MAUI через її можливість створення кросплатформних застосунків для Windows, macOS, iOS та Android із використанням єдиної кодової бази та підтримкою патерну MVVM, що забезпечує зручність розділення бізнес-логіки та інтерфейсу.

Результати дослідження можуть бути використані для подальшого розвитку систем управління завданнями, зокрема для інтеграції елементів штучного інтелекту, що підвищить ефективність роботи користувачів.

Ключові слова: .NET MAUI, MVVM, крос-платформний додаток, експертна система, управління завданнями, експертні системи.

ABSTRACT

Ostrovskyi V. D. Development of a Cross-Platform Task Manager on the .NET MAUI Platform Using Expert System Technologies.

This project for obtaining a master's degree in speciality 122 "Computer Science." - University of Customs and Finance, Dnipro, 2025.

The object of the research is software tools for task management.

The subject of the research is the process of developing cross-platform applications and expert systems for task management.

The purpose of the work is to develop and implement an expert system task manager on the .NET MAUI platform using the MVVM architectural pattern.

The work includes an analysis of modern task management methods and software solutions such as Any.DO, Trello, Remember the Milk, and Google Calendar, as well as a study of the concept of expert systems that automate the decision-making process based on data analysis. The choice of the .NET MAUI platform is justified due to its ability to create cross-platform applications for Windows, macOS, iOS, and Android using a single codebase and support for the MVVM pattern, which ensures a clear separation of business logic and user interface.

The results of the study can be used for further development of task management systems, particularly for the integration of artificial intelligence elements to enhance user productivity.

Keywords: .NET MAUI, MVVM, cross-platform application, expert system, task management, expert systems.

ЗМІСТ

ВСТУП

Розвиток сучасних інформаційних технологій значно вплинув на підходи до управління завданнями та організації робочих процесів. Зростання обсягів даних та необхідність ефективної координації роботи як в особистих, так і в командних проєктах обумовлюють потребу у створенні гнучких та зручних інструментів для планування. Одним із ключових напрямів у цій галузі є розробка програмного забезпечення для управління завданнями, зокрема Task-менеджерів, які автоматизують процеси організації та підвищують продуктивність користувачів.

Task-менеджери є програмними додатками, що дозволяють користувачам створювати, редагувати та відстежувати виконання завдань, встановлювати пріоритети та отримувати аналітичну інформацію щодо виконаної роботи. В умовах популяризації віддаленої роботи та зростання потреб у мобільності важливим аспектом стає кросплатформенність, тобто можливість роботи на різних пристроях із єдиною базою даних. Платформа .NET MAUI (Multi-platform App UI) дозволяє створювати кросплатформні додатки з використанням єдиного коду, що охоплює операційні системи Windows, macOS, iOS та Android.

Актуальність теми обумовлена потребою у створенні інструменту для управління завданнями, який би поєднував простоту у використанні, ефективність, автоматизацію рутинних процесів та підтримку роботи на різних пристроях. Розробка експертної системи Task-менеджеру з використанням .NET MAUI дозволить автоматизувати розподіл завдань за пріоритетами, відображення статусу їхнього виконання та надання користувачу інтелектуальних підказок для підвищення продуктивності.

Метою дослідження є розробка крос-платформного Task-менеджеру на платформі .NET MAUI з використанням архітектурного патерну MVVM.

Для досягнення мети поставлено такі завдання:

1. Провести аналіз існуючих рішень у сфері управління завданнями та експертних систем.
2. Дослідити можливості платформи .NET MAUI для розробки кросплатформних застосунків.
3. Розробити архітектуру застосунку на основі патерну MVVM.
4. Реалізувати функціональність створення, редагування, видалення та сортування завдань.
5. Забезпечити тестування додатка для перевірки його функціональності та продуктивності.

Об'єктом дослідження є програмні засоби для управління завданнями.

Предметом дослідження є процес розробки крос-платформного додатків та експертних систем.

Практичне значення отриманих результатів полягає у створенні зручного інструменту для управління завданнями, що може бути використаний як у персональних так і командних проєктах. Використання .NET MAUI забезпечує зручність підтримки застосунку та його масштабованість для роботи на декількох платформах з єдиною базою коду.

Наукова новизна роботи полягає у розробці експертної системи Task-менеджеру, яка реалізує автоматизовану систему розподілу завдань за пріоритетами з використанням принципів патерну MVVM.

Робота складається зі вступу, трьох розділів, висновків, списку використаної літератури та додатків. Загальний обсяг роботи становить 70 сторінок, містить 43 рисунків та 3 таблиці.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ. ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

1.1 Аналіз існуючих рішень по менеджменту задач

Планування — це заздалегідь визначений і чітко структурований порядок дій, необхідний для досягнення конкретної мети. Воно передбачає ефективний розподіл ресурсів з метою досягнення поставлених завдань. Під ресурсами зазвичай мається на увазі час. Багато сфер діяльності базуються на принципах якісного тайм-менеджменту [1].

Тайм-менеджмент — це система методик, спрямованих на ефективне управління часом для виконання поточних завдань, проєктів та запланованих подій. Основними підходами у тайм-менеджменті є визначення пріоритетів, поділ великих завдань та проєктів на менші підзадачі, а також делегування обов’язків. До ключових інструментів управління часом належать особистий календар, перелік поточних завдань і список проєктів.

Одним із візуальних інструментів для планування часу є діаграма Ґанта, яка використовується для наочного зображення плану або графіка робіт у межах певного проєкту. Цей інструмент є засобом управління проєктами, де завдання представлені у вигляді горизонтальних відрізків (графічних блоків) на часовій шкалі, кожен з яких відповідає окремій задачі [4].

1.1.1 Додаток Any.DO

Any.DO — це один із найпопулярніших планувальників завдань серед користувачів Android та iOS, який відзначається простим, зручним інтерфейсом та можливістю синхронізації між різними пристроями. Додаток підходить як для особистого використання, так і для командної роботи, забезпечуючи безперервний доступ до завдань з будь-якої платформи.

Важливою функцією Any.DO є голосове введення завдань, що активує інтелектуальне розпізнавання тексту з автоматичним підбором відповідних варіантів, що значно спрощує процес планування [6].

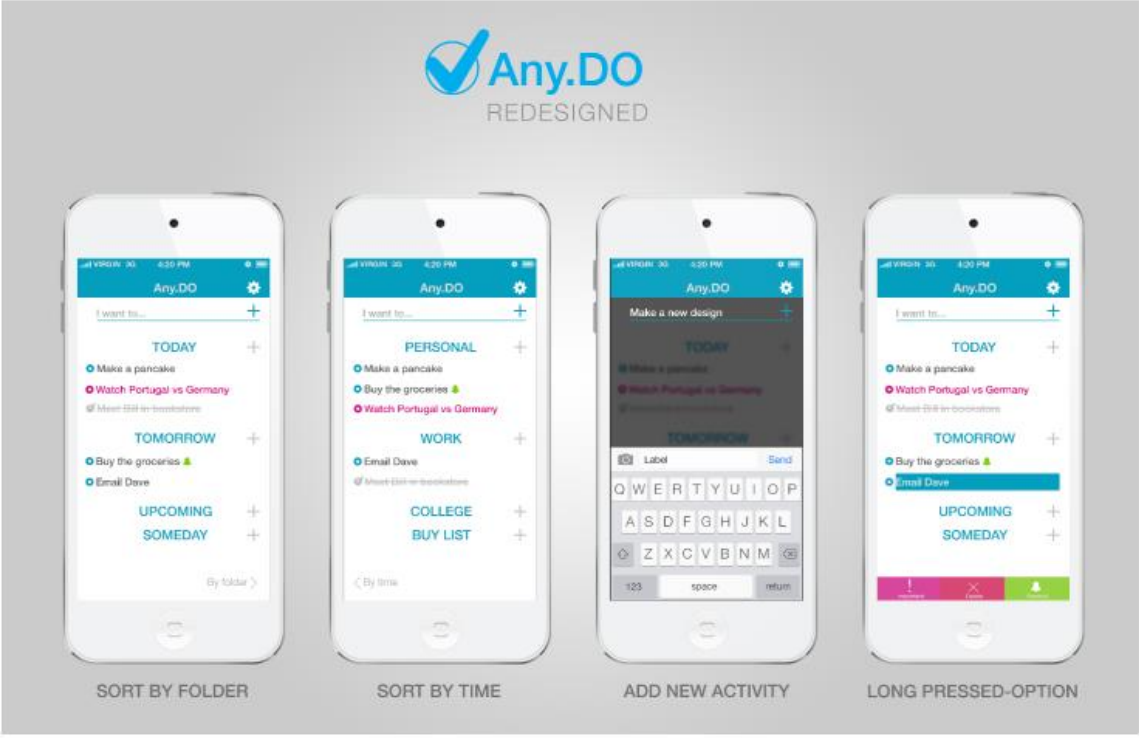


Рисунок 1.1 Any.DO

Ключові переваги Any.DO:

- можливість додавання текстових нотаток із прикріпленням медіафайлів (відео, фото, зображень), що відрізняє його від багатьох подібних додатків;
- створення персоналізованих списків завдань безпосередньо через головне меню;
- зручне перемикання між різними режимами відображення завдань;
- підтримка різноманітних форматів даних, зокрема списків справ, покупок і нотаток;

- інтуїтивне впорядкування завдань за категоріями для візуальної зручності.

Базова версія Any.DO надається безкоштовно, проте для розширеного функціоналу, включаючи розширені опції командної роботи, аналітики та інтеграції з іншими сервісами, доступна платна підписка [6].

1.1.2 Додаток Remember the milk

Програмне забезпечення для ефективного планування завдань «Remember the Milk» забезпечує збереження даних на сервері розробника, що дозволяє користувачам отримувати доступ до завдань із різних пристроїв та здійснювати синхронізацію даних у реальному часі. Основною функціональною можливістю додатка є управління завданнями, яке включає створення користувацьких списків, додавання тегів для оптимізації пошуку та спрощеного керування завданнями.

Інтерфейс програмного забезпечення представлений на рисунку 1.2.

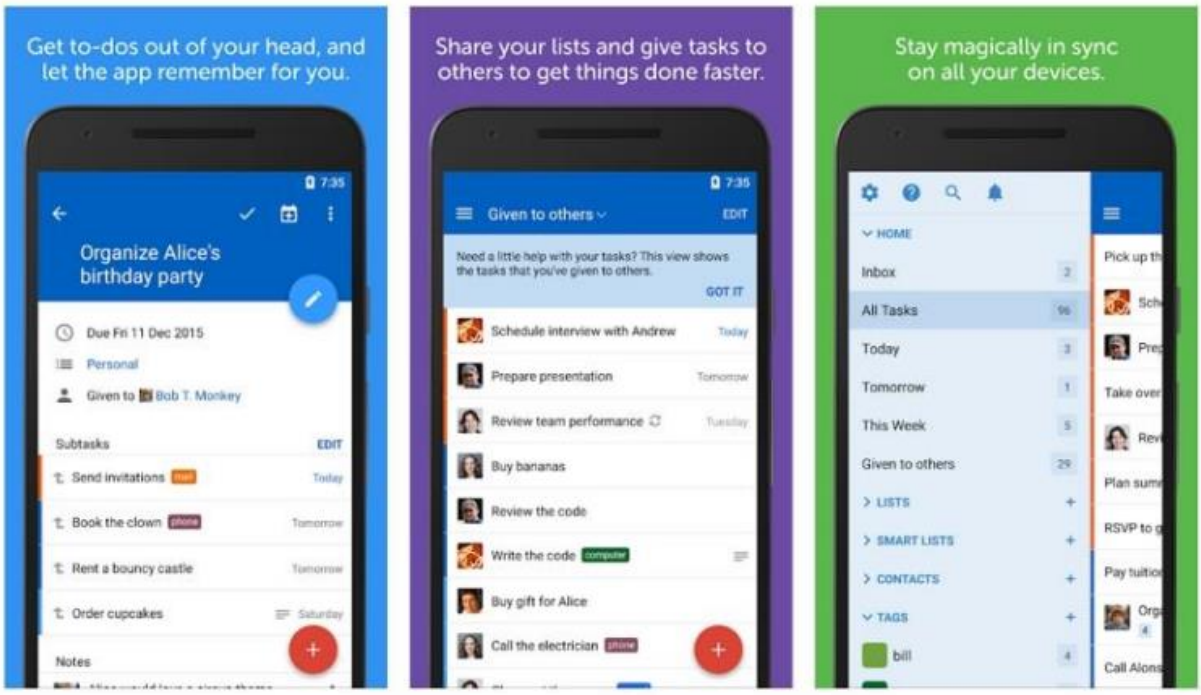


Рисунок 1.2 – Інтерфейс додатку «Remember the milk»

Серед основних переваг додатка «Remember the Milk» можна виокремити наступні:

1. Контроль робочих процесів

Програма забезпечує можливість сортування завдань за різними параметрами, такими як терміни виконання, рівень важливості та теги. Крім того, передбачена функціональність для командної роботи, що дозволяє декільком користувачам одночасно взаємодіяти із завданнями, спрощуючи управління спільними проєктами.

2. Організація завдань

Нові завдання за замовчуванням автоматично додаються до папки «Вхідні», що сприяє впорядкованості та швидкому доступу до новостворених елементів. Програма також підтримує створення користувацьких списків для більшої гнучкості у процесі планування.

3. Розширені можливості

Програмне забезпечення підтримує функцію прив’язки завдань до геолокації, що дає змогу отримувати нагадування про виконання справ у відповідному місці. Крім того, завдання можуть бути пов’язані з контактами користувача для ефективнішої організації заходів та взаємодії з іншими людьми.

4. Система нагадувань

Вбудована система нагадувань додатка «Remember the Milk» забезпечує взаємодію з популярними сервісами, зокрема Evernote, Microsoft Outlook та Twitter. Для сповіщення про заплановані завдання можуть використовуватися електронна пошта, SMS та повідомлення у месенджерах, що сприяє своєчасному інформуванню користувачів.

Застосунок має сумісність із широким переліком операційних систем, включаючи:

- Мобільні платформи: Android, iOS, BlackBerry OS.
- Настільні операційні системи: macOS, Windows.

Платна версія додатка надає розширені функціональні можливості, серед яких:

- Необмежена автосинхронізація даних між пристроями у режимі реального часу;
- PUSH-сповіщення для оперативного інформування про заплановані завдання та наближення термінів їх виконання.

Таким чином, програмне забезпечення «Remember the Milk» є комплексним інструментом для персонального та командного управління завданнями, що адаптоване для використання на більшості сучасних платформ і містить широкий набір функціональних можливостей для ефективної організації робочих процесів [6].

1.2.3 Trello

Додаток Trello є програмним забезпеченням для організації та управління завданнями, яке ґрунтується на принципах управління великими проєктами. Його функціональність дозволяє користувачам ефективно створювати, структурувати та відслідковувати завдання як для колективного, так і для особистого використання. Trello можна застосовувати для управління робочими процесами у командах, організації сімейних справ або навіть для управління завданнями в інтернет-магазинах.

Інтерфейс додатка Trello зображено на рисунку 1.3.

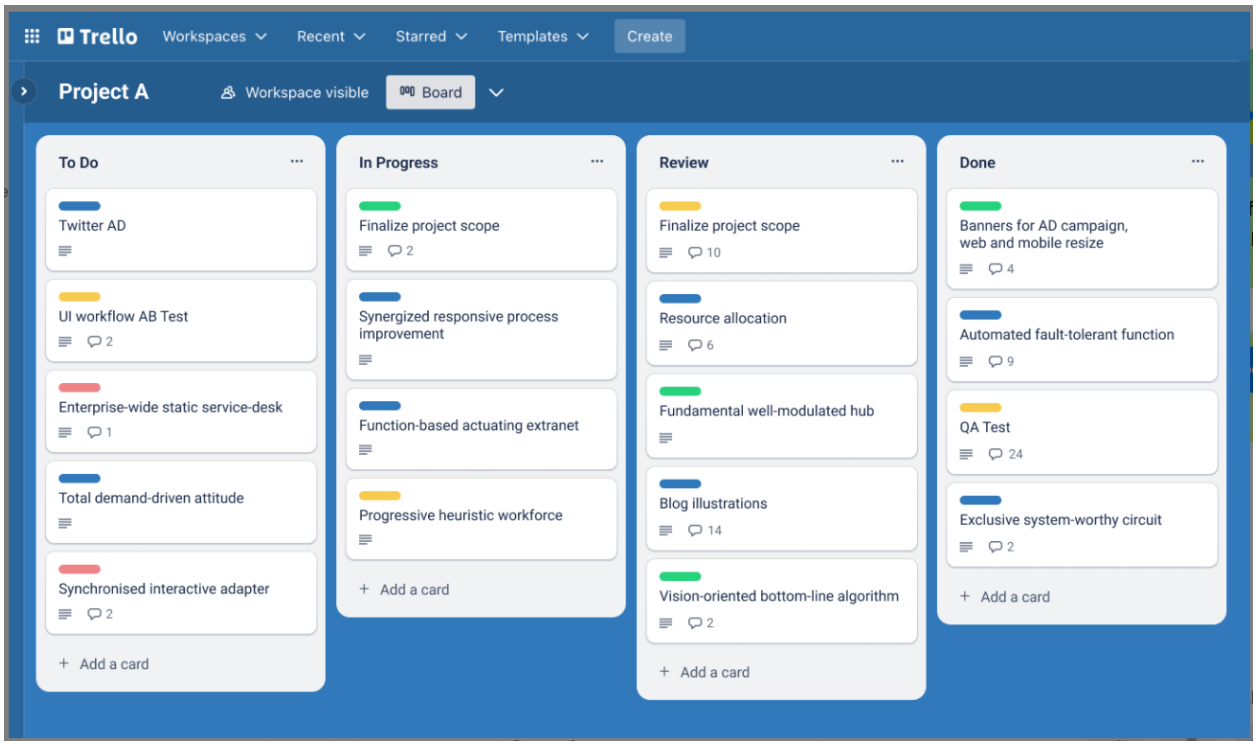


Рисунок 1.3 – Інтерфейс додатка Trello

Основні переваги планувальника Trello:

1. Створення списків завдань

Trello дозволяє створювати кілька списків завдань, які можна використовувати як для індивідуального, так і для командного застосування. Завдання в додатку організовані у вигляді карток, які можна переміщувати між списками, створюючи таким чином візуальне уявлення про хід виконання проєкту.

2. Можливості для спільної роботи

Додаток підтримує запрошення інших користувачів до спільних проєктів, зокрема колег, друзів та членів родини. Це дозволяє організовувати колективну роботу над завданнями та спільно відстежувати їхнє виконання.

3. Призначення відповідальних

Картки завдань у Trello можна призначати конкретним користувачам, що дає змогу чітко розподілити обов'язки між членами команди.

4. Функціональність коментування

Для полегшення комунікації всередині команди кожне завдання має розділ для коментарів, де користувачі можуть залишати відгуки, уточнення або обговорювати деталі виконання.

5. Прив’язка завдань до геолокації

Однією з функцій додатка є можливість прикріплення карток завдань до конкретних координат на карті. Це може бути корисно для організації заходів або проєктів, що залежать від фізичного місця проведення.

6. Візуалізація робочих процесів

Інтерфейс Trello базується на принципі візуального управління завданнями. Завдання подані у вигляді карток, розміщених у списках. Це спрощує контроль за прогресом виконання, оскільки картки можна переміщувати між списками відповідно до стадії виконання.

Trello є кросплатформним додатком, що забезпечує зручний доступ з різних пристроїв. Він підтримується такими операційними системами:

- Настільні платформи: Microsoft Windows, macOS.
- Мобільні платформи: iOS, Android.

Додаткові переваги додатка Trello:

- Контроль проєктів у реальному часі – додаток дозволяє стежити за оновленнями завдань одразу після їх внесення.
- Простий інтерфейс – візуальний стиль додатка інтуїтивно зрозумілий, що дозволяє швидко розпочати роботу без потреби у тривалому навчанні.
- Додавання вкладень: файли, зображення та документи для зручнішого управління завданнями.
- Можливість встановлення кінцевих термінів виконання завдань для своєчасного завершення проєктів.

Додаток Trello є безкоштовним у базовій версії, що робить його доступним для широкого кола користувачів. Основний функціонал, включаючи створення списків, додавання учасників і налаштування дедлайнів, доступний безоплатно. Додаткові функції, такі як розширені

автоматизації, аналітика та інтеграції з іншими сервісами, доступні у платних тарифних планах [6].

Таким чином, Trello є зручним та багатофункціональним інструментом для управління завданнями, який підходить як для особистого використання, так і для командної роботи завдяки простому інтерфейсу, візуалізації процесів та широкій підтримці платформ.

1.2.4 Додаток Google Calendar

Google Calendar — це програмне забезпечення для планування подій і управління завданнями, яке забезпечує збереження даних у хмарному середовищі з можливістю синхронізації між різними пристроями. Додаток розроблений для зручної організації як особистих, так і командних розкладів, а також дозволяє централізовано керувати подіями в інтеграції з іншими сервісами Google.

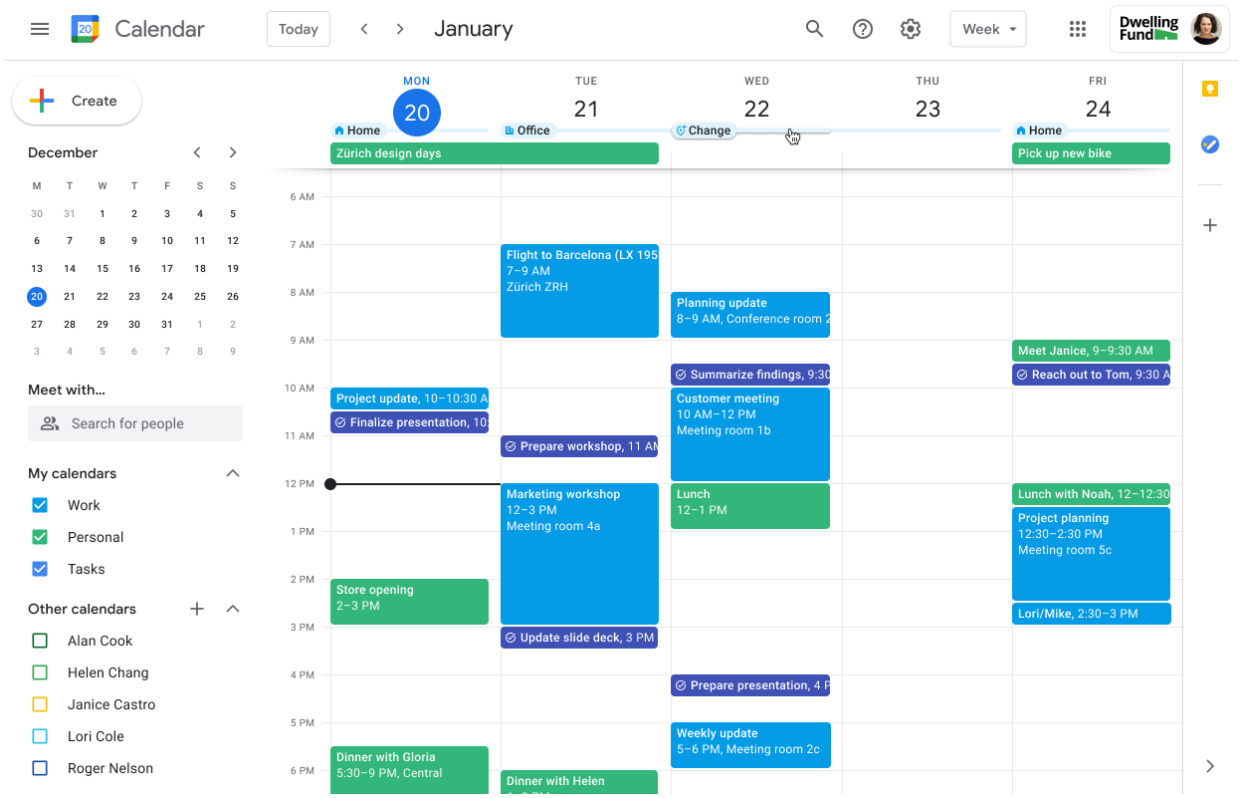


Рисунок 1.4 – Інтерфейс Google Calendar

Основні функціональні можливості додатка Google Calendar:

- Створення та управління подіями – можливість створення подій із зазначенням назви, опису, часу, місця проведення, а також параметрів повторюваності.
- Синхронізація з Gmail, Google Meet, Google Drive для автоматичного додавання подій та зручності у використанні файлів.
- Синхронізація та доступність.
- Функція створення спільних календарів із можливістю гнучкого налаштування прав доступу для інших користувачів.
- Вбудована система нагадувань через push-сповіщення, електронну пошту та SMS.
- Можливість візуальної організації завдань за допомогою кольорових міток.

Google Calendar підтримується на операційних системах Windows та macOS (через веб-версію), а також мобільних платформах Android та iOS.

Додаток доступний безкоштовно для персонального використання, проте для корпоративних клієнтів у складі Google Workspace передбачені додаткові функції, такі як аналітика, розширене адміністрування та інтеграція з іншими корпоративними сервісами.

Google Calendar є зручним інструментом для управління подіями, що забезпечує високу функціональність завдяки інтеграції з екосистемою Google, підтримці спільного використання календарів та доступності на різних платформах, що робить його ефективним рішенням для особистого та корпоративного застосування [6].

Кожен з вищерозглянутих мобільних додатків-аналогів має свої функціональні, візуальні та структурні особливості. Дані про них проаналізовано і складено порівняльну таблицю 1.1 особливостей кожного представлених застосунків

Таблиця 1.1

Порівняння застосунків

Характеристика	Google Calendar	Trello	Remember the Milk	Any.DO
Тип додатка	Планувальник подій та завдань	Дошка завдань для управління проєктами	Менеджер завдань зі збереженням у хмарі	Менеджер завдань із функцією нагадувань та планування
Призначення	Управління подіями, зустрічами та нагадуваннями	Управління завданнями для командних і особистих проєктів	Управління завданнями, створення списків	Управління завданнями, нагадування, планування дня
Основні функції	Створення подій, нагадування, повторювані завдання	Списки завдань, картки, коментування, дедлайни	Списки завдань, теги, нагадування, синхронізація	Списки завдань, нагадування, планування дня, голосове введення
Інтеграція з іншими сервісами	Gmail, Google Meet, Google Drive	Slack, Google Drive, Dropbox, Jira	Evernote, Microsoft Outlook, Twitter	Google Calendar, Alexa, Slack, WhatsApp, Zapier
Спільна робота	Спільні календарі з налаштуванням доступу	Запрошення користувачів до дошки, призначення завдань	Спільний доступ до списків, командна робота	Можливість обміну завданнями та списками із контактами
Візуалізація завдань	Календарна сітка з кольоровим кодуванням подій	Канбан-дошка, картки	Списки завдань, сортування	Списки завдань, хронологічне планування
Геоприв’язка	Так	Так	Так	Відсутня

Платформи	Web, Android, iOS, Windows, macOS	Web, Android, iOS, Windows, macOS	Web, Android, iOS, Windows, macOS, BlackBerry	Web, Android, iOS, Windows, macOS
Синхронізація	Автоматична синхронізація через Google Cloud	Автоматична синхронізація через хмару Trello	Автоматична синхронізація у платній версії	Автоматична синхронізація на всіх пристроях
Нагадування	Push-сповіщення, SMS, email	Push-сповіщення	Push-сповіщення, email, SMS	Push-сповіщення, email, голосові нагадування
Фінансова модель	Безкоштовний для персонального використання	Безкоштовний базовий, преміум-функції платні	Безкоштовний базовий, преміум-версія з розширеними функціями	Безкоштовний базовий, преміум-версія з розширеними можливостями
Вартість преміум-версії	Входить у Google Workspace	Trello Premium, Business Class	Платна версія з необмеженою синхронізацією	Платна версія із додатковими функціями (аналітика, автоматизація)
Підтримка командної роботи	Так, спільні календарі	Так, дошки для команд	Так, спільне використання списків завдань	Частково, через спільний доступ до завдань
Особливості інтерфейсу	Простий календарний інтерфейс	Візуальна Kanban-дошка	Класичний менеджер завдань із папками	Мінімалістичний, зручний для щоденного планування

Підсумовуючи, додатки Google Calendar, Trello, Remember the Milk та Any.DO пропонують різноманітні інструменти для управління завданнями та подіями, орієнтовані на різні потреби користувачів. Google Calendar найбільше підходить для організації подій і нагадувань із глибокою інтеграцією в екосистему Google, тоді як Trello є ефективним для управління командними проєктами завдяки візуальній Kanban-дошці. Remember the Milk фокусується на управлінні персональними завданнями з гнучкою системою нагадувань і геоприв'язкою, а Any.DO пропонує мінімалістичний підхід до щоденного планування з голосовим введенням і нагадуваннями. Усі ці додатки підтримують основні платформи та доступні як у безкоштовних, так і в розширених преміум-версіях.

1.2 Концепція та структура експертних систем

Експертні системи (ЕС) – це програмні продукти, створені для автоматизації процесу прийняття рішень на основі знань експертів у певних предметних областях. Вони виникли в середині 1970-х років як підгалузь прикладного штучного інтелекту (ШІ). Перші системи, такі як MYCIN (1974) для діагностики інфекційних захворювань та DENDRAL (1979) для хімічних досліджень, продемонстрували потенціал використання ЕС у складних аналітичних задачах. У 1980-х роках ЕС поширилися у виробничій сфері, фінансах і праві, а в 1990-х завдяки збільшенню обчислювальних потужностей – у медіа-індустрії та технологічному секторі [5].

Структура експертних систем:

1. База знань – центральний елемент системи, що містить факти, правила та евристики, необхідні для аналізу даних і прийняття рішень. Інформація може бути отримана як безпосередньо від експертів, так і через аналіз документальних джерел або методи машинного навчання.

2. Машина виведення – програмний модуль, який аналізує інформацію з бази знань та формує висновки. Складається з двох ключових підсистем: компонента висновування та інтерпретатора правил.

3. Інтерфейс користувача – забезпечує зручну взаємодію з системою, включаючи введення даних та отримання результатів у зручній формі.

4. Механізм пояснення – демонструє користувачу логіку прийнятих рішень, відображаючи використані правила та знання.

5. Додаткові компоненти: модуль навчання для оновлення бази знань та система моніторингу для контролю ефективності роботи.

Принцип роботи експертної системи полягає у введенні початкових даних користувачем, їх обробці у придатний для аналізу формат, пошуку відповідної інформації в базі знань, формуванні висновків і наданні користувачу результатів у вигляді рекомендацій або діагнозу.

Переваги експертних систем:

- Об'єктивність рішень, що базуються на формалізованих правилах.
- Економія ресурсів.
- Висока швидкість аналізу даних.
- Можливість застосування в реальному часі.

Обмеження експертних систем:

- Залежність від якості та повноти бази знань.
- Висока вартість створення та підтримки.
- Обмежена здатність до самонавчання [8].

Таким чином, експертні системи є потужними інструментами для автоматизації процесу прийняття рішень у різноманітних сферах, включаючи медицину, фінанси та техніку. Вони сприяють підвищенню точності аналізу, однак ефективність їх застосування залежить від актуальності бази знань та алгоритмів обробки інформації. Незважаючи на певні обмеження, експертні системи продовжують відігравати важливу роль у сучасних інформаційних технологіях.

1.3 Висновок до першого розділу

У даному розділі проведено дослідження методів та підходів до управління завданнями, включаючи принципи пріоритизації, використання візуальних інструментів планування та підходи до тайм-менеджменту. Проаналізовано ключові критерії оцінки систем управління завданнями, такі як функціональність, підтримка командної роботи, інтеграція з іншими сервісами, зручність інтерфейсу та системи нагадувань.

Особливу увагу приділено концепції експертних систем, які використовуються для автоматизації прийняття рішень на основі аналізу даних. Розглянуто основні компоненти експертних систем, зокрема базу знань, машину виведення та механізми пояснення.

Метою дослідження є створення експертної системи Task-менеджеру для ефективного управління завданнями на основі сучасних підходів до менеджменту часу.

Для досягнення поставленої мети було визначено та виконано такі завдання:

1. Проведено аналіз наукових джерел щодо принципів управління завданнями.
2. Досліджено можливості популярних систем управління завданнями, таких як Any.DO, Remember the Milk, Trello та Google Calendar.
3. Визначено ключові функціональні вимоги до систем управління завданнями.
4. Описано принципи роботи експертних систем у контексті Task-менеджерів.

Отримані результати стали основою для подальшого етапу роботи, присвяченого аналізу інструментів для розробки та проектування експертної системи Task-менеджеру.

РОЗДІЛ 2 АНАЛІЗ ЗАСОБІВ ДЛЯ РЕАЛІЗАЦІЇ TASK-МЕНЕДЖЕРУ

2.1 Програмні засоби для розробки крос-платформного додатку

Технології розробки крос-платформних додатків: аналіз та порівняння

Крос-платформна розробка мобільних та десктопних додатків дозволяє створювати програмні продукти, які можуть функціонувати на різних операційних системах, використовуючи єдину кодову базу. Це забезпечує скорочення часу розробки, спрощення підтримки коду та зниження витрат на розробку. У даному дослідженні розглянуто три популярні технології крос-платформної розробки: .NET MAUI, Flutter та React Native.

2.1.1 .NET MAUI (Multi-platform App UI)

.NET MAUI (Multi-platform App UI) є фреймворком від Microsoft, призначеним для створення крос-платформних додатків на основі платформи .NET. Основними перевагами MAUI є інтеграція з екосистемою .NET, підтримка мов програмування C# та XAML, а також можливість створення додатків для iOS, Android, Windows та macOS.

Особливості:

- Використання патерну MVVM для розділення логіки та інтерфейсу.
- Повна інтеграція з .NET 6+ та підтримка сучасних бібліотек, таких як Entity Framework Core.
- Можливість створення спільної кодової бази для кількох платформ із мінімальними змінами.
- Нативна продуктивність завдяки використанню відповідних рендерерів для кожної платформи.

Недоліки:

- Відносна новизна фреймворку, що може спричинити обмежену кількість ресурсів та прикладів.
- Обмежена підтримка платформ у порівнянні з іншими рішеннями [15].

2.1.2 Flutter

Flutter — це крос-платформний фреймворк від Google, який дозволяє створювати мобільні, веб- та десктопні додатки за допомогою мови Dart. Головною особливістю Flutter є використання власного графічного рушія Skia для рендерингу UI, що забезпечує однаковий вигляд додатків на всіх платформах.

Особливості:

- Використання єдиного коду для iOS, Android, вебу та десктопу.
- Підтримка гарячого перезавантаження (Hot Reload) для прискорення розробки.
- Вбудований набір віджетів для створення сучасних UI-компонентів.
- Висока продуктивність завдяки відмові від стандартних нативних компонентів.

Недоліки:

- Використання мови Dart, яка менш популярна порівняно з JavaScript та C#.
- Великі розміри кінцевих APK-файлів через використання власного рушія рендерингу.
- Відсутність повної нативності, що може впливати на доступ до деяких системних API [14].

2.1.3 React Native

React Native — це фреймворк, розроблений Facebook, який дозволяє створювати мобільні додатки за допомогою мови JavaScript та бібліотеки React. Його ключовою особливістю є використання декларативного підходу до створення UI та можливість повторного використання значної частини коду між платформами.

Особливості:

- Використання JavaScript, однієї з найбільш популярних мов програмування.
- Можливість використання нативних модулів через мости (Bridges).
- Велика спільнота та доступ до численних бібліотек та інструментів.
- Підтримка гарячого оновлення (Hot Reload).

Недоліки:

- Проблеми з продуктивністю при складних анімаціях через використання JavaScript Bridge.
- Потреба у створенні нативних модулів для складних функцій (наприклад, робота з Bluetooth) [7].

В таблиці 2.1 представлено порівняльний аналіз трьох популярних технологій для крос-платформної розробки: .NET MAUI, Flutter та React Native. Що дозволяє оцінити сильні та слабкі сторони кожного фреймворку для прийняття обґрунтованого рішення щодо вибору оптимальної технології для конкретного проекту.

Таблиця 2.1

Порівняння технологій

Критерій	.NET MAUI	Flutter	React Native
Мова програмування	C#, XAML	Dart	JavaScript
Підтримувані платформи	iOS, Android, Windows, macOS	iOS, Android, Web, Desktop	iOS, Android, Web (обмежено)
Продуктивність	Висока (нативні компоненти)	Висока (власний рушій)	Середня (JavaScript Bridge)
Простота у навчанні	Висока	Середня	Висока
Підтримка нативних функцій	Відмінна	Відмінна	Висока
Гарячий перезапуск	Hot Restart	Hot Reload	Hot Reload

Вибір крос-платформної технології залежить від конкретних вимог до проекту, досвіду команди розробників та потреб у продуктивності. .NET MAUI є оптимальним вибором для розробників у екосистемі Microsoft, що використовують C#. Flutter підходить для проектів, де важлива стабільність UI на всіх платформах, а React Native є чудовим варіантом для веб-орієнтованих команд завдяки використанню JavaScript.

2.2 Патерни проектування

Патерни проектування відіграють ключову роль у розробці веб-додатків, забезпечуючи структурованість, зрозумілість та масштабованість коду. Вони дозволяють розробникам застосовувати перевірені рішення для типових проблем, що виникають під час створення програмного забезпечення. Серед найпопулярніших патернів, які використовуються у веб-розробці, можна виділити Model-View-Controller (MVC), Model-View-ViewModel (MVVM), та VIPER. Кожен з цих патернів має свої переваги та недоліки, а також сферу застосування, що робить їх незамінними інструментами в арсеналі сучасного розробника. Розглянемо більш детально кожен з них [9].

MVC (Model-View-Controller) - це патерн проектування, що використовується для створення організованих і структурованих додатків, особливо в контексті веб-розробки. Він забезпечує розділення логіки додатка на три взаємодіючі компоненти: Model, View, і Controller. Основна мета MVC - розділити внутрішнє представлення інформації від способу її подання та взаємодії з користувачем.

1) Model - відповідає за управління даними додатка.
2) View - відповідає за відображення даних користувачу.
3) Controller - виступає як посередник між Model і View.

Переваги MVC

1) Розділення обов'язків

MVC забезпечує чітке розділення обов'язків між різними компонентами додатка, що полегшує його розробку, тестування та підтримку.

2) Модульність

Кожен компонент (Model, View, Controller) може бути розроблений, протестований і підтримуваний незалежно, що сприяє більшій масштабованості та зрозумілості коду.

3) Гнучкість

Завдяки розділенню логіки додатка, MVC дозволяє легко змінювати та оновлювати окремі частини додатка без впливу на інші компоненти.

4) Повторне використання коду

Компоненти Model і View можуть бути повторно використані у різних частинах додатка або навіть у різних проектах [11].

MVVM (Model-View-ViewModel) - це патерн проектування, що використовується для створення структурованих і масштабованих додатків, особливо в контексті розробки з використанням сучасних фреймворків та бібліотек, таких як WPF, Angular, та React. Основна мета MVVM - розділення представлення інтерфейсу користувача та бізнес-логіки, що полегшує розробку, тестування та підтримку додатка.

1) Model - відповідає за управління даними додатка. Це компоненти, які взаємодіють з базою даних, виконують бізнес-логіку і надають дані для ViewModel. Model містить чисту бізнес-логіку і не залежить від інших компонентів MVVM.

2) View - відповідає за відображення даних користувачу. Це інтерфейс користувача, який може бути реалізований за допомогою HTML, XAML, або іншого мови розмітки. View отримує дані з ViewModel і відображає їх користувачу. Крім того, View може містити анімації, стилі та інші елементи UI, але не містить бізнес-логіки.

3) ViewModel - виступає як посередник між View та Model. Він отримує дані з Model, обробляє їх і надає View у зручному для відображення форматі. ViewModel також обробляє події від View (наприклад, кліки на кнопки) і викликає відповідні методи Model. Основною перевагою ViewModel є те, що він дозволяє двосторонню прив'язку даних, що автоматично оновлює View при зміні даних у ViewModel і навпаки.

Переваги MVVM

- Розділення обов'язків:

MVVM забезпечує чітке розділення обов'язків між даними, логікою та інтерфейсом користувача, що полегшує розробку та підтримку додатка.

- Модульність

Кожен компонент (Model, View, ViewModel) може бути розроблений, протестований і підтримуваний незалежно, що сприяє більшій масштабованості та зрозумілості коду.

- Двостороння прив'язка даних

Однією з основних переваг MVVM є можливість двосторонньої прив'язки даних між View і ViewModel, що знижує необхідність вручну оновлювати інтерфейс при зміні даних.

- Повторне використання коду

ViewModel і Model можуть бути повторно використані у різних частинах додатка або навіть у різних проектах [12].

VIPER (View, Interactor, Presenter, Entity, Router) - це патерн проектування, що застосовується для розробки модульних і масштабованих додатків, особливо в контексті мобільної розробки, але також може бути адаптований для веб-додатків. Основна мета VIPER - розділення відповідальностей між різними компонентами для досягнення високої ступені модульності та тестованості коду.

1) View - відповідає за відображення даних і взаємодію з користувачем. Він отримує команди від Presenter і відображає відповідний контент.

2) Interactor - містить бізнес-логіку додатка. Він отримує запити від Presenter, обробляє їх, і повертає результат назад до Presenter.

3) Presenter - виступає як посередник між View та Interactor. Він отримує дані від Interactor і форматує їх для відображення у View. Також обробляє події з View і передає їх до Interactor.

4) Entity - містить модель даних. Це можуть бути об'єкти, що представляють дані додатка, такі як користувачі, продукти тощо.

5) Router - відповідає за навігацію між екранами додатка. Містить логіку переходів та навігаційних маршрутів [13].

Результатом є таблиця 2.2, що надає порівняння основних характеристик патернів MVC, MVVM та VIPER, показуючи їх переваги та недоліки, а також підказуючи, який патерн може бути найкращим вибором для конкретного проекту залежно від його вимог.

Таблиця 2.2

Порівняння патернів проектування

Параметр	MVC (Model-View-Controller)	MVVM (Model-View-ViewModel)	VIPER (View-Interactor-Presenter-Entity-Router)
Основна мета	Розділення представлення даних та бізнес-логіки	Розділення представлення інтерфейсу користувача та бізнес-логіки з підтримкою двосторонньої прив'язки даних	Розділення відповідальностей для досягнення модульності та тестованості
Компоненти	Model, View, Controller	Model, View, ViewModel	View, Interactor, Presenter, Entity, Router
Відповідальність Model	Управління даними та бізнес-логікою	Управління даними та бізнес-логікою	Управління даними та бізнес-логікою
Відповідальність View	Відображення даних користувачу	Відображення даних користувачу	Відображення даних користувачу
Відповідальність Controller/Presenter/ViewModel	Controller обробляє запити від View і взаємодіє з Model	ViewModel обробляє події з View і взаємодіє з Model	Presenter обробляє запити від View, взаємодіє з Interactor і оновлює View
Двостороння прив'язка даних	Ні	Так	Ні
Модульність	Середня	Висока	Дуже висока
Тестованість	Середня	Висока	Дуже висока
Масштабованість	Середня	Висока	Дуже висока

Придатність для великих проектів	Середня	Висока	Дуже висока
Навігація між екранами	Обробляєтьс я в Controller	Обробляється окремо від ViewModel	Виконується Router
Складність реалізації	Помірна	Середня	Висока

2.3 Вимоги до проекту

Вимоги до проекту визначають ключові характеристики, необхідні для створення ефективної експертної системи управління завданнями, яка забезпечить зручність використання, стабільність роботи та високу продуктивність. Визначення цих вимог гарантує відповідність функціональності потребам користувача та сучасним стандартам програмної розробки.

Функціональні вимоги:

1. Створення, редагування, видалення та перегляд завдань:
   - Можливість додавати нові завдання з визначенням назви, опису, пріоритету та дедлайну.
   - Підтримка редагування властивостей завдання.
   - Відображення статусу виконання завдання.
2. Автоматизація процесу управління завданнями:
   - Сортування завдань за категоріями: одне головне, три середньої важливості, п’ять меншої важливості.
   - Автоматичне відображення завдань відповідно до встановлених пріоритетів.
3. Кросплатформенність

Нефункціональні вимоги:

1. Висока швидкодія додатка для великих обсягів даних.
2. Коректна робота застосунку незалежно від обсягу завдань.
3. Інтуїтивно зрозумілий інтерфейс
4. Архітектура:

- Використання патерну MVVM (Model-View-ViewModel).
- Чіткий розподіл логіки, даних та інтерфейсу для полегшення тестування та масштабованості.

5. Сумісність:

- Підтримка стандартних елементів MAUI (ListView, CollectionView).
- Можливість використання стандартних API для збереження даних.

Ці вимоги забезпечують створення зручного, надійного та ефективного інструменту для управління завданнями, що підходить як для особистого, так і командного використання.

2.4 Проектування експертної системи

Проєктування експертної системи для управління завданнями базується на принципі розподілу задач за пріоритетами, яка спрямована на оптимізацію процесу планування та підвищення ефективності користувача. Цей підхід передбачає щоденний вибір одного великого, трьох середніх, п’яти невеликих завдань, та одного з терміном, які користувач має намір виконати протягом дня. Такий метод дозволяє структурувати робочий процес, допомагаючи сфокусуватися на найбільш важливих завданнях, водночас залишаючи місце для менш значущих, але корисних справ. Експертна система автоматизує цей принцип, забезпечуючи користувача інструментами для сортування, пріоритизації та моніторингу завдань.

У межах реалізації даного алгоритму експертна система здійснює автоматичний розподіл завдань на категорії за рівнем важливості. Алгоритм повинен дозволяти користувачеві позначати завдання як великі, середні чи малі, використовуючи числовий пріоритет.

Важливим елементом експертної системи є функціональність аналізу продуктивності, що дозволяє відстежувати виконання задач відповідно до

алгоритм експертної системи. Вона зберігає дані про кількість завершених завдань у кожній категорії та може формувати звіти, відображаючи, наскільки користувач дотримувався плану. Це сприяє самоконтролю та підвищенню дисципліни у плануванні.

Алгоритм дій для виведення завдань на екран за алгоритмом:

1. Збір завдань:

Система отримує всі завдання користувача, включаючи їхню назву, опис, пріоритет та статус виконання.

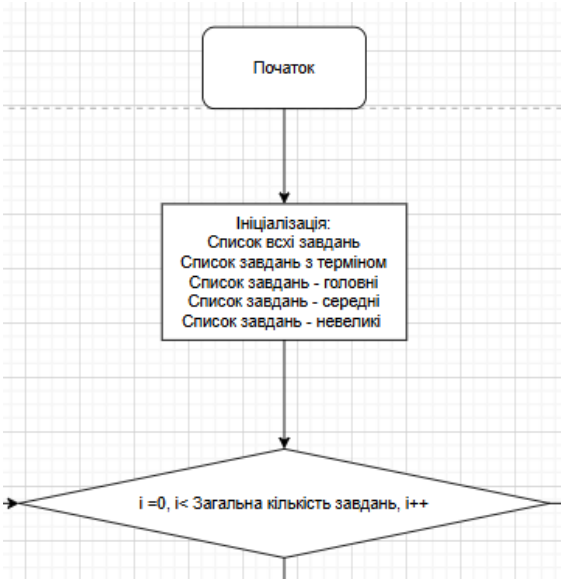


Рисунок 2.1 – Діаграма алгоритму розподілення завдання (ч.1)

2. Розподіл завдань за категоріями:

Завдання автоматично розподіляються за 4 категоріями (рис. 2.2). Задача, що має встановлений термін, наступна задача головна, три завдання — середньої важливості, і п’ять завдань — меншої важливості. Якщо завдань більше, система може запропонувати відкласти їх на інший день.

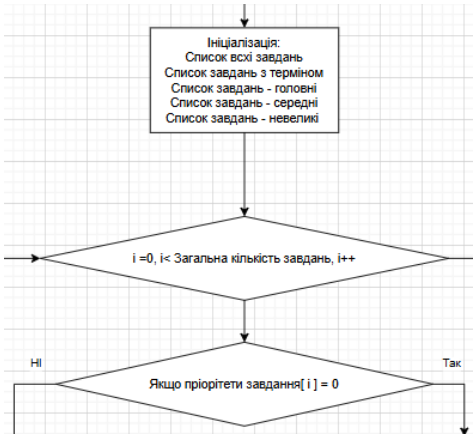


Рисунок 2.2 - Діаграма алгоритму розподілення завдання (ч.2)

3. Сортування завдань:

Система сортує завдання всередині кожної категорії за пріоритетністю або терміновістю (рис. 2.3). Завдання з найвищим рівнем важливості відображаються першими у відповідній групі, щоб користувач міг зосередитись на ключових цілях.

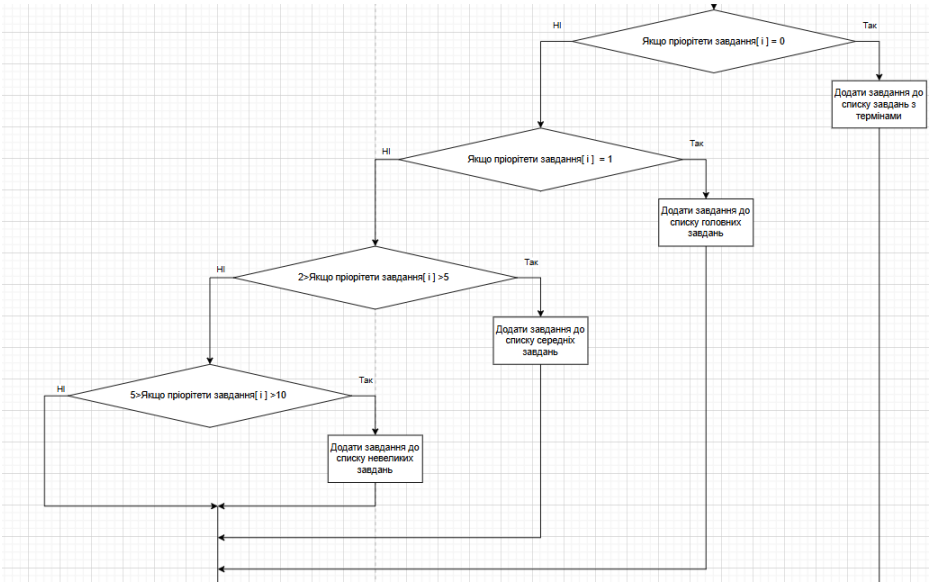


Рисунок 2.3 - Діаграма алгоритму розподілення завдання (ч.3)

4. Відображення категорій на екрані:

Завдання відображаються на екрані у 4 окремих секціях, кожна з яких відповідає за власну категорію.

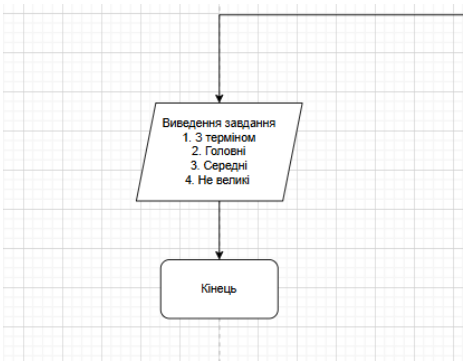


Рисунок 2.4 - Діаграма алгоритму розподілення завдання (ч.4)

2.5 Висновок до другого розділу

У даному розділі було проведено аналіз засобів для реалізації Task-менеджеру, зокрема програмних рішень для створення кросплатформних застосунків, таких як .NET MAUI, Flutter та React Native. Було здійснено порівняльний аналіз цих інструментів за критеріями функціональності, підтримки патернів проектування, продуктивності та гнучкості налаштувань.

Фреймворк .NET MAUI обрано основним середовищем для реалізації експертної системи Task-менеджеру. Такий вибір обумовлений його інтеграцією з екосистемою .NET, підтримкою мови програмування C#, можливістю створення спільної кодової бази для кількох платформ, а також зручністю застосування патерну MVVM для розділення логіки та інтерфейсу.

Для досягнення поставлених цілей було виконано такі завдання:

1. Проаналізовано доступні програмні засоби для розробки Task-менеджеру.
2. Оцінено функціональні можливості .NET MAUI, Flutter та React Native.
3. Обґрунтовано вибір .NET MAUI як оптимального середовища для розробки експертної системи.

Завдання автоматично розподіляються за 4 категоріями (рис. 2.2). Задача, що має встановлений термін, наступна задача головна, три завдання — середньої важливості, і п’ять завдань — меншої важливості. Якщо завдань більше, система може запропонувати відкласти їх на інший день.

Система сортує завдання всередині кожної категорії за пріоритетністю або терміновістю. Завдання з найвищим рівнем важливості відображаються першими у відповідній групі, щоб користувач міг зосередитись на ключових цілях.

Результати даного дослідження стали основою для подальшого етапу роботи, присвяченого проектуванню та програмній реалізації експертної системи Task-менеджеру.

РОЗДІЛ 3. РОЗРОБКА ЕКСПЕРТНОЇ СИСТЕМИ TASK-МЕНЕДЕРУ НА MAUI

3.1 Актуальність розробки

У сучасному цифровому середовищі ефективне управління завданнями та робочими процесами набуває особливої актуальності. Це зумовлено зростаючою складністю робочих процесів, популяризацією віддаленої роботи та необхідністю обробки великих обсягів інформації. Для користувачів важливо мати інструмент, який не тільки дозволяє фіксувати завдання, а й автоматизує процеси управління ними, пропонуючи інтелектуальні підказки, аналізуючи продуктивність і допомагаючи оптимізувати час. Саме тому розробка експертної системи Task-менеджеру на платформі .NET MAUI відповідає сучасним вимогам до програмного забезпечення.

Однією з ключових тенденцій у створенні таких додатків є забезпечення кросплатформенності та уніфікованості. Платформа .NET MAUI (Multi-platform App UI) дозволяє створювати застосунки з єдиною кодовою базою, які працюють на операційних системах Windows, macOS, iOS та Android. Це значно спрощує процес розробки, тестування та подальшої підтримки застосунку. Сучасний Task-менеджер має бути доступним на будь-якому пристрої, забезпечуючи користувачам однаковий досвід використання незалежно від платформи. Важливим аспектом є також адаптивний дизайн, який автоматично підлаштовується під розмір екрану та орієнтацію пристрою.

Важливу роль у сучасному програмному забезпеченні відіграє якісний та інтуїтивно зрозумілий користувацький інтерфейс. Додаток має бути мінімалістичним, зосередженим на основних функціях — створенні, редагуванні та моніторингу завдань. Сучасні тенденції передбачають використання гнучкого дизайну з можливістю перемикання між темною та світлою темами, підтримку жестів для мобільних пристроїв та функціональність drag-and-drop для зручного управління завданнями. Окрему

увагу слід приділити доступності: застосунок має бути зручним для користувачів із порушеннями зору, підтримувати масштабовані шрифти та озвучення елементів інтерфейсу.

Персоналізація та використання технологій штучного інтелекту є ще однією ключовою вимогою до сучасних Task-менеджерів. Система має адаптуватися до потреб конкретного користувача, пропонуючи розумні рекомендації щодо пріоритетності завдань, аналізуючи робоче навантаження та навіть автоматично прогнозуючи дедлайни. Впровадження аналітичних функцій, таких як статистика виконаних завдань, середній час на їх завершення та порівняння з минулими періодами, дозволяє користувачам краще оцінювати свою продуктивність.

Сучасні експертні системи орієнтовані також на автоматизацію рутинних процесів. Task-менеджер має підтримувати створення повторюваних завдань, автоматичне надсилання сповіщень про наближення термінів виконання, а також інтелектуальне сортування завдань за пріоритетністю та дедлайнами. Це дозволяє зменшити кількість ручних операцій та зосередитися на більш важливих аспектах роботи.

Важливою складовою сучасних Task-менеджерів є інтеграція із зовнішніми сервісами. Це включає можливість синхронізації з календарями (Google Calendar, Outlook), інтеграцію з хмарними сховищами (OneDrive, Google Drive) та популярними платформами для командної роботи (Trello, Asana, Slack). Така функціональність дозволяє створити єдине інформаційне середовище для управління завданнями, що особливо важливо для корпоративних користувачів.

Таким чином, розробка експертної системи Task-менеджеру на MAUI має відповідати сучасним стандартам, які включають кросплатформенність, адаптивний дизайн, персоналізацію, автоматизацію та інтеграцію із зовнішніми сервісами. Це дозволить створити потужний інструмент для управління завданнями, який підвищить ефективність як окремих користувачів, так і команд, сприяючи досягненню стратегічних цілей.

3.2 Структура проекту

Структура проєкту експертної системи Task-менеджеру на платформі .NET MAUI базується на концепції чіткого розділення логіки, даних та інтерфейсу користувача, що сприяє легкості підтримки, тестування та розширення функціональності. Основу цього підходу становить архітектурний патерн MVVM (Model-View-ViewModel), який забезпечує розділення бізнес-логіки та відображення даних у застосунку, що є особливо важливим для складних експертних систем.

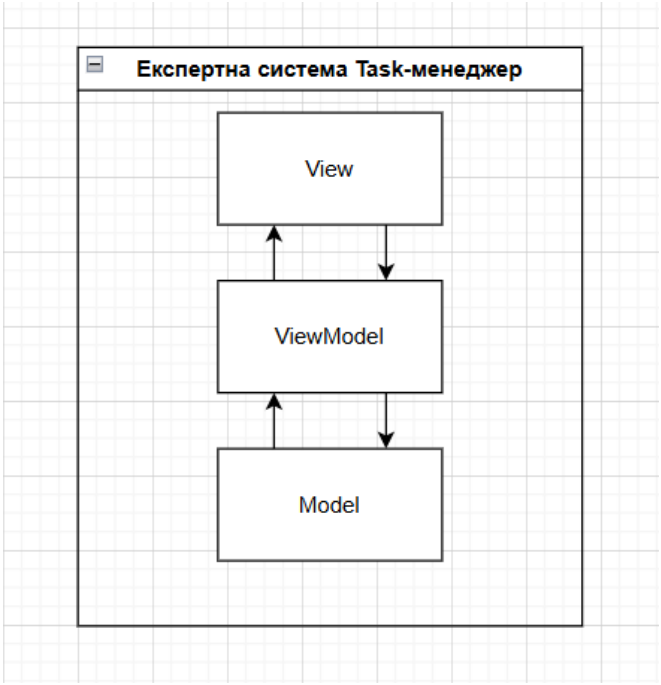


Рисунок 3.1 – Структура проекту

Архітектурний патерн MVVM (Model-View-ViewModel) є центральним елементом проєктування цієї експертної системи Task-менеджеру (рис. 3.1). Основна мета цього патерну — забезпечити слабке зв’язування між інтерфейсом користувача (View) та бізнес-логікою (ViewModel) шляхом використання моделей даних (Model).

Компонент Model у цьому патерні відповідає за представлення даних та їх обробку. У випадку Task-менеджеру, це об'єкт, який містить властивості,

такі як Title, Description, DueDate та Priority. Model також може взаємодіяти з базами даних або API для отримання та збереження даних. У в даному проєкті він буде використовуватися для збереження списку завдань та інформації про їх статус.

ViewModel виконує роль посередника між моделлю та уявленням. Він відповідає за обробку логіки, такої як сортування завдань, застосування фільтрів або автоматичне оновлення інтерфейсу при зміні даних.

View відповідає за відображення даних, що передає ViewModel. У MAUI для цього використовується XAML (eXtensible Application Markup Language), який дозволяє декларативно описувати інтерфейс.

Такий підхід має низку переваг для експертної системи Task-менеджеру. По-перше, він сприяє легшому тестуванню бізнес-логіки, оскільки вона відокремлена від інтерфейсу. Це особливо важливо для експертних систем, де складна логіка прийняття рішень може потребувати ретельного тестування. По-друге, MVVM спрощує підтримку та масштабування застосунку, дозволяючи легко додавати нові функції, такі як аналіз продуктивності або рекомендації завдань. По-третє, цей патерн ідеально підходить для MAUI, оскільки платформа спочатку розроблена з урахуванням принципів MVVM, включаючи вбудовану підтримку Data Binding та Commands.

У підсумку, структура проєкту MAUI для експертної системи Task-менеджеру, побудована на основі патерну MVVM, дозволяє створити масштабовану, гнучку та легку для супроводу архітектуру. Це забезпечує відокремлення логіки обробки завдань від інтерфейсу, спрощує тестування та сприяє створенню зручного, адаптивного застосунку, що відповідає сучасним вимогам до кросплатформених програм [2].

3.3 Розробка додатку

функціональних можливостей кросплатформенного додатка експертної системи Task-менеджеру на платформі .NET MAUI:

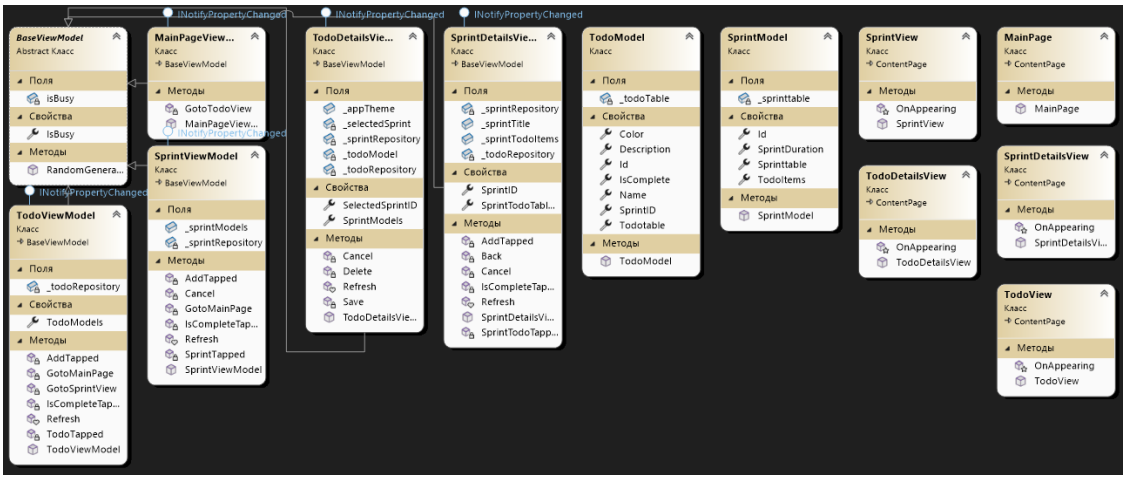


Рисунок 3.2 – Діаграма класів проекту

Класи можна розділити на кілька основних груп відповідно до їхньої ролі у застосунку:

1. ViewModel (Моделі представлення)

Ця група включає класи, відповідальні за бізнес-логіку та зв'язок між моделями даних і представленням (UI). До цієї групи належать:

- BaseViewModel – абстрактний клас, що реалізує базові методи та інтерфейси, такі як INotifyPropertyChanged.
- MainPageViewModel, SprintViewModel, TodoViewModel, SprintDetailsViewModel, TodoDetailsViewModel – реалізують логіку управління даними для відповідних сторінок (екранів). Містять методи для додавання, видалення, оновлення завдань, а також обробки взаємодії з інтерфейсом (наприклад, AddTapped, Refresh).

2. Model (Моделі даних)

Ці класи представляють структуру даних та використовуються для зберігання інформації про завдання та спринти:

- TodoModel – модель для представлення завдання з властивостями (Id, Name, Description, IsComplete, Priority).
- SprintModel – модель для представлення спринту з властивостями (Id, SprintDuration, TodoItems).

3. View (Подання або Інтерфейс користувача)

Ці класи відповідають за відображення даних у графічному інтерфейсі користувача. Вони реалізовані через XAML-сторінки:

- MainPage, SprintView, TodoView, SprintDetailsView, TodoDetailsView – окремі сторінки для різних частин застосунку.
- Кожна з цих сторінок має метод OnAppearing, який використовується для ініціалізації даних під час завантаження сторінки [3].

Спочатку потрібно розглянути класи, що відповідають за бізнес логіку проекту:

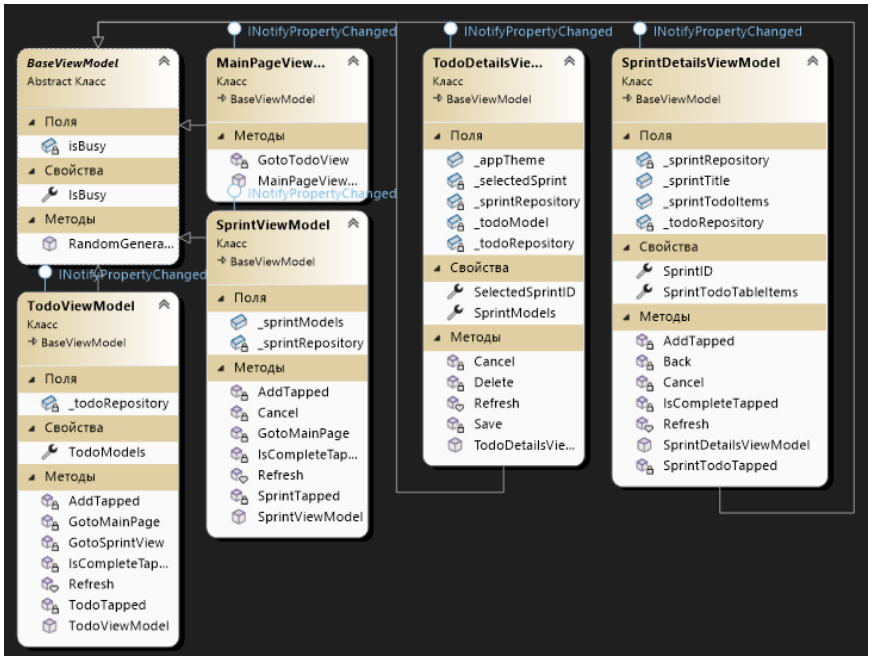


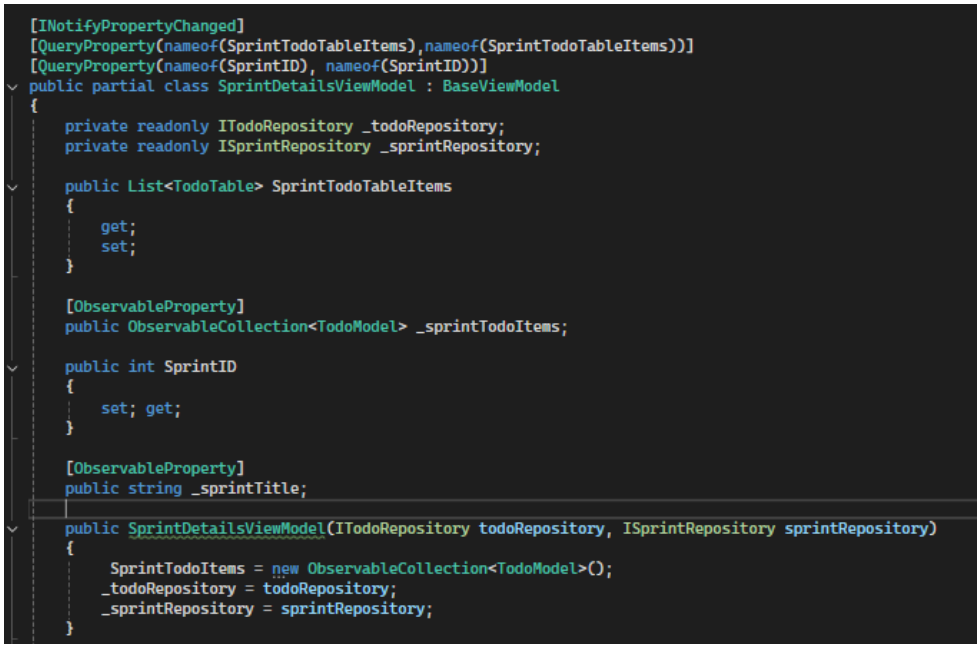
Рисунок 3.3 – Група класів ViewModel

Перший клас SprintDetailsViewModel відповідає за управління логікою для відображення та обробки даних, пов’язаних завданнями (рис. 3.4). Цей клас реалізує бізнес-логіку для сторінки, яка відображає список завдань у межах вибраного пріоритету, а також містить методи для взаємодії з користувачем, такі як додавання, завершення та перегляд завдань.

Ключовим аспектом SprintDetailsViewModel є використання механізму INotifyPropertyChanged, який дозволяє автоматично оновлювати інтерфейс користувача при зміні властивостей ViewModel. Це досягається через атрибут

[INotifyPropertyChanged], що автоматично генерує необхідний код для сповіщення про зміну даних у прив'язаних елементах інтерфейсу. Також застосовуються атрибути [QueryProperty], які використовуються для передачі даних між сторінками через механізм Shell у MAUI.

У класі реалізовані два основних джерела даних: SprintTodoTableItems — список завдань у вигляді стандартного List<TodoTable> та SprintTodoItems, представлений як ObservableCollection<TodoModel>. ObservableCollection використовується для зручної інтеграції з механізмом прив'язки даних (Data Binding), що дозволяє автоматично оновлювати інтерфейс при зміні колекції.



```
[INotifyPropertyChanged]
[QueryProperty(nameof(SprintTodoTableItems),nameof(SprintTodoTableItems))]
[QueryProperty(nameof(SprintID), nameof(SprintID))]
public partial class SprintDetailsViewModel : BaseViewModel
{
    private readonly ITodoRepository _todoRepository;
    private readonly ISprintRepository _sprintRepository;

    public List<TodoTable> SprintTodoTableItems
    {
        get;
        set;
    }

    [ObservableProperty]
    public ObservableCollection<TodoModel> _sprintTodoItems;

    public int SprintID
    {
        set; get;
    }

    [ObservableProperty]
    public string _sprintTitle;

    public SprintDetailsViewModel(ITodoRepository todoRepository, ISprintRepository sprintRepository)
    {
        SprintTodoItems = new ObservableCollection<TodoModel>();
        _todoRepository = todoRepository;
        _sprintRepository = sprintRepository;
    }
```

Рисунок 3.4 – Клас SprintDetailsViewModel

Конструктор класу приймає два сервіси ITodoRepository та ISprintRepository через механізм Dependency Injection (рис. 3.4). Це дозволяє ізольовано працювати з даними, які надходять із зовнішніх джерел, таких як бази даних або API, спрощуючи тестування та підтримку додатка. У конструкторі ініціалізується ObservableCollection<TodoModel> для зберігання завдань, а також передані сервіси зберігаються у приватних полях класу _todoRepository та _sprintRepository.

Клас також містить набір команд (RelayCommand), які використовуються для обробки дій користувача. Наприклад, метод SprintTodoTapped відкриває сторінку деталей конкретного завдання (TodoDetailsView), передаючи туди об’єкт TodoModel через Shell.Current.GoToAsync. Це дозволяє зручно організувати навігацію у MAUI. Команда AddTapped викликає відкриття сторінки для створення нового завдання, передаючи SprintID, що забезпечує створення нового завдання у контексті поточного спринту.

```
[RelayCommand]
async void SprintTodoTapped(TodoModel sprintTodoItem)
{
    if (IsBusy) return;
    IsBusy = true;
    await Shell.Current.GoToAsync(nameof(TodoDetailsView), true,
        new Dictionary<string, object>()
        {
            ["TodoModel"] = sprintTodoItem,
        });
    IsBusy = false;
}

[RelayCommand]
async void Cancel()
{
    await Shell.Current.GoToAsync("..");
}
```

Рисунок 3.5 – Метод SprintTodoTapped

Метод IsCompleteTapped викликається при позначенні завдання як завершеного. Це асинхронний метод, який використовує Task.Run для збереження змін через TodoRepository. Такий підхід дозволяє уникнути блокування основного потоку, зберігаючи швидкість роботи застосунку.

Функція Refresh (рис. 3.6) оновлює дані на сторінці, перезавантажуючи список завдань та сортує їх за статусом виконання. Вона спочатку перевіряє стан IsBusy, щоб уникнути повторного виклику під час активного оновлення. Завдання сортуються за принципом: спочатку незавершені, потім завершені. Цей метод також оновлює заголовок сторінки (SprintTitle) та додає нові завдання до ObservableCollection для оновлення UI.

```
internal void Refresh()
{
    if (IsBusy)
    {
        return;
    }
    IsBusy = true;

    Task.Run(() =>
    {
        SprintTitle = "Sprint " + SprintID.ToString();
        SprintTodoItems.Clear();
        SprintTodoTableItems.Sort((t1, t2) =>
        {
            if (t1.IsComplete == t2.IsComplete)
            {
                return t1.Id < t2.Id ? -1 : 1;
            }
            else if (t1.IsComplete)
            {
                return 1;
            }
            else
            {
                return -1;
            }
        });
        if (SprintTodoTableItems != null && SprintTodoTableItems.Count > 0)
        {
            SprintTodoTableItems.ForEach((st) => { SprintTodoItems.Add(new TodoModel(st)); });
        }
        IsBusy = false;
    });
}
```

Рисунок 3.6 – Метод Refresh

SprintDetailsViewModel є комплексною ViewModel, яка відповідає за управління завданнями, включаючи додавання, видалення, відмічення як завершених та перегляд деталей.

TodoDetailsViewModel є ViewModel-класом, який відповідає за управління логікою для сторінки перегляду та редагування конкретного завдання у застосунку Task-менеджеру, створеного на платформі MAUI. Цей клас реалізує обробку даних, взаємодію з репозиторіями та керування подіями через механізм команд (RelayCommand), що є важливим аспектом реалізації патерну MVVM.

Основною функцією класу є забезпечення роботи з одним конкретним завданням (TodoModel). Для спрощення прив’язки даних між сторінками використовується атрибут [QueryProperty]. Завдяки цьому механізму, об'єкти TodoModel та SelectedSprintID можуть передаватися між сторінками через параметри навігації Shell. Це забезпечує плавний перехід даних між сторінками без необхідності прямої передачі через конструктор.

Конструктор класу TodoDetailsViewModel приймає три залежності через механізм ін'єкції залежностей (Dependency Injection): AppThemeService, ITodoRepository та ISprintRepository. Сервіс AppThemeService ймовірно

використовується для управління темами застосунку, хоча в поточному фрагменті коду він безпосередньо не використовується. Репозиторії ITodoRepository та ISprintRepository відповідають за взаємодію з базою даних для збереження та отримання завдань і спринтів. У конструкторі ініціалізується порожній об'єкт TodoModel для створення нового завдання за замовчуванням.

```
public partial class TodoDetailsViewModel : BaseViewModel
{
    public AppThemeService _appTheme;

    private readonly ITodoRepository _todoRepository;
    private readonly ISprintRepository _sprintRepository;
    [ObservableProperty]
    private TodoModel _todoModel;
    [ObservableProperty]
    private SprintModel _selectedSprint;
    public int SelectedSprintID
    {
        get;set;
    }
    public ObservableCollection<SprintModel> SprintModels
    {
        private set; get;
    } = new ObservableCollection<SprintModel>();
    public TodoDetailsViewModel(AppThemeService appTheme, ITodoRepository _todoReposit
    {
        TodoModel = new TodoModel();
        this._todoRepository = _todoRepository;
        _sprintRepository = sprintRepository;
    }
```

Рисунок 3.7 – Клас TodoDetailsViewModel

Для управління даними та реакції на події в цьому класі використовується кілька RelayCommand. Команда Save (рис. 3.8) перевіряє, чи заповнені обов'язкові поля (Name та Description) у моделі завдання. Якщо дані відсутні, викликається метод Cancel, що повертає користувача на попередню сторінку. Якщо ж дані валідні, завдання зберігається у базі даних за допомогою асинхронного методу Task.Run. Примітно, що для асинхронної обробки використовується ContinueInMainThreadWith, який гарантує повернення до головного потоку після виконання фонових операцій.

Команда Delete (рис. 3.8) дозволяє видалити поточне завдання. Вона також виконується асинхронно, використовуючи Task.Run, і після видалення користувач автоматично повертається до попередньої сторінки

(Shell.Current.GoToAsync("..")). Це забезпечує зручність та автоматизацію взаємодії з користувачем.

```
void Save()
{
    var temp = SelectedSprint;
    if (TodoModel.Name == null && TodoModel.Description == null)
    {
        Cancel();
    }
    else
    {
        var t = Task.Run(async () =>
        {
            //var service = await TodoService.Instance;
            TodoModel.Color = RandomGenerator();
            TodoModel.SprintID = SelectedSprint.Id;
            await _todoRepository.SaveItem(TodoModel.Todotable);
        }).
            ContinueInMainThreadWith(async () =>
            {
                await Shell.Current.GoToAsync("..");
            });
    }
}
[RelayCommand]
void Delete()
{
    Task.Run(async () =>
    {
        //var service = await TodoService.Instance;
        await _todoRepository.DeleteItem(TodoModel.Id);

    }).ContinueInMainThreadWith(async () =>
    {
        await Shell.Current.GoToAsync("..");
    });
```

Рисунок 3.8 – Методи Delete та Save

Метод Refresh відповідає за оновлення списку пріорітетів та їх завантаження з бази даних. Якщо у сховищі приорітетів (SprintRepository) не виявляється жодного запису, створюється новий із тривалістю 7 днів за замовчуванням. Потім ці дані додаються до колекції SprintModels, яка представлена як ObservableCollection<SprintModel> для автоматичного оновлення інтерфейсу. Важливо, що цей метод перевіряє прапор IsBusy, щоб уникнути одночасного запуску кількох процесів оновлення, що може вплинути на продуктивність застосунку.

```
internal void Refresh()
{
    if (IsBusy)
    {
        return;
    }
    IsBusy = true;
    Task.Run<List<SprintTable>>(async () =>
    {
        List<SprintTable> tasks = await _sprintRepository.GetItems();
        //if no sprint exist create a new sprint with default duration as 7
        if (tasks.Count() == 0)
        {
            var task = new SprintTable()
            {
                SprintDuration = 7
            };
            await _sprintRepository.SaveItem(task);
            tasks.Add(task);
        }
        return tasks;
    }).
    ContinueInMainThreadWith((sprintTables) =>
    {
        if (sprintTables != null && sprintTables.Count > 0)
        {
            sprintTables.ForEach((t) =>
            {
                SprintModels.Add(new SprintModel(t));
            });
            SelectedSprint = SprintModels.Where(s => s.Id.Equals(SelectedSprintID)).FirstOrDefault();
```

Рисунок 3.9 – Метод Refresh

Даний клас ефективно працює з базою даних через репозиторії, обробляє команди для збереження, видалення та оновлення даних, а також використовує асинхронні операції для забезпечення високої продуктивності. Це робить клас важливим компонентом у структурі застосунку, забезпечуючи чіткий розподіл відповідальностей та спрощуючи розширення функціональності.

Наступний клас – SprintViewModel відповідає за управління логікою сторінки управління пріоритетами у Task-менеджері. Керує створенням, оновленням та переглядом списку пріоритетів, а також забезпечує навігацію між сторінками та збереження змін у базі даних через репозиторій ISprintRepository.

```
[INotifyPropertyChanged]
public partial class SprintViewModel : BaseViewModel
{
    private readonly ISprintRepository _sprintRepository;

    [ObservableProperty]
    public ObservableCollection<SprintModel> _sprintModels;


    public SprintViewModel(ISprintRepository sprintRepository)
    {
        SprintModels = new ObservableCollection<SprintModel>();
        _sprintRepository = sprintRepository;
    }

    [RelayCommand]
    async void SprintTapped(SprintModel sprintModel)
```

Рисунок 3.10 – Клас SprintViewModel

Основною відповідальністю SprintViewModel є управління колекцією об'єктів SprintModel, яка представлена у вигляді ObservableCollection. Це дозволяє автоматично оновлювати інтерфейс користувача при зміні даних у колекції. Колекція SprintModels містить список пріоритетів, які можуть бути додані, оновлені або переглянуті. Ініціалізація цієї колекції відбувається в конструкторі, де також відбувається передача репозиторію ISprintRepository через механізм Dependency Injection для взаємодії з базою даних.

Методи класу реалізовані через RelayCommand, що забезпечує асинхронне виконання команд у середовищі MAUI.

Метод SprintTapped (рис. 3.11) використовується для навігації до сторінки з детальною інформацією про обраний пріоритет (SprintDetailsView). Вона передає до нової сторінки ідентифікатор пріоритету (SprintID) та список завдань (TodoItems) через механізм Shell у MAUI, що дозволяє передавати параметри між сторінками.

```
[RelayCommand]
async void SprintTapped(SprintModel sprintModel)
{
    if (IsBusy) return;
    IsBusy = true;
    await Shell.Current.GoToAsync(nameof(SprintDetailsView), true,
        new Dictionary<string, object>()
        {
            ["SprintTodoTableItems"] = sprintModel.TodoItems,
            ["SprintID"] = sprintModel.Id
        });
    IsBusy = false;
}
```

Рисунок 3.11 – Метод SprintTapped

Команда AddTapped (рис. 3.12) відповідає за створення нового пріоритету. При її виклику створюється новий запис у базі даних через метод SaveItem репозиторію, а потім викликається метод Refresh() для оновлення списку пріоритетів у користувацькому інтерфейсі. Варто відзначити, що для уникнення дублювання викликів використовується флаг IsBusy, який блокує повторний виклик методів, якщо вже виконується поточний процес.

```
[RelayCommand]
async void AddTapped()
{
    if (IsBusy) return;
    IsBusy = true;
    await Task.Run(async () =>
    {
        await _sprintRepository.SaveItem(new SprintTable
        {
            SprintDuration = 7
        });
    });
    IsBusy = false;
    Refresh();
}
```

Рисунок 3.12 – Команда AddTapped

Метод IsCompleteTapped (рис. 3.13) дозволяє зберегти оновлений стан пріоритету. Він працює асинхронно за допомогою Task.Run, що дозволяє уникнути блокування основного потоку, при цьому зберігаючи зміни у базі даних. Використання асинхронних методів підвищує продуктивність застосунку, зберігаючи інтерфейс чуйним до дій користувача.

```
[RelayCommand]
void IsCompleteTapped(SprintModel sprintModel)
{
    Task.Run(async () =>
    {
        //var service = await TodoService.Instance;
        await _sprintRepository.SaveItem(sprintModel.Sprinttable);
    });
}
```

Рисунок 3.13 – Метод IsCompleteTapped

Метод Refresh (рис. 3.14) є ключовим для оновлення списку пріоритетів. Він виконує завантаження даних з репозиторію ISprintRepository, асинхронно отримуючи об'єкти SprintTable із бази даних. Дані сортуються за Id для забезпечення правильного порядку відображення, після чого колекція SprintModels оновлюється. Важливо, що IsBusy використовується як захист від повторних викликів оновлення, якщо поточний процес ще не завершено.

```
internal void Refresh()
{
    if (IsBusy)
    {
        return;
    }
    IsBusy = true;

    Task.Run<List<SprintTable>>(async () =>
    {
        var tasks = await _sprintRepository.GetItems();
        tasks.Sort((s1, s2) =>
        {
            return s1.Id < s2.Id ? -1 : 1;
        });
        return tasks;
    }).
    ContinueInMainThreadWith((sprintTables) =>
    {
        SprintModels.Clear();
        if (sprintTables != null && sprintTables.Count > 0)
        {
            sprintTables.ForEach((s) => { SprintModels.Add(new SprintModel(s)); });
        }

        IsBusy = false;
    });
```

Рисунок 3.14 – Метод Refresh

Крім того, методи GotoMainPage і Cancel (рис. 3.15) використовуються для навігації назад до головної сторінки, забезпечуючи базову функціональність повернення до попереднього стану застосунку через Shell.Current.GoToAsync(".."). Це забезпечує зручний контроль навігації між сторінками без прямого використання контролерів.

```
[RelayCommand]
async void GotoMainPage()
{
    if (IsBusy) return;
    IsBusy = true;
    await Shell.Current.GoToAsync("..");
    IsBusy = false;
}

[RelayCommand]
async void Cancel()
{
    await Shell.Current.GoToAsync("..");
}
```

Рисунок 3.15 – Методи GotoMainPage, Cancel

Використання ObservableCollection, RelayCommand та асинхронних методів в даному класі дозволяє зробити цей клас ефективним інструментом для управління даними та підтримки принципів MVVM.

Далі потрібно розглянути клас TodoViewModel. Він відповідає за логіку управління завданнями у Task-менеджері. Основною його функцією є управління списком завдань, їх створення, перегляд. Він взаємодіє з репозиторієм ITodoRepository для збереження та отримання даних з бази даних.

Основним джерелом даних у цьому класі є ObservableCollection<TodoModel> під назвою TodoModels. Цей тип колекції дозволяє автоматично оновлювати інтерфейс користувача під час змін у даних, що є ключовим у реалізації патерну MVVM. Колекція ініціалізується як порожня при створенні екземпляра TodoViewModel, і її наповнення здійснюється через метод Refresh.

Конструктор класу приймає залежність ITodoRepository, яка передається через механізм ін'єкції залежностей (Dependency Injection). Це забезпечує централізований доступ до бази даних для збереження, оновлення та видалення завдань. Репозиторій зберігається у приватному полі _todoRepository для подальшого використання у методах цього класу.

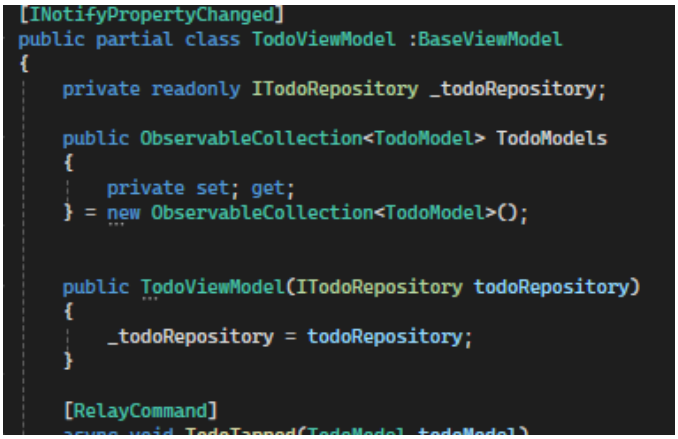


Рисунок 3.16 – Клас TodoViewModel

Методи взаємодії з даними та сторінками реалізовані через атрибут [RelayCommand], який автоматично генерує команди для MAUI. Команда TodoTapped (рис. 3.17) обробляє натискання на завдання у списку та виконує навігацію до сторінки TodoDetailsView, передаючи вибране завдання та його SprintID. Ця функціональність подібна до методу SprintTapped з класу SprintViewModel, де також використовується передача даних між сторінками через Shell.Current.GoToAsync.

Рисунок 3.17 – Метод TodoTapped

Метод IsCompleteTapped (рис. 3.18) використовується для оновлення статусу завершення завдання. Він викликає збереження змін через репозиторій у фоновому потоці. Аналогічний використовується у SprintViewModel.

Команда AddTapped (рис. 3.18) відкриває сторінку створення нового завдання, використовуючи Shell.Current.GoToAsync. Це дозволяє користувачу створити нове завдання, яке буде збережене після введення інформації. Подібним чином працює метод AddTapped у SprintViewModel, де створюється новий пріоритет.

```
[RelayCommand]
void IsCompleteTapped(TodoModel todoModel)
{
    Task.Run(async () =>
    {
        //var service = await TodoService.Instance;
        await _todoRepository.SaveItem(todoModel.Todotable);
    });
}
[RelayCommand]
async void AddTapped()
{
    if (IsBusy) return;
    IsBusy = true;
    await Shell.Current.GoToAsync(nameof(TodoDetailsView));
    IsBusy = false;
}
```

Рисунок 3.18 - Методи IsCompleteTapped, AddTapped

Методи GotoSprintView і GotoMainPage використовуються для навігації між сторінками. Перший відкриває сторінку управління пріоритетами, а другий повертає користувача на головну сторінку, використовуючи Shell.Current.GoToAsync. Функціонал аналогічний методу GotoMainPage у SprintViewModel.

Метод Refresh (рис. 3.19) відповідає за оновлення списку завдань на сторінці. Він завантажує дані з репозиторію ITodoRepository та сортує їх за статусом (IsComplete). Незавершені завдання відображаються першими. Сортування та оновлення ObservableCollection після завантаження даних забезпечує автоматичне оновлення інтерфейсу користувача. Подібний підхід до оновлення використовується в SprintViewModel у методі Refresh.

```
internal void Refresh()
{
    if (IsBusy)
    {
        return;
    }
    IsBusy = true;

    Task.Run<List<TodoTable>>(async () =>
    {
        //var service = await TodoService.Instance;
        var tasks = await _todoRepository.GetItems();
        tasks.Sort((t1, t2) =>
        {
            if (t1.IsComplete == t2.IsComplete)
            {
                return t1.Id < t2.Id ? -1 : 1;
            }
            else if (t1.IsComplete)
            {
                return 1;
            }
            else
            {
                return -1;
            }
        });
        return tasks;
    }).
    ContinueInMainThreadWith((taskTables) =>
    {
        TodoModels.Clear();
        if (taskTables != null && taskTables.Count > 0)
        {
            taskTables.ForEach((t) => { TodoModels.Add(new TodoModel(t)); });
        }

        IsBusy = false;
    });
}
```

Рисунок 3.19 – Метод Refresh

Наступний допоміжний клас TodoContext (рис. 3.20). Він відповідає за взаємодію з базою даних SQLite через Entity Framework Core (EF Core). Цей клас реалізує контекст бази даних (DbContext), що дозволяє зручно працювати з таблицями та здійснювати операції створення, оновлення, збереження й видалення даних.

```
namespace Todo.me.Context;
public class TodoContext : DbContext
{
    public TodoContext(DbContextOptions<TodoContext> options)
        : base(options)
    {
        SQLitePCL.Batteries_V2.Init();
        //this.ChangeTracker.QueryTrackingBehavior = QueryTrackingBehavior.NoTrackin
        //this.ChangeTracker.LazyLoadingEnabled = false;
        this.Database.EnsureCreated();
    }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        base.OnConfiguring(optionsBuilder);
        optionsBuilder.UseSqlite($"Filename={Constants.DbPath}");
```

Рисунок 3.20 – Клас TodoContext

Конструктор класу TodoContext приймає об'єкт DbContextOptions<TodoContext> як параметр, що дозволяє налаштовувати

підключення до бази даних під час реєстрації залежностей через Dependency Injection. Конструктор також викликає SQLitePCL.Batteries_V2.Init() для ініціалізації бібліотеки SQLite, яка забезпечує коректну роботу з базою даних. Метод this.Database.EnsureCreated (рис. 3.20) автоматично створює базу даних, якщо вона ще не існує, що зручно для простих застосунків, де складна міграція даних може бути не потрібна.

Метод OnConfiguring перевизначає базову конфігурацію підключення до бази даних. Він викликає UseSqlite, який вказує шлях до бази даних SQLite (Constants.DbPath). Це дозволяє зберігати базу даних у локальному файлі. Також у цьому методі налаштовано логування SQL-запитів через метод LogTo. Логи SQL-запитів відображаються у вікні відлагодження (Debug.WriteLine) з рівнем деталізації Information. Параметр EnableSensitiveDataLogging() дозволяє логувати конфіденційні дані, такі як значення параметрів запитів, що корисно для налагодження, але не рекомендовано для використання у production-середовищі.

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    base.OnConfiguring(optionsBuilder);
    optionsBuilder.UseSqlite($"Filename={Constants.DbPath}");
    optionsBuilder.LogTo(message => Debug.WriteLine(message), new[] {
        DbLoggerCategory.Database.Command.Name
    }, LogLevel.Information).EnableSensitiveDataLogging();
}
```

Рисунок 3.21 – Клас OnConfiguring

Метод OnModelCreating (рис. 3.22) використовується для налаштування моделей даних (TodoTable і SprintTable). Тут визначені ключі (HasKey) для кожної таблиці, а також встановлені зв’язки між таблицями. Зокрема, модель TodoTable має зовнішній ключ SprintID, який пов’язує завдання зі спринтом через відношення "один до багатьох" (HasOne...WithMany). Це дозволяє кожному спринту (SprintTable) мати кілька завдань (TodoItems).

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    base.OnModelCreating(modelBuilder);
    modelBuilder.Entity<TodoTable>()
        .HasKey(x => x.Id);

    modelBuilder.Entity<TodoTable>()
        .HasOne(x => x.Sprint)
        .WithMany(x=>x.TodoItems)
        .HasForeignKey(x => x.SprintID);

    modelBuilder.Entity<SprintTable>()
        .HasKey(x => x.Id);
}
public DbSet<TodoTable> Todos
{
    get; set;
}
public DbSet<SprintTable> Sprints
{
    get; set;
}
```

Рисунок 3.22 – Метод OnModelCreating

Клас також містить два DbSet-властивості: Todos та Sprints. Вони представляють таблиці бази даних і використовуються для виконання CRUD-операцій (Create, Read, Update, Delete). Через ці властивості Entity Framework Core надає доступ до даних у вигляді об'єктів TodoTable та SprintTable. Використання DbSet спрощує взаємодію з базою, оскільки дозволяє виконувати запити через LINQ, наприклад, context.Todos.ToList().

3.4 Проектування користувацького інтерфейсу

Проєктування інтерфейсу користувача є одним із ключових етапів розробки сучасних програмних застосунків, особливо у випадку кросплатформених додатків, таких як експертна система Task-менеджеру, створена за допомогою MAUI. Ефективний інтерфейс не лише відповідає сучасним стандартам візуального дизайну, але й забезпечує зручність та інтуїтивність взаємодії користувача з додатком. Головною метою під час розробки інтерфейсу є створення естетично привабливого, доступного та функціонального дизайну, який відповідає принципам мінімалізму, послідовності та фокусування на ключових функціях програми. Використання

XAML (Extensible Application Markup Language) у MAUI забезпечує декларативний підхід до створення інтерфейсу, дозволяючи розробникам ефективно розділити візуальну частину додатка від бізнес-логіки, що реалізується у ViewModel відповідно до патерну MVVM.

Спочатку розглянемо головну сторінку застосунку, яка відіграє роль вітальної панелі, з якої користувач може розпочати роботу із застосунком. Основними елементами інтерфейсу цієї сторінки є анімований логотип, вітальний текст, кнопка для переходу до перегляду завдань та мотивуюча цитата. Головна сторінка реалізована за допомогою контейнера ScrollView, що дозволяє зручно відображати контент на різних екранах, включаючи мобільні пристрої. Всі елементи сторінки розміщені у вертикальному стековому контейнері VerticalStackLayout (рис. 3.23), який забезпечує послідовне розміщення компонентів один під одним із заданими відступами для зручності читання.

```
<!-- Градієнтний фон сторінки -->
<ContentPage.Background>
    <LinearGradientBrush StartPoint="0,0" EndPoint="1,1">
        <GradientStop Color="#FFA07A" Offset="0.0" />
        <GradientStop Color="#FF6347" Offset="1.0" />
    </LinearGradientBrush>
</ContentPage.Background>

<!-- Основний контейнер -->
<ScrollView>
    <VerticalStackLayout
        Spacing="25"
        Padding="30"
        VerticalOptions="Center">

        <!-- Анімація з тінню -->
        <Frame
            HasShadow="True"
            CornerRadius="20"
            BackgroundColor="White"
            Padding="10"
            HorizontalOptions="Center">
```

Рисунок 3.23 - Контейнер VerticalStackLayout

Анімований логотип реалізується через елемент SKLottieView з бібліотеки SkiaSharp (рис. 3.24), який дозволяє відтворювати анімацію у

форматі Lottie JSON. Його розміщено всередині контейнера Frame з тінню (HasShadow="True") та заокругленими кутами (CornerRadius), що додає сучасного вигляду. Вітальний текст реалізований через елемент Label із заданим розміром шрифту (FontSize="36") та жирним шрифтом (FontAttributes="Bold"), що дозволяє одразу привернути увагу користувача до назви застосунку. Допоміжний текст нижче має менший розмір (FontSize="20") і нейтральний колір, щоб не відволікати від основного повідомлення.

```
<!-- Анімація з тінню -->
<Frame
    HasShadow="True"
    CornerRadius="20"
    BackgroundColor="White"
    Padding="10"
    HorizontalOptions="Center">
    <skia:SKLottieView
        Source="todo.json"
        RepeatCount="-1"
        SemanticProperties.Description="Ласкаво просимо до Todo.me!"
        HeightRequest="150"
        WidthRequest="150"
        HorizontalOptions="Center" />
</Frame>

<!-- Привітальний текст -->
<Label
    Text="Ласкаво просимо до Todo.me!"
    FontSize="36"
    FontAttributes="Bold"
    HorizontalOptions="Center"
    TextColor="White"/>
```

Рисунок 3.24 – Анімований логотип та вітальний текст

Ключовим інтерактивним елементом головної сторінки є кнопка переходу до списку завдань, реалізована через елемент Button (рис. 3.25) із закругленими кутами (CornerRadius) та виразним зеленим фоном (BackgroundColor="#4CAF50"). Для забезпечення сучасного вигляду вона також поміщена у Frame для створення ефекту тіні. До кнопки прив’язана команда GotoTodoViewCommand, що ініціює навігацію до сторінки управління завданнями. Це реалізується через механізм прив’язки даних у XAML (Command="{Binding GotoTodoViewCommand}"), що дозволяє забезпечити прямий зв’язок між інтерфейсом і логікою, реалізованою у ViewModel.

```
<!-- Кнопка з тінню -->
<Frame
    HasShadow="True"
    CornerRadius="20"
    BackgroundColor="White"
    Padding="5"
    HorizontalOptions="Center">
    <Button
        Text="Перейти до завдань"
        Command="{Binding GotoTodoViewCommand}"
        BackgroundColor="#4CAF50"
        TextColor="White"
        FontSize="20"
        Padding="15"
        CornerRadius="25"
        WidthRequest="250"
        SemanticProperties.Hint="Відкрити список завдань" />
</Frame>
```

Рисунок 3.25 – Дизайн кнопки

На завершення сторінки розташована мотивуюча цитата, представлена елементом Label зі стилізованим текстом (FontAttributes="Italic") та стриманим сірим кольором, що підсилює загальну естетику сторінки. Для візуального розділення елементів використовується BoxView у вигляді горизонтальної лінії. Завдяки використанню таких компонентів, як Frame, Label, Button та SKLottieView, сторінка виглядає структуровано, збалансовано та відповідає принципам сучасного дизайну, поєднуючи функціональність із естетикою.

Наступне вікно – це список завдань. Використовується для відображення основного списку завдань, а також надає інструменти для навігації між сторінками. Сторінка складається з кількох ключових компонентів: заголовка сторінки, списку завдань, інтерактивних елементів управління та нижньої панелі навігації.

Центральним елементом є CollectionView (рис. 3.26), який використовується для відображення списку завдань, де кожен елемент представлений через шаблон DataTemplate. У цьому шаблоні використовується Frame з тінями, закругленими кутами та фоном, що змінюється відповідно до стану завдання за допомогою

ValueToColorConverter. Це дозволяє візуально розділяти завдання за пріоритетністю або статусом виконання.

```
<CollectionView  x:Name="xCollectionView" Grid.Row="1" Grid.Column="0" Grid.ColumnSpan="3"
                 HorizontalOptions="FillAndExpand"  ItemsSource="{Binding TodoModels}"
                 VerticalOptions="FillAndExpand">
    <CollectionView.Shadow>
        <Shadow Brush="{AppThemeBinding Light={StaticResource White}, Dark={StaticResource Black}}"
                Radius="5"
                Opacity="0.3"/>
    </CollectionView.Shadow>
    <CollectionView.ItemsLayout>
        <LinearItemsLayout Orientation="Vertical"
                           ItemSpacing="8"/>
    </CollectionView.ItemsLayout>
    <CollectionView.ItemTemplate>
        <DataTemplate x:DataType="model:TodoModel">
            <Frame BackgroundColor="{Binding Color, Converter={StaticResource ValueToColorConverter}}"
                   CornerRadius="15"
                   IsClippedToBounds="True"
                   BorderColor="{Binding Color, Converter={StaticResource ValueToColorConverter}}"
                HasShadow="True">

                <Frame.Shadow>
                    <Shadow Brush="{AppThemeBinding Light={StaticResource White}, Dark={StaticResource Black}}"
                            Offset="6,10"
                            Opacity="0.8"/>
                </Frame.Shadow>

                <SwipeView HorizontalOptions="Center" VerticalOptions="Center" >
                    <SwipeView.LeftItems>
                        <SwipeItems SwipeBehaviorOnInvoked="RemainOpen">
                            <SwipeItem
```

Рисунок 3.26 – Елемент CollectionView

Кожен елемент списку підтримує функціональність «свайпів» (SwipeView), яка дозволяє користувачеві видаляти або відмічати завдання як виконані (рис. 3.27). З лівого боку при «свайпі» відображається іконка видалення, а з правого — завершення завдання. Команди прив’язані до методів у ViewModel через Command="{Binding IsCompleteTappedCommand}". Крім того, кожен елемент містить CheckBox для відмітки виконання завдання, що також інтерактивно оновлює його стан через прив’язку даних.

```
<SwipeView HorizontalOptions="Center" VerticalOptions="Center" >
    <SwipeView.LeftItems>
        <SwipeItems SwipeBehaviorOnInvoked="RemainOpen">
            <SwipeItem
                    IconImageSource="delete"
                    BackgroundColor="{Binding Color, Converter={StaticResourc
        </SwipeItems>
    </SwipeView.LeftItems>

    <SwipeView.RightItems>
        <SwipeItems SwipeBehaviorOnInvoked="RemainOpen">
            <SwipeItem
                    IconImageSource="approve"
                     BackgroundColor="{Binding Color, Converter={StaticResour
                     Command="{Binding Path=IsCompleteTappedCommand, Mode=One
                     CommandParameter="{Binding .}"/>
        </SwipeItems>
    </SwipeView.RightItems>
    <StackLayout Orientation="Vertical">
        <StackLayout.GestureRecognizers>
            <TapGestureRecognizer Command="{Binding Path=TodoTappedCommand, S
                                  CommandParameter="{Binding .}" />
        </StackLayout.GestureRecognizers>
        <Label
                    TextColor="{Binding Source={Reference xIsComplete}, P
                    Text="{Binding Name}"
                    MaxLines="3"
                    FontSize="Large"
```

Рисунок 3.27 – Додавання роботи «свайпів»

Заголовок сторінки реалізований через Label з великим розміром шрифту (FontSize="35") та жирним накресленням (FontAttributes="Bold"), що допомагає чітко відобразити мету сторінки. Для візуальної гармонії елементи розміщені у VerticalStackLayout з відступами та центруванням по горизонталі.

Нижня панель навігації (Border) містить три основні кнопки (ImageButton): повернення на головну сторінку, додавання нового завдання та перехід до сторінки управління пріорітетами (рис. 3.28). Кожна кнопка має стиль із тінями та заокругленими кутами, що забезпечує візуальну консистентність із рештою інтерфейсу. Команди для кнопок (GotoMainPageCommand, AddTappedCommand, GotoSprintViewCommand) прив'язані до методів ViewModel.

```
</CollectionView>
<Border Grid.Row="2"
        Grid.Column="0"
        Grid.ColumnSpan="3"
        StrokeThickness="0"
        Padding="10"
        BackgroundColor="{AppThemeBinding Light={StaticResource White}, Dark={StaticResource Black}}"
        HorizontalOptions="Center">
    <Grid RowDefinitions="*" ColumnDefinitions="*,*,*"
          ColumnSpacing="50"
          VerticalOptions="Center">
        <ImageButton Grid.Column="0" Source="todo" HeightRequest="35"
                     WidthRequest="30" HorizontalOptions="Center" VerticalOptions="Center"
                     Command="{Binding GotoMainPageCommand}"/>
        <Border Grid.Column="1"
                StrokeThickness="0" StrokeShape="RoundRectangle 10"
                BackgroundColor="CadetBlue"
                HeightRequest="48" WidthRequest="48">
            <Border.Shadow>
                <Shadow Brush="{AppThemeBinding Light={StaticResource White}, Dark={StaticResource Black}}"
                    Radius="5"
                    Opacity="0.3"/>
            </Border.Shadow>
            <ImageButton Source="add" HeightRequest="32" WidthRequest="22"
                         HorizontalOptions="Center" VerticalOptions="Center"
                         Command="{Binding AddTappedCommand}"/>
        </Border>
        <ImageButton Grid.Column="2" Source="setting" HeightRequest="35"
                     WidthRequest="30"
                     HorizontalOptions="Center" VerticalOptions="Center"
                     Command="{Binding GotoSprintViewCommand}"/>
    </Grid>
</Border>
```

Рисунок 3.28 – Панель навігації

Сторінка TodoDetailsView у застосунку Task-менеджеру призначена для відображення деталей конкретного завдання, його редагування, вибору пріорітету, оновлення статусу виконання, а також для збереження або видалення завдання.

Головним елементом сторінки є Grid, який структурно поділяє простір на кілька секцій. Верхній рядок сітки (Grid.Row="0") містить кнопку ImageButton для повернення на попередню сторінку, прив’язану до команди CancelCommand (рис. 3.29). Вона розміщена у лівій частині екрана, використовуючи стандартний для мобільних додатків підхід до навігації.

```
<Grid Margin="15,0" HorizontalOptions="FillAndExpand" VerticalOptior
    <Grid.RowDefinitions>
        <RowDefinition Height="auto" />
        <RowDefinition Height="*"/>
        <RowDefinition Height="*"/>
        <RowDefinition Height="auto"/>
        <RowDefinition Height="auto"/>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="auto"/>
        <ColumnDefinition Width="*"/>
        <ColumnDefinition Width="*"/>
        <ColumnDefinition Width="auto"/>
    </Grid.ColumnDefinitions>
    <ImageButton Command="{Binding CancelCommand}" Source="back"
                 Grid.Row="0" Grid.Column="0" Padding="0,10"
                 HorizontalOptions="Start"
                 VerticalOptions="Start"/>
    <StackLayout Spacing="5"
                 Grid.Row="1"
                 Grid.RowSpan="3"
```

Рисунок 3.29 – Елементи Grid та кнопка ImageButton

Основний контент сторінки розташований у центральній секції (Grid.Row="1-3"). Він представлений у вигляді вертикального стекового контейнера StackLayout (рис. 3.30), що містить кілька ключових елементів. Поле для введення назви завдання (Entry) має двосторонню прив’язку (TwoWay) до властивості TodoModel.Name у TodoDetailsViewModel, що забезпечує автоматичне оновлення даних у моделі під час змін у текстовому полі. Далі розташований елемент Picker для вибору спринту, прив’язаний до SprintModels і властивості SelectedSprint. Це дозволяє користувачу вибрати, до якого спринту буде віднесене завдання, використовуючи джерело даних, завантажене із ViewModel.

Окремо виділений текстовий редактор Editor для введення опису завдання (TodoModel.Description). Він підтримує двосторонню прив’язку, автоматично оновлюючи дані в моделі під час введення тексту. Всі ці елементи

розміщені у зручному контейнері StackLayout із відступами для естетичного розташування компонентів.

```
<ImageButton Command="{Binding CancelCommand}" Source="back"
             Grid.Row="0" Grid.Column="0" Padding="0,10"
             HorizontalOptions="Start"
             VerticalOptions="Start"/>
<StackLayout Spacing="5"
             Grid.Row="1"
             Grid.RowSpan="3"
             Grid.ColumnSpan="4"
             VerticalOptions="FillAndExpand"
             HorizontalOptions="FillAndExpand"
             Orientation="Vertical">

    <VerticalStackLayout Spacing="20" Grid.Row="1">
        <Entry Text="{Binding TodoModel.Name, Mode=TwoWay}" Placeholder="Title"
          FontSize="Title"/>
    </VerticalStackLayout>
        <HorizontalStackLayout Spacing="7" VerticalOptions="Center" Grid.Row="2">
            <Label Text="Sprint:" FontSize="Medium"
                   VerticalOptions="Center" HorizontalOptions="Start"/>
            <Picker x:Name="SprintPicker" Title="Select a Sprint"
                    ItemsSource="{Binding SprintModels}"
                    ItemDisplayBinding="{Binding Id}"
                    SelectedItem="{Binding SelectedSprint, Mode=TwoWay}"
                    FontSize="Medium" VerticalOptions="Center" HorizontalOptions="Start"/>
    </HorizontalStackLayout>
```

Рисунок 3.30 – Основний контент сторінки

Нижня панель керування (Border) містить основні кнопки управління завданням. Тут розташовані:

CheckBox завершення завдання (CheckBox) з командою IsCompleteTappedCommand, яка дозволяє змінити статус виконання завдання. Чекбокс змінює свій колір залежно від стану (BoolToColorIsCompleteConverter).

- Кнопка збереження (ImageButton із Source="approve"), яка викликає команду SaveCommand для збереження змін.
- Кнопка скасування (ImageButton із Source="cancel"), яка закриває сторінку без збереження даних (CancelCommand).
- Кнопка видалення (ImageButton із Source="delete"), що видаляє завдання з бази даних. Вона відображається лише у випадку, коли у моделі вже існує TodoModel.Id (перевіряється через конвертер IntToBoolConverter).

Зовнішній вигляд сторінки досягнута за допомогою кольорових акцентів, які динамічно змінюються відповідно до кольору, прив’язаного до завдання (TodoModel.Color). Це реалізовано через конвертер

ValueToColorConverter, який застосовується як до фону сторінки, так і до панелі керування.

3.5 Тестування додатку

Для перевірки роботи, додатку використовуються середовище VisualStudio, за допомогою нього можна зручно створити пристрій для емуляції через Android Studio (рис. 3.31):

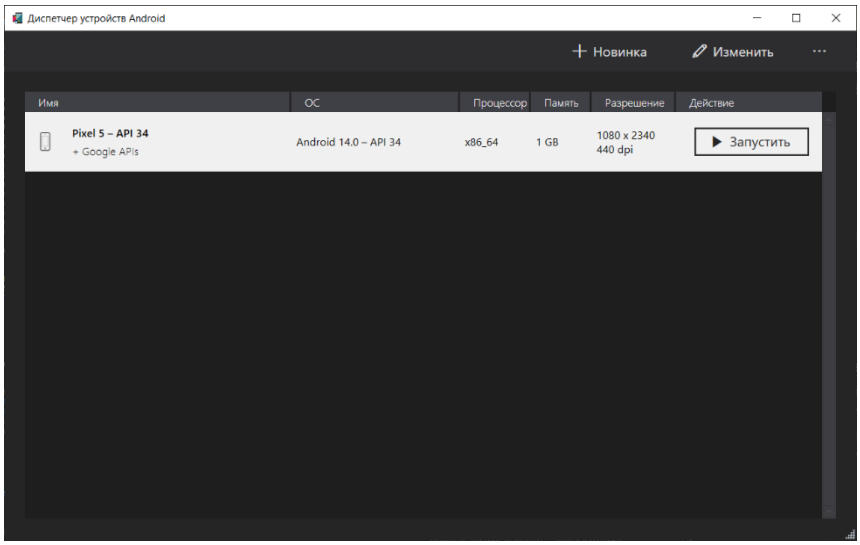


Рисунок 3.31 – Список доступних пристроїв для тестування

Після виконання цієї дії має активуватися емулятор мобільного пристрою, що дозволить перевірити роботу застосунку у віртуальному середовищі, максимально наближеному до реального пристрою (рис. 3.31):

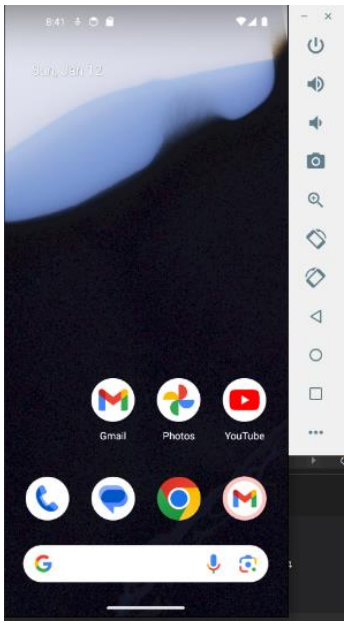


Рисунок 3.31 – Емуляція мобільного пристрою

Потрібно далі перейти в меню та завантажити саму програму. Емулятор має відобразити графічний інтерфейс програми.

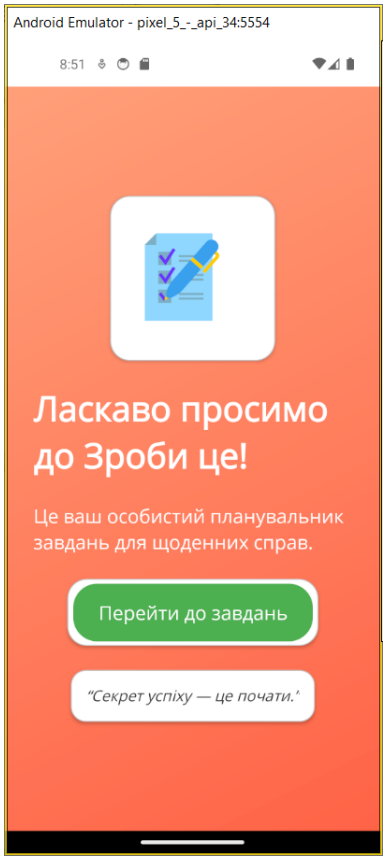


Рисунок 3.32 – Завантаження додатку

Так як проект коректно завантажився, в користувача є можливість натиснути на кнопку «Перейти до завдань». Після цього з’явиться екран щ усіма створюваними завданнями (рис. 3.33):

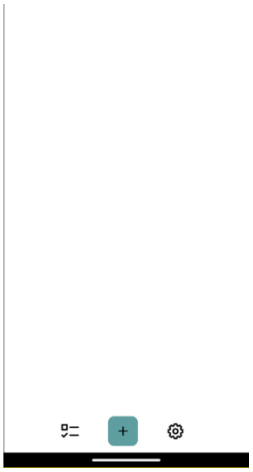


Рисунок 3.33 – Вікно завдань

Для створення нового завдання користувачу потрібно натиснути на кнопку «+». Перед користувачем відкриється вікно в якому він може ввести назву завдання та основний текст (рис. 3.34):

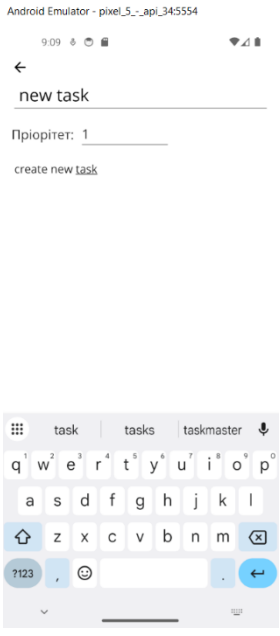


Рисунок 3.34 – Створення нового завдання

На головному екрані з’явиться вже нове завдання в якому зображена тема та текст (рис. 3.35):

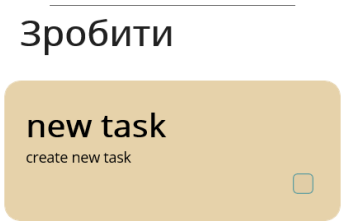


Рисунок 3.35 – Поява завдання на головному екрані

Для створення нового пріоритету потрібно перейти на наступне вікно та та теж натиснути «+»:

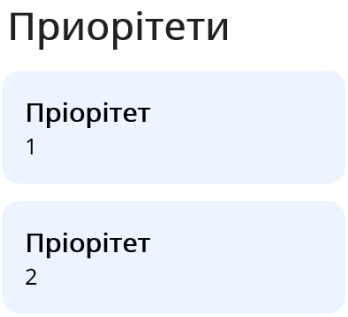


Рисунок 3.35 – Створення нового пріоритету

Тепер користувач отримує можливість взаємодіяти з експертною системою для визначення важливості завдань, встановлюючи пріоритет відповідно до певного числового значення. Це дозволяє систематизувати завдання за рівнем важливості, де нижчий номер пріоритету може відповідати більш критичним завданням, тоді як вищий — менш терміновим. Такий підхід сприяє ефективнішій організації робочого процесу, оскільки завдання сортуються за рівнем значущості, допомагаючи користувачу зосередитися на найважливіших завданнях у першу чергу.

3.6 Висновок третього розділу

У результаті було створено функціональний кросплатформенний додаток для управління завданнями, побудований на основі платформи .NET MAUI з використанням архітектурного патерну MVVM. Додаток забезпечує інтуїтивний користувацький інтерфейс, що дозволяє ефективно працювати зі списками завдань, встановлювати пріоритети, редагувати, видаляти та відмічати завдання як виконані. Створений додаток є зручним інструментом для планування, що сприяє підвищенню продуктивності користувача, забезпечуючи доступність на різних платформах та адаптивний дизайн.

ВИСНОВОК

У процесі виконання кваліфікаційної роботи на тему «Розробка крос-платформного Task-менеджеру на платформі .NET MAUI» було проведено комплексне дослідження методів управління завданнями, сучасних технологій розробки програмного забезпечення та архітектурних підходів до створення програмних систем. Робота включала теоретичний аналіз предметної області, дослідження існуючих рішень, обґрунтування вибору технологій, розробку програмної реалізації та тестування розробленого продукту.

Основними завданнями, які були вирішені під час виконання роботи, стали:

1. Проведення аналізу сучасних підходів до управління завданнями та принципів роботи експертних систем.
2. Огляд існуючих програмних рішень для управління завданнями, таких як Any.DO, Trello, Remember the Milk та Google Calendar.
3. Обґрунтування вибору платформи .NET MAUI для розробки Task-менеджеру, з огляду на її функціональність, кросплатформність та інтеграцію з C#.
4. Розробка експертної системи Task-менеджеру з використанням архітектурного патерну MVVM, що забезпечило розділення логіки, даних та інтерфейсу користувача.
5. Тестування програмного продукту для перевірки відповідності функціональним і нефункціональним вимогам, включаючи стабільність роботи, продуктивність та зручність використання.

У ході роботи було проведено порівняльний аналіз інструментів для розробки кросплатформних застосунків, зокрема .NET MAUI, Flutter та React Native. Вибір платформи .NET MAUI було обґрунтовано її можливостями для створення застосунків із єдиною кодовою базою, використанням C#, а також вбудованою підтримкою патерну MVVM, що значно спрощує розробку та підтримку програмного забезпечення.

У процесі роботи були сформовані вимоги до програмного забезпечення:

- Функціональні вимоги: створення, редагування та пріоритизація завдань, автоматизація управління завданнями, сортування за важливістю, візуалізація прогресу.
- Нефункціональні вимоги: кросплатформність, висока продуктивність, інтуїтивний інтерфейс, надійність роботи з великими обсягами даних.

Розроблений Task-менеджер було успішно протестовано, що підтвердило його ефективність для управління завданнями як у персональних, так і командних проєктах. Система може бути використана як основа для подальшого розвитку, зокрема шляхом інтеграції з хмарними сервісами та додавання аналітичних модулів для оцінки продуктивності користувачів.

Таким чином, поставлена мета щодо розробки експертної системи Task-менеджеру була досягнута, а отримані результати можуть бути використані для подальших досліджень у сфері управління завданнями та кросплатформної розробки програмного забезпечення.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Jordan B. Barlow, Justin Scott Giboney, Mark Jeffrey Keith, David W. Wilson, Ryan M. Schuetzler, Paul Benjamin Lowry, Anthony Vance. Overview and Guidance on Agile Development in Large Organizations. Communications of the Association for Information Systems. 2011, №29(2).

2. Rodenburg J. Code like a Pro in C# / Jort Rodenburg., 2021. – pp. 15-32.

3. Smith J. Entity Framework Core in Action / Jon Smith., 2018. – pp. 27-58.

4. Koloro. Тайм-менеджмент: принципи управління своїм часом – [Електронний ресурс] – Режим доступу: http://surl.li/yrcysm

5. MFlow. Що таке діаграма Ганта – [Електронний ресурс] – Режим доступу: http://surl.li/uxihnn

6. InvoDigital. Топ-100 популярних додатків світу – [Електронний ресурс] – Режим доступу: http://surl.li/tftegc

7. React Native learning book. – [Електронний ресурс] – Режим доступу: http://surl.li/ltigeg

8. . Бізнес-планування, сутність і поняття ризиків. – [Електронний ресурс] – Режим доступу: http://surl.li/fuxolr

9. Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions / Gregor Hohpe, Bobby Woolf – Addison-Wesley, 2004 – 736p.

10. Офіційний документація C# [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/overview

11. MVC Framework Introduction – [Електронний ресурс] – Режим доступу: https://www.geeksforgeeks.org/mvc-framework-introduction/

12. Introduction to MVVM Architecture – [Електронний ресурс] – Режим доступу: http://surl.li/yjbvta

13. Understanding VIPER Architecture – [Електронний ресурс] – Режим доступу: http://surl.li/prptzq

14. Документація Flutter. – [Електронний ресурс] – Режим доступу: https://flutter.dev

15. NET MAUI [Електронний ресурс] – Режим доступу: http://surl.li/fxxvpg