Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інформаційних технологій та комп'ютерної інженерії
(повне найменування факультету)

Кафедра обчислювальної техніки
(повна назва кафедри)

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«КОМП'ЮТЕРНА СИСТЕМА ДЛЯ ПЛАНУВАННЯ ТА КОНТРОЛЮ ЗАДАЧ КОРИСТУВАЧА»

08-54.БКР.026.00.000 ПЗ

Виконав: студент 4 курсу, групи 2КІ-23мс
спеціальності 123 – Комп'ютерна інженерія
(шифр і назва напряму підготовки, спеціальності)
Крижанівський Д. В.
(прізвище та ініціали)

Керівник: к.т.н., доц. доцент каф. ОТ
Томчук М. А.
(прізвище та ініціали)
«____» _____ 2025 р.

Рецензент: к.т.н., доц., доцент каф. ПЗ
Ракитянська Г. Б.
(прізвище та ініціали)
«____» _____ 2025 р.

Допущено до захисту
Завідувач кафедри ОТ
д.т.н., проф. Азаров О.Д.
«13» 06 2025 року

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Освітньо-кваліфікаційний рівень бакалавр
Галузь знань – 12 – Інформаційні технології
Спеціальність – 123 – Комп'ютерна інженерія
Освітньо-професійна програма – Комп'ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н., проф. Азаров О.Д.
«21» березня 2025 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Крижанівському Даниїлу Вікторовичу

1 Тема роботи: «Комп'ютерна система для планування та контролю задач користувача», керівник роботи Томчук М.А., к.т.н., доц., доцент кафедри ОТ затверджені наказом вищого навчального закладу від "20" березня 2025 року № 97.

2 Термін подання студентом роботи 13.05.2025 р.

3 Вихідні дані до роботи: сучасні технології створення клієнтської частини на основі WPF, використання шаблону MVVM, засоби організації локального зберігання даних, а також методи реалізації інтерфейсів користувача, сортування, фільтрації та перевірки введених даних.

4 Зміст розрахунково-пояснювальної записки: вступ, огляд сучасного стану проблеми, проектування програмного засобу, реалізація програмного засобу, висновки, список використаних джерел, додатки

5 Перелік графічного матеріалу: Блок-схема логіки взаємодії компонентів.

Перелік ілюстративного матеріалу: вигляд головного вікна програми, прик
відображення задач зі статусами, вигляд архіву виконаних задач, при
повідомлення про валідацію, приклад повідомлення про вихід з програми.

6 Консультанти розділів роботи наведені в таблиці 1.

Таблиця 1 — Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдан прийн
Розділ 1–3	к. т. н., доцент каф. ОТ Томчук М.А.	21.03.2025	13.05.2
Нормоконтроль	ас. каф. ОТ Швець С.І.	14.05.2025	30.05.2

7 Дата видачі завдання 21.03.2025 р.

8 Календарний план виконання БКР наведено в таблиці 2.

Таблиця 2 — Календарний план

№	Назва та зміст етапу	Термін виконання		Під
		початок	закінчення	
1	Вибір, узгодження та затвердження теми БКР	21.03.2025	22.03.2025	вико
2	Аналіз задачі дослідження	23.03.2025	24.03.2025	вико
3	Огляд існуючих систем управління задачами	25.03.2025	27.03.2025	вико
4	Вибір оптимальних технологій і компонентів для реалізації	28.03.2025	30.03.2025	вико
5	Розробка структури бази даних	31.03.2025	01.04.2025	вико
6	Розробка архітектури та інтерфейсу	02.04.2025	10.04.2025	вико
7	Реалізація функціоналу, тестування	11.04.2025	12.04.2025	вико
8	Попередній захист	13.05.2025	13.05.2025	вико

Студент Крижанівський Д.В.

Керівник роботи Томчук М.А.

АНОТАЦІЯ

УДК 621.3

Крижанівський Д.В. Комп’ютерна система для планування та контролю задач користувача. Бакалаврська кваліфікаційна робота зі спеціальності 123 «Комп’ютерна інженерія», освітня програма «Комп’ютерна інженерія». Вінниця: ВНТУ, 2025. 83 с.

На укр. мові. Бібліогр.: 21; рис.: 20; табл.: 6.

Бакалаврська кваліфікаційна робота присвячена розробці комп’ютерної системи для планування та контролю задач користувача. Метою створення системи є забезпечення зручного інструменту для організації особистих справ, збереження задач, їх фільтрація, сортування, архівація та візуальна індикація статусу. Проведено аналіз функціональних можливостей подібних програмних засобів, обґрунтовано вибір архітектури та програмних технологій. Розроблено структуру бази даних, реалізовано інтерфейс користувача з підтримкою шаблону MVVM та збереженням інформації у локальній базі даних SQLite. Проведено тестування функціоналу, описано основні сценарії використання програми.

Ключові слова: управління задачами, WPF, SQLite, MVVM, десктопний застосунок.

ABSTRACT

Kryzhanivsky D.V. Computer System for User Task Planning and Control. Bachelor qualification work on specialty 123 «Computer engineering», educational program «Computer engineering». Vinnytsa: VNTU, 2025. 67 p.

In Ukrainian. Bibliography: 21; Figures: 20; Tables: 6.

The bachelor's qualification work is dedicated to the development of a computer system for user task planning and control. The goal of the system is to provide a convenient tool for organizing personal tasks, saving them, filtering, sorting, archiving, and visually indicating their status. An analysis of the functionality of similar software solutions was carried out, and the choice of architecture and technologies was justified. The database structure was designed, and a user interface was implemented using the MVVM pattern, with local data storage based on SQLite. The functionality was tested and the main use cases of the application were described.

Key words: task management, WPF, SQLite, MVVM, desktop application.

ЗМІСТ

ВСТУП

У сучасних умовах інформаційного суспільства темпи життя вимагають високої організованості, планування та ефективного розподілу часу. Із розвитком цифрових технологій усе більшої популярності набувають програмні системи для управління задачами**,** які дозволяють оптимізувати повсякденні справи, підвищити особисту продуктивність і покращити контроль за виконанням важливих завдань. Особливої актуальності такі рішення набувають в умовах віддаленої роботи, навчання або індивідуальної проєктної діяльності, де користувачеві необхідно самостійно контролювати дедлайни та пріоритети задач. Комп'ютерна система планування задач виступає в ролі персонального помічника, що допомагає структурувати справи, встановлювати пріоритети, групувати задачі за категоріями та відстежувати виконання. Ефективність таких систем визначається не лише їх функціональністю, а й зручністю інтерфейсу, швидкістю роботи, адаптивністю до потреб користувача та здатністю зберігати й обробляти інформацію.

Актуальність теми даної кваліфікаційної роботи зумовлена зростаючою потребою сучасного суспільства в ефективному управлінні особистим часом та завданнями. У добу цифрових технологій дедалі більше користувачів стикаються з проблемою організації власної діяльності, відстеження строків виконання задач, встановлення пріоритетів та підтримки стабільного особистого плану. Це особливо актуально в умовах дистанційної роботи, навчання, багатозадачності та динамічного темпу життя, де втрата контролю над щоденними завданнями призводить до зниження ефективності, пропущених строків і стресу. Сьогоднішні рішення для планування часто є складними для освоєння, перевантаженими зайвими функціями або орієнтованими на колективну роботу в мережі, що не завжди відповідає індивідуальним потребам користувача. Існує потреба у створенні простого, зручного, локального програмного продукту, який дозволить швидко та зручно формувати графік виконання задач, контролювати їхній стан, визначати пріоритети та строки, а також зберігати історію виконання без необхідності підключення до інтернету.

Мета дослідження полягає у розробці комп’ютерної системи для планування та контролю задач користувача, яка забезпечує зручну взаємодію з задачами, надає можливість формувати особистий графік виконання, визначати пріоритети, встановлювати дедлайни, архівувати завершені задачі та зберігати дані у локальній базі з використанням сучасних технологій програмування.

Задачі дослідження:

— проаналізувати вимоги до сучасних програм для планування задач;

— обґрунтувати вибір мови програмування, технологій та архітектури системи;

— розробити структуру бази даних для збереження задач користувача;

— реалізувати механізми додавання, редагування, видалення та архівації задач;

— забезпечити можливість фільтрації задач за різними параметрами;

— реалізувати валідацію введених даних та індикацію пріоритетів і дедлайнів;

— виконати тестування програмного продукту та надати інструкцію користувача.

Об'єкт дослідження — процес організації та управління задачами користувачів за допомогою комп’ютерних засобів.

Предмет дослідження — архітектурні та функціональні рішення програмного забезпечення для управління задачами на базі WPF, MVVM та SQLite.

Практичне застосування — розроблена в рамках даної кваліфікаційної роботи, система може бути використана як персональний застосунок для студентів, фахівців, працівників офісів або будь-яких користувачів, які прагнуть ефективно керувати своїм часом та завданнями. Крім того, вона може використовуватись під час освітнього процесу або в професійній діяльності для формування персонального графіку виконання завдань. Програма не вимагає підключення до інтернету, має простий інтерфейс, забезпечує візуалізацію пріоритетів, дедлайнів і

підтримує зберігання інформації у локальній базі. Вона може бути інтегрована у повсякденну діяльність як самостійне рішення або як складова частина більшої інформаційної системи.

Апробація результатів кваліфікаційної роботи: доповідь на міжнародній конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)» Вінницького національного технічного університету.

Публікації. Комп’ютерна система для планування та контролю задач користувача //Крижанівський Д. В., Томчук М. А. Матеріали міжнародної конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)» (Вінниця, 2025 р.). На момент захисту бакалаврської кваліфікаційної роботи, дослідження знаходиться на етапі публікації.

1 ОГЛЯД ТА КЛАСИФІКАЦІЯ ІСНУЮЧИХ РІШЕНЬ ДЛЯ УПРАВЛІННЯ ЗАДАЧАМИ ТА ОРГАНІЗАЦІЇ РОБОЧОГО ПРОЦЕСУ

1.1 Поняття та роль інформаційних систем у сучасному суспільстві.

Інформаційні системи відіграють надзвичайно важливу роль у сучасному суспільстві, де ключовими ресурсами є знання, дані та ефективність управління ними. В умовах стрімкого розвитку технологій, збільшення обсягів інформації та зростаючих вимог до оперативності прийняття рішень, використання інформаційних систем стало невід’ємною складовою функціонування більшості організацій, установ та навіть окремих користувачів.

Під інформаційною системою зазвичай розуміють організовану сукупність апаратних, програмних, мережевих та людських компонентів, які забезпечують збирання, обробку, збереження, передавання та представлення інформації з метою підтримки процесів управління, аналізу й прийняття рішень. Це програмно-апаратний комплекс, який надає можливість системно та ефективно працювати з інформацією.

Інформаційні системи класифікуються за різними ознаками: за функціональним призначенням (управлінські, експертні, інформаційно-пошукові), за рівнем автоматизації (автоматизовані, інтегровані, інтелектуальні), за середовищем використання (корпоративні, персональні, мобільні) тощо. Залежно від цілей, які ставляться перед системою, її компоненти та структура можуть варіюватися, проте базовими елементами залишаються: джерело даних, оброблювальний модуль, база даних, інтерфейс користувача та канали обміну інформацією.

Станом на сьогодні, інформаційні системи охоплюють практично всі сфери діяльності людини: від фінансів і медицини до освіти, та побуту. Зокрема, у сфері персонального планування, інформаційні системи забезпечують зберігання й організацію особистої інформації, контроль за виконанням завдань, нагадування про важливі події, аналіз ефективності користувацької діяльності тощо.

Особливе місце інформаційні системи посідають у процесах автоматизації рутинних задач. Замість того, щоб вручну обробляти великі обсяги даних або вести

паперові записи, сучасний користувач має змогу застосовувати інтелектуальні інструменти для планування, контролю, обліку та взаємодії з іншими системами. Це не лише підвищує продуктивність, але й знижує кількість помилок, мінімізує людський фактор, а також сприяє швидкому доступу до необхідної інформації.

Разом з тим, значення інформаційних систем полягає не лише у спрощенні процесів — вони створюють передумови для ухвалення обґрунтованих рішень. Завдяки структурованій обробці даних, системи можуть надавати рекомендації, визначати пріоритети або навіть прогнозувати наслідки певних дій. Це набуває особливої актуальності в умовах динамічного середовища, де своєчасна реакція має вирішальне значення.

У сучасних умовах глобалізації та цифровізації, інформаційні системи виступають основою для формування так званого інформаційного суспільства — суспільства, в якому переважаючим ресурсом є саме інформація. У цьому контексті інформаційні системи стають базовим інструментом для забезпечення сталого розвитку, конкурентоспроможності та стратегічного управління як на рівні окремих організацій, так і держав в цілому.

Із поширенням концепцій “розумного офісу” та “персональної ефективності” особливого значення набувають персоналізовані інформаційні системи, орієнтовані на індивідуального користувача. Це можуть бути застосунки для керування часом (time-management), таск-трекери, системи нагадувань, цифрові щоденники, а також більш комплексні платформи, які дозволяють формувати та оптимізувати плани дій, контролювати досягнення поставлених цілей, враховувати навантаження та розподіляти ресурси.

Окрім утилітарної ролі, такі системи впливають і на формування поведінки користувача, спонукаючи до впорядкованості, плановості, відповідальності у прийнятті рішень. Наприклад, наявність автоматичного нагадування про важливу подію сприяє зменшенню ризику її пропуску, а система пріоритетів дозволяє не загубити головне серед другорядного. В результаті, інформаційна система не просто виконує функцію записника, а фактично виступає як «цифровий помічник».

Ще одним важливим аспектом розвитку інформаційних систем є їх інтеграція

з іншими технологіями, зокрема зі штучним інтелектом, машинним навчанням та аналітичними модулями. Це дозволяє не лише реагувати на дії користувача, а й практично передбачати його потреби, пропонуючи оптимальні рішення або виявляючи неефективні шаблони поведінки. Такі можливості вже реалізуються у корпоративних CRM-системах, цифрових асистентах, а також у мобільних планувальниках нового покоління.

Крім того, сучасні інформаційні системи дедалі частіше реалізуються з урахуванням модульності та масштабованості, що дозволяє адаптувати їх до змін у середовищі користувача. Наприклад, невелика система персонального планування може з часом розширюватися до багатофункціонального середовища, яке підтримує командну взаємодію, календар подій, облік витрат, інтеграцію з зовнішніми сервісами (електронна пошта, календарі, меседжери тощо). Це відкриває широкі перспективи для створення адаптивних рішень, орієнтованих на конкретні потреби користувача.

З технологічного погляду, широке впровадження мобільних пристроїв, хмарних сервісів, штучного інтелекту та Big Data ще більше підсилює можливості інформаційної системи. Наприклад, сучасні програми можуть здійснювати синхронізацію між пристроями, автоматично сортувати завдання, аналізувати поведінку користувача, пропонуючи оптимальні сценарії дій. Такі функції, з одного боку, підвищують зручність, а з іншого — вимагають від розробників підвищеної уваги до безпеки даних, інтерфейсу користувача та надійності системи в цілому.

Усе вищезазначене свідчить про надзвичайно важливу роль інформаційних систем у повсякденному житті сучасної людини. Вони не лише виконують допоміжні функції, а стають невід’ємною частиною персональної та професійної ефективності. Саме тому вивчення, аналіз та подальша розробка нових, більш досконалих ІС є актуальним завданням як у науково-дослідницькій, так і в практичній площині.

1.2 Класифікація та структура інформаційних систем

Інформаційні системи, як складні організаційно-технічні утворення,

підлягають класифікації за різними критеріями. Такий підхід дозволяє глибше аналізувати особливості інформаційних систем, визначати їх призначення, структуру та умови використання. Класифікація також відіграє ключову роль у виборі відповідних технічних та програмних рішень при розробці нових систем або вдосконаленні вже існуючих.

Одним з основних критеріїв є сфера застосування інформаційної системи. У цьому контексті інформаційні системи поділяються на такі типи, як: комерційні (використовуються для автоматизації бізнес-процесів), медичні (електронні картки пацієнтів, системи моніторингу), освітні (електронні журнали, системи дистанційного навчання), побутові (персональні планувальники, календарі, замітки) тощо. Відповідно, структура, функціональність та інтерфейс інформаційних систем тісно пов’язані з особливостями предметної області, яку вони обслуговують.

Інший важливий критерій — ступінь автоматизації процесів. За цим параметром системи можуть бути:

— ручні, де всі дії виконує людина без технічної підтримки;

— автоматизовані, де частина операція виконується за допомогою технічних засобів або програм;

— автоматичні, де майже повністю виключають участь людини (наприклад, системи обліку або автоматичні системи сповіщення).

Ще одним критерієм є кількість користувачів і масштаб використання. Тут можна виділити:

— індивідуальні інформаційні системи, які використовуються одним користувачем на одному пристрої;

— локальні мережеві системи, призначені для колективного використання в межах локальної мережі;

— корпоративні системи, охоплюють кілька структурних підрозділів, часто інтегровані з іншими сервісами (наприклад, ERP-системи) [2].

Крім того, інформаційні системи класифікують за типом обробки даних:

— оперативні системи, обробляють інформацію в режимі реального часу

(наприклад, банківські транзакції);

— аналітичні системи — використовуються для аналізу накопичених даних;

— інтелектуальні системи — застосовують елементи штучного інтелекту та логічного виводу.

Більш детальний огляд класифікаційних підходів до інформаційних систем наведено у таблиці 1.1, де узагальнено основні критерії поділу, типові приклади та особливості застосування систем у різних контекстах.

Таблиця 1.1 — Класифікація інформаційних систем за різними критеріями

Критерій	Приклади класифікації	Опис
За сферою застосування	Бізнес, медицина, освіта, побут	Визначає, де саме використовується ІС
За ступенем автоматизації	Автоматизовані, напівавтоматизовані	Відображає участь людини у процесах
За кількістю користувачів	Персональні, локальні, корпоративні	Масштаб і цільова аудиторія
За типом обробки даних	Оперативні, аналітичні, інтелектуальні	Функціональні можливості системи

Окрім класифікації, важливим аспектом вивчення інформаційних систем є аналіз їх внутрішньої структури. Незалежно від сфери застосування чи рівня складності, будь-яка інформаційна система складається з низки функціональних компонентів, які взаємодіють між собою з метою забезпечення основних інформаційних процесів: збирання, обробки, зберігання, передавання та використання даних.

До основних компонентів інформаційної системи традиційно належать:

— інформаційні ресурси, сукупність вхідних даних, що використовуються в системі. Це можуть бути як первинні, так і оброблені дані, які надходять із зовнішніх джерел або формуються в процесі функціонування самої системи;

— апаратне забезпечення, це фізичні пристрої (сервери, клієнтські машини, мобільні гаджети тощо), що забезпечують виконання програмного коду та обробку

інформації;

— програмне забезпечення, це набір інструкцій, що керують апаратною частиною і реалізують логіку функціонування інформаційної системи. Включає операційні системи, бази даних, прикладні модулі, інтерфейс користувача та інші компоненти;

— база даних, структуроване сховище інформації, що дозволяє швидко зберігати, шукати, змінювати та аналізувати дані. Важливим елементом є система управління базами даних (СУБД), яка забезпечує доступ до БД та підтримує її цілісність;

— користувацький інтерфейс, засоби взаємодії користувача з системою: графічний інтерфейс, команди, меню, елементи управління тощо. Від його зручності значною мірою залежить ефективність користування інформаційною системою.

— комунікаційні засоби, канали та протоколи передавання даних між компонентами ІС, особливо в умовах мережевого середовища, забезпечують обмін інформацією між локальними і віддаленими модулями системи.

Сукупність вказаних елементів формує архітектуру інформаційної системи, яка може бути централізованою (усі компоненти на одному сервері), децентралізованою (розподілені по кількох вузлах), або гібридною. Сучасні системи, особливо в хмарному середовищі, часто реалізуються як модульні, що дозволяє масштабувати функціонал без зміни базової структури.

Загалом, структура інформаційної системи визначає її продуктивність, стабільність, надійність і масштабованість. Від правильної організації компонентів залежить здатність системи адаптуватися до нових вимог, оновлюватися, підтримувати інтеграцію з іншими сервісами, а також зберігати безперебійність роботи при зростанні навантаження.

1.3 Програмні засоби для організації діяльності користувача

В умовах стрімкого розвитку цифрових технологій та зростаючого обсягу інформаційного навантаження, сучасні користувачі дедалі частіше звертаються до

спеціалізованих програмних засобів, які дозволяють ефективно організовувати особисту чи професійну діяльність. До таких засобів належать програмні додатки для планування завдань, керування роботами, контролю дедлайнів, зберігання нотаток та інтеграції з іншими цифровими сервісами.

Загалом, ці інструменти можна умовно поділити на кілька категорій:

— застосунки для персонального планування (Microsoft To Do, Todoist);

— засоби для візуального управління задачами (Trello, Kanban-системи);

— багатофункціональні цифрові органайзери (Notion, ClickUp);

— календарні та нагадувальні сервіси (Google Calendar, Any.do);

— корпоративні інструменти (Asana, Jira, Monday).

Одним із найпопулярніших засобів персонального планування є Microsoft To Do — багатофункціональний, але водночас простий у використанні застосунок для керування особистими завданнями. Його основна перевага — гнучкість в організації задач: користувач може створювати кілька списків (наприклад, «Робота», «Особисте», «Навчання»), додавати до них задачі, встановлювати дати виконання, пріоритети, а також нагадування.

Застосунок підтримує повторювані завдання, інтелектуальні підказки (My Day — система рекомендацій задач на день), прикріплення файлів, а також можливість додавати нотатки до кожної задачі. Завдяки повній інтеграції з Microsoft Outlook і Microsoft 365, задачі з календаря Outlook автоматично синхронізуються із застосунком. Крім того, Microsoft To Do підтримує спільну роботу: список задач можна надіслати іншому користувачу або використовувати у командній роботі. Програма доступна на Windows, Android, iOS і у веб-версії, з автоматичною синхронізацією між усіма пристроями. Інтерфейс застосунку лаконічний, зручний і повністю адаптивний під різні роздільні здатності екранів [4].

На рисунку 1.1 зображено інтерфейс програми Microsoft To Do, зокрема головне вікно із вкладками завдань та прикладом активного списку задач користувача.

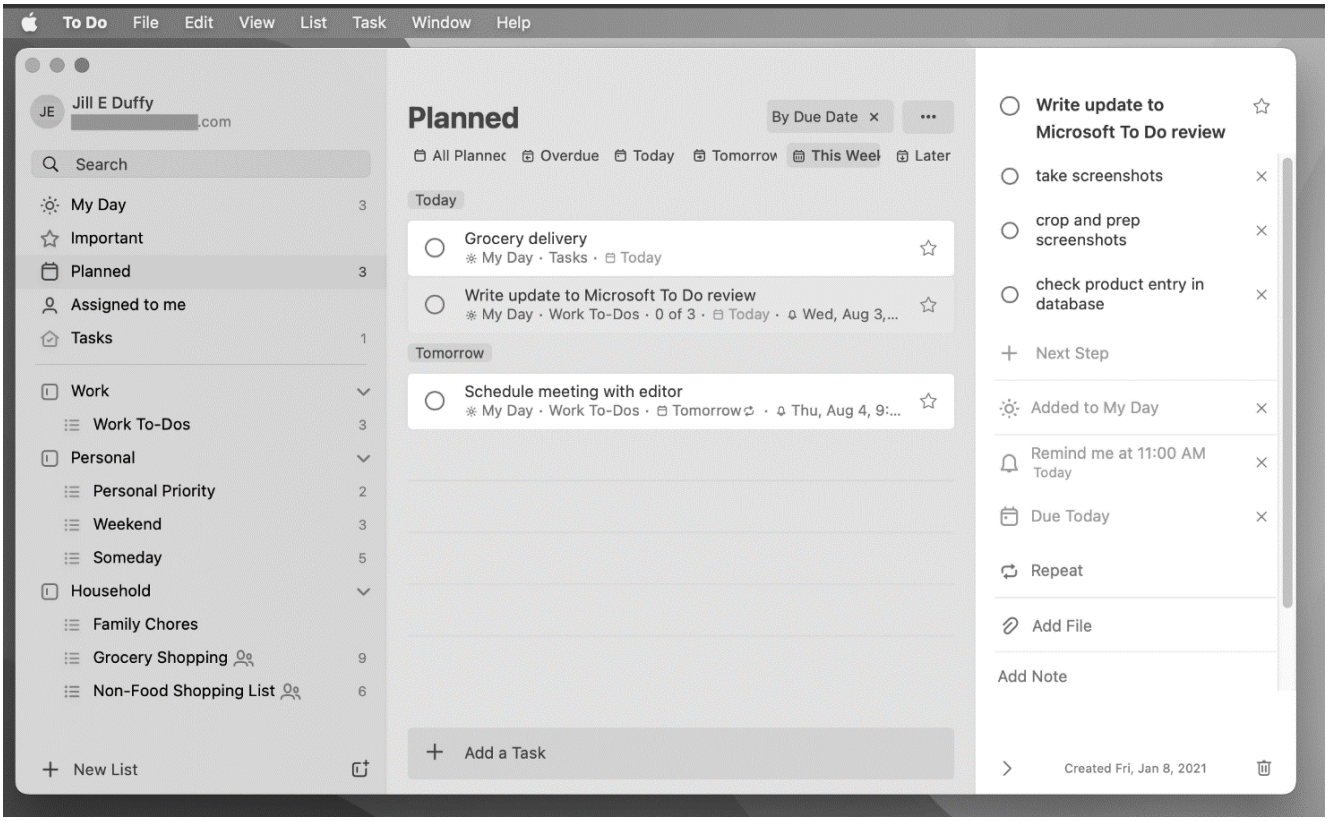


Рисунок 1.1 — Інтерфейс програми Microsoft To Do

Ще один потужний інструмент, який набув популярності серед як окремих користувачів, так і команд, — це Trello. Це веб-застосунок, який реалізує принцип Kanban-системи — метод візуального управління завданнями. Основою інтерфейсу Trello є дошка, що складається з колонок (наприклад, «Очікує на виконання», «У роботі», «Готово»), в які користувач розміщує картки-завдання.

Кожна картка є окремим завданням і може містити опис, дедлайн, чек-листи, прикріплені файли, мітки, а також коментарі учасників. Завдання можна перетягувати мишкою між колонками — це дозволяє в реальному часі змінювати статус задачі. Завдяки такому підходу, користувач бачить повну картину прогресу у виконанні всіх поточних справ.

Trello підтримує спільну роботу, дозволяє додавати учасників до конкретних задач, створювати спільні дошки для команд, організацій або проєктів. Додаткові функції, такі як автоматизація (через Butler), інтеграція з Slack, Google Drive та іншими сервісами, роблять Trello ефективним засобом не лише для особистого, а й для корпоративного використання.

На рисунку 1.2 представлено фрагмент дошки Trello з прикладом структури колонок і карток, що ілюструє процес управління задачами у форматі Kanban.

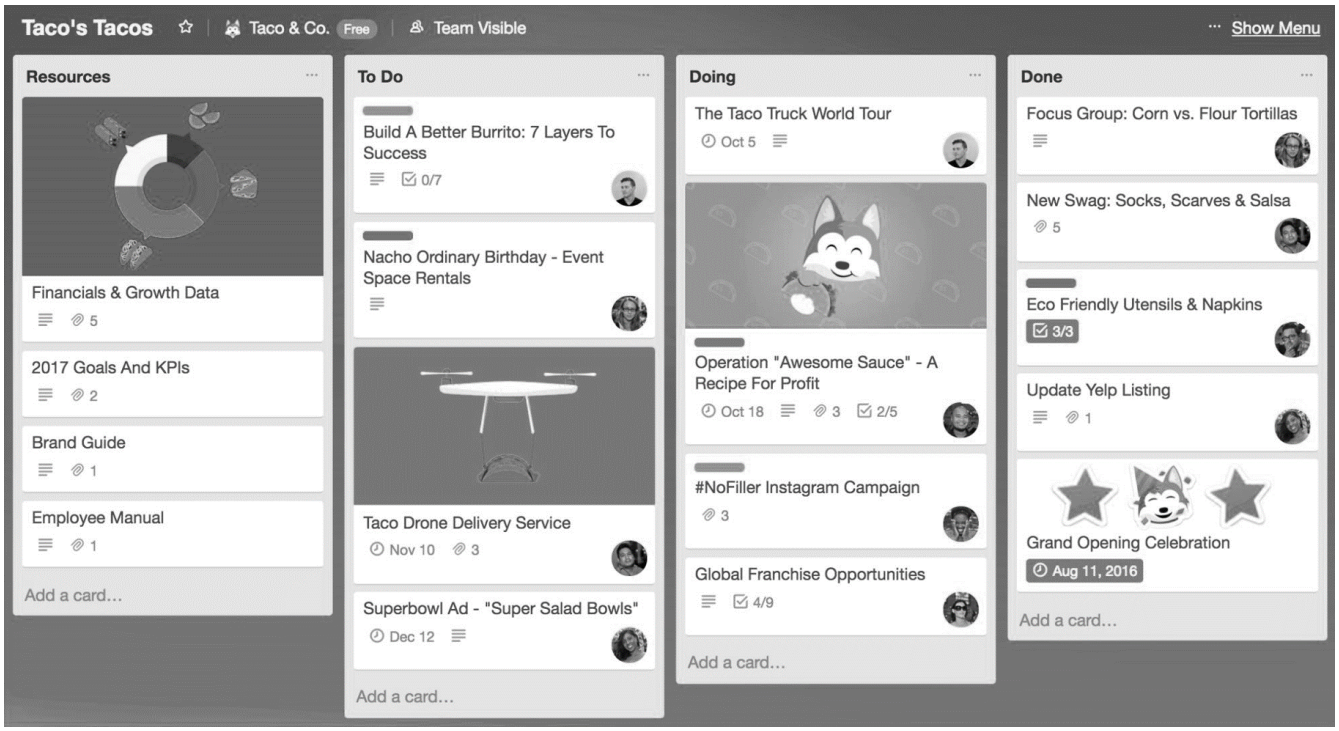


Рисунок 1.2 — Фрагмент дошки веб-застосунку Trello

Одним із найбільш універсальних і водночас функціонально гнучких інструментів для управління особистими та робочими завданнями є Todoist. Цей застосунок поєднує простоту створення задач із розширеними можливостями для їхньої організації, контролю та аналітики. Застосунок підтримує створення проєктів, задач і підзадач, використання тегів (ярликів), пріоритетів, дедлайнів і повторюваних завдань.

Todoist дозволяє групувати задачі у проєкти, кожен з яких може містити субзадачі або вкладені структури. Наприклад, у межах проєкту "Навчання" можуть бути задачі "Підготувати презентацію", "Здати курсову" тощо. До кожної задачі можна додати опис, коментарі, мітки, прикріпити файли. Також реалізована інтеграція з календарем Google, а в преміум-версії — з Outlook, Slack, Dropbox тощо.

Користувач має змогу фільтрувати задачі за різними параметрами (дата, пріоритет, тег), переглядати завдання у вигляді списку, таймлайну або календаря.

Система також відображає графік продуктивності — аналітику виконаних задач за тиждень чи місяць, а також мотивує користувача підтримувати серію виконань (так звані streaks).

Ще однією перевагою Todoist є підтримка розширеного синтаксису при введенні задач. Інтерфейс Todoist простий, структурований та адаптивний. Програма доступна як для ПК (Windows, macOS), так і для мобільних платформ (Android, iOS), а також у веб-версії та як розширення для браузера.

Todoist також має систему нагород та рівнів для користувача, яка ґрунтується на активності та регулярності виконання задач. Кожна виконана дія приносить бали продуктивності, що мотивує користувача до постійного саморозвитку. Крім того, застосунок підтримує шаблони проєктів, що дозволяє швидко запускати повторювані типи завдань (наприклад, щотижневе планування або робочі спринти).

На рисунку 1.3 зображено інтерфейс застосунку Todoist з прикладом списку задач користувача, фільтрації по тегу та графіком продуктивності.

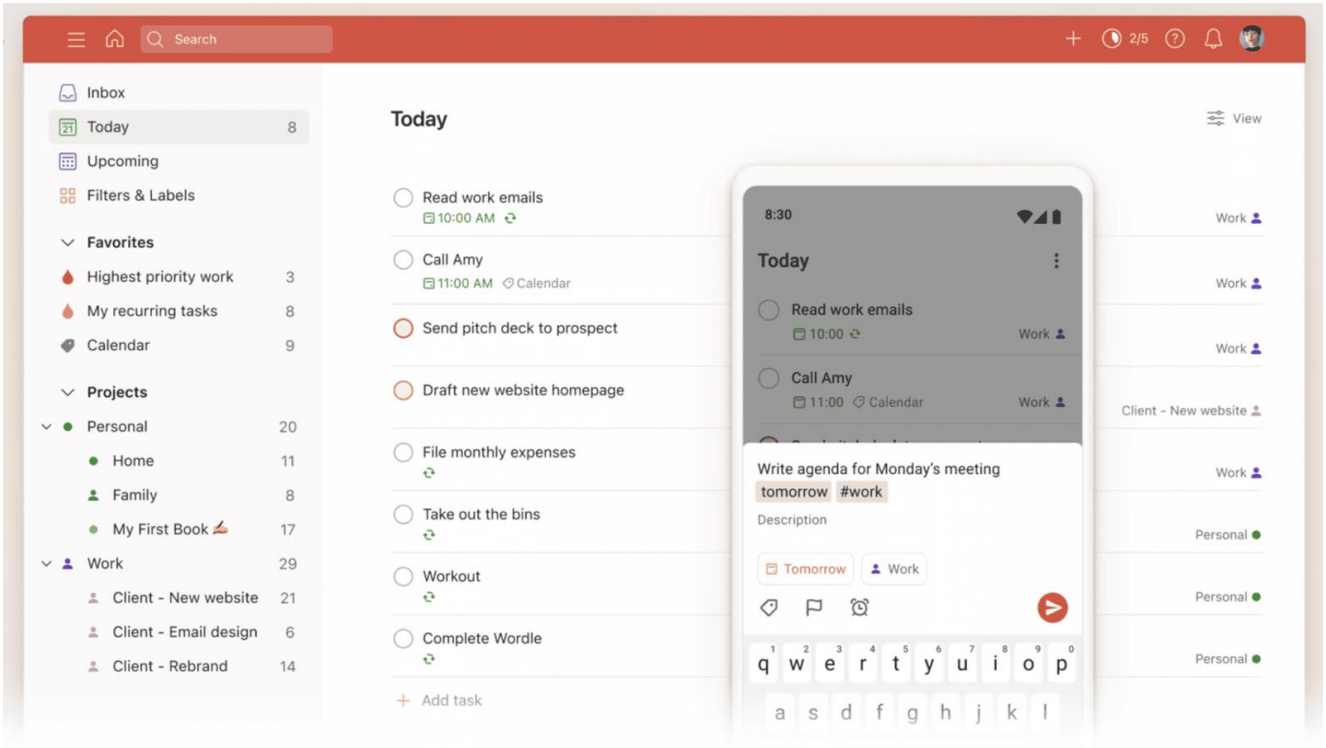


Рисунок 1.3 — Інтерфейс застосунку Todoist

Google Calendar — це потужний календарний сервіс, який став одним із найпопулярніших інструментів для організації особистого та професійного часу.

Його головна перевага полягає в тісній інтеграції з іншими сервісами Google: Gmail, Google Meet, Google Tasks, а також сторонніми платформами, такими як Zoom, Slack, Trello тощо. Завдяки цьому Google Calendar стає центром цифрового планування, що охоплює як особисті події, так і робочі зустрічі.

Користувачі можуть створювати події із зазначенням дати, часу, тривалості, місця проведення, списку учасників і навіть прикріплювати файли. Кожну подію можна налаштувати як одноразову або повторювану, додати кольорову позначку для візуальної ідентифікації, встановити нагадування, або автоматично надсилати запрошення учасникам на електронну пошту.

Однією з ключових функцій є підтримка кількох календарів одночасно, які можуть бути прив’язані до різних облікових записів або поділені за тематикою (робота, навчання, особисте). Календарі можна відображати одночасно, а також приховувати або показувати окремо. Google Calendar підтримує гнучке відображення подій — день, тиждень, місяць, розклад або план на 4 дні.

Серед додаткових можливостей — інтеграція з Google Tasks (для перегляду списку завдань поруч із календарем), використання цілей (Goals) для особистого розвитку (наприклад, "читати 3 рази на тиждень"), а також додавання нагадувань (Reminders), які залишаються в календарі доти, поки їх не буде виконано або видалено користувачем.

Сервіс доступний у веб-версії, а також як окремий застосунок для Android та iOS. Усі події синхронізуються між пристроями в режимі реального часу. Інтерфейс Google Calendar мінімалістичний, інтуїтивно зрозумілий, із широкими можливостями кастомізації.

Google Calendar також дозволяє надавати спільний доступ до окремих календарів із правами перегляду або редагування, що особливо зручно для командної роботи. Крім того, сервіс має функцію автоматичного додавання подій із Gmail, наприклад, бронювання квитків, зустрічей чи нагадувань про подорожі.

На рисунку 1.4 наведено інтерфейс Google Calendar з прикладом подій, доданих до календаря користувача, та можливостями перегляду розкладу в різних форматах.

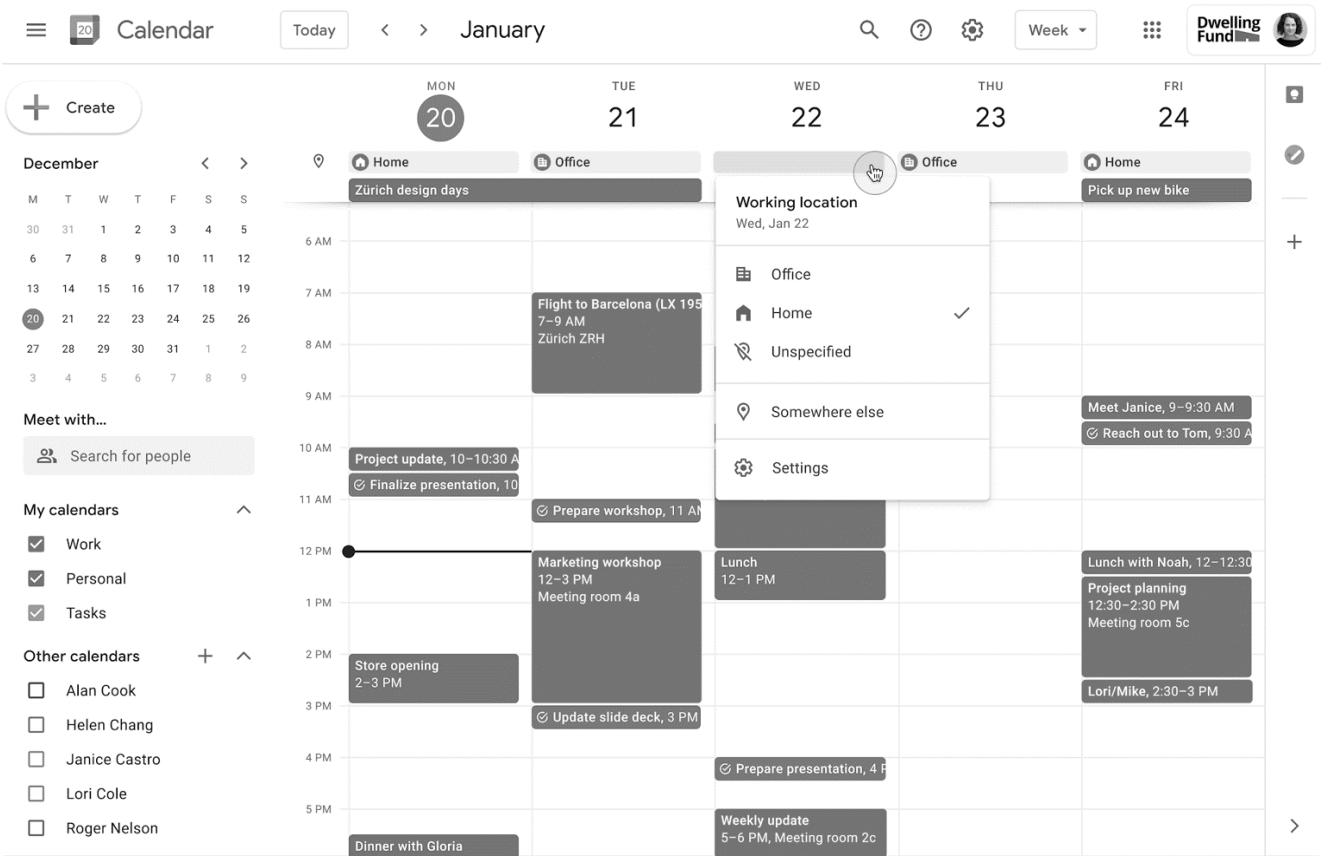


Рисунок 1.4 — Інтерфейс Google Calendar

Notion — це універсальна платформа для створення, зберігання та організації інформації, яка поєднує функціональність нотатника, бази даних, таск-менеджера, вікі-системи та інструмента для спільної роботи. Завдяки своїй гнучкості, Notion набув широкого поширення серед студентів, фрілансерів, IT-команд, проєктних менеджерів та навіть цілих компаній.

Ключовим елементом платформи є "блокова система", де кожен елемент сторінки (текст, заголовок, список, таблиця, база даних, кнопка, відео тощо) — це окремий блок, який можна переміщувати, копіювати, вкладати один в інший або перетворювати. Завдяки цьому користувач може створювати повноцінні структуровані документи, бази знань, чек-листи, особисті щоденники тощо.

Notion підтримує різні типи представлення інформації: списки, таблиці, календарі, дошки у стилі Kanban, галереї, вкладені бази даних. Усі ці форми легко налаштовуються, фільтруються, сортуються за потрібними критеріями та дозволяють створювати взаємозв’язані структури — наприклад, список завдань,

пов'язаний із календарем і базою контактів.

Платформа доступна в браузері, як десктопна програма, а також як мобільний застосунок. Усі дані автоматично синхронізуються між пристроями. Попри величезну кількість функцій, інтерфейс Notion залишається інтуїтивно зрозумілим, а можливість створювати власні шаблони дозволяє налаштувати систему під себе. Notion також активно використовується в освітньому середовищі: студенти створюють власні бази конспектів, розклади, фінансові трекери та щоденники.

У свою чергу, викладачі та наставники використовують платформу для ведення навчальних матеріалів, спільного доступу до файлів та організації командної роботи над проєктами. Завдяки підтримці шаблонів та швидкому дублюванню сторінок, користувачі можуть легко створити повноцінну систему самоменеджменту без необхідності програмування чи складних налаштувань.

На рисунку 1.5 зображено інтерфейс платформи Notion з прикладом комбінованої структури сторінки, що містить список завдань, календар і базу даних.

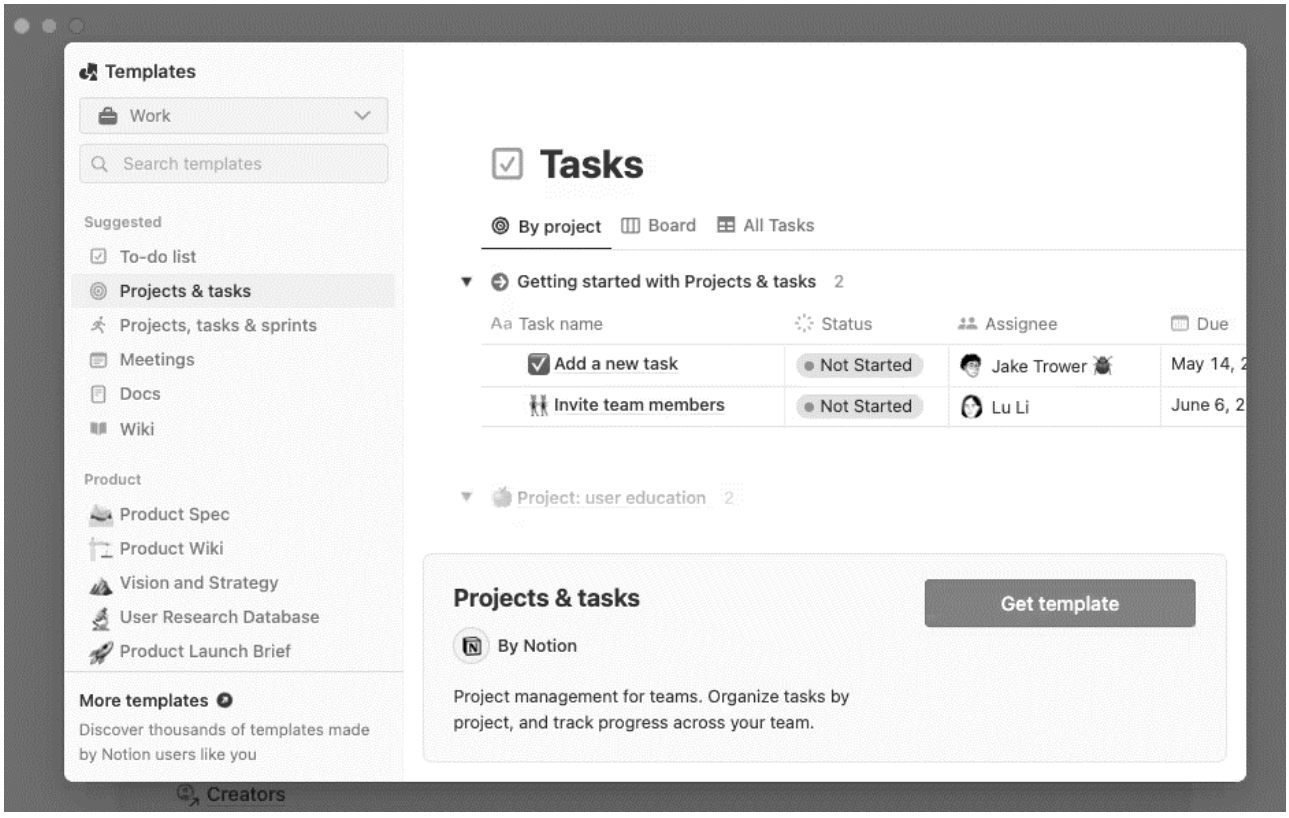


Рисунок 1.5 — Інтерфейс платформи Notion

Таким чином, сучасні програмні засоби для організації діяльності користувача пропонують широкий спектр функцій — від базових списків справ до комплексних систем управління завданнями, календарями та інформаційними блоками. Кожен із розглянутих застосунків — Microsoft To Do, Trello, Todoist, Google Calendar та Notion — має свої особливості, орієнтовані на різні категорії користувачів та сценарії використання. Їхня гнучкість, доступність та інтеграція з іншими сервісами значно підвищують ефективність роботи, що підтверджує доцільність використання таких інструментів як у професійній, так і в особистій діяльності [14].

1.4 Поняття та моделі управління завданнями

Управління завданнями — це процес планування, контролю та координації виконання конкретних дій, спрямованих на досягнення поставлених цілей. Цей процес охоплює низку функцій: постановку задач, визначення термінів і пріоритетів, розподіл відповідальності, моніторинг прогресу та оцінку результатів. Ефективне управління завданнями дозволяє оптимізувати використання часу, ресурсів та зменшити кількість помилок при виконанні повсякденних чи робочих справ.

Залежно від контексту, управління задачами може здійснюватись як у ручному режимі (за допомогою списків або нотаток), так і за допомогою спеціалізованих програмних рішень, що автоматизують ключові етапи. Важливу роль у цьому процесі відіграють обрані моделі управління, які задають загальний підхід до організації робочого процесу.

Значною перевагою сучасних моделей управління задачами є їх адаптивність до змін середовища та потреб користувача. У практиці планування дедалі частіше використовуються гібридні підходи, які поєднують елементи з різних методологій. Наприклад, система GTD може бути реалізована в середовищі Kanban, або Scrum може доповнюватись елементами візуального контролю через дошку завдань. Такі комбінації дозволяють створювати індивідуально налаштовані підходи до продуктивності, що особливо важливо у сфері IT, фрилансу та навчання.

На практиці найбільш поширеними є такі моделі:

— GTD (Getting Things Done), система, орієнтована на вивільнення уваги користувача, шляхом фіксації усіх завдань та подальшої їхньої організації;

— Kanban, візуальний підхід із розподілом задач за статусами, що дозволяє контролювати потік роботи;

— Agile / Scrum, інкрементальні підходи, характерні для командної роботи, особливо в IT-проєктах.

Процес управління завданнями умовно поділяється на кілька етапів:

— планування — встановлення цілей, дедлайнів, підзадач;

— виконання — реалізація завдання, комунікація;

— моніторинг — відстеження прогресу, контроль термінів;

— завершення — фіксація результату, аналіз ефективності.

У таблиці 1.2 наведено порівняльну характеристику найбільш поширених моделей управління завданнями відповідно до ключових параметрів.

Таблиця 1.2 — Порівняння моделей управління завданнями

Критерій	GTD	Kanban	Scrum
Основна ідея	Звільнення розуму від задач через запис	Візуальне управління потоком задач	Інкрементальна робота за спринтами
Структура задач	Списки, контексти	Дошка з колонками	Беклог, спринт, таски
Командна робота	Не передбачена	Частково	Основна ціль
Контроль прогресу	Огляд списків, щоденне планування	Візуальний статус задач	Щоденні стендапи, Review
Гнучкість	Висока	Висока	Середня
Підходить для індивідуального користувача	Так	Так	Обмежено

Крім загальних принципів, кожна модель передбачає власний підхід до

визначення пріоритету, що відіграє вирішальну роль при обмеженому часі або ресурсах. Наприклад, у GTD пріоритет визначається контекстом та енергією, доступною користувачу, у Kanban — завантаженістю колонок, а у Scrum — порядком задач у «беклозі», визначеним на етапі планування спринту. Це означає, що ефективне управління вимагає не лише інструменту, а й правильного розуміння логіки його застосування.

Окрему увагу слід приділити психологічним аспектам використання моделей. Наприклад, Kanban із чітким візуальним відображенням прогресу допомагає відчувати мотивацію, оскільки користувач бачить, як задачі переходять у стовпець «Виконано». У свою чергу, GTD знижує стрес, викликаний забутими завданнями, оскільки забезпечує повне «вивантаження» справ із голови у систему. Scrum, зі своєю командною динамікою, сприяє розвитку відповідальності та взаємодії між учасниками.

Розгляд моделей управління завданнями демонструє, що організація діяльності не є одноманітним процесом, а базується на виборі відповідної методології з урахуванням особистих або командних потреб. Кожна з моделей — GTD, Kanban, Scrum — пропонує власну логіку структурування роботи, яка формує різні підходи до планування, контролю та досягнення результатів. Здатність адаптувати ці підходи під конкретну ситуацію є ключем до підвищення ефективності як в індивідуальному, так і в колективному контексті.

Таким чином, моделі управління завданнями не є універсальними рішеннями, проте вони створюють чітку рамку, всередині якої користувач може структурувати свою діяльність, відстежувати прогрес і досягати поставлених цілей більш системно. Їх правильне використання — це не лише про ефективність, а й про якість життя, розумову гігієну та збереження уваги.

1.5 Системи автоматизації планування, огляд сучасних рішень

У сучасному цифровому середовищі зростає потреба в ефективному управлінні часом, ресурсами та процесами. Для цього все ширше впроваджуються системи автоматизації планування — програмні комплекси, що дозволяють не

лише фіксувати задачі, а й комплексно керувати робочими процесами, відстежувати дедлайни, координувати роботу команд, будувати розклади та формувати звітність.

Такі системи активно застосовуються у бізнесі, логістиці, IT, маркетингу, освіті та управлінні проєктами. На відміну від простих таск-менеджерів, вони мають значно ширший функціонал — інтеграцію з іншими сервісами, аналітику, шаблони, автоматизацію процесів, систему ролей, командну роботу та навіть зв’язок із фінансами або документообігом.

Системи автоматизації планування умовно можна поділити на:

— індивідуальні, для фрілансерів, викладачів, особистого планування (наприклад, Notion, Todoist);

— командні — для малих і середніх команд, відділів або навчальних груп (ClickUp, Asana);

— корпоративні — з комплексною структурою, ролями, інтеграцією у великі бізнес-процеси (Jira, Microsoft Project, SAP, Bitrix24).

ClickUp — одна з найбільш гнучких сучасних платформ для автоматизації планування, яка поєднує в собі функції таск-менеджера, CRM, системи управління знаннями та календаря. Ця система особливо популярна серед команд, які хочуть об’єднати кілька сервісів в один, щоб уникнути розпорошеності задач і зберігати всю інформацію в єдиному робочому просторі.

Основною одиницею в ClickUp є завдання (task), які можна групувати в списки, папки та простори (spaces). До кожного завдання додаються описи, підзадачі, контрольні списки, теги, дедлайни, пріоритети, файли, а також коментарі з підтримкою тегів користувачів. ClickUp підтримує кілька варіантів представлення задач: списки, дошки, календарі, діаграми Ґанта, таблиці — що дозволяє кожному користувачу адаптувати інтерфейс під свої потреби.

Крім цього, платформа пропонує інструменти автоматизації, шаблони проєктів, аналітичні панелі (dashboard) та можливість створювати форми, що інтегруються із задачами. Також реалізована підтримка цілей — користувачі можуть задавати мету, розбивати її на мікрозадачі та відстежувати прогрес.

ClickUp також відзначається активною підтримкою інтеграцій із популярними інструментами, такими як Slack, Zoom, Google Calendar, Microsoft Teams, GitHub та багатьма іншими. Це дозволяє безперешкодно включити систему у вже існуючі бізнес-процеси. Крім того, користувачі мають змогу створювати глобальні звіти, які відображають стан проєктів у реальному часі. Система також підтримує контроль робочого часу, що дозволяє відстежувати витрати часу на конкретні задачі та формувати звіти по ефективності. Завдяки багаторівневій структурі доступів.

ClickUp доступний у вебі, на десктопі та мобільних пристроях. Його інтерфейс адаптивний, з потужною системою кастомізації, де кожен учасник команди може обрати зручне представлення даних, не змінюючи при цьому робочу область інших.

На рисунку 1.6 зображено інтерфейс системи ClickUp із прикладом використання дошки задач у форматі Kanban та відображенням інформації про виконання.

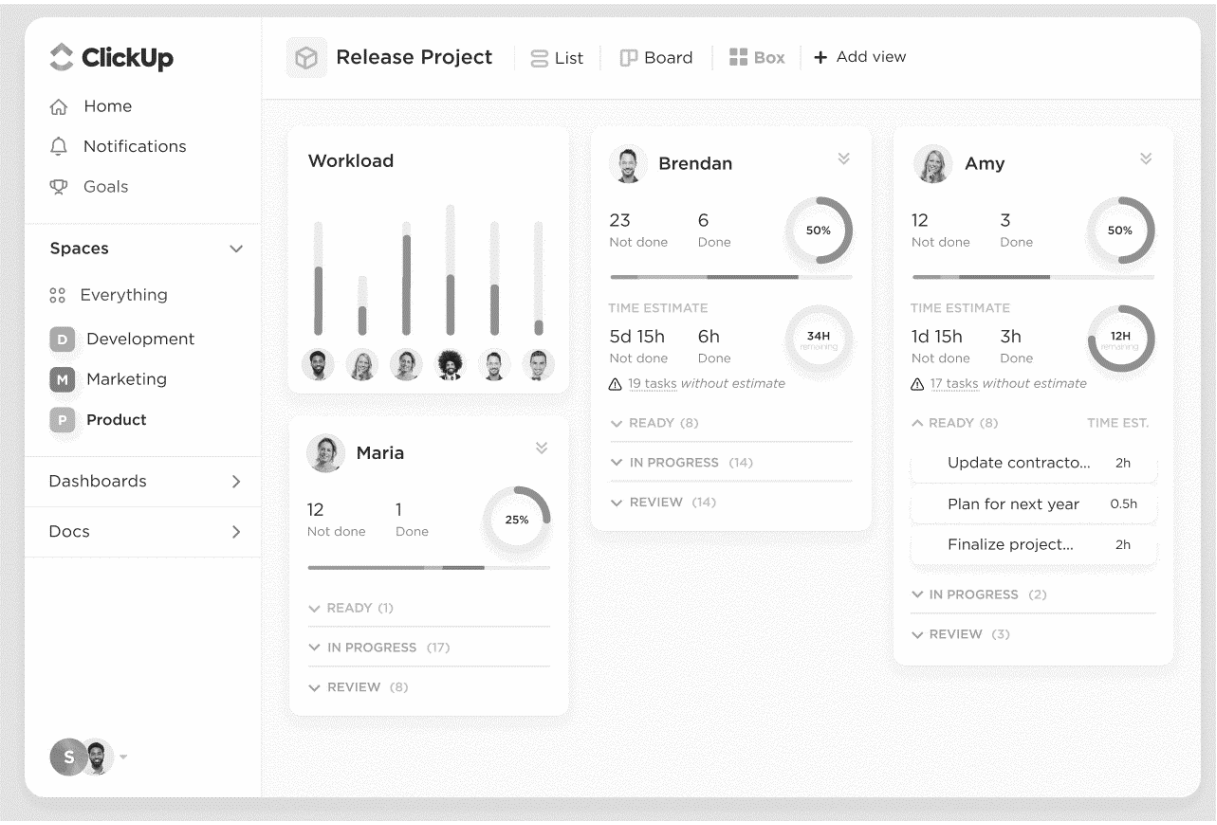


Рисунок 1.6 — Інтерфейс системи ClickUp

Asana — це потужна система для управління проєктами й задачами, яка спеціалізується на організації командної роботи. Платформа орієнтована на гнучке планування, контроль термінів і забезпечення прозорості в процесі виконання завдань. Основною метою Asana є полегшення координації між членами команди, особливо у великих або розподілених колективах.

Кожне завдання в Asana можна детально описати, прикріпити до нього файли, встановити дедлайн, пріоритет, обрати відповідального виконавця та додати залежності (тобто завдання, які мають бути виконані до або після). Задачі можна структурувати в проєкти, які, своєю чергою, можуть мати кілька режимів відображення: список, дошка (Kanban), календар або часову шкалу (Timeline), схожу на діаграму Ґанта. Це дозволяє візуалізувати планування на різних рівнях складності.

Asana також має вбудовану систему повідомлень, що дозволяє учасникам швидко отримувати інформацію про зміни в завданнях, коментарі, дедлайни тощо. Завдяки інтеграції з електронною поштою, Slack, Google Drive, Zoom та десятками інших сервісів, Asana легко вбудовується у вже існуючі цифрові середовища компаній.

Asana підтримує рольове управління доступом, що дозволяє встановлювати права для різних учасників роботи — від перегляду до повного редагування. У системі також передбачено аналітичні звіти та дашборди, які відображають продуктивність команди та стан задач у реальному часі. Мобільні додатки для Android та iOS забезпечують повноцінну роботу з будь-якого пристрою. Asana має інструменти для масштабування — від персонального трекінгу до керування складними роботами великих компаній. Інтерфейс платформи адаптивний і дозволяє кастомізувати вигляд проєктів під специфіку роботи команди.

Особливістю Asana є "Rules" — система автоматизації, яка дозволяє створювати сценарії для автоматичної зміни статусу задач, переміщення між розділами, призначення виконавців та багато іншого. Наприклад, задача може автоматично переходити в іншу колонку після завершення підзадач, або змінювати виконавця при зміні тегу.

На рисунку 1.7 зображено інтерфейс Asana з прикладом організації проєкту у форматі Kanban-дошки, із зазначенням статусів та виконавців завдань.

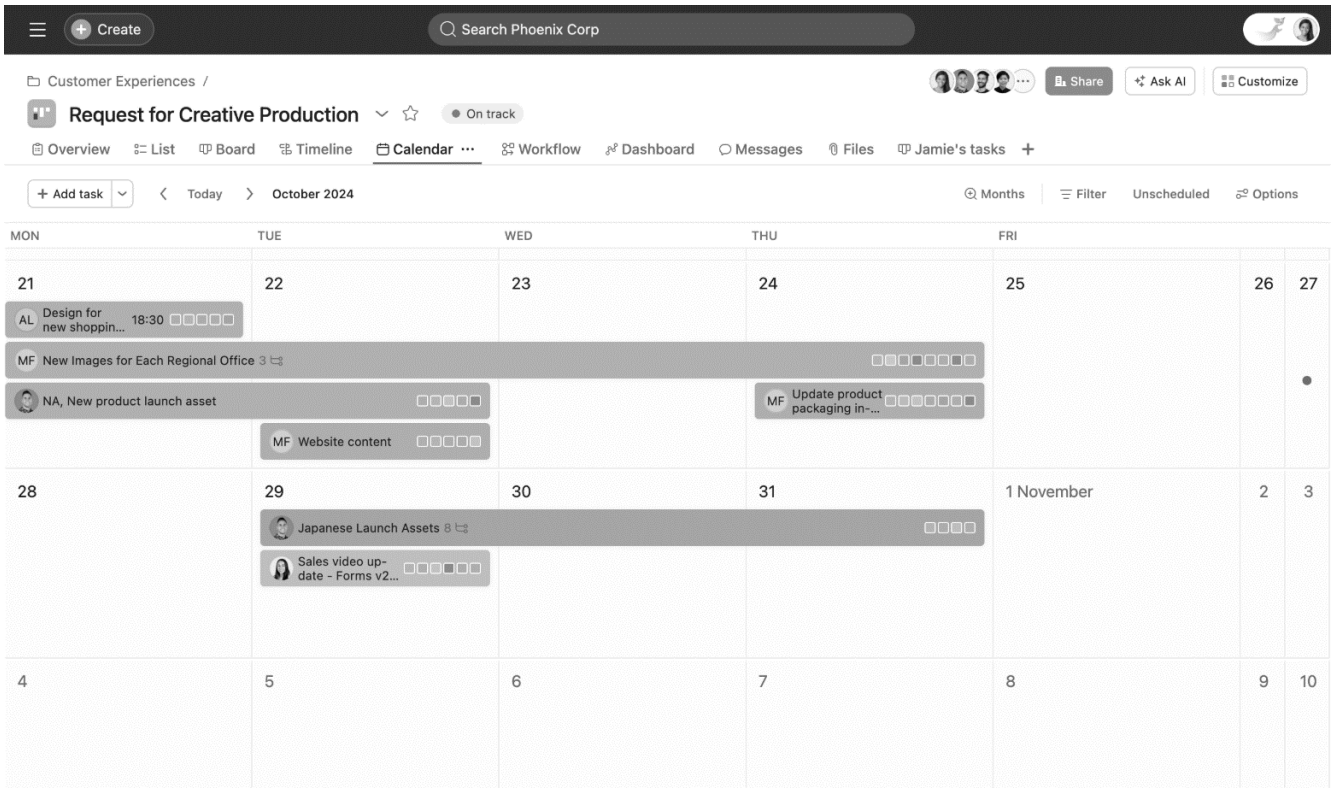


Рисунок 1.7 — Інтерфейс застосунку Asana

Jira — це професійна система управління проєктами, розроблена компанією Atlassian, яка широко використовується в IT-компаніях для організації розробки програмного забезпечення. Вона дозволяє планувати, відстежувати та аналізувати весь життєвий цикл задач — від постановки технічного завдання до його реалізації та тестування. Jira ідеально підходить для реалізації гнучких методологій управління, зокрема Scrum та Kanban.

У центрі системи — так звані "issues" (записи), які можуть представляти задачі, помилки, підзадачі або технічні запити. Кожен елемент має розширені атрибути: опис, категорію, пріоритет, призначення виконавця, дедлайни, теги, статуси, залежності та історію змін. Завдяки цьому Jira забезпечує повну прозорість і контроль над усіма активностями в межах команди.

Однією з найсильніших сторін Jira є широкий вибір налаштувань. Користувачі можуть створювати власні поля, фільтри, шаблони, а також візуальні

дошки для кожного проєкту. Scrum-дошки дозволяють проводити спринти, планування, оцінку задач, а Kanban-дошки — організовувати потік без жорсткої ітераційної структури. Додатково Jira підтримує епіки, story points, беклоги, roadmaps — усе, що потрібно для серйозного проєктного менеджменту.

Система інтегрується з Bitbucket, Confluence, GitHub, Jenkins, і багатьма іншими сервісами, що робить її надзвичайно зручною для технічних команд. Також доступна потужна система звітності: діаграми Ґанта, burn-down графіки, velocity charts та різноманітна аналітика, яка дозволяє оцінювати ефективність спринтів, завантаженість виконавців і динаміку виконання задач.

На рисунку 1.8 показано приклад Scrum-дошки в Jira з переліком задач, спринтом та статусами виконання, що демонструє підхід до організації роботи в команді розробників.

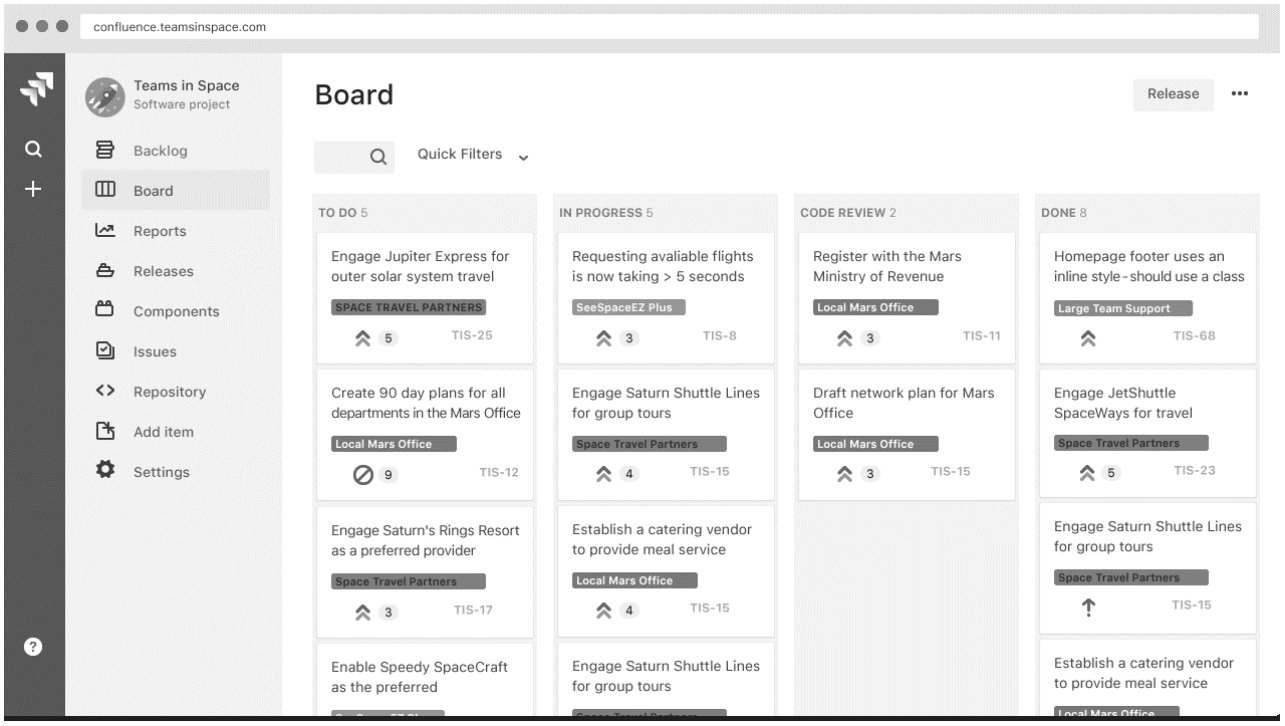


Рисунок 1.8 — Приклад Scrum-дошки в Jira

Microsoft Project — це професійна система для класичного (водоспадного) управління проєктами, яка вже багато років використовується в державному, будівельному, інженерному та корпоративному секторах. Її головна перевага — детальне планування, структуризація проєктів, побудова діаграм, управління

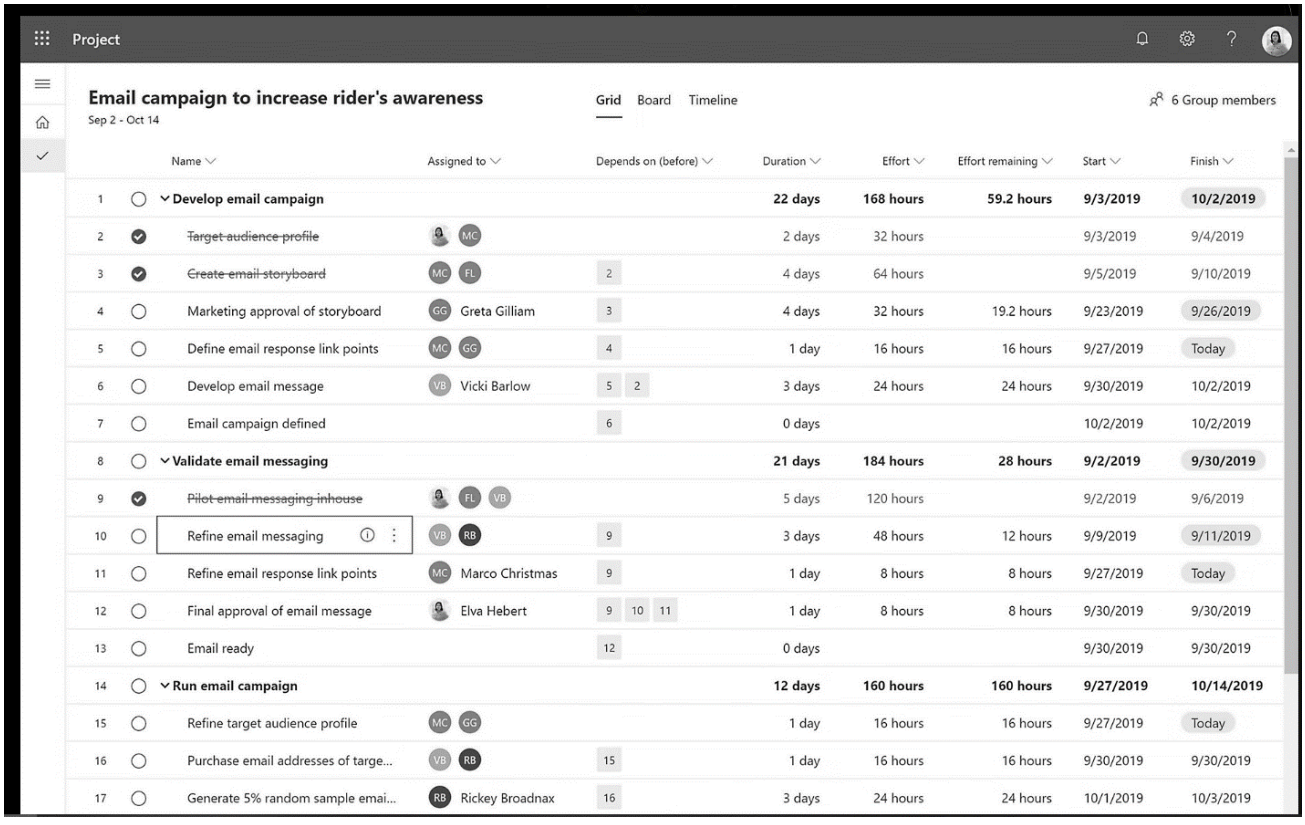
ресурсами та точний контроль над усіма етапами реалізації.

Однією з ключових можливостей є діаграма Ґанта — візуальне представлення проєкту у вигляді шкали часу, де показано всі завдання, їхню тривалість і зв’язки між ними. Це надає змогу не лише бачити загальну картину, а й ефективно виявляти вузькі місця, прогнозувати затримки та коригувати планування в режимі реального часу.

Microsoft Project також має потужну систему управління ресурсами: користувачі можуть задавати кількість доступних працівників, обладнання, бюджет та розподіляти ці ресурси між задачами. Програма автоматично підраховує завантаження виконавців, витрати та прогнозовані строки завершення.

Платформа інтегрується з іншими продуктами Microsoft (Excel, Teams, SharePoint), а також підтримує звіти, графіки, KPI-панелі, які допомагають контролювати стан проєкту як на високому, так і на деталізованому рівні.

На рисунку 1.9 продемонстровано вікно застосунку Microsoft Project із прикладом планування задач та тривалості виконання.



	Name	Assigned to	Depends on (before)	Duration	Effort	Effort remaining	Start	Finish
1	**Develop email campaign**			**22 days**	**168 hours**	**59.2 hours**	**9/3/2019**	**10/2/2019**
2	~~Target audience profile~~	MC		2 days	32 hours		9/3/2019	9/4/2019
3	~~Create email storyboard~~	MC FL	2	4 days	64 hours		9/5/2019	9/10/2019
4	Marketing approval of storyboard	GG Greta Gilliam	3	4 days	32 hours	19.2 hours	9/23/2019	9/26/2019
5	Define email response link points	MC GG	4	1 day	16 hours	16 hours	9/27/2019	Today
6	Develop email message	VB Vicki Barlow	5 2	3 days	24 hours	24 hours	9/30/2019	10/2/2019
7	Email campaign defined		6	0 days			10/2/2019	10/2/2019
8	**Validate email messaging**			**21 days**	**184 hours**	**28 hours**	**9/2/2019**	**9/30/2019**
9	~~Pilot email messaging inhouse~~	FL VB		5 days	120 hours		9/2/2019	9/6/2019
10	Refine email messaging	VB RB	9	3 days	48 hours	12 hours	9/9/2019	9/11/2019
11	Refine email response link points	MC Marco Christmas	9	1 day	8 hours	8 hours	9/27/2019	Today
12	Final approval of email message	Elva Hebert	9 10 11	1 day	8 hours	8 hours	9/30/2019	9/30/2019
13	Email ready		12	0 days			9/30/2019	9/30/2019
14	**Run email campaign**			**12 days**	**160 hours**	**160 hours**	**9/27/2019**	**10/14/2019**
15	Refine target audience profile	MC GG		1 day	16 hours	16 hours	9/27/2019	Today
16	Purchase email addresses of targe...	VB RB	15	1 day	16 hours	16 hours	9/30/2019	9/30/2019
17	Generate 5% random sample emai...	RB Rickey Broadnax	16	3 days	24 hours	24 hours	10/1/2019	10/3/2019

Рисунок 1.9 — Вікно програми Microsoft Project

Отже, сучасні системи автоматизації планування охоплюють широкий спектр інструментів — від універсальних платформ, як-от ClickUp та Asana, до спеціалізованих рішень, таких як Jira та Microsoft Project. Їх використання дозволяє значно підвищити ефективність керування завданнями, покращити командну взаємодію, забезпечити прозорість процесів і своєчасне досягнення поставлених цілей. Залежно від потреб користувача чи організації, обирається система з відповідним рівнем функціональності, складності та гнучкості.

1.6 Основні принципи побудови програм для контролю задач

Програми для контролю задач є важливим класом програмного забезпечення, що забезпечують користувачеві можливість організовувати, планувати, відстежувати та аналізувати свою діяльність. Такі програми можуть використовуватись як окремими користувачами для особистих потреб, так і командами в рамках спільної роботи над проєктами. Основна мета цих програм полягає в підвищенні продуктивності, структуризації завдань і забезпеченні контролю за термінами їх виконання.

Незалежно від масштабу чи складності, типова програма для управління задачами повинна реалізовувати базовий набір функцій, який охоплює:

— створення нових задач із можливістю вказати назву, опис, дедлайн, пріоритет, категорію;

— редагування та видалення задач, які вже внесено до системи;

— групування та сортування задач за статусом, категорією або датою;

— пошук задач за ключовими словами;

— встановлення нагадувань або сповіщень про наближення дедлайнів;

— ведення історії виконаних задач для подальшого аналізу.

Додатковими, але бажаними можливостями можуть бути: повторювані задачі, підтримка підзадач або вкладених рівнів, синхронізація з календарем, експортування або резервне копіювання даних.

Більшість застосунків для управління задачами мають схожу загальну архітектуру, яка дозволяє реалізувати необхідний функціонал ефективно та зручно

для користувача. У центрі такої архітектури зазвичай знаходиться користувацький інтерфейс, який дозволяє переглядати задачі, змінювати їх стан, додавати нові події або редагувати наявні. Успішна реалізація графічного інтерфейсу часто ґрунтується на патернах на кшталт MVVM або MVC, які забезпечують чіткий поділ логіки і представлення даних [19].

За інтерфейсом стоїть логіка програми, яка відповідає за обробку даних, сортування, фільтрацію та інші операції. Вона тісно взаємодіє з модулем збереження інформації — наприклад, з базою даних або внутрішнім сховищем файлів. У реальних проєктах часто використовується проміжний рівень — сервіс або репозиторій, який абстрагує доступ до даних і дозволяє масштабувати програму або перенести її до клієнт-серверної архітектури. Ще одним важливим компонентом є система нагадувань або сповіщень, яка постійно перевіряє стан задач і, за необхідності, повідомляє користувача про наближення терміну виконання. Її робота зазвичай базується на таймерах, фонових процесах або окремих службах, які дозволяють не перевантажувати головний потік застосунку. Наявність добре структурованої архітектури дозволяє також легко оновлювати окремі компоненти, не порушуючи цілісність роботи всієї системи.

При створенні програм для контролю задач розробники дотримуються кількох базових принципів. Насамперед — це модульність, яка передбачає чіткий поділ на окремі частини системи: наприклад, блок збереження даних, блок логіки взаємодії з користувачем, модуль обробки подій тощо. Такий підхід не лише полегшує підтримку та масштабування проєкту, а й сприяє повторному використанню коду в інших проєктах. Ще одним ключовим принципом є подієво-орієнтована логіка. Це означає, що програма реагує на дії користувача у вигляді подій — наприклад, натискання кнопки створює задачу, зміна статусу автоматично запускає оновлення інтерфейсу тощо. У мовах типу C# цей підхід реалізується через обробники подій, що забезпечує високу гнучкість реакцій програми.

У центрі кожної подібної програми знаходиться робота з даними: необхідно забезпечити стабільне збереження інформації про задачі, її швидке завантаження та актуалізацію. Часто використовуються механізми XML або JSON, локальні бази

даних, або власні формати збереження. У поєднанні з кешуванням або тимчасовим буфером це дозволяє програмі працювати стабільно навіть в офлайн-режимі.

Окрему увагу слід приділяти внутрішньому стану програми, який має точно відображати статус кожної задачі, їхню кількість, строки виконання, категорії тощо. Для цього використовуються змінні стану, логічні прапорці, або навіть цілі таблиці станів у більш складних системах. Програма має оперативно реагувати на будь-які зміни — зміну дати, статусу, тексту задачі — і миттєво оновлювати візуальне представлення даних.

Користувацький досвід у програмах для контролю задач є надзвичайно важливим, адже такі системи використовуються регулярно, іноді — по кілька разів на день. Тому інтерфейс повинен бути не лише привабливим, а й інтуїтивно зрозумілим, швидким у використанні та ненавантаженим зайвими елементами. Чим легше користувач орієнтується у додатку — тим вища ймовірність, що він продовжить ним користуватись. У таких програмах критично важливо забезпечити простий доступ до ключових функцій: створення задачі, її редагування, зміну статусу, перегляд дедлайнів. Особливо цінується можливість використовувати гарячі клавіші, контекстні меню або швидкі дії прямо зі списку задач — без необхідності відкривати кожен елемент окремо. Це значно підвищує швидкість взаємодії, особливо при великій кількості задач.

Важливо також, щоб для виконання базових дій потрібно було якомога менше натискань. Розміщення елементів на екрані має бути логічним і звичним для користувача, з чіткою ієрархією інформації. Вдалий інтерфейс допомагає користувачеві не лише швидко орієнтуватись у своїх завданнях, а й відчувати контроль над часом, знижуючи навантаження та покращуючи організацію повсякденних справ.

1.7 Проблеми та обмеження існуючих систем

Незважаючи на значну кількість доступних програмних рішень для управління завданнями, більшість із них мають певні недоліки, які знижують ефективність або обмежують комфорт користування. Ці проблеми часто виникають

через надмірну універсалізацію інструментів або, навпаки, занадто вузьку спеціалізацію, що не дозволяє адаптувати їх під індивідуальні потреби. Однією з найбільш поширених проблем є перевантаженість функціоналом, коли програма включає десятки модулів, функцій і опцій, значна частина з яких ніколи не використовується некваліфікованим користувачем. Це не лише ускладнює інтерфейс, а й робить програму менш зручною для новачків, які змушені витрачати час на адаптацію. Особливо це стосується корпоративних систем, де навіть базове створення задачі може вимагати заповнення кількох обов'язкових полів.

Ще однією проблемою є відсутність підтримки офлайн режиму. Багато популярних платформ — зокрема, ті, що працюють через браузер або потребують постійної синхронізації з сервером — не дозволяють користуватись усім функціоналом без підключення до Інтернету. Це унеможливлює ефективну роботу в польових умовах, у поїздках або при нестабільному зв'язку. Крім того, значна частина систем мають загальний, непридатний до кастомізації підхід, без врахування особливостей різних категорій користувачів. Наприклад, студенту потрібні зовсім інші функції, ніж проєктному менеджеру в IT-компанії, однак більшість сервісів орієнтовані саме на бізнес-сценарії. Через це користувачам доводиться або ігнорувати зайві модулі, або шукати інші, більш гнучкі рішення.

Ще одне обмеження — низька продуктивність на слабших пристроях. Великі веб-застосунки або кросплатформенні рішення часто споживають надмірну кількість ресурсів системи, що робить їх повільними на офісних ПК або ноутбуках середнього рівня. Для простої задачі на кшталт «додати нагадування» користувач змушений чекати завантаження десятків фонових процесів та компонентів. Важливо також згадати залежність від підписки або платних функцій. Часто базовий функціонал, який потрібен більшості користувачів (наприклад, повторювані задачі, нагадування, аналітика), доступний лише у платній версії або через обмеження після пробного періоду. Це значно знижує доступність інструменту для широкого кола користувачів. До типових обмежень таких систем належать також труднощі з інтеграцією у вже існуючі процеси — наприклад, неможливість налаштувати синхронізацію з локальними файлами або іншими

десктопними програмами. У багатьох випадках користувачі змушені повністю змінювати спосіб організації своєї роботи, щоб підлаштуватися під логіку зовнішнього сервісу. Крім того, деякі системи не дозволяють локально зберігати дані, що може бути критичним у контексті конфіденційності або роботи в захищених середовищах. Також відсутність повноцінної кастомізації у безкоштовних версіях призводить до того, що користувач не може змінити, наприклад, структуру відображення задач чи типи категорій під свої потреби [17].

З огляду на перелічені проблеми, можна зробити висновок, що існує потреба в розробці легкого, функціонального та адаптивного десктопного застосунку, який поєднував би базовий функціонал планування, гнучкість структури, простоту інтерфейсу та можливість працювати автономно.

1.8 Актуальність створення програмних рішень для персонального планування

У сучасному динамічному світі обсяг інформаційного навантаження на людину постійно зростає. Усе частіше ми стикаємось із багатозадачністю, дедлайнами, необхідністю координувати навчання, роботу, особисті справи та побут. У цих умовах вкрай важливою стає здатність ефективно керувати власним часом, контролювати виконання завдань і зберігати продуктивність протягом дня. Саме тому програмні засоби для персонального планування перетворюються на невіддільний елемент повсякденного життя користувачів.

Наявність цифрового планувальника дозволяє не лише зберігати задачі у впорядкованому вигляді, а й системно керувати пріоритетами, розподіляти час, вчасно реагувати на зміни та уникати перевантаження. Важливо підкреслити, що в умовах постійного цифрового шуму — повідомлень, сповіщень, соціальних мереж — планувальник відіграє роль опори для концентрації уваги й усвідомленого виконання завдань. Проте значна частина існуючих рішень має вузький фокус або навпаки — перевантаженість функціоналом. Деякі застосунки орієнтовані переважно на бізнес-користувача, інші мають надмірно складний інтерфейс, який відштовхує новачків. Багато сервісів працюють виключно онлайн або зберігають

дані на віддалених серверах, що не завжди допустимо в умовах конфіденційності або нестабільного підключення до Інтернету. Індивідуальне програмне рішення дозволяє врахувати особливості конкретного типу користувача — наприклад, студента, офісного працівника, фрілансера або навіть користувача з особливими потребами. У таких випадках є сенс реалізовувати лише ті функції, які реально потрібні: просте створення задач, візуальний контроль дедлайнів, нагадування, можливість сортування і категоризації, робота в офлайн-режимі, локальне зберігання даних.

Особливої актуальності набувають застосунки, які не лише виконують роль електронного списку справ, а й адаптуються під стиль роботи конкретного користувача: дозволяють налаштовувати повторення задач, працювати з пріоритетами, отримувати сповіщення у потрібний час або переглядати завдання у вигляді календаря. Саме такі рішення не лише допомагають користувачу не забувати про важливе, а й виховують дисципліну, створюють відчуття контролю над часом і формують сталі звички. Розробка такого програмного забезпечення є не лише практичним завданням для реалізації технічних навичок, але й відповіддю на реальний запит сучасного суспільства. Враховуючи, що персональна ефективність безпосередньо впливає на рівень досягнень у навчанні, роботі та житті в цілому, створення доступного, зрозумілого та адаптивного інструменту планування є завданням, яке має соціальну, освітню й практичну цінність. Таким чином, актуальність створення програмного рішення для персонального планування зумовлена як реальними потребами користувачів, так і обмеженнями наявних засобів. Це створює підґрунтя для ініціативної розробки нового подібного застосунку.

2 АНАЛІЗ ТА ОБҐРУНТУВАННЯ ВИБОРУ ТЕХНОЛОГІЙ

2.1 Постановка задачі та вимоги до програмного засобу

У сучасних умовах цифрового навантаження користувачі потребують інструментів, які допомагають організовувати щоденну діяльність, планувати задачі, контролювати дедлайни та зберігати баланс між особистими і професійними справами. Існуючі сервіси або орієнтовані на корпоративне використання, або мають надлишковий функціонал, що ускладнює їхнє використання для побутових потреб. Крім того, багато з них працюють лише онлайн, не мають локального зберігання або вимагають підписки для доступу до базових функцій.

У зв’язку з цим постає задача створення десктопного застосунку для планування та контролю задач користувача, орієнтованого на індивідуальне використання. Програмний засіб має надавати простий, інтуїтивно зрозумілий інтерфейс, базовий набір функцій (додавання, редагування, видалення задач), можливість встановлювати дедлайни, пріоритети, а також працювати без постійного підключення до Інтернету. Рішення повинно бути реалізоване у вигляді десктопного застосунку на платформі Windows з використанням технології WPF та мови програмування C#. Дані повинні зберігатися локально у вигляді реляційної бази даних SQLite, що забезпечує стабільність, простоту розгортання та відсутність залежності від зовнішніх сервісів.

Важливою особливістю проєкту є вибір десктопного формату реалізації. Попри широке розповсюдження веб та мобільних застосунків, локальні програми залишаються актуальними у випадках, коли користувачеві необхідна стабільна робота без доступу до мережі, швидкий запуск, мінімальні вимоги до ресурсів системи та повна автономність. Крім того, десктопна реалізація дозволяє краще контролювати файлову структуру, зберігати дані у власному форматі та забезпечити більш надійний захист інформації.

Ідея реалізації застосунку полягає в поєднанні простоти організації особистих завдань із базовими можливостями автоматизації: нагадуванням про майбутні події, системою категорій, підтримкою дедлайнів, індикацією важливості задач тощо. Основна ціль — забезпечити користувачеві відчуття контролю над

своїм часом без необхідності опановувати складні системи чи витрачати ресурси на зайві налаштування.

Одним із ключових акцентів у програмі стане локальне зберігання даних у базі SQLite, що дозволяє уникнути передачі інформації на сторонні сервери та забезпечити повну конфіденційність. Такий підхід є особливо важливим для користувачів, які не бажають або не мають можливості зберігати свої особисті дані у хмарних сервісах. Крім того, використання SQLite гарантує швидку роботу, легкість резервного копіювання та простоту у відновленні після збою, у разі його настання.

Функціональні вимоги визначають основні можливості, які повинні бути реалізовані в програмному засобі:

— додавання нової задачі з обов’язковими полями: назва, дедлайн, пріоритет, категорія;

— редагування існуючих задач з можливістю змінювати опис, назву, дедлайн, пріоритет або категорію;

— видалення задачі зі списку користувача;

— категоризація задач (наприклад: “Робота”, “Навчання”, “Особисте”) для зручності структурування;

— візуальна індикація задач, які прострочені (через зміну кольору);

— збереження всіх даних у локальній базі даних;

— автоматичне збереження при кожній зміні або при виході з програми;

— підтримка роботи програми без доступу до мережі Інтернет.

Нефункціональні вимоги описують властивості та обмеження, які не стосуються конкретних дій користувача, але важливі для ефективності та зручності:

— платформа: програма повинна працювати в середовищі Windows 10/11;

— середовище виконання: .NET 6 або новіша версія;

— технологія інтерфейсу: WPF, як засіб побудови сучасного UI;

— інтерфейс має бути інтуїтивно зрозумілим, з використанням елементів керування WPF.

— час відгуку інтерфейсу не повинен перевищувати 200 мс при виконанні основних дій;

— програма повинна запускатися з ярлика та не потребувати попереднього встановлення зовнішніх компонентів;

2.2 Обґрунтування вибору середовища розробки та бази даних

Для розробки десктопного застосунку, призначеного для організації та контролю особистих задач користувача, було обрано мову програмування C# у поєднанні з технологією WPF, середовищем Visual Studio, системою керування базами даних SQLite та засобом об'єктно-реляційного відображення Entity Framework Core. Такий набір технологій є оптимальним для створення продуктивного, зручного у використанні та стабільного програмного засобу, орієнтованого на повністю автономну роботу без підключення до мережі.

Мова C# поєднує об’єктно-орієнтований підхід із високим рівнем безпеки, строгим типуванням і широкою бібліотечною підтримкою. Вона активно розвивається компанією Microsoft, має глибоку інтеграцію з операційною системою Windows та надає всі необхідні інструменти для реалізації десктопних рішень. Її вибір зумовлений також зручністю в обробці подій, побудові віконних інтерфейсів та роботі з локальними файлами, що є критичними у рамках реалізації обраної теми [3].

У свою чергу, технологія WPF, яка є частиною NET-платформи, дозволяє створювати сучасні інтерфейси з підтримкою анімацій, стилів, прив’язки даних та розділення логіки застосунку від його візуального представлення. Завдяки моделі MVVM, яка активно підтримується WPF, можна реалізувати чітке розмежування між даними, представленням та логікою взаємодії, що значно спрощує супровід та масштабування коду. Порівняно з класичними рішеннями на базі Windows Forms, технологія WPF надає більше можливостей для створення гнучкого і привабливого інтерфейсу користувача [7].

На початковому етапі проєкту розглядались також альтернативні варіанти — WinForms та JavaFX. Хоча WinForms є простішим у реалізації, його

функціональність обмежена в частині адаптивного дизайну, гнучкої прив’язки даних та масштабування інтерфейсу. JavaFX, своєю чергою, потребує сторонніх середовищ виконання та не забезпечує достатньої інтеграції з Windows, що є критично важливим у випадку десктопного локального застосунку. З метою об’єктивного порівняння характеристик розглянутих середовищ розробки у таблиці 2.1 наведено ключові критерії вибору технології інтерфейсу та середовища реалізації.

Таблиця 2.1 — Порівняльна характеристика середовищ розробки

Критерій	WinForms	WPF	Java(Swing/FX)
Складність розробки	Низька	Середня	Середня/Висока
Гнучкість UI	Обмежена	Висока	Середня
Адаптивність	Слабка	Висока	Відносна
Підтримка MVVM	Немає	Повна підтримка	Частково реалізується
Інтеграція з Windows	Висока	Висока	Відсутня

Середовище розробки Visual Studio було обране як стабільна та багатофункціональна платформа, що забезпечує повну інтеграцію з .NET та підтримку WPF. Воно включає зручний редактор коду, візуальний дизайнер інтерфейсу, засоби налагодження, управління пакетами NuGet і вбудовані шаблони проєктів. Це значно пришвидшує розробку, дозволяє миттєво перевіряти зміни в інтерфейсі, зручно керувати залежностями та вести контроль версій. Наявність підтримки C#, XAML та Entity Framework у Visual Studio забезпечує узгоджену роботу всіх компонентів проєкту в єдиному середовищі.

З огляду на необхідність локального зберігання даних, для реалізації бази даних було обрано систему SQLite. Вона є реляційною, не потребує встановлення серверу та зберігає інформацію у вигляді одного «.db» файлу, що розміщується локально на пристрої користувача. Серед переваг SQLite – швидкість доступу до даних, простота інтеграції, підтримка стандарту SQL та мінімальне навантаження на систему. Для персонального застосунку, що не потребує спільної роботи чи підключення до віддаленого сховища, використання SQLite є обґрунтованим і ефективним рішенням [6].

Для роботи з базою даних обрано технологію Entity Framework Core, яка виступає об'єктно-реляційним відображенням і дозволяє маніпулювати даними за допомогою об’єктів мови C#. EF Core усуває потребу у прямому написанні SQL-запитів, забезпечує автоматичне створення та оновлення структури бази через механізм міграцій, а також підтримує зв’язки між таблицями, фільтрацію, пошук і сортування за допомогою LINQ. Це зменшує кількість помилок при взаємодії з даними, спрощує логіку програми та робить її легшою в підтримці [9].

Таким чином, обраний стек технологій – C#, WPF, Visual Studio, SQLite та EF Core – забезпечує реалізацію функціональних вимог до системи з урахуванням потреб користувача, простоти обслуговування, стабільної роботи в автономному режимі та сучасного візуального оформлення. Обґрунтованість цих рішень підтверджується широкою підтримкою з боку розробників, документацією та можливістю масштабування функціоналу в майбутньому.

2.3 Архітектура системи та взаємодія компонентів

Розробка програмного забезпечення для планування та контролю задач вимагає чіткого структурування архітектури, що дозволяє забезпечити зрозумілу логіку виконання, легкість супроводу та модульність. У запропонованій системі реалізовано трирівневу архітектуру, де виділено наступні компоненти: інтерфейс користувача, логічна частина, а також взаємодія з базою даних (рівень доступу до даних). Такий підхід дозволяє розділити відповідальність між окремими модулями, зменшуючи кількість залежностей між ними та підвищуючи стабільність системи.

Інтерфейс користувача створено з використанням засобівWPF, що дозволяє реалізувати подієвий підхід до обробки дій користувача. При натисканні на кнопку, зміні значення у полі або виборі елемента зі списку відбувається виклик події, яка обробляється у відповідному методі логіки програми. Таким чином, будь-яка взаємодія з інтерфейсом запускає конкретні функції у внутрішній частині системи, не змішуючи UI із логічною частиною.

Логічна частина відповідає за обробку подій, перевірку вхідних даних, взаємодію з об’єктами та формування запитів до бази даних. Тут реалізовані

основні функції: додавання нової задачі, редагування, видалення, зміна статусу виконання, фільтрація, сортування та пошук. Для зберігання інформації використовується локальна база даних на основі SQLite, доступ до якої реалізовано через засіб об'єктно-реляційного відображення Entity Framework Core. Це дозволяє працювати із сутностями у вигляді об’єктів C#, не використовуючи прямі SQL-запити [15].

Взаємодія між компонентами реалізована за принципами шаблону MVVM, який є рекомендованим при роботі з WPF. У ViewModel використано колекцію ObservableCollection, що забезпечує автоматичну синхронізацію списку задач із графічним інтерфейсом користувача. Інтерфейс відображає дані, ViewModel виступає посередником між представленням і логікою, а модель забезпечує зберігання і структурування даних. Зв'язок між інтерфейсом та ViewModel здійснюється за допомогою механізму прив’язки даних, що дозволяє оновлювати інтерфейс автоматично при зміні властивостей об'єктів. Хоча реалізація MVVM у даній системі є спрощеною, її базова структура дозволяє чітко відокремити логіку від візуального шару. Відповідно до наведеної архітектури, взаємодія між компонентами побудована послідовно та логічно. Користувач здійснює дії у графічному інтерфейсі, які ініціюють відповідні події в логічному шарі застосунку. Ці події обробляються відповідними методами, які або оновлюють стан об’єктів, або формують запити до бази даних. Усі запити реалізовано через Entity Framework Core, що забезпечує зручну та безпечну взаємодію з локальною базою даних SQLite.

Крім цього, архітектура системи передбачає розширюваність і підтримку майбутніх змін. Завдяки чіткому розділенню логіки, інтерфейсу та доступу до даних, можливе безболісне впровадження нових функцій або модулів без необхідності істотної модифікації існуючих компонентів. Такий підхід особливо важливий при супроводі програмного забезпечення та забезпечує його довготривалу життєздатність у реальних умовах використання.

На рисунку 2.1 представлено загальну архітектуру системи, що демонструє взаємозв’язки між ключовими модулями.

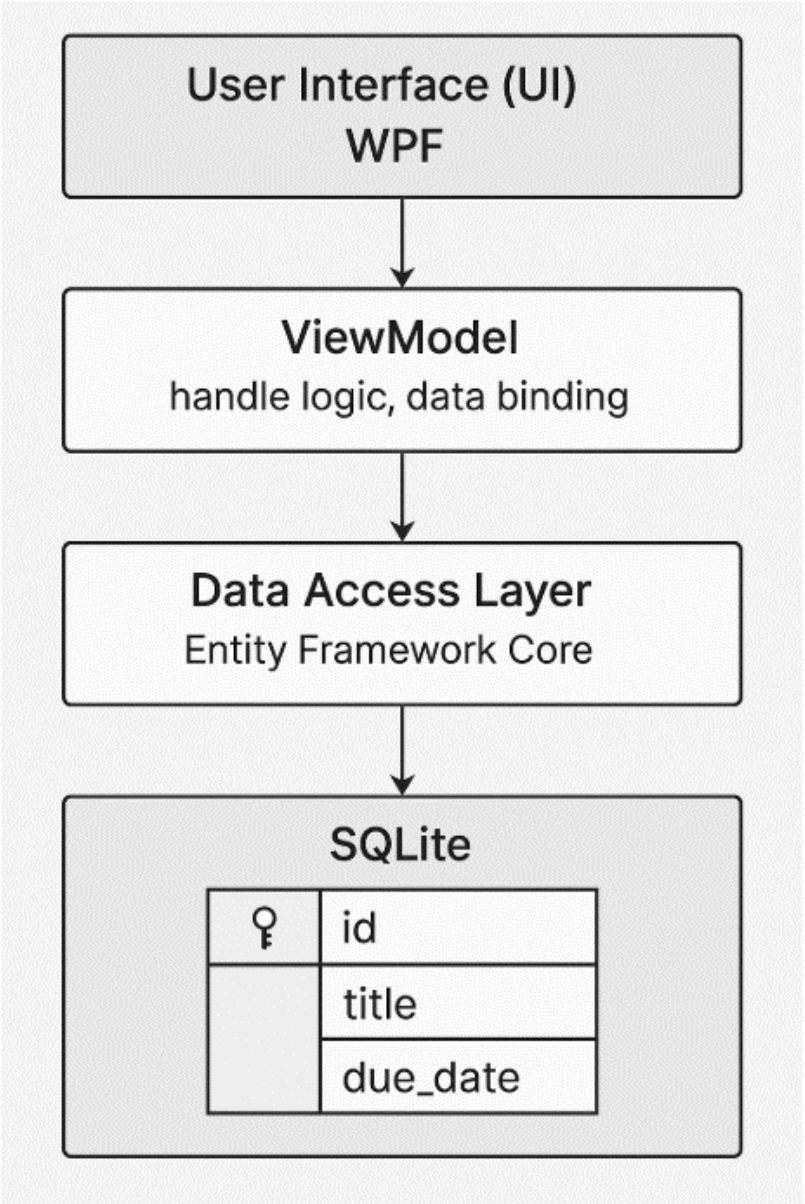


Рисунок 2.1 — Архітектура та взаємодія компонентів системи

Завдяки такій структурі інтерфейс користувача працює окремо від логіки обробки даних, що дозволяє розробляти, змінювати або тестувати кожен компонент незалежно. Зміни у структурі зберігання даних або способах їх обробки не впливають на зовнішній вигляд та поведінку інтерфейсу, оскільки взаємодія між ними відбувається через окремий посередницький шар. Це ж стосується і зворотного зв’язку: зміни в даних автоматично відображаються у вікні програми завдяки використанню прив’язки даних.

Обрана модель дозволяє не лише створити стабільний і логічно побудований застосунок, але й забезпечити потенціал для подальшого масштабування. У разі

розширення функціональності (наприклад, додавання хмарної синхронізації або мережевого режиму) можна буде розширити відповідні компоненти без повної перебудови архітектури. Крім того, підтримка шаблону MVVM створює передумови для якісного тестування та багаторазового використання компонентів [5].

Таким чином, описана архітектура забезпечує чіткий поділ відповідальностей, зменшує взаємозалежності між модулями, спрощує налагодження та підтримку системи, а також створює зручне підґрунтя для розвитку та удосконалення застосунку в майбутньому.

2.4 Моделювання структури бази даних

Для зберігання інформації про задачі користувача у розроблюваному програмному засобі використовується локальна реляційна база даних, реалізована за допомогою SQLite. Такий підхід дозволяє зберігати всі дані у вигляді одного локального файлу, що не потребує підключення до серверу або зовнішніх сервісів. Це забезпечує автономність, високу швидкодію, стабільність роботи та зручність резервного копіювання. База даних створюється автоматично при першому запуску програми та інтегрується безпосередньо в структуру застосунку.

Архітектура бази даних побудована з урахуванням простоти, зручності обробки та можливості розширення в майбутньому. У системі реалізовано дві основні сутності: задачі та категорії. Задачі є головною складовою системи, саме з ними безпосередньо взаємодіє користувач. Кожна задача містить набір основних параметрів, таких як унікальний ідентифікатор, назва, опис, дата дедлайну, пріоритет, статус виконання, а також належність до певної категорії. Для реалізації можливості групування задач за тематиками або типами в базі також передбачена таблиця категорій, у якій зберігається список доступних категорій з унікальними назвами. Враховуючи необхідність підтримки зв’язків між таблицями, у структурі реалізовано зовнішній ключ — кожна задача містить посилання на ідентифікатор категорії, що дозволяє швидко відфільтровувати задачі за категоріями та забезпечити цілісність даних при оновленні або видаленні записів. Такий підхід

дозволяє реалізувати просту, але повноцінну логіку збереження інформації, яку легко масштабувати у разі розширення функціональності системи.

У рамках структури бази даних ключовим елементом є таблиця «Tasks», що відповідає за зберігання всіх параметрів створених задач. Саме з цією таблицею безпосередньо працює як логіка програми, так і користувач через інтерфейс. Її структура повинна бути не лише технічно коректною, але й зручною для використання в контексті фільтрації та сортування задач. З огляду на це, структура Tasks була спроєктована з урахуванням мінімального, але достатнього набору полів для повноцінної роботи застосунку.

Усі поля таблиці мають чітко визначене призначення. Поля «Title» і «DueDate» забезпечують базову ідентифікацію та контроль виконання задачі, «Priority» дозволяє організовувати завдання за важливістю, а «IsCompleted» — оперативно фіксувати факт виконання. Додаткове поле «Description» не є обов’язковим, однак дозволяє надати більше контексту до задачі. Посилання на «CategoryId» забезпечує зв’язок між задачами та категоріями, що створює можливість для гнучкої класифікації та подальшої фільтрації записів у застосунку. Завдяки чіткому поділу відповідальностей між полями таблиця залишається простою, але достатньо гнучкою, щоби підтримати увесь запланований функціонал системи. Також її структура забезпечує зручність при реалізації запитів у рамках ORM-технологій, зокрема з використанням Entity Framework Core, який буде застосовано при розробці програмного продукту. Така структура дозволяє ефективно зберігати дані про задачі у зручному форматі, забезпечуючи швидкий доступ до них у процесі роботи програми. Крім того, вона є базою для реалізації фільтрації, сортування та відображення задач у графічному інтерфейсі. Ретельно продумана схема полів сприяє збереженню цілісності даних та уникненню дублювання записів. Завдяки цьому база даних залишається стабільною та придатною для масштабування в разі розширення функціоналу.

З метою кращого розуміння структури збереження даних у системі, у таблиці 2.2 наведено опис основних полів таблиці «Tasks». Ця таблиця є центральною в моделі бази даних і містить усі ключові атрибути, необхідні для створення, обробки

та контролю задач користувача в межах програмного засобу.

Таблиця 2.2 — Структура таблиці Tasks

Назва поля	Тип даних	Призначення
Id	Integer	Первинний ключ, унікальний ідентифікатор задачі
Title	Text	Назва задачі, короткий опис суті
Description	Text	Розгорнутий опис задачі (необов’язковий)
DueDate	DateTime	Дата і час, до якого потрібно виконати задачу
Priority	Integer	Числове значення пріоритету
IsCompleted	Boolean	Статус виконання задачі
CategoryId	Integer	Зовнішній ключ, що вказує на відповідну категорію в таблиці «Categories»

Також реалізовано навігаційне поле «Category», що дозволяє безпосередньо отримати пов’язані дані з таблиці «Categories» через механізм зв’язків у Entity Framework Core. Це поле не зберігається фізично у базі, але використовується у програмі для зручного відображення категорій.

У моделі даних також передбачено кілька додаткових властивостей, які не зберігаються у базі «[NotMapped]», але використовуються у програмі для форматованого виводу, зокрема «PriorityDisplay», «DeadlineStatusDisplay», «CategoryDisplay» тощо [16].

Наведена структура забезпечує оптимальний баланс між простотою

реалізації та можливістю подальшого розширення функціоналу, відповідаючи вимогам до локального десктопного застосунку для управління завданнями.

2.5 Інтерфейс користувача з урахуванням принципів UX/UI

Інтерфейс користувача є ключовим елементом будь-якого застосунку, що визначає зручність, швидкість та ефективність взаємодії між користувачем і програмною системою. Особливо важливим це стає у випадку програм, орієнтованих на щоденне використання, таких як системи управління задачами. Саме тому при розробці інтерфейсу у даному проєкті було приділено значну увагу застосуванню сучасних підходів у сфері UX та UI-дизайну.

Головною метою є створення максимально зрозумілого, інтуїтивного та адаптивного інтерфейсу, який не перевантажує користувача зайвими елементами, але при цьому забезпечує доступ до всього необхідного функціоналу. При проєктуванні інтерфейсу враховувалися такі ключові принципи UX/UI:

— простота, лише необхідні елементи без зайвого візуального шуму;

— логічна структура, кожна дія доступна у два-три кліки, без зайвих переходів;

— візуальні акценти, завдання з наближеними або простроченими термінами виділяються кольором, що дозволяє користувачу швидко їх ідентифікувати;

— сталість, однакові дії мають однакову поведінку в різних місцях інтерфейсу;

— доступність, інтерфейс зберігає зручність використання на екранах з різним розміром вікна, забезпечуючи коректне масштабування елементів [8].

Інтерфейс програми складається з кількох логічних зон. У верхній частині розташована панель керування задачами — додавання нової задачі, введення її назви, опису, дати виконання та вибору категорії. Нижче — основний робочий простір, де відображається список задач із можливістю фільтрації за статусом виконання, категорією або пріоритетом. Кожна задача відображається у вигляді окремого блоку з можливістю редагування, видалення або відмітки про наближення дедлайну. Також передбачено візуальні індикатори задач, дедлайни

яких вже минули — вони автоматично підсвічуються, що дозволяє швидко виявляти прострочені завдання.

Структура головного вікна побудована за принципом односторінкового інтерфейсу — користувач взаємодіє з усім функціоналом на одній формі без необхідності перемикання між вкладками або відкриття додаткових вікон. Такий підхід забезпечує швидкість роботи та мінімізує навантаження на користувача. Усі основні дії — додавання, редагування, сортування та перегляд задач — доступні з головного вікна, що відповідає сучасним вимогам до UX [11].

Для реалізації інтерфейсу обрано платформу WPF, яка дозволяє створювати сучасні, гнучкі й динамічні UI-рішення з використанням мови розмітки XAML. Завдяки підтримці шаблонів, стилів і механізму прив’язки даних, WPF дозволяє відокремити логіку від презентаційного рівня, що спрощує подальше обслуговування та розширення інтерфейсу. Крім того, використання подій у поєднанні з MVVM-архітектурою дає змогу реалізувати реактивну взаємодію без зайвого дублювання коду. Таким чином, інтерфейс користувача програми відповідає сучасним стандартам зручності, простоти та доступності, що робить систему ефективною у повсякденному використанні та зрозумілою навіть для недосвідчених користувачів.

2.6 Забезпечення цілісності даних та взаємозв’язків у базі даних

Цілісність даних у системі управління базами даних означає збереження правильності, узгодженості та достовірності інформації, що зберігається в таблицях. У межах розроблюваного застосунку підтримка цілісності бази даних є надзвичайно важливою, оскільки від неї залежить коректність роботи всіх функцій програми: відображення задач, фільтрації, групування та оновлення записів.

Для забезпечення цілісності використовується набір стандартних механізмів, зокрема:

— первинні ключі, гарантують унікальність кожного запису, у таблицях «Tasks» та «Categories» поля «Id» виступають первинними ключами, що не допускають дублювання і є обов’язковими для кожного рядка;

— зовнішні ключі, дозволяють зберігати зв’язки між таблицями. У таблиці «Tasks» поле «CategoryId» є зовнішнім ключем, що вказує на запис у таблиці «Categories». Такий зв’язок реалізується як “багато до одного”;

— обмеження типів даних, поля мають строго визначені типи що запобігає некоректному введенню;

— обов’язковість заповнення полів, важливі поля, такі як «Title», не можуть бути порожніми, що забезпечує мінімально необхідну інформацію для кожної задачі.

Для зручності розгляду в таблиці 2.3 наведено основні аспекти цілісності, реалізовані у базі даних роботи.

Таблиця 2.3 — Засоби забезпечення цілісності даних у БД

Механізм	Реалізація у проєкті
Первинні ключі	Поля «Id» в таблицях «Tasks» та «Categories»
Зовнішні ключі	Поле «CategoryId» у таблиці «Tasks» пов’язане з «Categories.Id»
Обмеження типів	Типи полів: Integer, Text, DateTime, Boolean
Обов’язкові поля	«Title», «DueDate», «CategoryId» не допускають збереження порожніх значень
Уникнення дублів	Завдяки первинним ключам; дублікати записів неможливі

З боку програмної реалізації, додаткову перевірку на рівні застосунку забезпечує Entity Framework Core. При створенні нової задачі або зміні існуючої ORM автоматично перевіряє відповідність структури сутностей визначеним обмеженням у моделі. Крім того, у процесі застосування міграцій схема бази даних оновлюється без порушення вже існуючих зв’язків. Крім того, при запуску програми використовується метод EnsureCreated(), який автоматично створює базу даних відповідно до структури моделей, якщо вона ще не існує. Це дозволяє забезпечити коректну ініціалізацію даних без необхідності виконання окремих міграцій або налаштувань, що особливо зручно для настільного застосунку. Таким

чином, реалізована система гарантує стабільну роботу, відсутність “висячих” записів, а також запобігає введенню некоректної інформації, що сприяє підвищенню загальної надійності та стабільності програмного забезпечення [13].

2.7 Організація обробки задач, фільтрації, сортування

Однією з ключових функцій розроблюваного програмного засобу є підтримка повного життєвого циклу задач: створення, збереження, відображення, оновлення стану та видалення. Для реалізації цієї логіки в архітектурі програми передбачено взаємодію між інтерфейсом, логікою ViewModel та базою даних через ORM-засіб Entity Framework Core.

Додавання нової задачі ініціюється користувачем через відповідну форму, яка розташована у верхній частині головного вікна. Після заповнення полів і натискання кнопки “Додати” запускається відповідний метод логіки, що створює об’єкт типу «TaskModel» із заданими параметрами. Перед збереженням перевіряються обов’язкові поля, у разі валідності, об'єкт передається до контексту бази даних. Команда «context.Tasks.Add(newTask)» додає запис до таблиці, після чого викликається метод SaveChanges(), який зберігає зміни у фізичному файлі SQLite. Дані автоматично оновлюються на екрані завдяки оновленню зв’язаної колекції «ObservableCollection», яка автоматично синхронізується з візуальним компонентом через прив’язку даних.

Редагування задачі реалізовано через механізм двостороннього зв’язку, що дозволяє завантажити дані вибраного запису у форму введення. Після зміни параметрів і підтвердження нові значення записуються в об’єкт задачі, після чого викликається команда «context.Update(task)» і застосовуються зміни у базі. Статус виконання змінюється безпосередньо зі списку задач через чекбокс або кнопку — подія викликає зміну значення у відповідному полі, а після виклику SaveChanges() база оновлюється.

Видалення задачі виконується лише після підтвердження дії користувачем, аби запобігти випадковим втратам даних. Для цього використовується метод «context.Tasks.Remove(task)» з наступним збереженням змін. Перед видаленням

перевіряється наявність об’єкта у контексті — це запобігає помилкам при видаленні неактуального запису.

Усі ці операції реалізовані з урахуванням принципів безпеки та узгодженості. Обробка винятків, перевірка даних та повторна ініціалізація візуальних елементів після дій із базою — усе це дозволяє уникнути ситуацій втрати зв’язку між UI та моделлю завдяки реалізації шаблону MVVM, який забезпечує автоматичну синхронізацію стану між інтерфейсом користувача та моделлю даних.

Другим важливим аспектом організації роботи із задачами є реалізація механізмів фільтрації та сортування. Вони дозволяють користувачу швидко знаходити потрібні записи, аналізувати інформацію та фокусуватись лише на актуальних елементах. У програмному засобі реалізовано кілька фільтрів, які можуть застосовуватись незалежно один від одного або у поєднанні: фільтрація за статусом виконання, за обраною категорією та за рівнем пріоритету. Фільтрація відбувається на рівні ViewModel — дані не завантажуються повністю з UI, а формуються як результат LINQ-запитів через Entity Framework Core, що дозволяє програмно виконувати фільтрацію без прямого використання SQL. Всі запити виконуються через ORM-рівень, що забезпечує безпечну та ефективну взаємодію із SQLite.

Сортування задач реалізоване за кількома напрямками: за датою виконання, за алфавітним порядком назви та за рівнем пріоритету. При зміні критерію сортування виконується оновлення списку задач, який відображається у головному вікні. Сортування також відбувається на рівні бази або у ViewModel в залежності від обраної стратегії оптимізації. Важливою особливістю реалізації є те, що жодна фільтрація чи сортування не призводить до повного перезавантаження форми. Оновлюється лише колекція об’єктів, прив’язана до елемента керування (наприклад, списку задач), що забезпечує високу швидкість оновлення та зменшує навантаження на систему. Завдяки цьому користувач отримує миттєвий результат без затримок або перемальовування всього інтерфейсу. Також варто зазначити, що вся обробка фільтрації та сортування є

реактивною — кожна зміна параметра призводить до негайного оновлення даних. Сортування та фільтрація реалізовані за допомогою LINQ-запитів до колекції задач, що дозволяє виконувати ефективні операції без прямого звернення до бази даних. Це забезпечує динамічну взаємодію між користувачем і програмою, що є ключовим фактором у зручності використання системи. Архітектурно це реалізовано через реалізацію шаблону MVVM, де ViewModel виступає проміжною ланкою між UI та базою даних, координуючи логіку завантаження та відображення задач.

У комплексі реалізовані підходи дозволяють підтримувати високу продуктивність, гнучкість налаштування інтерфейсу та зручність у роботі з даними, що повністю відповідає функціональним вимогам до даного програмного продукту.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

3.1 Реалізація основних функцій додатку: створення, редагування та видалення задач

Однією з основних функціональних складових розроблюваного застосунку є можливість повноцінного керування задачами — від їх створення до остаточного видалення. Ці функції формують базову логіку взаємодії користувача із системою, і реалізовані з урахуванням принципів зручності, надійності та цілісності даних. Процес створення нової задачі починається з введення користувачем інформації у відповідні поля інтерфейсу. Це включає назву задачі, опис (необов'язково), дату та час дедлайну, рівень пріоритету та обрану категорію. Для цього у верхній частині головного вікна програми передбачена спеціальна форма з текстовими полями, випадаючими списками та кнопкою "Додати". Після заповнення користувач натискає кнопку, що ініціює виклик методу ViewModel, який реалізує відповідну логіку обробки події. У коді «ViewModel» створюється екземпляр «TaskModel», який заповнюється на основі введених користувачем даних, та який перед збереженням проходить перевірку на валідність. Зокрема, поля «Title», «DueDate» та «CategoryId» є обов'язковими, і їхня наявність перевіряється до передачі об'єкта у контекст бази даних. У разі успішної перевірки задача додається до колекції задач за допомогою стандартного методу ORM — додавання в контекст Entity Framework і збереження змін. Після цього оновлений список задач миттєво відображається у інтерфейсі, що забезпечується двосторонньою прив'язкою даних. Для забезпечення стабільності та уникнення дублювання об'єктів, кожна нова задача автоматично отримує унікальний ідентифікатор, який виступає первинним ключем у таблиці Tasks [19].

Редагування задачі реалізовано з урахуванням можливості швидкого оновлення вмісту без необхідності переходу на інше вікно або форму. При виборі задачі зі списку користувач має можливість завантажити її параметри у поля введення. Для цього передбачено логіку, яка при виборі об'єкта з колекції виконує прив'язку його властивостей до відповідних елементів інтерфейсу. Після внесення змін, ініціюється оновлення запису у базі даних.

З технічного боку оновлення здійснюється шляхом отримання об’єкта задачі за його Id, внесення змін до його властивостей та виклику методу «Update». Перед оновленням перевіряється унікальність запису та прив’язка до об'єкта контексту, щоб уникнути дублювання чи помилок прив'язки. Після застосування методу SaveChanges() зміни зберігаються у фізичному файлі бази SQLite. У результаті список задач оновлюється автоматично, без необхідності ручного перезавантаження форми. Такий підхід забезпечує зручність та зменшує кількість дій з боку користувача [10].

Функція видалення задачі також реалізована без зайвих ускладнень, але з урахуванням вимог до безпеки даних. Перед видаленням задача обов’язково має бути вибрана користувачем. Після цього запускається метод, який видаляє об’єкт із контексту Entity Framework за допомогою методу Remove. Обов’язковою частиною реалізації є запит на підтвердження дії з боку користувача, який захищає від випадкового видалення. Якщо підтвердження отримано, зміни застосовуються, і запис повністю вилучається з таблиці Tasks. Після видалення або редагування задачі інтерфейс автоматично оновлює список, оскільки колекція задач пов’язана з елементом керування через механізм прив’язки даних. Це забезпечує синхронність між базою даних і візуальним відображенням інформації на формі користувача.

Таким чином, реалізовані функції створення, редагування та видалення задач охоплюють базовий функціонал системи та виконуються відповідно до сучасних вимог до десктопних застосунків. Взаємодія між інтерфейсом, логікою ViewModel і базою даних побудована таким чином, щоб забезпечити стабільність, швидкодію та зрозумілу поведінку програми у процесі роботи з даними.

3.2 Впровадження категорій, пріоритетів і фільтрації задач

Для підвищення зручності візуального сприйняття та управління завданнями, у програмі реалізована система категорій задач. Категорії поділяють задачі за типом активності, наприклад: «Робота», «Навчання», «Особисте» тощо. Ці категорії представлені у вигляді випадаючого списку під час створення задачі, а також використовуються у фільтрах для швидкого перегляду лише певного типу

записів. З технічного боку, категорії зберігаються у окремій таблиці Categories бази даних SQLite. Зв'язок між задачами і категоріями реалізований через зовнішній ключ «CategoryId». Таким чином, кожна задача може належати лише до однієї категорії, але сама категорія — до необмеженої кількості задач.

Для задач також передбачено встановлення пріоритету, що дозволяє користувачу виокремлювати більш важливі записи. Пріоритет задається при створенні або редагуванні задачі через ще один ComboBox, значення якого обмежено до трьох рівнів: «Низький», «Середній» і «Високий». Ці значення зберігаються як числове поле «Priority» у таблиці задач, що дозволяє легко здійснювати сортування або умовне форматування інтерфейсу. У візуальному представленні задач пріоритет позначається кольором або іконкою, відповідно до рівня важливості. Наприклад, задачі з високим пріоритетом можуть мати червоне тло або спеціальний індикатор поряд з назвою. Цей візуальний акцент реалізується через стилі WPF, із використанням «DataTemplate» або стилів для «ListViewItem», залежно від рівня пріоритету що дозволяє автоматично змінювати відображення елементів на основі значень властивостей.

Для забезпечення гнучкого перегляду та аналізу задач у додатку реалізована система фільтрації, що дозволяє користувачу миттєво обмежувати відображення списку задач відповідно до заданих критеріїв.

Основними параметрами фільтрації є:

— категорія задачі, дозволяє показати лише задачі, що належать до вибраної категорії;

— статус виконання, фільтрація за виконаними або невиконаними задачами;

— пріоритет — дозволяє відібрати задачі з певним рівнем важливості;

— тимчасові межі, наприклад, задачі «термінові», «заплановані» та «прострочені».

Інтерфейс фільтрації реалізовано у вигляді панелі з елементами ComboBox, CheckBox. Кожна зміна параметра фільтра автоматично оновлює відображуваний список задач без перезавантаження вікна — завдяки використанню механізму прив’язки даних в WPF та оновлюваних колекцій ObservableCollection. Усі фільтри

реалізовані як властивості у ViewModel, що дозволяє забезпечити правильну двосторонню прив’язку елементів інтерфейсу та зручне керування станом фільтрації. Логіка фільтрації реалізована на основі LINQ-запитів, що застосовуються до колекції задач, через контекст бази даних, із урахуванням встановлених користувачем значень фільтрів. Комбінація умов (AND/OR) враховується динамічно, в залежності від встановлених користувачем значень фільтрів. Окрім цього, передбачено кнопку «За замовчуванням», яка скидає усі параметри та повертає повний список задач.

Результатом впровадження фільтрації є підвищення ефективності роботи користувача з великим обсягом даних що забезпечує високу ефективність роботи навіть з великим обсягом задач, підвищуючи зручність користувача у повсякденному використанні системи.

3.3 Реалізація візуальної індикації дедлайнів та пріоритетів задач

З метою підвищення інформативності та зручності користування програмою, в додатку реалізована візуальна індикація рівня пріоритету задач. Цей механізм дозволяє швидко визначити важливість завдань без потреби заглиблюватися в додаткові деталі.

Кожна задача в системі має параметр «Priority», що зберігається в базі даних як числове або перераховане значення. На основі цього значення формується візуальна індикація на рівні інтерфейсу. У WPF для відображення пріоритету задач використовується:

— кольорове оформлення рядка задачі (фон або рамка);

— іконка або маркер важливості (вставлена в шаблон елемента списку);

Це реалізується за допомогою «DataTemplate» та «DataTriggers» — стандартного механізму WPF, що дозволяє змінювати стилі елементів управління залежно від значення властивості, прив’язаної до джерела даних.

Оформлення пріоритетів підбиралось з урахуванням психологічного сприйняття кольорів:

— червоний, позначає терміновість або критичність;

— жовтий, попередження або підвищену важливість;

— зелений, відносно незначні задачі.

Завдяки цьому користувач отримує візуальні підказки щодо того, які задачі потребують негайної уваги.

Структура індикації була розроблена таким чином, щоб її можна було розширити або змінити без втручання у логіку збереження або в базу даних. Наприклад, у разі введення додаткового пріоритетного рівня достатньо буде додати нове правило відображення в XAML.

Окрім індикації пріоритетів, у застосунку реалізовано візуальне виділення задач залежно від їхнього дедлайну. Це дозволяє користувачу миттєво оцінити терміновість завдань та зосередити увагу на тих, що потребують першочергового виконання.

Визначення статусу дедлайну виконується у ViewModel, де для кожної задачі розраховується різниця між поточним часом і полем «DueDate». На основі цієї інформації формується властивість, яка може набувати наступних значень:

— прострочено, дедлайн уже минув;

— термінове, дедлайн сьогодні або до нього залишилось менше доби;

— заплановане, до дедлайну залишилось понад добу.

Ця властивість не зберігається в базі даних, а динамічно визначається під час відображення задач, що дозволяє адаптивно керувати візуальним оформленням інтерфейсу без додаткового навантаження на збережену модель [18].

3.4 Створення інтерфейсу в середовищі WPF та робота з подіями

Інтерфейс користувача в додатку реалізовано з використанням технології WPF, що забезпечує розділення логіки відображення та бізнес-логіки, а також дозволяє створювати гнучкі, адаптивні та стильові UI-рішення. Розмітка вікон здійснюється за допомогою мови XAML, де кожен елемент інтерфейсу має чітке призначення та може бути стилізований або прив'язаний до даних.

Основне вікно програми включає кілька функціональних блоків:

— форма додавання задач, поля введення назви, опису, дати та вибору

пріоритету, всі ці поля розміщені у верхній частині інтерфейсу для зручності користувача;

— список задач, динамічний блок, який відображає перелік усіх активних задач, для його реалізації використано елемент «ListView», який підтримує шаблони представлення кожного запису;

— панель фільтрації та сортування, розташована над списком задач і дозволяє швидко змінювати відображення за обраним критерієм;

— кнопки взаємодії, створити, видалити, редагувати, ці кнопки зв’язані з подіями через «Command» або обробники «Click».

Для організації макету застосовано гнучку систему контейнерів — таких як Grid, StackPanel, WrapPanel, які дозволяють адаптувати інтерфейс до вмісту, зберігаючи при цьому порядок і чистоту відображення. Реакція на взаємодію користувача реалізується за допомогою подій, зокрема:

— подія «Click», для кнопок дій (створення, видалення, редагування);

— подія «SelectionChanged», для обробки вибору задачі зі списку.

Один із ключових принципів побудови інтерфейсу в середовищі WPF — це двостороння прив’язка даних. У додатку використовується прив’язка між властивостями об’єктів задач і візуальними елементами інтерфейсу, що дозволяє оновлювати дані в обох напрямках — від користувача до моделі та навпаки. Наприклад, введення тексту у поле автоматично оновлює відповідну властивість об'єкта ViewModel, без потреби в ручному виклику логіки оновлення [12].

Для представлення колекцій задач використовується тип «ObservableCollection», який автоматично інформує інтерфейс про зміни — додавання, видалення чи редагування записів. Це дозволяє динамічно оновлювати список задач без повного перезавантаження вікна, що покращує продуктивність програми. Робота з подіями у WPF реалізується як безпосередньо через обробники, так і через шаблон «Command», особливо у випадках використання MVVM. Такий підхід дозволяє відокремити логіку обробки подій від коду інтерфейсу, що покращує масштабованість застосунку [20].

Особлива увага при розробці інтерфейсу приділялась зручності

використання. Усі елементи логічно згруповані, мають чітке візуальне розмежування, зручні відступи та зрозумілі підписи. Колірна палітра, розміри шрифтів та іконки підібрані таким чином, щоб зменшити навантаження на зір та забезпечити швидке орієнтування навіть при великій кількості задач. Інтерфейс програми побудований за принципом односторінкового доступу — усі основні дії виконуються в межах одного вікна, що значно спрощує навігацію і мінімізує кількість дій, необхідних для керування задачами. Таким чином, реалізація інтерфейсу за допомогою WPF дозволила досягти високої гнучкості, ефективної роботи з даними та комфортної взаємодії користувача з програмою.

3.5 Тестування функціональності застосунку

Метою тестування програмного забезпечення є перевірка функціональності розробленої системи на відповідність вимогам, що були сформовані на етапі планування. Тестування дозволяє виявити можливі помилки та переконатися в стабільності роботи системи при різних сценаріях використання. Основною задачею є забезпечення належного рівня якості програмного продукту перед передачею його кінцевому користувачу.

Для перевірки функціональності застосунку було використано ручне тестування, що полягало в покроковому виконанні основних дій користувача з подальшою перевіркою результатів. Усі функціональні модулі було протестовано на відповідність технічному завданню. Крім того, було проведено інтеграційне тестування, з метою перевірки взаємодії між основними компонентами системи: інтерфейсом, базою даних SQLite та логікою процесів у «ViewModel».

Основними протестованими сценаріями були:

— додавання нової задачі з усіма обов’язковими параметрами;

— редагування існуючої задачі;

— видалення задачі;

— фільтрація задач за категорією, пріоритетом та статусом виконання;

— робота з архівом виконаних задач;

— збереження та зчитування задач з бази даних;

— валідація полів при створенні задачі;

— відображення дедлайну та індикації пріоритету.

Кожен із цих сценаріїв перевірявся окремо, а також у комбінації з іншими для перевірки взаємодії компонентів системи. Таким чином, тестування охоплювало не лише базові сценарії взаємодії з користувачем, але й комплексну перевірку логіки роботи застосунку при зміні даних у базі даних [21].

З метою узагальнення процесу перевірки функціональності програмного застосунку, у таблиці 3.1 наведено перелік основних дій, що були протестовані в рамках ручного тестування. Таблиця відображає виконання ключових функцій системи «TaskManager» та їх відповідність очікуваним результатам.

Таблиця 3.1 — Таблиця перевірки функціональних можливостей

Номер функції	Перевірка функції	Результат
1	Додавання нової задачі	Успішно
2	Редагування існуючої задачі	Успішно
3	Видалення задачі	Успішно
4	Збереження та зчитування задач із бази даних SQLite	Успішно
5	Фільтрація задач за категорією, пріоритетом та статусом	Успішно
6	Індикація пріоритету та статусу дедлайну	Успішно
7	Переміщення задач до архіву	Успішно
8	Валідація обов’язкових полів при створенні та редагуванні задачі	Успішно
9	Автоматичне заповнення полів при редагуванні задачі	Успішно
10	Відображення статусу задач (Прострочено, Термінове, Заплановано)	Успішно
11	Візуальне оновлення інтерфейсу після додавання, редагування, архівації задач	Успішно

Проведене тестування підтвердило правильність реалізації функціональних можливостей застосунку. Усі основні сценарії використання були перевірені

вручну й успішно виконані без виявлених критичних помилок. Система коректно взаємодіє з базою даних, відображає актуальну інформацію, виконує валідацію та забезпечує стабільну роботу користувацького інтерфейсу. Отже, програмне забезпечення повністю відповідає вимогам, поставленим на етапі проєктування, і є готовим до використання кінцевим користувачем.

3.6 Інструкція користувача та демонстрація запуску програми

Щоб розпочати роботу з програмним застосунком «TaskManager», користувачу необхідно запустити програму за допомогою відповідного ярлика на робочому столі або відкривши виконуваний файл «TaskManager.exe», що розміщується у каталозі встановлення. Після запуску відкривається головне вікно програми, яке забезпечує доступ до всіх основних функціональних можливостей.

Після успішної компіляції або відкриття програми через ярлик «TaskManager.exe», користувач потрапляє у головне вікно застосунку. У центральній частині інтерфейсу розміщено список задач з основними атрибутами: назва, опис, категорія, дедлайн, пріоритет та статус.

Інтерфейс програми побудований таким чином, щоб забезпечити максимально зручну взаємодію користувача з системою управління задачами. Усі елементи розміщено логічно та інтуїтивно зрозуміло, що дозволяє швидко освоїтися навіть користувачам без технічної підготовки. Головне вікно містить необхідні інструменти для перегляду та керування задачами, з чітким розмежуванням функцій за допомогою кнопок та фільтрів. Завдяки використанню сучасних елементів управління та адаптивному дизайну інтерфейсу, користувач може ефективно виконувати всі необхідні дії з мінімальними витратами часу.

У верхній частині розташовані поля для створення або редагування задачі:

— Назва, Опис — текстові поля для введення основної інформації;

— Категорія — випадаючий список для вибору категорії задачі;

— Дедлайн — вибір дати виконання;

— Пріоритет — список для вибору важливості задачі.

Нижче знаходяться додаткові елементи керування:

— Фільтр — дозволяє фільтрувати задачі за статусом;

— Сортування — відповідає за порядок відображення задач.

У нижній частині — кнопки для основних дій:

— Додати, Редагувати, Видалити — керування задачами;

— Архів — перегляд виконаних задач;

— Додати до архіву — позначення задачі як виконаної;

— Вийти — закриття програми.

З метою наочного ознайомлення з інтерфейсом застосунку, на рисунку 3.1 представлено головне вікно програми «TaskManager» після запуску. У цьому вікні користувач може переглядати поточні задачі, застосовувати фільтрацію, а також створювати, редагувати, видаляти або архівувати задачі.

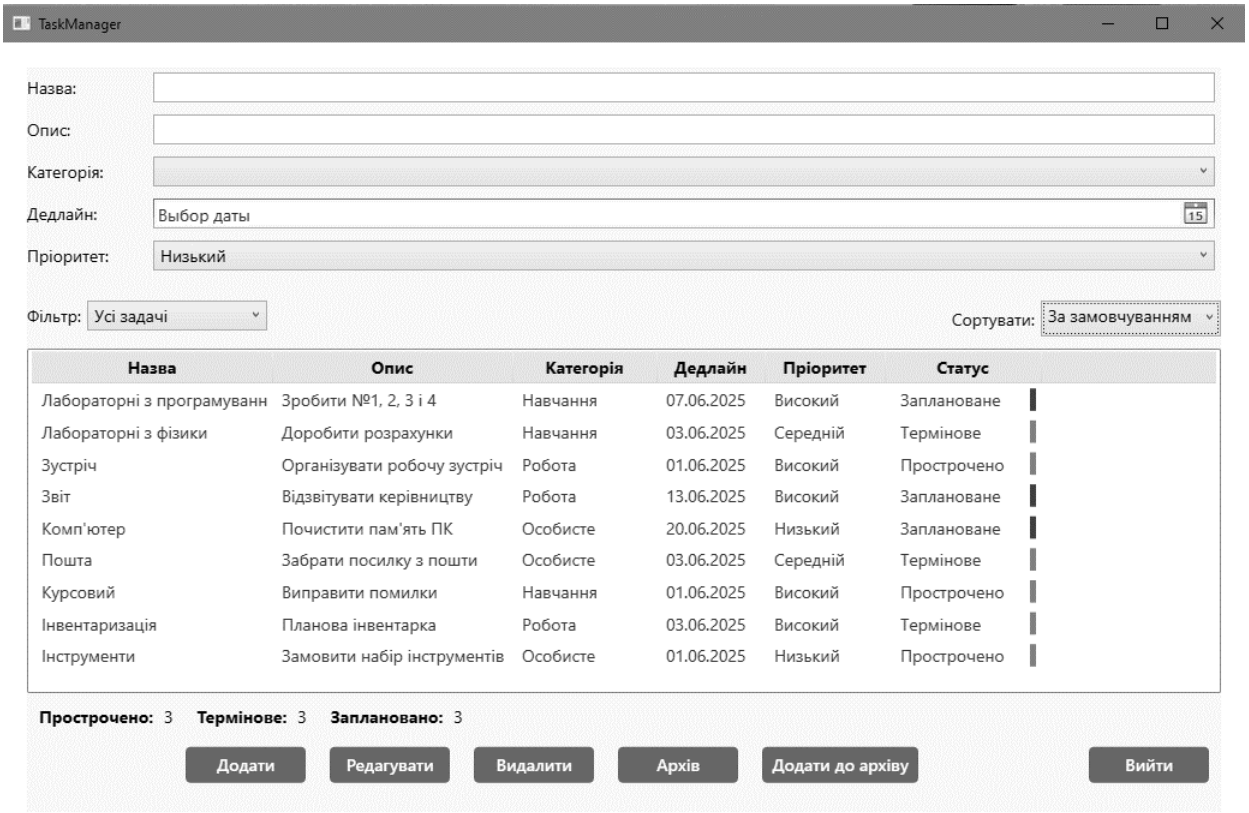


Рисунок 3.1 — Головне вікно програми TaskManager після запуску

Для зручності навігації в застосунку реалізовано можливість фільтрації та сортування задач. За допомогою випадаючого списку «Фільтр» користувач може обрати потрібну категорію задач — наприклад, лише виконані, лише заплановані

або всі задачі. Це дозволяє швидко відфільтрувати перелік задач відповідно до поточних потреб.

Окрім фільтрації, доступне сортування задач за різними критеріями. У правій частині інтерфейсу знаходиться меню сортування, де можна вибрати один з варіантів: за замовчуванням, за назвою, за дедлайном або за пріоритетом. Це дозволяє впорядкувати задачі у зручному порядку, наприклад, спершу показати ті, що мають найближчий строк виконання. Також користувач може комбінувати фільтрацію та сортування, що дозволяє ефективно управляти великою кількістю задач. Завдяки цьому, навіть при значному навантаженні програма забезпечує швидкий доступ до найактуальнішої інформації.

З метою демонстрації роботи інструментів фільтрації та сортування, на рисунку 3.2 представлено відображення задач при активному фільтрі «Усі задачі» та сортуванні за дедлайном.

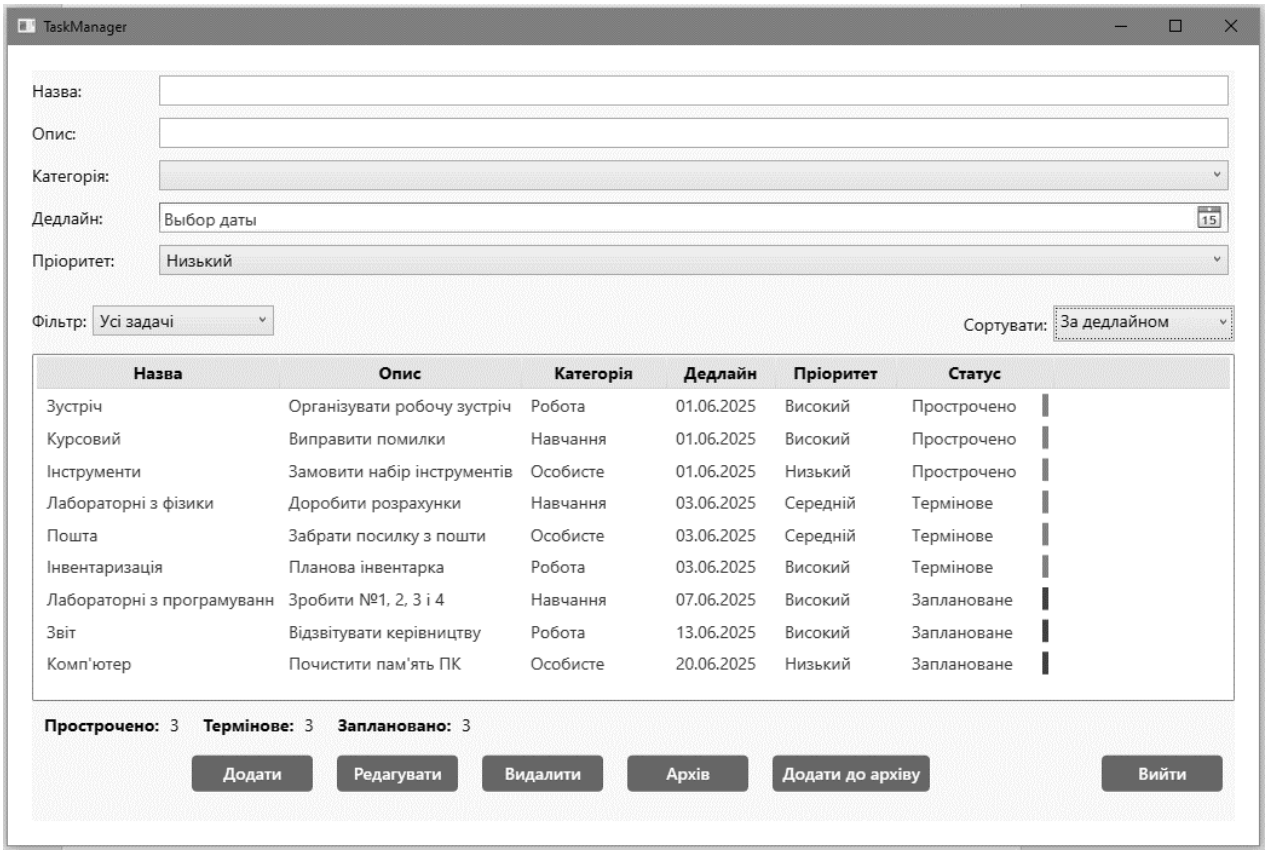


Рисунок 3.2 — Список задач при сортуванні «За дедлайном»

У застосунку реалізована можливість переміщення виконаних задач до

архіву. Це дозволяє зберігати історію виконаних завдань окремо від основного списку, зберігаючи чистоту та актуальність головного інтерфейсу. Користувач може натиснути кнопку «Архів», після чого відкривається окреме діалогове вікно зі списком усіх задач, позначених як виконані.

У цьому вікні представлено основні параметри задачі — назву, опис, категорію, дедлайн та пріоритет. За потреби користувач може видалити будь-яку задачу з архіву, використовуючи відповідну кнопку у нижній частині вікна, або закрити вікно, повернувшись до основного інтерфейсу. Можливість керування вмістом архіву дозволяє підтримувати його в упорядкованому стані та уникати накопичення застарілої інформації. Це дозволяє не лише впорядкувати задачі, але й зберігати доступ до завершених справ у разі потреби повторного перегляду або аналізу виконаного обсягу роботи.

Для демонстрації роботи, на рисунку 3.3 наведено приклад відкритого вікна «Архів виконаних задач», де відображаються задачі, які переніс користувач.

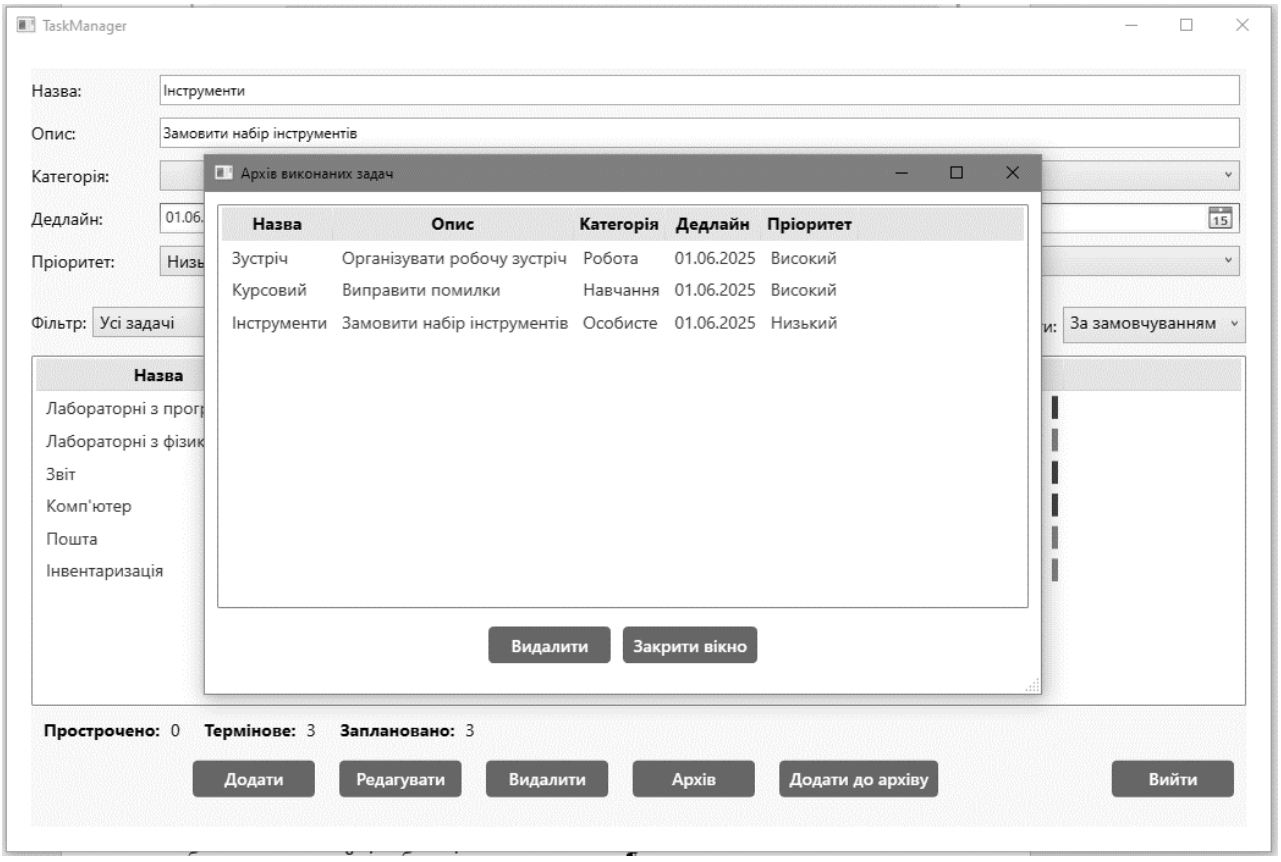


Рисунок 3.3 — Вікно перегляду архіву виконаних задач

Для забезпечення коректної роботи застосунку та уникнення збереження неповної інформації, реалізовано механізм валідації обов’язкових полів. Якщо користувач намагається додати або редагувати задачу без заповнення ключових полів — програма автоматично виводить повідомлення про помилку.

На екрані з’являється попереджувальне вікно з відповідним текстом та кнопкою підтвердження, після натискання якої користувач може внести необхідні виправлення. При цьому обов’язкові поля візуально підсвічуються червоним кольором, що додатково привертає увагу до некоректно заповнених даних.

Така сама перевірка виконується і під час редагування існуючої задачі, що гарантує однакові правила введення інформації для обох режимів. Це дозволяє підтримувати цілісність даних та забезпечує зручний механізм взаємодії.

На рисунку 3.4 продемонстровано приклад ситуації, коли користувач намагається здійснити додавання або редагування існуючої задачі без заповнення всіх обов’язкових полів.

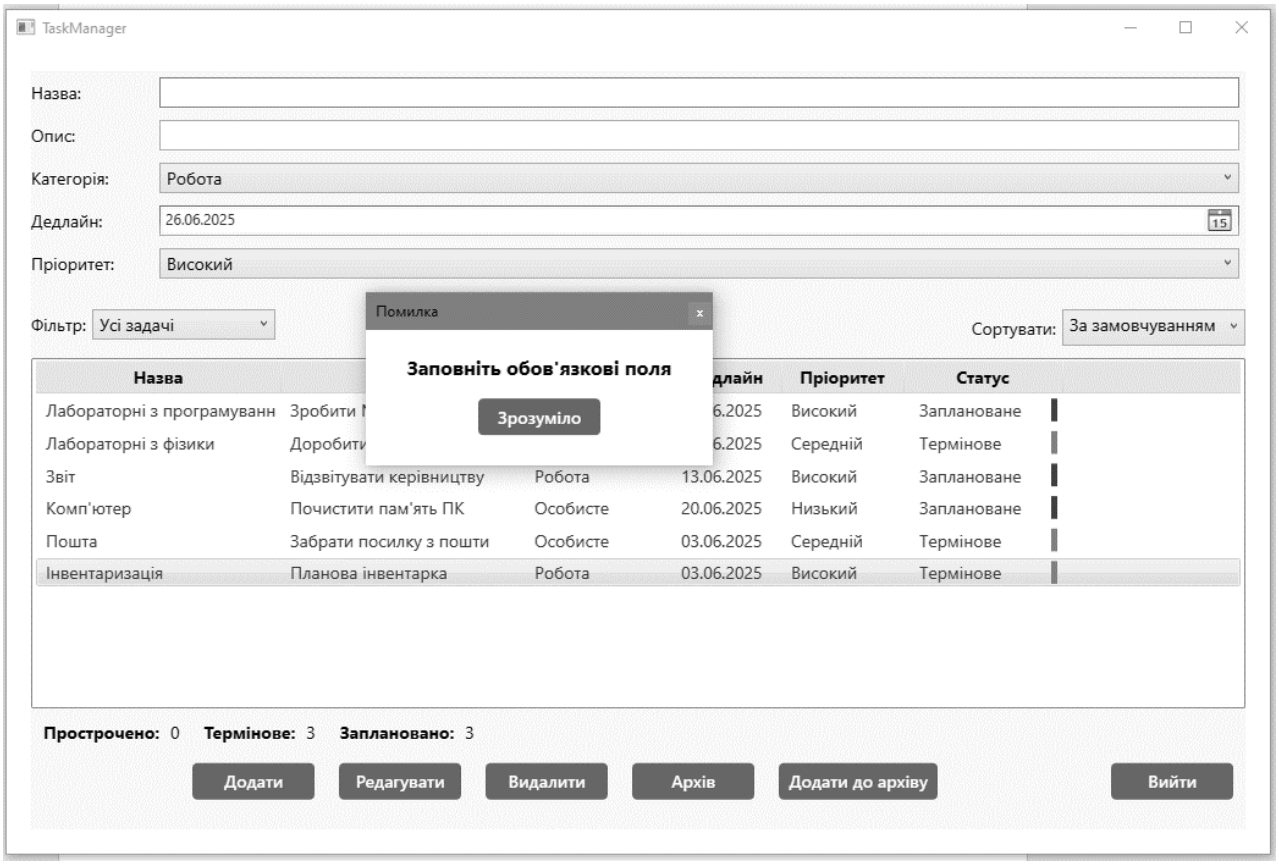


Рисунок 3.4 — Повідомлення про помилку при відсутності важливих полів

У результаті тестової експлуатації застосунку «TaskManager» було підтверджено зручність та інтуїтивність користувацького інтерфейсу. Основні функції — додавання, редагування, фільтрація, архівація задач та перевірка правильності введених даних — реалізовано у доступній формі, що забезпечує комфортне використання навіть для недосвідчених користувачів. Візуальні елементи, діалогові вікна та повідомлення про помилки сприяють кращій взаємодії з програмою, а також підвищують загальну якість користувацького досвіду.

3.7 Можливості розвитку та вдосконалення системи

Розроблений програмний засіб має модульну архітектуру, що дозволяє у майбутньому масштабувати та вдосконалювати його без необхідності повної перебудови системи. Подальший розвиток може охоплювати як функціональні, так і нефункціональні компоненти. У контексті розширення функціональності першочерговими можуть бути:

— додавання системи сповіщень, реалізація локальних повідомлень про наближення дедлайнів або зміну статусу задач;

— синхронізація з хмарними сервісами, можливість зберігати дані в хмарі для доступу з різних пристроїв, зокрема через Google Drive або OneDrive;

— мобільна версія застосунку, розробка супутнього додатку для Android/iOS, який би дозволив управляти задачами з телефону або планшета;

— імпорт та експорт задач, підтримка обміну даними у форматах CSV, XML або JSON, що полегшить міграцію та інтеграцію з іншими інструментами.

Крім того, існує потенціал для впровадження додаткових засобів аналітики, наприклад, візуалізація виконання задач у вигляді діаграм, статистики за періодами чи категоріями. Такі функції дозволять користувачам оцінювати власну продуктивність та переваги чи недоліки розпорядку задач. Окрему увагу варто звернути на підтримку розширених налаштувань користувача — кастомізація інтерфейсу, вибір тем оформлення, налаштування частоти оновлення задач та сортування за складеними правилами. Ще одним вектором розвитку системи є автоматизація повторюваних задач. У перспективі можна реалізувати функцію

створення циклічних завдань із заданою періодичністю (щодня, щотижня, щомісяця тощо). Це дозволить користувачам не витрачати час на вручну повторне додавання однакових задач, що є актуальним для рутинної або регламентованої діяльності.

У перспективі система може бути масштабована до корпоративного рівня, що дозволить впровадити багатокористувацький режим. Це дасть змогу користувачам об’єднуватись у групи, розподіляти ролі (наприклад, адміністратор, співробітник, переглядач), керувати спільними списками задач та отримувати централізований контроль над поточним прогресом. Такий функціонал значно підвищить ефективність використання програми в невеликих командах, навчальних закладах та малих підприємствах. Також перспективним є впровадження системи розширених прав доступу. Залежно від ролі користувача, можна обмежувати або надавати повний доступ до певних функцій застосунку. Наприклад, для корпоративного використання — одні учасники команди зможуть лише переглядати задачі, інші — редагувати, а адміністратори — керувати структурою категорій, пріоритетів та доступу. Це забезпечить додаткову гнучкість та безпеку при масштабуванні проєкту.

Завдяки модульному підходу до архітектури, більшість із зазначених вдосконалень можуть бути реалізовані поступово, без істотного впливу на стабільну роботу вже наявної функціональності.

ВИСНОВКИ

У ході виконання бакалаврської кваліфікаційної роботи було створено план, та реалізовано комп’ютерну систему для планування, організації та контролю задач користувача. Розроблений програмний застосунок дозволяє ефективно керувати особистими справами, встановлювати дедлайни, визначати пріоритетність завдань, здійснювати фільтрацію задач та зберігати інформацію у локальній базі даних. Реалізація програми виконувалась із використанням сучасних технологій: WPF, архітектурного патерну MVVM, мови C# та системи керування базами даних SQLite.

У процесі дослідження проаналізовано актуальні проблеми цифрової самоорганізації користувачів, особливо в умовах багатозадачності та відсутності централізованих інструментів керування часом. Визначено функціональні вимоги до системи управління задачами та обґрунтовано вибір програмних засобів для її реалізації. Проведено проєктування структури бази даних, що забезпечує зберігання задач, категорій та пріоритетів. Застосовано механізми валідації обов’язкових полів**,** візуальної індикації стану задач, фільтрації за параметрами, а також реалізовано архів задач, що зберігає завершені завдання окремо від активних.

Розроблений застосунок забезпечує інтуїтивно зрозумілий інтерфейс, що дозволяє кінцевому користувачу без зайвих зусиль додавати, редагувати, групувати та архівувати задачі. Програма не потребує підключення до інтернету та працює локально, що підвищує її зручність, безпеку використання та швидкодію. Реалізація візуальних елементів через XAML, логіки у ViewModel та збереження даних у SQLite забезпечила повну відповідність поставленим функціональним вимогам.

У результаті тестування було підтверджено працездатність усіх компонентів системи, зокрема функцій додавання, редагування, архівації, валідації та фільтрації задач. Інтерфейс програми стабільно оновлюється відповідно до дій користувача, забезпечуючи високий рівень зручності та функціональності. Програмний продукт може бути використаний як самостійний

інструмент організації особистого часу або як частина більшої системи управління діяльністю.

Запропоновано можливі напрями подальшого розвитку: додавання нагадувань, синхронізація з календарем, підтримка експорту та імпорту задач, багатокористувацький режим. Реалізація таких функцій дозволить розширити сферу застосування програми та зробити її універсальним інструментом для широкого кола користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1 Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 123 «Комп'ютерна інженерія» (освітня програма «Комп'ютерна інженерія») [Електронний ресурс] / уклад. О. Д. Азаров, С. В. Богомолов, С. І. Швець. – Вінниця : ВНТУ, 2023. – 105 с.

2 Нагорний О. М. Розробка інтерфейсів користувача у WPF : навч. посіб. / О. М. Нагорний. — Київ : Наукова думка, 2020. — 192 с.

3 Чалий В. В. Мова програмування C# : теорія та практика / В. В. Чалий. — Харків : Факт, 2021. — 284 с.

4 Система управління задачами Microsoft To Do. [Електронний ресурс] — URL: https://todo.microsoft.com/, дата звернення 20.04.2025.

5 MVVM Design Pattern. [Електронний ресурс] — URL: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/data/mvvm/, дата звернення 19.05.2025.

6 Система управління базами даних SQLite. [Електронний ресурс] — URL: https://www.sqlite.org/index.html, дата звернення 18.05.2025.

7 Глушаков С. А. Windows Presentation Foundation. Програмування інтерфейсів у C# / С. А. Глушаков. — Дніпро : Універсум, 2022. — 164 с.

8 UI/UX Design for Desktop Applications. [Електронний ресурс] — URL: https://www.nngroup.com/articles/desktop-apps/, дата звернення 22.04.2025.

9 Entity Framework Core Documentation. [Електронний ресурс] — URL: https://learn.microsoft.com/en-us/ef/core/, дата звернення 20.05.2025.

10 Програмування на C# з використанням шаблону MVVM / С. Бойко // Вісник НТУУ «КПІ». Серія: Програмна інженерія. — 2022. — № 2. — С. 41—47.

11 Інтерфейси користувача: принципи проектування та зручність використання / Дж. Рубін // Програмні системи. — 2021. — № 3. — С. 24—31.

12 Шевченко І. І. Розробка локальних додатків з використанням .NET Core / І. І. Шевченко // Інформаційні технології і засоби навчання. — 2023. — № 2. — С. 56—61.

13 Гнатюк С. П. Бази даних: моделювання, проєктування та реалізація / С. П.

Гнатюк. — Львів : Новий Світ, 2020. — 248 с.

14 Comparison of task management apps: Trello, Asana, Todoist. [Електронний ресурс] — URL: https://zapier.com/blog/best-task-management-apps/, дата звернення 24.04.2025.

15 SQLite vs SQL Server: What to choose? [Електронний ресурс] — URL: https://learn.microsoft.com/en-us/sql/sql-server/sql-server-vs-sqlite, дата звернення 23.04.2025.

16 Гнатишин Р. Ю. Використання шаблонів проєктування в розробці desktop-застосунків / Р. Ю. Гнатишин // Системи управління, навігації та зв’язку. — 2022. — № 4. — С. 88—94.

17 TaskManager – Simple WPF To-Do List App. [Електронний ресурс] — URL: https://github.com/kerrry/TaskManagerWPF, дата звернення 25.04.2025.

18 Principles of Interaction Design. [Електронний ресурс] — URL: https://www.interaction-design.org/literature/topics/interaction-design, дата звернення 22.04.2025.

19 Хрестоматія з проектування інтерфейсів / за ред. П. І. Сидоренка. — Одеса : ТЕС, 2021. — 156 с.

20 Візуалізація інформації в управлінських системах / М. Д. Кузьменко // Вісник ХНУРЕ. — 2020. — № 5. — С. 72—78.

21 Створення ефективного користувацького досвіду у настільних додатках / Т. М. Костюк // Програмна інженерія. — 2023. — № 1. — С. 38—44.

ДОДАТОК А

Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ

Завідувач кафедри ОТ

проф., д.т.н. Азаров О. Д.

«21» березня 2025р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи

«КОМП'ЮТЕРНА СИСТЕМА ДЛЯ ПЛАНУВАННЯ ТА КОНТРОЛЮ ЗАДАЧ КОРИСТУВАЧА»

08-54.БКР.026.00.000 ТЗ

Науковий керівник: доцент к.т.н.
_____ Томчук М.А.
Студент групи: 2КІ-23мс
_____Крижанівський Д.В.

ВНТУ 2025

1 Підстава для виконання бакалаврської кваліфікаційної роботи.

1.1 Актуальність цього дослідження зумовлена необхідністю ефективної організації особистих задач користувача в умовах високої інтенсивності інформаційного навантаження, дистанційної роботи, та навчання. Більшість доступних рішень є або занадто складними у використанні, або залежать від підключення до мережі, що обмежує їхню доступність. Створення зручного, автономного програмного засобу для управління задачами користувача — є актуальним завданням, яке має як теоретичне, так і прикладне значення.

1.2 Наказ про затвердження теми бакалаврської дипломної роботи.

2 Мета і призначення БКР

2.1 Мета роботи полягає у розробці комп’ютерної системи для планування та контролю задач користувача, яка забезпечить зручну та ефективну взаємодію з особистими завданнями, дозволить відслідковувати строки, фільтрувати задачі, змінювати їхній стан, а також зберігати дані в локальній базі з використанням сучасних програмних технологій.

2.2 Призначення розробки — створення комп’ютерної системи управління задачами як частини бакалаврської кваліфікаційної роботи, з можливістю її практичного застосування кінцевими користувачами у повсякденному житті, а також із подальшими перспективами розширення функціоналу.

3 Вихідні дані для виконання БКР

3.1 Запропонувати підхід до реалізації функціоналу управління задачами користувача, з урахуванням актуальних вимог до зручності, автономності та візуалізації даних.

3.2 Визначити технічні та функціональні вимоги до програми:

— підтримка текстових даних (назва, опис);

— фільтрація задач за категоріями, пріоритетом і статусом;

— кольорова індикація дедлайнів;

— локальне збереження даних у базі даних SQLite;

— автономна робота без доступу до мережі.

3.3 Вибір архітектури та технологій:

— розробка у середовищі Visual Studio;

— застосування WPF і архітектурного патерну MVVM;

— використання Entity Framework Core.

4 Вимоги до виконання БКР

Розробити застосунок з інтуїтивно зрозумілим інтерфейсом, що дозволяє створювати, редагувати, видаляти, фільтрувати та архівувати задачі. Реалізувати валідацію обов'язкових полів, візуальну індикацію стану задачі, сортування та категоризацію. Здійснити збереження всіх даних у локальній базі SQLite та забезпечити відповідність між інтерфейсом користувача та логікою обробки даних.

5 Етапи БДР та очікувані результати

Етапи роботи та очікувані результати приведено в таблиці А.1.

Таблиця А.1 — Етапи БДР

№ з/п	Назва етапу	Термін виконання		Очікувані результати
		початок	кінець	
1	Аналіз предметної області	21.03.2025	24.03.2025	Формування функціональних вимог
2	Вивчення інструментів і технологій	25.03.2025	29.03.2025	Розділ 1
3	Розробка структури бази даних	30.03.2025	01.03.2025	Розділ 2
4	Реалізація інтерфейсу та логіки	02.04.2025	10.04.2025	Розділ 2
5	Впровадження фільтрів, валідації, архіву	11.04.2025	29.04.2025	Розділ 3
6	Тестування та налагодження, перевірка працездатності	30.04.2025	31.04.2025	Розділ 3
7	Оформлення пояснювальної записки та презентації	01.05.2025	13.05.2025	ПЗ, графічний матеріал, презентація

6 Матеріали, що подаються до захисту БКР

До захисту подаються: пояснювальна записка БКР, ілюстративні матеріали, протокол попереднього захисту БКР на кафедрі, відгук наукового керівника,

анотації до БКР українською та іноземною мовами, довідка про відповідність оформлення БКР діючим вимогам.

7 Порядок контролю виконання та захисту БДР

Виконання етапів графічної та розрахункової документації БКР контролюється науковим керівником згідно зі встановленими термінами. Захист БКР відбувається на засіданні Екзаменаційної комісії, затвердженої наказом ректора.

8 Вимоги до оформлення та порядок виконання БДР

8.1 При оформлюванні БКР використовуються:

— ДСТУ 3008 : 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302 : 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— ГОСТ 2.104-2006 «Єдина система конструкторської документації. Основні написи»;

— Методичні вказівки до виконання бакалаврських кваліфікаційних робіт зі спеціальності 123 «Комп’ютерна інженерія» (освітня програма «Системне програмування»). Кафедра обчислювальної техніки ВНТУ 2022;

— документами на які посилаються у вище вказаних.

8.2 Порядок виконання БДР викладено в «Положення про кваліфікаційні роботи на першому (бакалаврському) рівні вищої освіти СУЯ ВНТУ-03.02.02-П.001.01:21»

ДОДАТОК Б

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Комп'ютерна система для планування та контролю задач користувача

Тип роботи: _____ бакалаврська кваліфікаційна робота _____
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ _____кафедра обчислювальної техніки_____
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism ___4___%

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

✓ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

_____Азаров О.Д., зав. каф. ОТ _____ _____
(прізвище, ініціали, посада) (підпис)

____Крупельницький Л.В., доц. каф ОТ_____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ ___Захарченко С.М.____
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ _____
(підпис) (прізвище, ініціали, посада)

Здобувач _____ _____
(підпис) (прізвище, ініціали)

ДОДАТОК В

ГРАФІЧНА ЧАСТИНА

КОМП’ЮТЕРНА СИСТЕМА ДЛЯ ПЛАНУВАННЯ ТА КОНТРОЛЮ ЗАДАЧ КОРИСТУВАЧА

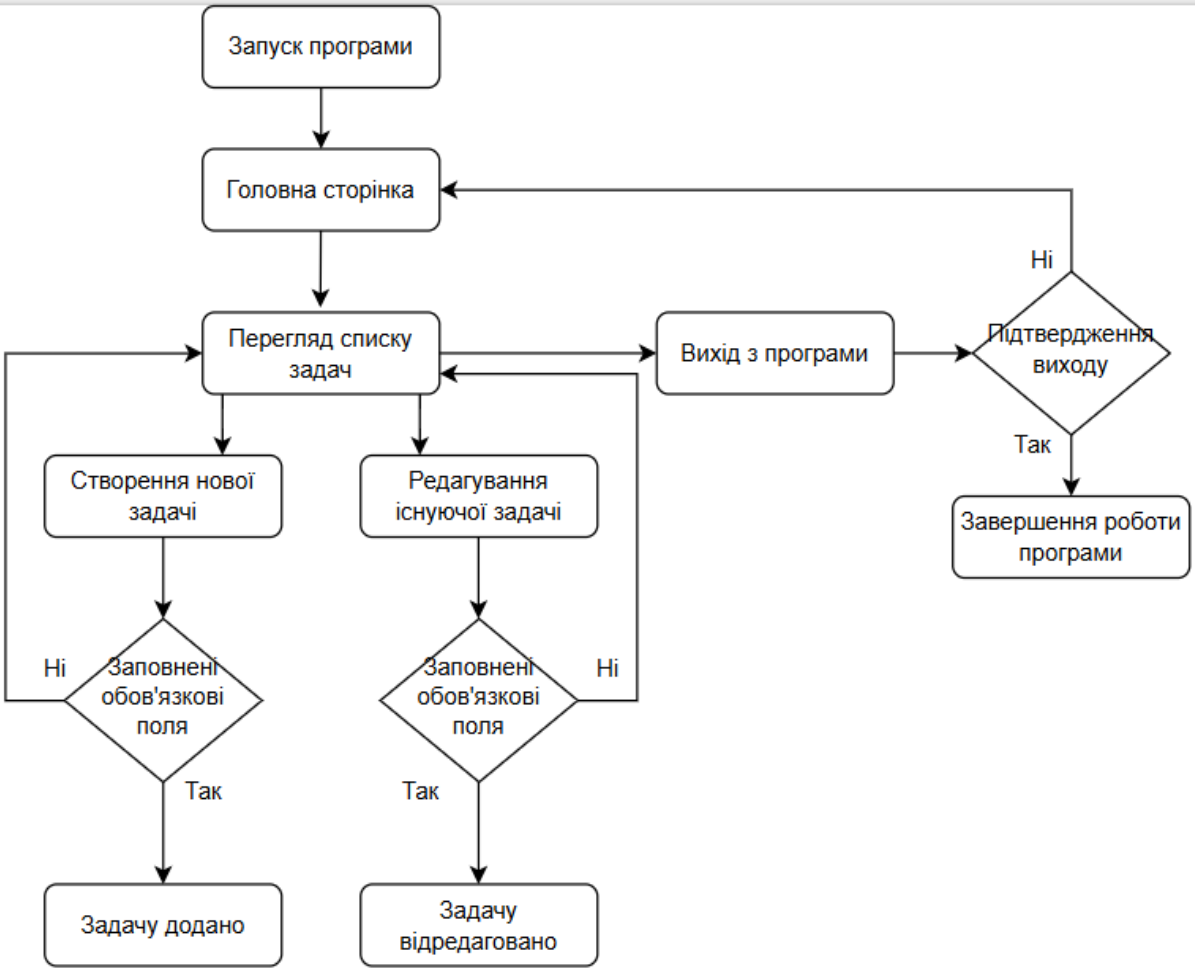


Рисунок В.1 — Блок-схема алгоритму роботи логіки валідації даних та підтвердження дій користувача

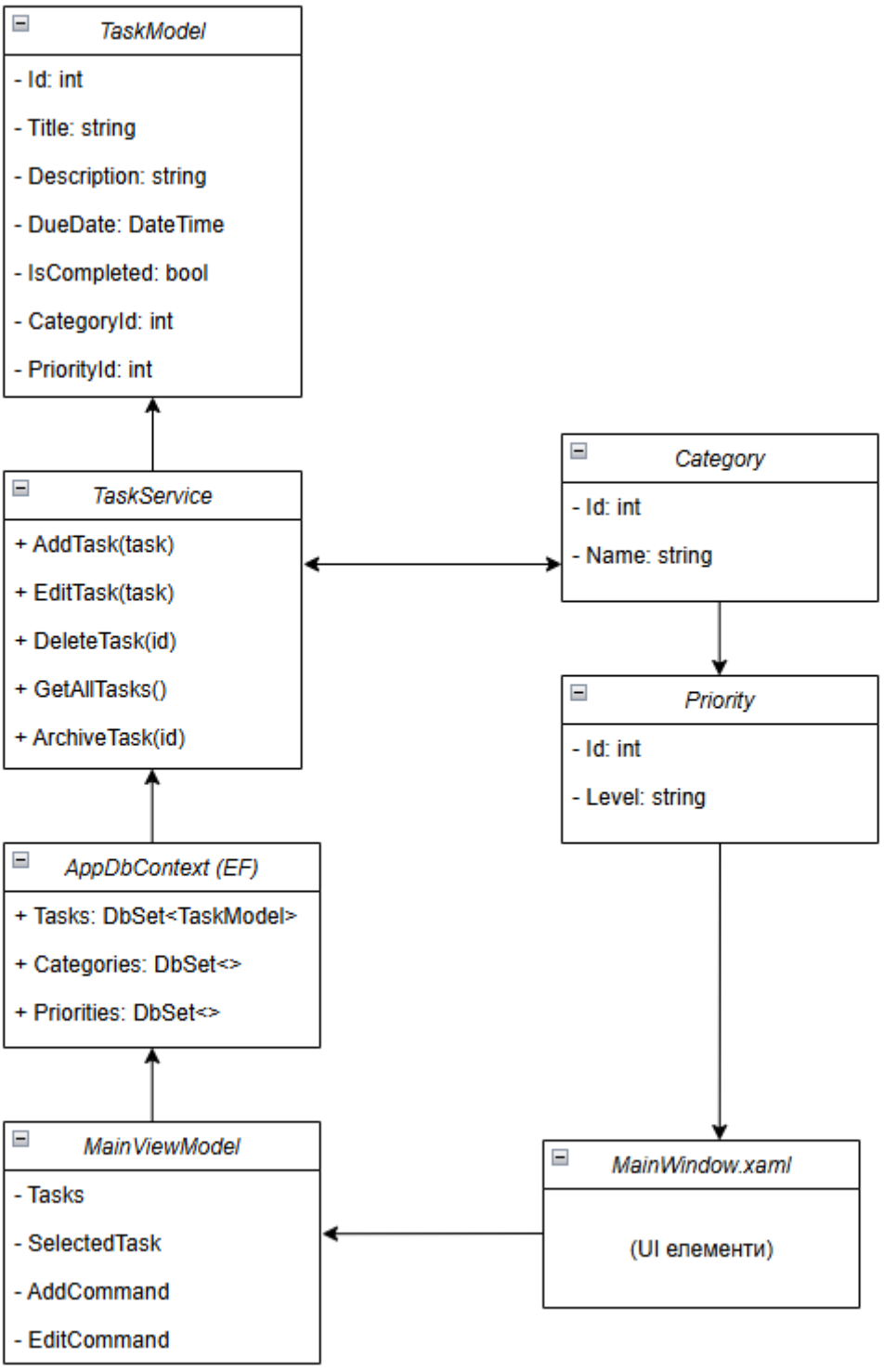


Рисунок В.2 — UML-діаграма класів системи TaskManager

ДОДАТОК Г

ІЛЮСТРАТИВНА ЧАСТИНА

КОМП’ЮТЕРНА СИСТЕМА ДЛЯ ПЛАНУВАННЯ ТА КОНТРОЛЮ ЗАДАЧ КОРИСТУВАЧА

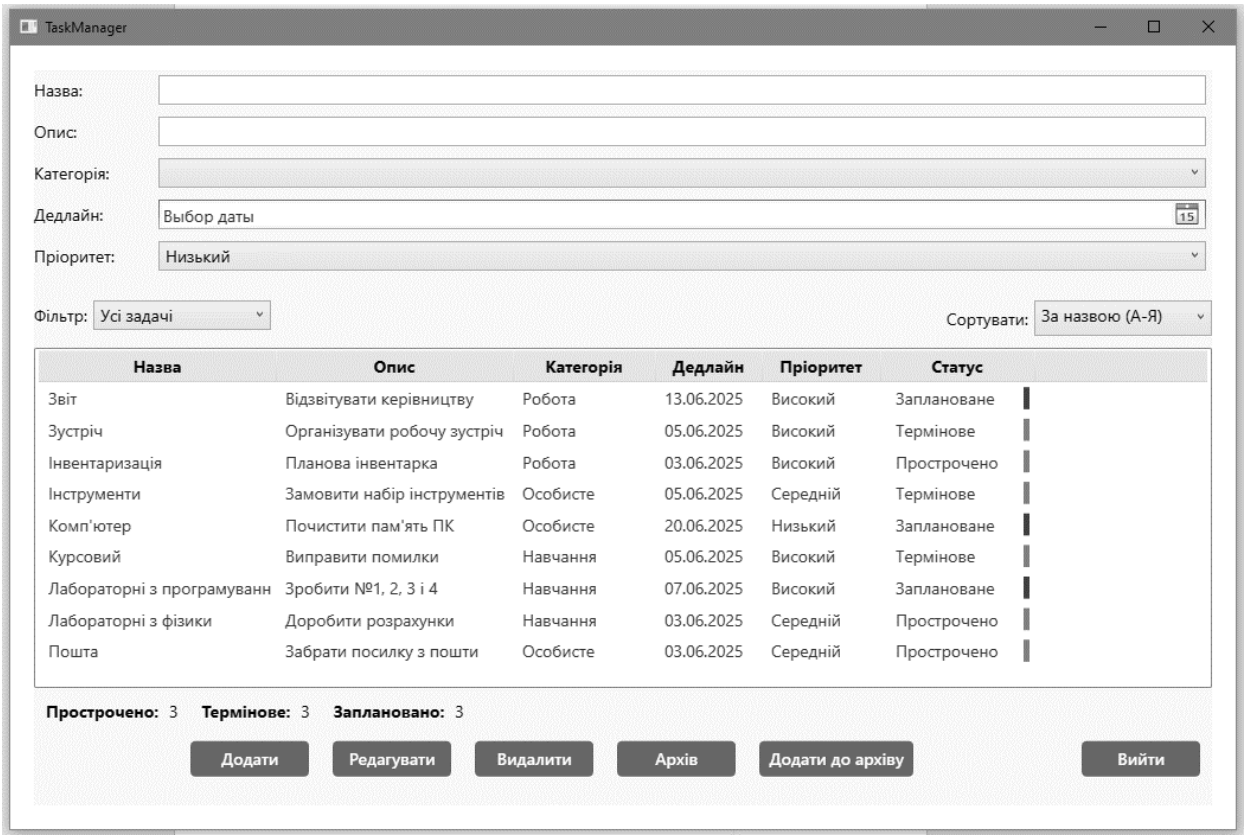


Рисунок Г.1 — Вигляд головної сторінки програмного застосунку

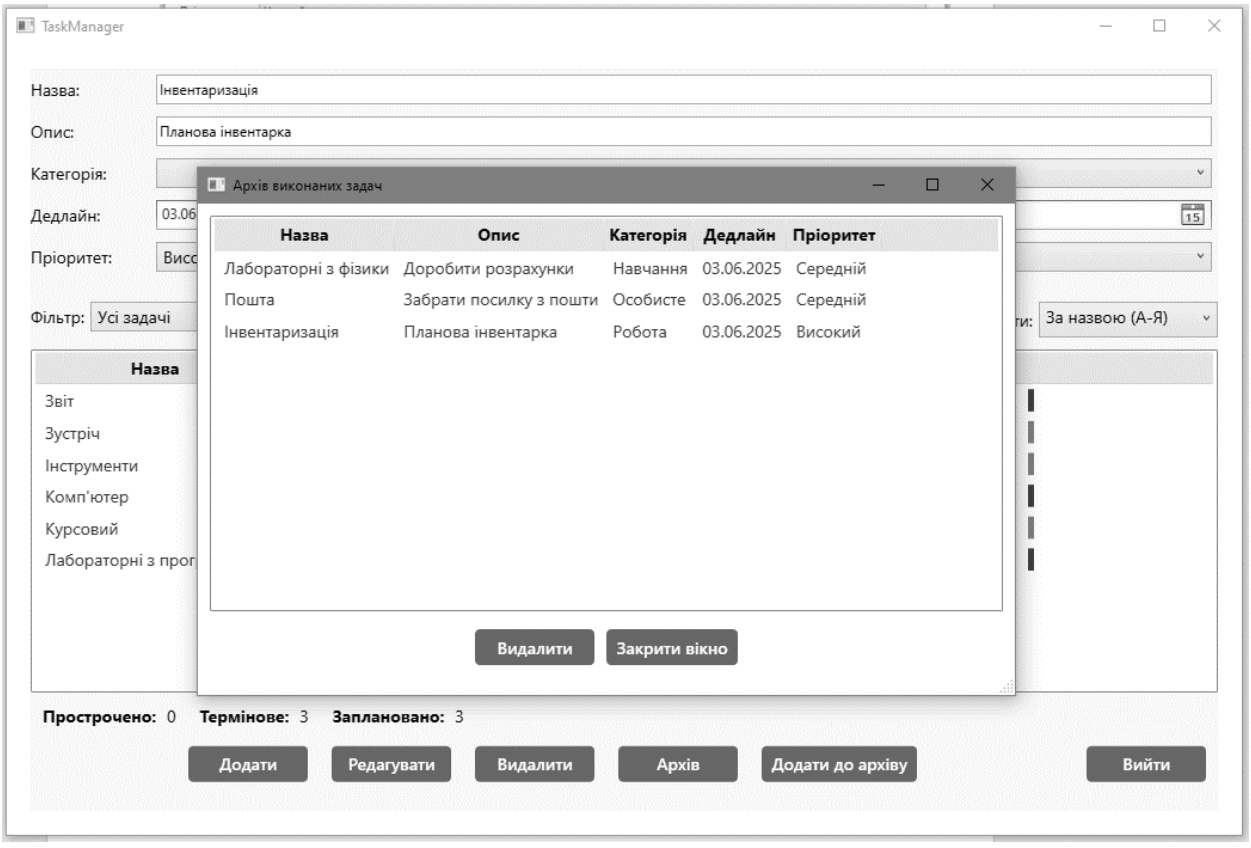


Рисунок Г.2 — Вигляд вікна для зберігання виконаних задач

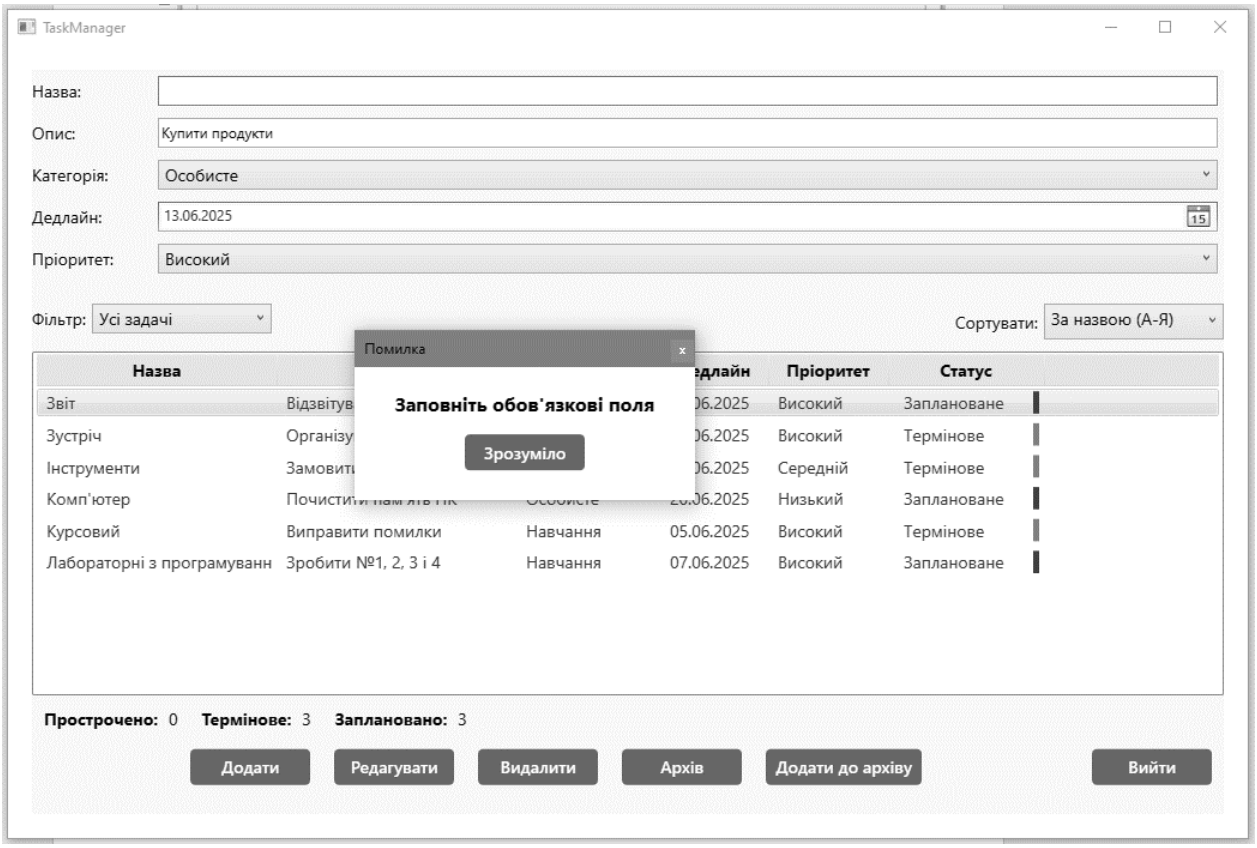


Рисунок Г.3 — Реакція програми на неправильне введення полів

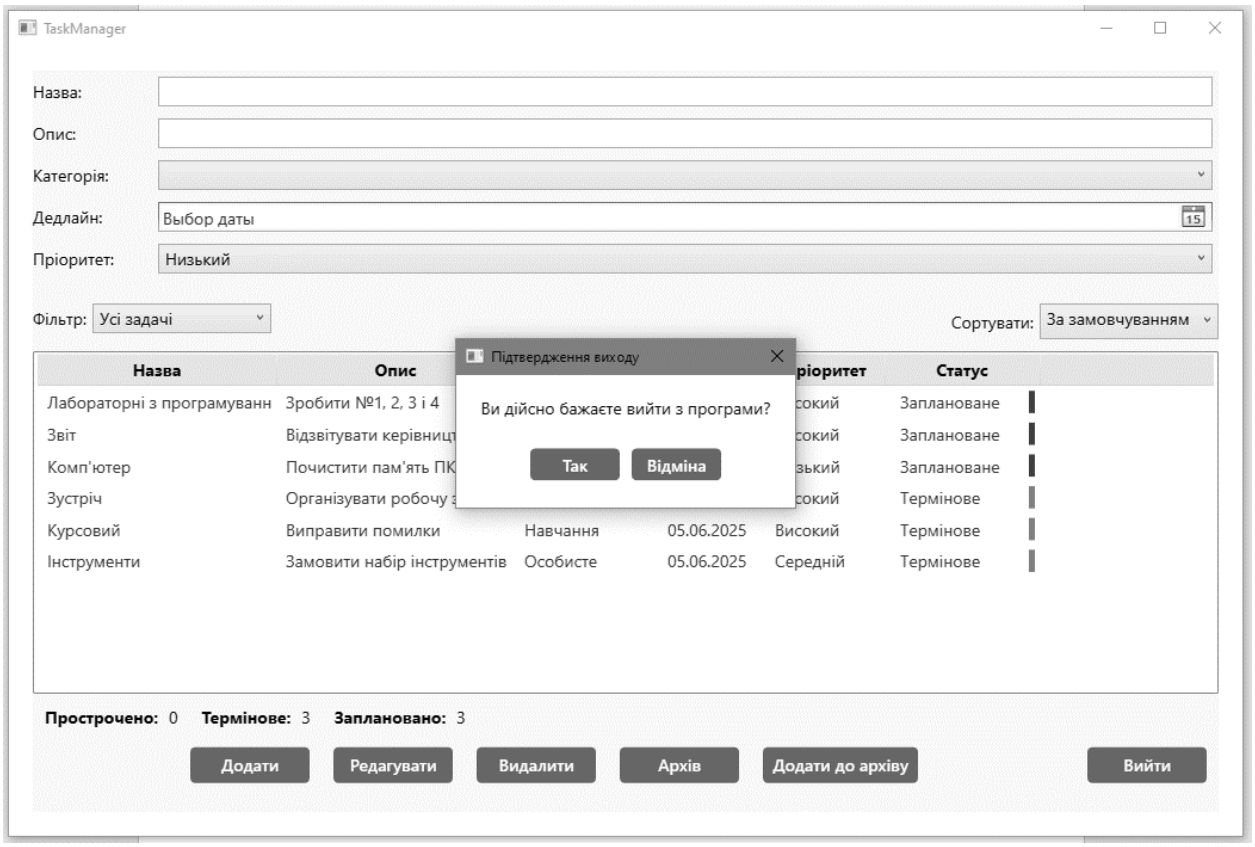


Рисунок Г.4 — Вікно підтвердження виходу з програмного застосунку

ДОДАТОК Д

Лістинг Д.1 — Основна логіка класу MainViewModel

```
using System.Collections.ObjectModel;
using TaskManager.Models;
using TaskManager.Services;
using System.ComponentModel;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;
using System.Linq;
using System.Collections.Generic;
using System;

namespace TaskManager.ViewModels
{
    public class MainViewModel : INotifyPropertyChanged
    {
        private readonly TaskService _taskService;

        public ObservableCollection<TaskModel> Tasks { get; } = new();

        public MainViewModel(AppDbContext context, TaskService taskService)
        {
            _context = context;
            _taskService = taskService;

            LoadCategoriesAsync();
        }
        public async Task AddTask(TaskModel task)
        {
            _taskService.AddTask(task);
            await LoadTasksAsync();
            UpdateTaskCounters();
        }
        public async Task UpdateTaskAsync(TaskModel task)
        {
            await _taskService.UpdateTaskAsync(task);
        }

        public async void RemoveTask(TaskModel task)
        {
            _taskService.RemoveTask(task);
            await LoadTasksAsync();
            UpdateTaskCounters();
        }

        public void UpdateTask(TaskModel updatedTask)
        {
            _taskService.UpdateTask(updatedTask);
            OnPropertyChanged(nameof(Tasks));
            UpdateTaskCounters();
        }

        public event PropertyChangedEventHandler PropertyChanged;

        protected void OnPropertyChanged(string propertyName)
        {
            PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
        }
        public async Task LoadTasksAsync()
        {
            var allTasks = await _taskService.GetAllTasksAsync();
```

```
            var activeTasks = allTasks.Where(t => !t.IsCompleted).ToList();

            Tasks.Clear();
            foreach (var task in activeTasks)
            {
                Tasks.Add(task);
            }

            await LoadCategoriesAsync();
            UpdateTaskCounters();
        }

        private readonly AppDbContext _context;
        public ObservableCollection<Category> Categories { get; } = new();
        public int? SelectedCategoryId { get; set; }

        public async Task LoadCategoriesAsync()
        {
            Categories.Clear();

            var categoriesFromDb = await _context.Categories.ToListAsync();

            if (!categoriesFromDb.Any())
            {
                var defaultCategories = new List<Category>
        {
            new Category { Title = "Навчання" },
            new Category { Title = "Робота" },
            new Category { Title = "Особисте" }
        };

                _context.Categories.AddRange(defaultCategories);
                await _context.SaveChangesAsync();

                categoriesFromDb = defaultCategories;
            }

            foreach (var category in categoriesFromDb)
            {
                Categories.Add(category);
            }

            OnPropertyChanged(nameof(Categories));
        }
        public ObservableCollection<PriorityItem> Priorities { get; set; } = new()
        {
            new PriorityItem { Id = 0, Name = "Низький" },
            new PriorityItem { Id = 1, Name = "Середній" },
            new PriorityItem { Id = 2, Name = "Високий" }
        };

        public int SelectedPriority { get; set; }

        public ObservableCollection<FilterOption> FilterOptions { get; set; } =
    new()
        {
            new FilterOption { Name = "Усі задачі", Value = "All" },
            new FilterOption { Name = "Прострочено", Value = "Overdue" },
            new FilterOption { Name = "Термінове", Value = "Urgent" },
            new FilterOption { Name = "Заплановане", Value = "Planned" },
        };

        public ObservableCollection<SortOption> SortOptions { get; set; } = new()
        {
            new SortOption { Name = "За замовчуванням", Value = "Default" },
```

```
            new SortOption { Name = "За назвою (А–Я)", Value = "ByName" },
            new SortOption { Name = "За дедлайном", Value = "ByDeadline" },
        };
        public string SelectedFilter { get; set; } = "All";
        public string SelectedSort { get; set; } = "Default";

        private int _overdueCount;
        public int OverdueCount
        {
            get => _overdueCount;
            set { _overdueCount = value; OnPropertyChanged(nameof(OverdueCount));
}
        }

        private int _urgentCount;
        public int UrgentCount
        {
            get => _urgentCount;
            set { _urgentCount = value; OnPropertyChanged(nameof(UrgentCount)); }
        }

        private int _plannedCount;
        public int PlannedCount
        {
            get => _plannedCount;
            set { _plannedCount = value; OnPropertyChanged(nameof(PlannedCount));
}
        }
        public void UpdateTaskCounters()
        {
            int overdue = Tasks.Count(t => GetStatus(t) == "Прострочено");
            int urgent = Tasks.Count(t => GetStatus(t) == "Термінове");
            int planned = Tasks.Count(t => GetStatus(t) == "Заплановане");

            OverdueCount = overdue;
            UrgentCount = urgent;
            PlannedCount = planned;
        }
        private string GetStatus(TaskModel task)
        {
            if (task.DueDate < DateTime.Now)
                return "Прострочено";
            else if ((task.DueDate - DateTime.Now).TotalHours <= 24)
                return "Термінове";
            else
                return "Заплановане";
        }

    }
}
```