Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр
(назва освітнього ступеня)

на тему: «Розробка високорівневої програмної платформи для управління подіями та тайм-менеджментом на основі WPF»

Виконав(ла): студент(ка) VI курсу, групи СПм-61
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва спеціальності)

_____ Масловський В.Б.
(підпис) (прізвище та ініціали)

Керівник _____ Цуприк Г.Б.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Стоянов Ю.М.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Петрик М.Р.
(підпис) (прізвище та ініціали)

Рецензент _____ Тиш Є. В.
(підпис) (прізвище та ініціали)

Тернопіль 2025

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра Програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Петрик М. Р.
(підпис) (прізвище та ініціали)
« » 2025 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістр
(назва освітнього ступеня)

за спеціальністю 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

студенту Масловському Віталію Богдановичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка високорівневої програмної платформи для управління подіями та тайм-менеджментом на основі WPF

Керівник роботи Цуприк Галина Богданівна, канд. техн. наук, доц.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від « 27 » листопада 2025 року № 4/7 - 1045
2. Термін подання студентом завершеної роботи 22 грудня 2025 р.
3. Вихідні дані до роботи Завдання на розробку програмної платформи

4. Зміст роботи (перелік питань, які потрібно розробити)
Сформулювати, погодити та затвердити тему кваліфікаційної роботи; розробити та затвердити завдання; провести аналіз предметної області тайм-менеджменту та огляд існуючих програмних рішень; сформулювати мету та задачі дослідження; описати технології та інструментальні засоби розробки; спроєктувати архітектуру програмної системи з використанням MVC-подібного підходу; розробити модель даних та структуру локальнох бази даних SQLite; реалізувати програмну платформу для управління подіями та тайм-менеджментом; розробити графічний інтерфейс користувача з використанням WPF та XAML; розробити план тестування програмного продукту; виконати аналіз питань охорони праці та безпеки в надзвичайних ситуаціях; сформулювати висновки за результатами виконаної роботи.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Ілюстративна частина кваліфікаційної роботи оформляється у вигляді слайдів та включає: тему, мету, об'єкт і предмет дослідження; перелік основних задач кваліфікаційної роботи; діаграму варіантів використання системи; архітектурну схему програмної платформи; UML-діаграму основних класів; ER-схему бази даних SQLite; знімки екрана розробленого програмного забезпечення; результати функціонального тестування.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Вибір та погодження теми з керівником. Створення та затвердження технічного завдання	16.06.2025 – 22.06.2025	виконано
2.	Аналіз технічного завдання, підбір та аналіз бібліографічних матеріалів необхідних для виконання кваліфікаційної роботи	23.06.2025 – 01.07.2025	виконано
3.	Проєктування та розробка програмного забезпечення.	22.09.2025	виконано
4.	Тестування та дослідна експлутація програмного забезпечення.	17.11.2025	виконано
5.	Створення допоміжної документації. Написання та перевірка на відповідність вимогам пояснювальної записки.	01.12.2025	виконано
6.	Апробація результатів роботи	17-18.12.2025	виконано
7.	Перевірка програмною антиплагіат	03.12.2025	виконано
8.	Охорона праці	06.12.2025	виконано
9.	Безпека в надзвичаний ситуація	06.12.2025	виконано
10.	Формування записки кваліфікаційної роботи	11.12.2025	виконано
11.	Нормоконтроль	12.12.2025	виконано
12.	Попередній захист	15.12.2025	виконано
13.	Рецензування кваліфікаційної роботи	22.12.2025	виконано
14.	Захичт кваліфікаційної роботи	23.12.2025	виконано

Студент _____ Масловський В. Б.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Цуприк Г. Б.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Масловський Віталій Богданович. Розробка високорівневої програмної платформи для управління подіями та тайм-менеджментом на основі WPF. Кваліфікаційна робота освітнього ступеня «магістр». Тернопільський національний технічний університет імені Івана Пулюя, кафедра програмної інженерії, група СПм-61 спеціальність 121 «Інженерія програмного забезпечення». Тернопіль, 2025. Сторінок 79, таблиць 1, рисунків 20, додатків 2, бібліографічних посилань 32.

Метою роботи є створення високорівневої програмної платформи для управління подіями та тайм-менеджментом із використанням технології (WPF), що забезпечує зручне середовище для організації часу користувача.

Об’єктом дослідження є процес створення високрівневої програмної платформи для управління подіями та тайм-менеджментом на основі WPF.

Предметом дослідження є методи та технології розробки настільного програмного забезпечення для керування календарем, подіями та нагадуваннями, реалізовані засобами C#, WPF, SQLite та MVC-подібної архітектури.

Методи дослідження охоплюють аналіз існуючих програмних рішень у сфері тайм-менеджменту, проєктування системної архітектури, моделювання бази даних, розробку користувацького інтерфейсу та тестування основних функціональних модулів.

У межах роботи описано весь цикл створення автономного настільного застосунку для управління подіями. У проєкті реалізовано програмну платформу з підтримкою авторизації користувачів, локальним збереженням даних у базі SQLite, графічним інтерфейсом на базі WPF, а також системою нагадувань. Функціонал застосунку включає можливість додавання, редагування, видалення та фільтрації подій, забезпечуючи своєчасні сповіщення для зручності користувачів.

Ключові слова: тайм-менеджмент, WPF, C#, SQLite, MVC-подібна архітектура, календар подій, нагадування, програмна платформа.

ABSTRACT

Maslovsky Vitaly Bogdanovich. Development of a high-level software platform for event management and time management based on WPF. Qualification work for the educational degree of “Master”. Ivan Pul'uj Ternopil National Technical University, Department of Software Engineering, Group SPm-61, Specialty 121 “Software Engineering.” Ternopil, 2025. 79 pages, 1 tables, 34 figures, 2 appendices, 32 bibliographic references.

The goal of this work is to create a high-level software platform for event management and time management using technology (WPF), which provides a convenient environment for organizing the user's time.

The object of research is the process of creating a high-level software platform for event management and time management based on WPF.

The subject of the study is the methods and technologies for developing desktop software for managing calendars, events, and reminders, implemented using C#, WPF, SQLite, and MVC-like architecture.

The research methods include analysis of existing software solutions in the field of time management, system architecture design, database modeling, user interface development, and testing of basic functional modules.

The work describes the entire cycle of creating a standalone desktop application for event management. The project implements a software platform with user authorization support, local data storage in an SQLite database, a WPF-based graphical interface, and a reminder system. The application's functionality includes the ability to add, edit, delete, and filter events, providing timely notifications for user convenience.

Keywords: time management, WPF, C#, SQLite, MVC-like architecture, event calendar, reminders, software platform.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

MVC – Model–View–Controller, архітектурний шаблон розробки ПЗ

WPF – Windows Presentation Foundation, графічний фреймворк .NET

IDE – Integrated Development Environment (інтегроване середовище розробки)

UI/UX – User Interface / User Experience (інтерфейс користувача / користувацький досвід)

SQL – Structured Query Language (мова структурованих запитів до бази даних)

API – Application Programming Interface – інтерфейс прикладного програмування

GUI – Graphical User Interface – графічний інтерфейс користувача

.NET – платформа для розробки застосунків від Microsoft

C# – об'єктно-орієнтована мова програмування

WPF – Windows Presentation Foundation – технологія для створення графічних застосунків

Visual Studio – середовище розробки (IDE) для програмування на C#, C++, .NET тощо

XAML – Extensible Application Markup Language – мова розмітки інтерфейсів WPF

БД – база даних

ER (Entity-Relationship) – сутність-зв'язок

N – кількість користувачів системи, осіб

ЗМІСТ

ВСТУП

У сучасних умовах цифровізації та зростаючого обсягу інформації проблема ефективного управління часом стає особливо актуальною. Величезна кількість навчальних, професійних і побутових завдань потребує чіткої організації, ретельного планування та контролю виконання. У постійній багатозадачності люди часто стикаються з ризиком пропуску важливих подій, дедлайнів чи зустрічей, що негативно позначається на продуктивності та якості життя загалом. З цієї причини програмні засоби тайм-менеджменту відіграють важливу роль у забезпеченні організованості користувачів.

Хоча на ринку присутня велика кількість рішень для планування часу, більшість із них орієнтовані на мобільні платформи, вимагають постійного інтернет-з’єднання або мають надто складний інтерфейс. Це створює необхідність розробки автономного настільного застосунку, який поєднує простоту використання, зручний графічний інтерфейс і локальне збереження даних із високою надійністю. Особливу увагу слід приділяти архітектурі програмної системи, її масштабованості та потенціалу для подальшого розширення функціональних можливостей.

У магістерській роботі розглядається створення програмної платформи для управління подіями та тайм-менеджменту на базі технології Windows Presentation Foundation (WPF). Ця платформа дає змогу розробляти сучасні настільні застосунки з багатим інтерфейсом, підтримкою прив’язки даних і чітким розділенням логіки від представлення.

Мета роботи полягає в розробці платформи для управління подіями та тайм-менеджменту з опорою на WPF. Вона має забезпечувати легке створення, редагування, збереження та перегляд подій, включати механізм нагадувань і реалізувати базові функції контролю доступу користувачів.

Для досягнення мети було поставлено такі задачі:

- проаналізувати галузь тайм-менеджменту та існуючі програмні рішення;

- сформулювати функціональні й нефункціональні вимоги до системи;
- обґрунтувати вибір технологій (C#, .NET, WPF, SQLite);
- спроєктувати архітектуру застосунку за модельно-орієнтованим підходом MVC;
- розробити модель даних і структуру локальної бази даних;
- побудувати графічний інтерфейс користувача за допомогою XAML;
- реалізувати механізми авторизації, управління подіями та нагадувань;
- протестувати систему й перевірити її коректність роботи;
- описати процес впровадження і використання застосунку.

Об'єктом дослідження виступає процес розробки високорівневої програмної платформи, призначеної для управління подіями та організації тайм-менеджменту, із застосуванням технології WPF.

Предмет дослідження охоплює методи, моделі та технології проєктування й створення настільного програмного забезпечення для тайм-менеджменту. У процесі роботи використовуються мова програмування C#, технологія WPF, локальна база даних SQLite, а також архітектура, що нагадує концепцію MVC.

Обов’язковим компонентом кваліфікаційної роботи магістра є апробація отриманих результатів. Основні висновки магістерської роботи були оформлені у вигляді тез доповіді та подані для публікації у збірнику матеріалів XIII науково-технічної конференції «Інформаційні моделі, системи та технології», яка проводитиметься в період 17-18 грудня 2025 року.

1. АНАЛІЗ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

У цьому розділі проводиться аналіз предметної області, а також визначаються ключові вимоги до програмної платформи для управління подіями та тайм-менеджментом. Розглядаються потреби користувачів, основні цілі системи та можливі сценарії її застосування.

1.1. Аналіз предметної області

Сьогоднішній світ кожного разу вимагає від людей все більшої організованості та ефективного керування часом. Саме тому з даної причини програмні засоби для планування подій стають все більше і більше необхідними інструментами для багатьох людей, що не залежить від їхньої сфери діяльності. Однією із головних потреб користувачів має бути змога зручно та швидко створювати події, а також нагадувати про них. При цьому досить важливо мати можливість налаштовувати час нагадування, щоб відповідати особистим потребам та графіку дня.

Крім того, більшість користувачів цінують зручний інтерфейс, який буде дозволяти легко взаємодіяти із програмою та швидко знаходити усю необхідну інформацію. Ергономіка та простота використання є важливими у розробленні програмного забезпечення для того, щоб даний засіб був доступним для досить широкого кола користувачів.

Зростаюча популярність мобільних технологій також змогла підкреслити значення наявності синхронізації із іншими пристроями та платформами. Переважно більшість користувачів все ж таки хочуть мати можливість отримувати нагадування на своїх смартфонах, планшетах та комп’ютерах, що не залежить від того, на якому пристрої вони працюють. Аналіз потреб користувачів у програмних засобах для планування подій показав, що ідеальний інструмент повинен поєднати

в собі не тільки зручний інтерфейс, а і також розширені можливості для налаштування подій і нагадувань, а також синхронізацію із іншими пристроями.

Звісно, що крім базових функцій, сучасні програмні засоби для планування подій повинні відповідати багатьом специфічним вимогам користувачів. Можливість гнучкого налаштування – це одна із таких, яка буде включати не лише налаштування часу нагадувань, але і персоналізацію інтерфейсу, зокрема вибір тем оформлення, шрифти або кольорові схеми. Саме такі функції будуть дозволяти кожному користувачеві адаптувати програму під свої власні потреби та стиль виконання. Як приклад, якщо використовувати кольорові маркери для категоризації подій, то тоді це буде допомагати візуально розділяти їх на різні види завдань або пріоритети, що тим самим буде значно спрощувати орієнтацію у щільному розкладі.

Ще одним із важливих аспектів є забезпечення роботи у режимі офлайн. Хоч і технології значно йдуть вперед, проте не завжди користувач має стабільний доступ до інтернету, саме тому можливість переглядати та змінювати свій розклад у будь який момент є критичною. Окрім того, важливо також навчитися передбачувати надійну систему резервного копіювання даних, що буде дозволяти уникати втрати важливої інформації. Кібербезпека стає все більше сучасною та актуальною у теперішньому світі, саме тому користувачі очікують, що їхні дані будуть захищені від несанкціонованого доступу. Тому інтеграція сучасних засобів шифрування та захисту є однією із невід’ємних частин для створення якісного програмного забезпечення.

Для багатьох користувачів інтеграція програмного забезпечення із іншими сервісами, такими як хмарні сховища, електронна пошта, месенджери або інші календарі є дуже важливою та необхідною. Адже це саме це буде надає дозвіл об’єднувати робочі процеси та особисте життя в єдину екосистему, що буде значно підвищувати зручність використання додатка. Одним із хороших прикладів, є можливість автоматично створювати події на основі отриманих листів або повідомлень, що як раз таки економить час та уникає помилок, які можуть бути пов’язані із ручним введенням інформації.

Включення технологій штучного інтелекту (ШІ) також може значно підвищувати функціональність програмного забезпечення для того, щоб можна було ефективно планувати події. Як приклад, ШІ може аналізувати розклад користувача та запропонувати досить оптимальні часові слоти для нових подій. Окрім того, функції прогнозування мають можливість передбачити певні конфлікти між подіями та рекомендувати зміни, що буде забезпечувати більше ефективне керування часом. Сучасне програмне забезпечення ще має враховувати міжнародні потреби користувачів. Тобто, можливість вибору мови інтерфейсу та враховувати регіональні особливості, такі як формат дати та часу, є досить таким важливим для того, щоб забезпечити доступності додатка для ще більше широкої аудиторії.

Синхронізація програмного забезпечення для планування подій із популярними голосовими асистентами, такими як Siri, Google Assistant або Alexa, наддасть дозвіл користувач легко створювати події та змінювати події за допомогою голосових команд. Саме це буде забезпечувати зручність та швидкість, особливо для певних користувачів, які перебувають у русі .

Таким чином, сучасні програмні засоби для планування подій мають не просто забезпечувати базові функції нагадувань, але і відповідати сучасним вимогам щодо їхньої персоналізації, безпеки, інтеграції та доступності. Лише поєднання даних факторів дозволить створювати продукт, який дійсно буде відповідати потребам сучасного користувача, а головне сприяти ефективному управлінню часом.

1.2. Постановка завдання та цілей

Розроблення додатку вимагає вирішення конкретних задач. Для початку потрібно розробити зручний та ергономічний користувацький інтерфейс, що буде дозволяти користувачам легко та швидко створювати події, налаштовувати нагадування та взаємодіяти з програмою.

Наступним кроком у розробці є реалізація модуля керування подіями, який буде відповідати за зберігання та організацію списку подій, що буде включати можливість їх редагування та видалення. Третя задача – це створити модуль нагадування, що буде відповідати за вчасне та точне нагадування користувачам про наближення якоїсь певної події згідно встановлених параметрів.

Розроблення механізму, що буде зберігати дані, які будуть забезпечувати надійне збереження інформації про події та нагадування між сеансами роботи програми. Останнім завершальним етапом є тестування та валідація додатку для перевірення правильності його роботи, а також вияв та виправлення можливих помилок чи недоліків. Вирішення даних задач наддасть дозвіл досягнути поставленої мети розроблення додатку і забезпечити користувачам зручний та функціональний інструмент для керування своїм часом.

Вирішення поставлених задач потребує впровадження сучасних технологій та підходів у процес розроблення програмного забезпечення. Зручний та ергономічний інтерфейс повинен стати ключовим аспектом, що буде забезпечувати легкість взаємодії з програмою для користувачів різного рівня цифрової грамотності. Дизайн має бути в першу чергу продуманим та інтуїтивно зрозумілим, для того, щоб можна було скоротити час на освоєння додатка та зробити його використання максимально комфортним для користувачів. Використання певних графічних засобів, такі як WPF, відкриває широкі можливості для створення інтерфейсу, який не тільки буде виглядати сучасно, але і повинен враховувати усі аспекти доступності для людей з особливими потребами.

Реалізація модуля керування подіями має включати не просто базові функції зберігання та редагування списку подій, але і змогу пошуку та сортувати інформацію за категорією чи пріоритетами. Саме це буде дозволяти користувачам ефективно організувати великий обсяг даних без витрати зайвого часу, що в теперішньому житті є досить важливим. Окрім цього, розроблення механізму нагадувань повинна врахувати потребу у гнучкому налаштуванні параметрів, а саме такі як частота нагадувань або способи доставки, що буде включати не тільки повідомлення на смартфон, але і на електронному пошту чи інші платформи. Така

функціональність буде допомагати робити систему нагадувань максимально адаптивною до індивідуальних вимог користувачів, а найголовніше зручною у використанні .

Таким чином, особливу увагу слід і приділяти механізму збереження даних. Адже крім локального зберігання, важливим є і забезпечення функції синхронізації із хмарними сервісами, що буде гарантувати доступ до інформації з будь якого пристрою. Це також посприяє і створенні резервних копій, що має знижувати ризики втрати важливих даних. Тестування програми слід проводити у декілька етапів, що в першу чергу повинно включати перевірку на функціональність, продуктивність, сумісність із різними платформами, а також зручність використання. Валідація додатка буде дозволяти не тільки виявляти помилки, але і забезпечувати відповідність очікуванням кінцевих користувачів, що дає можливість гарантувати стабільну та ефективну роботу продукту.

1.3. Пошук акторів та варіантів використання

Визначення цільової авдиторії є одним із ключових етапів у процесі розроблення будь якого програмного продукту, адже саме це надає змогу визначити потреби, функціональні очікування та вимоги до інтерфейсу додатку. Аналіз цільової авдиторії дозволить адаптувати програмне забезпечення під реальні умови його використання, забезпечити зручність взаємодії та підвищувати ефективність використання системи. Тому для побудови портрету цільової авдиторії в даному дослідження було використано методику SW Марка Шеррінгтона, яка надає змогу передбачити аналіз за п’ятьма основними питаннями. Перше питання – це «Who? (Хто?)», тобто користувачами додатку переважно є студенти, офісні працівники, фрілансери та представники творчих професій, які насамперед мають потребу у плануванні робочого дня, дедлайнів, подій, зустрічей або нагадувань. Віковий діапазон від 18 до 40 років, адже більшість користувачів володіє базовими навичками роботи з ПК або мобільними додатками.

Друге питання це «What? (Що?)» – користувачі прагнуть отримати насамперед простий та гнучкий інструмент для щоденного планування. Тому вони очікують швидкого додавання подій, нагадувань, можливості редагування, категоризації задач, а також фільтрації подій за різними критеріями. Третє питання – «Where? (Де?)», де використання додатку відбувається як у стаціонарних умовах, тобто це може бути як і робоче місце, так і навчання, та і в мобільному режимі, а саме на ноутбуках чи планшетах. Отже, додаток має бути адаптованим до різних середовищ використання. Четверте питання – «When? (Коли?)», застосування додатку може бути щоденним, із переважним акцентом на ранкове або вечірнє планування, або ж, наприклад, протягом дня – для швидкої актуалізації розкладу. І останнє п`яте питання це «Why? (Чому?)» – мотивація користувача базується на потребі підвищити продуктивність, уникнути забування важливих подій, мати чітку структуру дня або тижня, а також можливість зменшити рівень стресу, який пов’язаний із неорганізованістю.

Додатково до методики 5W ще також були враховані поради. Саме тому рекомендовано комбінувати демографічні, поведінкові та мотиваційні характеристики при створенні профілю користувача. Тому, основні типи користувачів можна згрупувати наступним чином:

- Тип 1 – студенти, які використовують додаток для занотовування пар, дедлайнів, іспитів, зустрічей із викладачами.
- Тип 2 – офісні працівники, які планують зустрічі, робочі задачі, мітинги, звіти та презентації.
- Тип 3 – фрілансери або творчі спеціалісти, що використовують для організації роботи над проєктами, домовленостей із клієнтами, дотримання графіку.
- Тип 4 – молоді батьки або сім’ї, переважно використовують для того, щоб запланувати домашні справи, розклад дітей, візити до лікаря тощо.

Ще також на основі дослідження було сформовані типові сценарії використання додатку, які зможуть допомогти краще зрозуміти вимоги до функціоналу. Додавання події – це перший сценарій, де користувачі вранці

відкривають додаток та додають подію, наприклад, лекція, мітинг, дзвінок. Нагадування – це другий сценарій, коли користувач має змогу отримати нагадування про подію за 30 хвилин до її початку. Фільтрація – це третій сценарій, де користувачу потрібно швидко переглянути лише події, які є пов’язаними із навчанням або роботою. Модифікація – це четвертий сценарій, коли користувач змінює дату або опис події через зміну планів. Перегляд тижня – це п’ятий сценарій, який допомагає користувачу переглянути календар на 7 днів уперед для розподілу навантажень.

Таким чином, проведений аналіз цільової авдиторії надав змогу виявити реальні потреби користувачів та сформувати функціональні вимоги до додатку. Тому це забезпечило обґрунтованість архітектурних та інтерфейсних рішень, роблячи програму максимально адаптованою до очікувань та способу життя користувача.

1.4. Опис ключових варіантів використання

Ключові сценарії використання програмної платформи для управління подіями та тайм-менеджментом відображають основні способи взаємодії користувачів із системою. Вони визначають функціональні можливості платформи в рамках повсякденного використання. Формалізація цих сценаріїв дозволяє чітко окреслити логіку роботи програмного забезпечення, регламентувати послідовність дій користувачів та встановити очікувану реакцію системи на певні операції. Платформа розроблена для індивідуального використання та надає інструменти для організації часу, планування подій, управління завданнями й нагадуваннями, що є надзвичайно актуальним у сучасному інформаційно-насиченому середовищі.

Процес взаємодії з платформою починається з реєстрації або авторизації. Реєстрація нового користувача передбачає введення облікових даних, які перевіряються системою на правильність та унікальність, а потім зберігаються у базі даних. Після завершення реєстрації користувач отримує доступ до особистого кабінету, де може користуватися всіма можливостями платформи. У випадку

авторизації вже зареєстрованого користувача забезпечується безпечний доступ до раніше збережених даних та особистих налаштувань, що дає змогу системі ідентифікувати користувача для подальшої роботи. Цей базовий функціонал гарантує персоналізацію досвіду, захист інформації та забезпечує чіткий розподіл доступу до даних.

Після входу до системи користувач отримує доступ до головного робочого простору з основними функціями платформи. Один із ключових сценаріїв використання – створення подій та завдань. Користувач може додавати нові записи, вказуючи назву події, дату, час початку й завершення, опис і додаткові параметри для уточнення деталей. Процес створення супроводжується перевіркою введених даних, що допомагає уникнути помилок і забезпечити коректне збереження інформації у системі. Такий функціонал дозволяє формувати індивідуальний розклад та структурувати особисту діяльність.

Окрім створення нових записів, платформа надає можливість редагувати або видаляти існуючі події. У разі зміни обставин чи оновлення інформації користувач може швидко внести необхідні корективи, що робить систему більш гнучкою та адаптованою до змін. Видалення неактуальних даних допомагає підтримувати чистоту розкладу й уникати плутанини через застарілі записи. Усі зміни автоматично фіксуються у базі даних і відображаються в інтерфейсі в реальному часі, забезпечуючи узгодженість інформації та зручність користування платформою.

Однією з ключових можливостей програмної платформи є функція налаштування нагадувань. Користувач може встановлювати сповіщення для окремих подій, зазначаючи час їх активації. Завдяки цьому нагадування своєчасно інформують про заплановані завдання, зменшуючи вірогідність їх пропуску. Ця функція особливо корисна для тих, хто працює з великим обсягом завдань або має насичений графік. Впровадження механізму нагадувань покращує практичну цінність платформи та сприяє ефективнішому плануванню й управлінню часом.

Для оптимізації роботи із численними подіями реалізовано інструменти пошуку та фільтрації даних. Користувач може шукати події за ключовими словами,

а також використовувати фільтри за датами чи часовими проміжками. Це значно полегшує швидкий доступ до потрібної інформації та допомагає аналізувати графік у визначений період. Такі можливості помітно підвищують зручність користування системою та скорочують час на обробку даних.

Платформа підтримує різні режими відображення подій, що дозволяє користувачам обирати найбільш зручний формат перегляду. Це полегшує аналіз розкладу, сприяє кращому розумінню запланованих завдань і допомагає ефективно спланувати подальші дії. Такий підхід забезпечує максимально наочну подачу інформації та робить використання програми комфортнішим.

Усі дії в системі супроводжуються перевіркою коректності введених даних і логіки виконуваних операцій. Інформація автоматично зберігається в базі даних з гарантією її цілісності, що забезпечує стабільність і надійність роботи продукту. Реалізовані механізми сприяють зручності, безпеці та передбачуваній роботі програмної платформи.

Таким чином, зазначені функції охоплюють основні сценарії використання платформи для управління подіями та тайм-менеджменту, відповідаючи сучасним вимогам інформаційних систем. Набір можливостей платформи забезпечує ефективну організацію часу, комфортну взаємодію з користувачем і відкриває перспективи її подальшого вдосконалення відповідно до нових потреб.

1.5. Загальна інформація про органайзери

Ефективне керування часом – це одне із актуальних питань для будь якої людини, що взагалі не залежить від її професії. Даній дисципліни, відомій як тайм-менеджмент, присвячено не одну книгу, де регулярно проводиться багато курсів, які пропонують різні техніки.

Організатори, які допомагають із плануванням часу – саме цю область програмного забезпечення використовують не тільки для встановлення порядку денного, але і для зберігання даних. Переважно більшості людям буває досить складно запам’ятати або зберегти усю необхідну інформацію, яка повинна бути під

рукою в невпорядкованому вигляді. Розробники стараються вкладати чимало грошей, і саме тому ретельно вивчають кожний крок для того, щоб зрозуміти конкретні потреби людей. Для них ця робота називається дослідженням досвіду користувачів, і таким чином, вона має враховувати всі емоції, переконання, уподобання, відчуття, фізіологічні та психологічні реакції, поведінку та досягнення, які виникають до, під час і після використання системи.

Дизайн інтерфейсу користувача – це процес, який використовують дизайнери для створення інтерфейсів у програмному забезпеченні чи комп'ютеризованих пристроях, де особлива увага приділяється зовнішньому вигляду чи стилю. Дизайнери прагнуть створювати інтерфейси, які для користувачів будуть простими у використанні та просто приємними. Окрім того, інтерфейс повинен додавати цінність бренду та підвищувати довіру користувачів. Селектор слід використовувати для вибору значення зі списку. В одному інструменті вибору можна використати лише декілька типів списків. Одним із хороших прикладів його застосування є вибір дати народження, де слід використовувати вибір дня, місяця та року в одному вікні.

Рис. 1.1 – Селектор

Проєктування інтерфейсу менеджера є одним із важливим елементів у розробленні, саме тому слід звертати увагу на певні аспекти. Зручність інтерфейсу – один із аспектів, де органайзер це мобільний додаток, в якому інтерфейс відіграє досить ключову роль, особливо з точки зору швидкості доступу до необхідних функцій. Завдання та календар – функції календаря під час створення завдань, списків справ, категорії, підзадач та створення подій. Нагадування та сповіщення –

тут потрібно встановити нагадування про події, дні народження і тому подібне, можна створювати листівки та методи спілкування. Організація контактів – це функціональність адресної книги, імпорт чи експорт даних. Спільний доступ – змога створювати проєкти, інструменти для колективної роботи. Мобільність та синхронність – це наявність мобільних платформ, інтеграція з онлайн-сервісами. Безпека – що є дуже важливим аспектом, де потрібно встановлювати паролі для запуску програм, баз даних або окремих частин органайзера, захист даних та шифрування.

Розроблення інтерфейсів органайзерів вимагає того, щоб було враховано принцип адаптивності. Сучасний користувач взаємодіє із програми на різних пристроях, таких як смартфони, планшети та комп’ютери. Саме тому створення інтерфейсів, які будуть досить коректно відображатися на будь якому екрані, є надзвичайно важливим, адже це буде зберігати зручність та функціональність. Адаптивний дизайн зможе забезпечити цілісне користувацьке враження незалежно від платформи. Досить таким ключовим аспектом є використання кольорової гами та типографіки. Адже вдалий вибір кольорів, що будуть відповідати стилю та бренду програми, може як раз таки посприяти більш кращому сприйняттю, що буде відповідати стилю та бренду програми. Саме це відіграє важливу роль, адже це посприяє кращому сприйняттю інформації користувачем. Контрастні кольори та великі шрифти надають змогу краще виділити ключові функції або важливі функції. Проте, при цьому, слід дотримуватися балансу між естетикою та зручністю.

Інтерактивні елементи інтерфейсу, такі як кнопки, повзунки та випадаючі меню, впершу чергу повинні бути інтуїтивно зрозумілі для користувача. Наприклад, важливим буде забезпечити достатній розмір та зручне розташування кнопок для роботи на сенсорних екранах. Саме це покращить користувацьке враження, адже візуальний зворотній зв'язок, такий як підсвітка кнопок або анімація при взаємодії із елементами додають більше елегантності та одночасно зручності. Не менш ключовим є створення досить чіткої та логічної структури навігації. Користувач повинен мати змогу досить швидко знаходити усі необхідні

функції без зайвих зусиль. Для цього слід використовувати зрозумілі іконки, вкладки та пошук, який буде забезпечувати доступ до даних. Інтуїтивна навігація є однією із основ комфортного використання органайзера.

В загальному, робота типової програми-органайзера є пов’язаною з багатьма функціями. Основні із них є календар, зв'язок із менеджером, блокнот, записування події, які пов’язані із конкретною датою або часом. Планування завдання є також досить ключовим, адже це змога контролювати їх незалежного або стороннього виконання. Будильника та послуга нагадування про події, що будуть встановлені користувачами та змога використати електронну пошту – це ще одні із важливих функцій.

2. ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОЇ ПЛАТФОРМИ

У цьому розділі розглянуто процес створення та впровадження програмної платформи для управління подіями та тайм-менеджментом. Розглянуто вибір архітектурного підходу, технологій і інструментів розробки, а також описано структуру програмних компонентів і бази даних.

2.1. Вибір процесу розробки

Проєктування та впровадження програмної платформи для управління подіями та тайм-менеджментом у середовищі WPF здійснювалися за ітеративною моделлю, близькою до гнучких підходів (Agile). Цей метод вибрано через специфіку настільних застосунків такого типу, у яких функціональність формується поступово – від мінімального життєздатного прототипу до повністю розвиненої системи зі підтримкою авторизації, збереження даних, редагування подій та системою нагадувань [1,2,12].

Ітеративний процес об’єднав риси класичних методологій, наприклад, формалізоване визначення функціональних вимог, проєктування структури бази даних і архітектури застосунку, з перевагами гнучкого підходу – поетапним розвитком функціоналу і можливістю оперативно вносити зміни. На початковій стадії було створено основний каркас застосунку, що охоплював ключові вікна, структуру даних та основні сценарії взаємодії користувача із системою. Далі функціональність вдосконалювалась у серії послідовних ітерацій [1, 5].

Завершення кожної ітерації знаменувалося реалізацією логічно завершеного функціонального модуля, який легко тестувався в реальній роботі програми. До таких модулів відносилися функції створення та перегляду подій у календарі, збереження та завантаження даних із локальної бази, авторизація користувачів, редагування і видалення подій, налаштування фільтрів і сортування списку подій, а також система нагадувань. Такий метод дозволив розширювати функціональність поступово без ризику втрати стабільності вже реалізованих частин системи.

Особливе значення ітеративний підхід набув через специфіку тематики тайм-менеджменту. Тут важливу роль відіграють зручність інтерфейсу і поведінка його елементів. Вимоги до відображення дат і часу, функції фільтрації подій, способу підтвердження дій користувача й загального стилю інтерфейсу уточнювалися безпосередньо в процесі роботи з прототипом. Це забезпечило можливість швидкого впровадження змін без необхідності переглядати архітектурні рішення.

Окрім того, цей підхід позитивно вплинув на якість кінцевого продукту. Кожна нова функція тестувалася з урахуванням уже реалізованих сценаріїв використання, що сприяло своєчасному виявленню помилок і їх усуненню ще на ранніх стадіях. Завдяки продуманій організації процесу розробки вдалося досягти стабільності платформи, мінімізувати ризик серйозних помилок у системі та створити міцну основу для майбутнього розширення функцій застосунку [2,6,7].

2.2. Порівняння WPF з альтернативами UWP, Win Forms

У процесі розроблення платформи потрібно було розглянути насамперед три основні технології для створення графічного інтерфейсу користувача: Windows Forms (Win Forms), Windows Presentation Foundation (WPF) та Universal Windows Platform (UWP). Кожна із них має свої певні особливості, переваги та обмеження.

WinForms – це одна із найстаріших технологій для того, щоб створювати GUI-додатки у середовищі .NET. Вона базується на бібліотеці GDI+ та надає можливість розробникам простий спосіб створення форм та елементів управління. Перевагами даної технології є простота у використання та швидке створення прототипів, а також широка підтримка у спільноті та велика кількість навчальних матеріалів. Ще однією особливістю є те, що вона підходить для простих додатків із невеликою кількістю візуальних ефектів. Проте, незважаючи на значні переваги, дана технологія має певні недоліки. А саме обмежені можливості щодо сучасного дизайну та анімацій, меншу гнучкість у налаштування інтерфейсу порівняно із новішими технологіями. Також використання застарілих технологій рендерингу, що може вплинути на продуктивність [9].

Windows Presentation Foundation (WPF) – це сучасна та потужна технологія, яка служить для створення багатофункціональних настільних додатків у середовищі .NET. Суттєвою перевагою WPF є використання мови розмітки XAML для опису користувацьких інтерфейсів, що забезпечує зручність проєктування і модифікації додатків. Для рендерингу, WPF базується на технології DirectX, яка дозволяє досягати високої якості графіки та продуктивності. Однією з ключових переваг цієї платформи є можливість створювати адаптивні інтерфейси, які легко пристосовуються до екранів різних розмірів і дозволів. WPF також підтримує впровадження складних візуальних ефектів, анімацій, переходів та гнучку систему стилізації, що дозволяє налаштовувати зовнішній вигляд елементів додатка під конкретні потреби. Окрім того, технологія надає розширений механізм прив'язки даних до шаблонів, що значно спрощує роботу з динамічними даними в інтерфейсі.

Водночас, WPF має свої недоліки, серед яких слід виділити вищу складність навчання та реалізації порівняно з іншими технологіями, такими як Windows Forms. Для розробників-початківців процес освоєння WPF може бути більш тривалим через багатство функцій і потужність платформи. Крім того, для забезпечення стабільної високої продуктивності додатки на базі WPF потребують більших обчислювальних ресурсів, що може стати обмеженням у випадках роботи на слабших пристроях [8,13-16].

Універсальна платформа Windows (UWP) являє собою ще одну сучасну технологію розробки, яка створена для реалізації універсальних додатків, здатних працювати на широкому спектрі пристроїв під управлінням Windows 10 і новіших версій операційної системи. Подібно до WPF, UWP також використовує XAML для створення користувацького інтерфейсу. Однією з основних переваг цієї платформи є її універсальність – додатки можуть виконуватись на різних типах пристроїв, включаючи персональні комп'ютери, планшети та смартфони. Платформа забезпечує глибоку інтеграцію з функціями Windows 10, такими як система сповіщень, живі плитки чи підтримка голосових асистентів. Розробники також отримують легкий доступ до платформи для публікації своїх застосунків у

Microsoft Store, що спрощує розповсюдження програмного забезпечення серед широкого кола користувачів.

Однак UWP має й слабкі сторони, які варто враховувати. Серед них обмежений доступ до системних ресурсів, що може суттєво впливати на створення складних або специфічних рішень. У порівнянні з WPF, UWP поступається у гнучкості налаштування інтерфейсу, особливо тоді, коли йдеться про реалізацію нестандартних або багатофункціональних компонентів. Ще один недолік полягає в тому, що платформа має обмежену підтримку для старіших версій Windows, що звужує її застосовність у випадках, коли необхідно охопити користувачів із попередніми поколіннями операційної системи [21].

Таблиця 2.1 – Порівняльна характеристика технологій

Характеристика	WinForms	WPF	UWP
Рік випуску	2002	2006	2015
Мова розмітки	Немає	XAML	XAML
Рендеринг	GDI+	DirectX	DirectX
Підтримка анімацій	Обмежена	Так	Так
Адаптивний інтерфейс	Ні	Частково	Так
Підтримка різних пристроїв	Ні	Ні	Так
Розповсюдження через Store	Ні	Ні	Так

Враховуючи усі функціональні вимоги до програмного забезпечення «Каландер-Нагадувач», такі як реалізація зручного та адаптивного інтерфейсу, підтримка графічних елементів, наявність системи авторизації, а також можливість масштабування структури за принципом MVC – технологія Windows Presentation Foundation (WPF) є дійсно оптимальним вибором серед наявних альтернатив. На відміну від Win Forms, яка є на теперішній час є однією із застарілих платформ із обмеженою підтримкою візуальних стилів та слабкою гнучкість у компонуванні інтерфейсу. WPF може надати засоби для реалізації багатошарових шаблонів UI, даних із прив’язкою, модульної архітектури, що є досить таки критично важливим для подальшого розвитку проєкту. Окрім того, у WPF є значно покращена та вища

інтеграція з XAML, що надає змогу забезпечити відокремлення логіки та дизайну, а саме це покращить підтримуваність та масштабованість коду.

UWP хоч і має дійсно ряд переваг у плані універсальності та інтеграції із Microsoft Store, проте її використання є обмежене вимогами до ОС Windows 10, а тому це ускладнює роботу із класичними СУБД, такими як My SQL, що є використані у даному проєкті. Крім того, UWP є більше орієнтованою на мобільні або універсальні застосунки, а не на класичні настільні додатки із традиційною логікою. Таким чином, вибір WPF для використанні у розробленні додатку є добре обґрунтованим за сукупністю таких факторів:

- Підтримка сучасного адаптивного та стилізованого інтефейсу;
- Ефективна робота із архітектурою MVC, яка є реалізованою у структурі додатку;
- Зручна прив’язка до компонентів бази даних My SQL;
- Можливість легко реалізовувати фільтрацію, сповіщення та авторизацію.

Таким чином, WPF надає дозвіл створити зручний у користування, інтуїтивно зрозумілий та функціонально гнучкий інструмент для того, щоб було ефективно керувати часом. Саме це відповідає сучасним технічним вимогам, а також запитам кінцевих користувачів, що було виявлено [8,9.13].

2.3. Вибір архітектурного шаблону MVC та обґрунтування доцільності використання

У процесі розробки програмної платформи було застосовано архітектурний підхід, подібний до MVC, адаптований до особливостей настільних додатків із використанням WPF. Класична модель Model–View–Controller у чистому вигляді не є поширеною для WPF, оскільки сама платформа надає інструменти взаємодії між інтерфейсом і логікою, такі як прив’язка даних, подієва модель і командний механізм. Проте ключовий принцип MVC – поділ відповідальності між основними компонентами – був збережений та покладений в основу архітектури [3,4,26].

Модель (Model) у цьому додатку представлена класами предметної області, що інкапсулюють дані та базові властивості сутностей системи. Основним елементом доменної моделі є клас ReminderEvent, який визначає структуру події та містить такі атрибути, як ідентифікатор, дата й час, назва, опис і статус. Виділення цих даних в окремий клас дозволяє працювати з подіями як із самостійними об’єктами, що спрощує передачу між компонентами системи, інтеграцію з базою даних та візуалізацію в інтерфейсі користувача.

Представлення (View) реалізоване через XAML-описи вікон і елементів керування WPF. Шар представлення відповідає за відображення даних користувачеві, організацію компонентів інтерфейсу та взаємодію з ними. Використання XAML забезпечує чітке відокремлення опису зовнішнього вигляду від програмної логіки, що полегшує модифікацію інтерфейсу та сприяє кращій підтримуваності коду. Для демонстрації списку подій використовується елемент DataGrid, прив’язаний до колекції об’єктів моделі, який автоматично оновлюється при зміні джерела даних.

Контролерна логіка (Controller) у рамках WPF реалізується через обробники подій та команди у файлах code-behind відповідних форм. Цей рівень відповідає за реагування на дії користувача, наприклад натискання кнопок, вибір події, підтвердження видалення або застосування фільтрів. Важливо, що контролерна логіка не взаємодіє безпосередньо із базою даних – такі операції здійснює сервісний шар. Це забезпечує ізоляцію інтерфейсу від деталей зберігання даних [3,5].

Сервісний клас DbService виконує роль проміжного шару між контролером і базою даних. Він об’єднує всі операції доступу до даних – створення, читання, оновлення та видалення подій – в одному місці. Така архітектурна організація підвищує модульність системи, зменшує залежність між компонентами й полегшує розширення її функціоналу.

Застосування структури на основі MVC-підходу дозволило оптимально розділити завдання між компонентами програмного забезпечення. Модель формулює дані предметної області, представлення відповідає за їх візуалізацію, а

контролерна логіка координує дії користувача та взаємодію компонентів. Це зробило код зрозумілішим, структурованим і більш зручним для підтримки.

Обраний підхід також забезпечує основу для майбутнього розвитку системи. Зокрема, є можливість переходу до більш формалізованих патернів, таких як MVVM, або розширення сервісного шару без значного переписування представлення. Таким чином, застосування MVC-подібної архитектури стало продуманим рішенням, що оптимально балансує простоту реалізації й упорядкованість структури програмної платформи.

2.4. Проектування архітектури системи

Архітектура програмної платформи розроблена відповідно до принципів модульності, підтримуваності, розширюваності та чіткого розподілу відповідальності між компонентами системи. Такий підхід зменшує складність коду, полегшує його супровід і створює основу для подальшого розвитку функціональності застосунку.

Архітектурне рішення базується на багатошаровому підході, який адаптований до структури, схожої на MVC, у WPF-застосунках. У цій моделі інтерфейс користувача, логіка керування і доступ до даних чітко розділені. Кожен шар виконує визначену роль і взаємодіє з іншими через стандартизовані інтерфейси.

Інтерфейс користувача (View) реалізований за допомогою XAML-файлів, які описують вікна й елементи керування WPF. Його основним завданням є відображення інформації та обробка дій користувача, таких як натискання кнопок, вибір елементів або введення даних. Інтерфейс не містить доступу до бази даних та бізнес-логіки, що покращує його зручність і ергономіку.

Логіка керування (Controller-подібний рівень) впроваджена у файлах code-behind відповідних форм. Цей шар обробляє події інтерфейсу, перевіряє вхідні дані, формує моделі предметної області та викликає методи сервісного шару. Таким чином, взаємодія користувача із системою централізована, а код не дублюється.

Ключову роль у системі відіграє сервісний шар, втілений у вигляді класу DbService, який служить проміжним елементом між інтерфейсом і локальною базою даних SQLite. Усі операції збереження, отримання, оновлення та видалення даних реалізовані через методи цього сервісу. Завдяки цьому забезпечується мінімальна залежність між компонентами і підвищується загальна стабільність системи [3,27,28].

Сервісний шар уніфікує роботу з базою даних, забезпечуючи єдину точку контролю за операціями. Це спрощує тестування, оскільки вся логіка доступу до даних зосереджена в одному класі. Крім того, така структура полегшує можливість замінити сховище даних у майбутньому, не вносячи змін до інтерфейсу.

Компоненти системи взаємодіють за схемою (див. рисунок 2.1). Кожен компонент виконує конкретно визначені функції, що відповідає принципам єдиної відповідальності та низької зв’язаності між модулями [4,5].

Обрана архітектура забезпечує високу гнучкість для розширення можливостей системи. Наприклад, можна додати нові сервіси, впроваджувати складні механізми фільтрації чи змінити тип сховища даних без значного переписування інших частин програми. Таким чином, архітектура платформи є продуманою та відповідає сучасним вимогам до настільного програмного забезпечення.

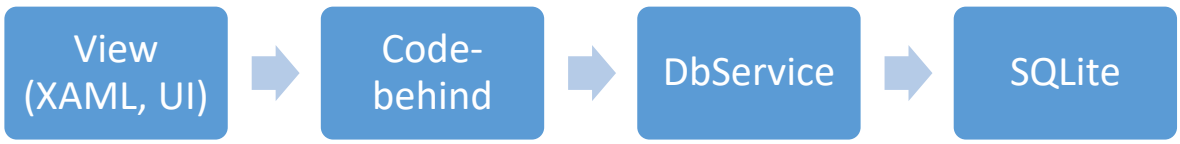

Рисунок 2.1 – Узагальнена архітектура застосунку (MVC-подібний поділ у WPF)

Рисунок 2.2 – Діаграма варіантів використання системи тайм-менеджменту

Діаграма варіантів використання демонструє основні сценарії взаємодії користувача з програмною системою. Користувач має можливість автентифікуватися, додавати, редагувати та видаляти події, а також переглядати список подій. Дана діаграма дозволяє наочно представити функціональні можливості системи та логіку її використання. Див. рисунок 2.2.

2.5. Побудова схем бази даних

Для забезпечення зберігання та обробки даних у програмній платформі управління заходами й тайм-менеджментом було застосовано локальну реляційну базу даних SQLite. Вибір саме цієї СКБД обумовлений орієнтацією платформи на автономну роботу в рамках настільних WPF-додатків, що не потребують розгортання серверної інфраструктури. SQLite пропонує оптимальну продуктивність для роботи зі структурованими даними, легкість перенесення між різними пристроями, а також зручність налаштування, оскільки її структура представлена єдиним файлом формату .db [18,19].

Проєктування структури бази даних здійснювалося з дотриманням принципів реляційної моделі, забезпечення цілісності даних та перспективного розширення функціональних можливостей системи. У межах актуальної реалізації, а також потенційних напрямів розвитку програмної платформи були виділені

кілька логічно взаємопов’язаних сутностей, які представляють ключові елементи предметної області: користувач, подія, нагадування та журнал дій.

Основою бази даних є таблиця Users, що зберігає облікові дані користувачів і підтримує функцію автентифікації. Її структура включає первинний ключ Id типу INTEGER, унікальне текстове поле Login, поле Password для паролів користувачів та поле CreatedAt, що реєструє дату створення облікового запису. Унікальність поля Login забезпечує уникнення дублювання записів користувачів і є базою для контролю доступу до системи. Хоча наразі паролі зберігаються в незашифрованому вигляді, структура бази даних дозволяє без зміни її архітектури перейти до збереження хешованих паролів [15,18].

Однією з ключових таблиць предметної області є Events, яка містить дані про події календаря. Кожен запис має унікальний первинний ключ Id та прив’язаний до конкретного користувача через зовнішній ключ UserId, що реалізує зв’язок типу «один-до-багатьох» між таблицями користувачів і подій. Це дозволяє в майбутньому інтегрувати багатокористувацький режим у систему. Крім того, таблиця включає поля EventDateTime для фіксації часу події, Title і Description для текстового опису та поле Status, яке представляє перелік станів подій відповідно до програмної моделі. Для оптимізації швидкодії реалізоване індексування полів, які часто використовуються при фільтрації чи сортуванні даних, зокрема EventDateTime та UpdatedAt.

Для підтримки механізму нагадувань передбачена таблиця EventReminders, яка доповнює функціонал базової системи. Вона має зв’язок із таблицею Events через зовнішній ключ EventId і містить значення моменту спрацювання нагадування (RemindAt), тип каналу сповіщення (Channel) і прапорець IsSent для контролю факту відправлення нагадування. Використання окремої таблиці для нагадувань дозволяє створювати кілька нагадувань для однієї події без порушення цілісності даних, надаючи масштабованість системі.

Спеціальну роль у структурі відіграє таблиця EventLogs, яка виконує функцію журналювання операцій над подіями. Вона зберігає деталізовану інформацію про дії користувачів — створення, редагування або видалення подій —

а також час їхнього виконання і додаткову інформацію. Такий підхід сприяє впровадженню механізмів аудиту, аналізу та усунення помилок у роботі системи, що є важливою складовою надійності програмного продукту [15,18].

Загалом структура бази даних спроєктована так, щоб забезпечити зручність реалізації та сприяти подальшому розвитку. Поточна модель підтримує всі основні сценарії взаємодії із системою, гарантує узгодженість даних і створює базу для інтеграції нових функціональних можливостей — зокрема багатокористувацького режиму, розширених нагадувань й історії змін. Взаємозв’язки між таблицями представлені на рисунку 2.3.

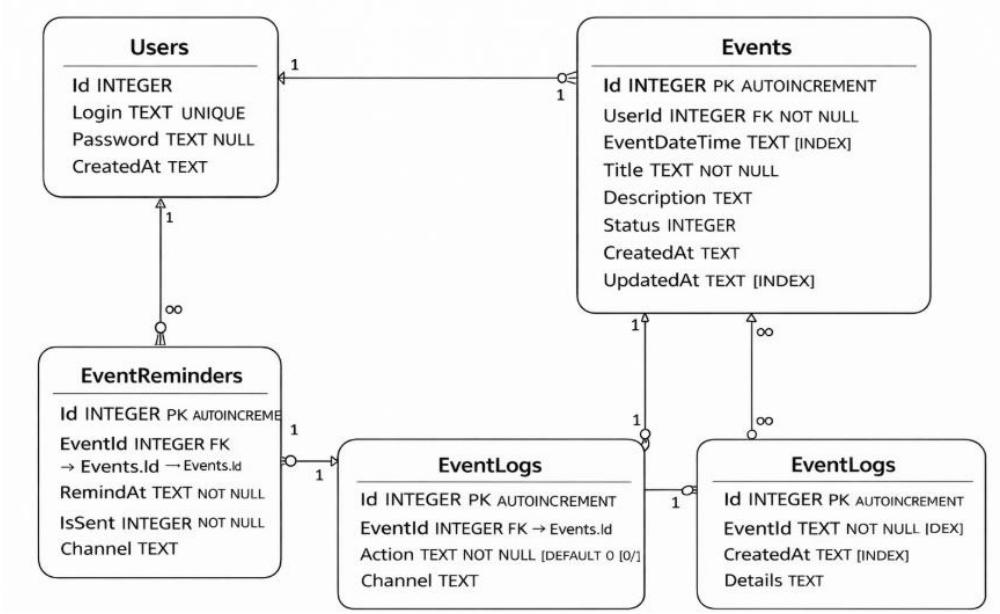

Рисунок 2.3 – Розширена ER-схема бази даних SQLite (Users–Events–EventReminders–EventLogs)

2.6. Побудова UML-діаграм класів

Проєктування програмної платформи для тайм-менеджменту здійснювалося відповідно до принципів об'єктно-орієнтованого підходу, зосередженого на виділенні ключових об'єктів предметної області та сервісних компонентів. Це дозволило сформувати логічно структурований код із чітким розподілом завдань

між класами, а також забезпечити зручність його подальшої модифікації й розширення.

Центральним елементом доменної моделі є клас ReminderEvent, який представляє подію в системі. Цей клас містить основні атрибути події, такі як ідентифікатор, дату, час виконання, назву, опис і статус. Об'єкти ReminderEvent використовуються для збереження, відображення і обробки подій у межах користувацького інтерфейсу. При цьому ReminderEvent не включає логіки доступу до бази даних, що забезпечує дотримання принципів низької зв’язаності й високої когерентності в об'єктно-орієнтованому програмуванні.

Для роботи з базою даних використовується окремий клас-сервіс DbService. Він функціонує як проміжна ланка між користувацьким інтерфейсом і базою даних SQLite, реалізуючи всі основні операції CRUD (створення, читання, оновлення і видалення). Логіка обробки SQL-запитів централізована в цьому класі, що дозволяє спростити тестування та зменшити залежність користувацького інтерфейсу від конкретної технології зберігання даних.

Клас MainWindow відповідає за представлення даних і реалізацію контролерної логіки в рамках архітектури, побудованої за принципом MVC. Він забезпечує взаємодію з користувачем, обробляє події інтерфейсу, такі як натискання кнопок або редагування даних, і викликає відповідні методи сервісного шару. Таким чином, MainWindow виступає координатором між візуальною частиною програми та її бізнес-логікою.

Для забезпечення механізму автентифікації користувачів використовується окреме вікно LoginWindow. Воно дозволяє вводити облікові дані та перевіряти доступ до основного функціоналу застосунку через методи класу DbService. Після успішного входу користувачу відкривають доступ до головного функціоналу системи.

Структура взаємодії між основними компонентами системи представлена у вигляді UML-діаграми класів. Ця діаграма ілюструє, що MainWindow взаємодіє з DbService для отримання необхідних даних, тоді як DbService оперує об'єктами

класу ReminderEvent. Такий підхід гарантує слабку зв'язаність між компонентами та відповідає вимогам модульності й масштабованості.

UML-діаграма слугує інструментом документування архітектури системи, наочно демонструючи структуру компонентів програмного забезпечення і зв'язки між ними. Це значно полегшує підтримку та розвиток платформи у майбутньому. Див. рисунок 2.4. [4,26].

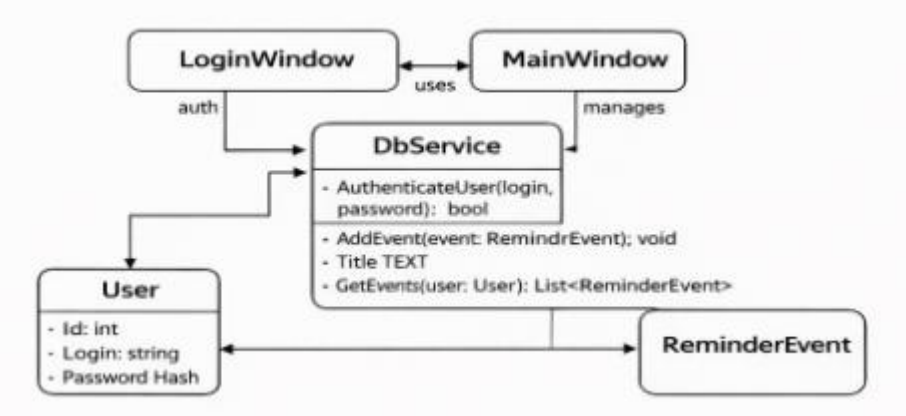

Рисунок 2.4 – UML-діаграма основних класів системи

Як приклад потенційного розширення створеної доменної моделі, система забезпечує еволюційний розвиток без необхідності змінювати існуючу архітектуру. Зокрема, базуючись на реалізованому класі ReminderEvent, можна інтегрувати нові сутності для поліпшення функціоналу управління часом.

Для впровадження функцій повторюваних подій можна створити окремий клас, що визначає параметри повторення (тип повторюваності, інтервал, кінцеву дату). Цей клас логічно пов'язується з подією через унікальний ідентифікатор і використовується на рівні сервісного шару під час генерації списку подій для відображення користувачеві.

Крім того, одним із можливих покращень є категоризація подій за допомогою введення класу категорій або міток, що дозволяє групувати завдання за напрямками (наприклад, навчання, робота, особисті справи). Це можна реалізувати через додаткову таблицю з відповідними зв’язками «один-до-багатьох» або «багато-до-багатьох», при цьому зберігаючи існуючу функціональність інтерфейсу незмінною.

Завдяки використанню архітектури, близької до моделі MVC, і сервісного шару (DbService), подібні вдосконалення не потребуватимуть значних змін у фронтенді. Основна бізнес-логіка залишається чітко відокремленою, а додавання нових можливостей здійснюється за рахунок розширення моделей даних і сервісних методів. Це підкреслює масштабованість і гнучкість запроектованої програмної платформи.

2.7. Вибір мови та середовища розробки

Для розробки програмної платформи тайм-менеджменту та управління подіями була обрана мова програмування C# у поєднанні з потужною платформою .NET, оскільки цей вибір забезпечує високий рівень надійності, продуктивності та адаптації до сучасних підходів у створенні настільних додатків. Мова програмування C# є об'єктно-орієнтованою, що дозволяє ефективно реалізовувати складні багатошарові архітектури. Це дає змогу впроваджувати ключові концепції розробки, такі як модульність, інкапсуляція та повторне використання коду, що безпосередньо впливає на продуктивність розробленої системи та її гнучкість [8,9].

Для створення графічного інтерфейсу користувача було обрано технологію Windows Presentation Foundation (WPF) у поєднанні з мовою розмітки XAML. Використання WPF забезпечує широкі можливості для створення сучасних інтерактивних додатків, які підтримують різноманітні стилі, шаблони, ресурси, а також механізми зручної прив'язки даних. Завдяки застосуванню XAML вдалося досягти чіткої сегрегації між логічною частиною програми та її візуальним компонентом. Такий підхід значно покращив читабельність коду, спростив його обслуговування та майбутню модернізацію.

Однією з найважливіших особливостей обраної технології стала підтримка механізму прив'язки даних (Data Binding), що забезпечує автоматизовану синхронізацію між даними моделі та відповідними елементами інтерфейсу користувача. У межах розробки додатку це дозволило оперативно оновлювати

інформацію в таблицях подій під час їхнього додавання, редагування або видалення без необхідності прямого втручання в програмний код представлених компонентів.

Такий підхід значно спрощує підтримку програми і робить її більш надійною у використанні.

В якості інструменту для розробки обрали Microsoft Visual Studio. Це середовище є найпоширенішим і найзручнішим стандартом для створення WPF-додатків завдяки своїм широким можливостям. Visual Studio пропонує інтегрований дизайнер XAML для проектування інтерфейсів користувача, потужний налагоджувач, засоби аналізу помилок прив'язки даних, а також зручну систему керування залежностями через пакетний менеджер NuGet. Завдяки цим інструментам процес інтеграції бібліотек і керування залежностями був виконаний швидко та без зайвих складнощів. Крім того, вбудовані засоби налагодження суттєво спростили виявлення і виправлення логічних помилок та налагодження взаємодії з базою даних [20].

Для збереження даних у застосунку було обрано легковаговий і водночас потужний механізм у вигляді локальної бази даних SQLite. Ця база відмінно інтегрується з платформою .NET і не потребує окремого серверного середовища, що робить її ідеальним вибором для настільного застосунку. Взаємодія із базою даних реалізована за допомогою спеціалізованого провайдера Microsoft.Data.Sqlite, який характеризується простотою налаштування та забезпечує стабільну роботу з реляційними даними. Додатково, усі SQL-запити реалізовані із застосуванням параметризації, що мінімізує ризики виникнення проблем безпеки, зокрема SQL-ін’єкцій.

Використання технологічного стеку C# з WPF, SQLite та Visual Studio забезпечило ідеальний баланс між функціональністю, зручністю розробки та потенціалом для подальшого розширення системи. Завдяки цьому підходу вдалося створити автономний настільний додаток, спрямований на щоденне використання. Додаток підтримує збереження даних, користувацьку авторизацію та механізми нагадувань, що повністю відповідає цілям магістерської роботи.

2.8. Реалізація основних класів та методів

Реалізація програмних класів платформи тайм-менеджменту здійснювалася з дотриманням принципів об’єктно-орієнтованого програмування.

Дані предметної області інкапсульовано у вигляді окремих класів-моделей, тоді як логіка взаємодії з базою даних винесена в окремий сервісний клас. Див. лістинг 2.1, 2.2.

Кожен клас відповідає за чітко визначену функціональну область, що забезпечує модульність, масштабованість та зручність супроводу програмного продукту [4,5,27].

Лістинг 2.1 – Клас-модель події ReminderEvent

```
public class ReminderEvent
{
    public int Id { get; set; }
    public DateTime EventDateTime { get; set; }
    public string Title { get; set; }
    public string Description { get; set; }
    public EventStatus Status { get; set; }
}
```

Лістинг 2.2 – Фрагмент логіки керування подіями головного вікна

```
private void Add_Click(object sender, RoutedEventArgs e)
{
    var ev = new ReminderEvent
    {
        EventDateTime = CalendarControl.SelectedDate.Value,
        Title = TitleBox.Text,
        Description = DescBox.Text,
        Status = (EventStatus)StatusBox.SelectedItem
    };

    _db.AddEvent(ev);
    LoadEvents();
}
```

Реалізація моделі користувача здійснена на рівні бази даних у вигляді окремої таблиці Users, що містить облікові дані для автентифікації. Див. лістинг 2.3.

Перевірка прав доступу виконується централізовано за допомогою сервісного класу DbService, який реалізує метод автентифікації користувача. Див. лістинг 2.4.

Подальший доступ до основного функціоналу системи дозволяється лише після успішної перевірки облікових даних, що забезпечує підвищений рівень інформаційної безпеки програмної платформи. Див. лістинг 2.5.

Лістинг 2.3 – Фрагмент реалізації моделі користувача на рівні бази даних

```
CREATE TABLE IF NOT EXISTS Users(
    Id INTEGER PRIMARY KEY AUTOINCREMENT,
    Login TEXT UNIQUE,
    Password TEXT
);
```

Лістинг 2.4 – Реалізація автентифікації користувача на основі централізованої моделі доступу

```
public bool CheckLogin(string login, string password)
{
    using var con = Open();
    con.Open();
    var cmd = con.CreateCommand();
    cmd.CommandText =
        "SELECT COUNT(*) FROM Users WHERE Login=$l AND
Password=$p;";
    cmd.Parameters.AddWithValue("$l", login);
    cmd.Parameters.AddWithValue("$p", password);
    return Convert.ToInt32(cmd.ExecuteScalar()) > 0;
}
```

Лістинг 2.5 – Логіка контролю доступу до функціональних можливостей системи

```
private void Login_Click(object sender, RoutedEventArgs e)
{
```

```
    if (_db.CheckLogin(LoginBox.Text, PasswordBox.Password))
    {
        new MainWindow().Show();
        Close();
    }
    else
    {
        MessageBox.Show("Невірний логін або пароль");
    }
}
```

Реалізація сервісного шару здійснюється за допомогою окремого класу DbService, який інкапсулює всю логіку доступу до бази даних. Користувацький інтерфейс взаємодіє з базою даних виключно через методи сервісного шару, що дозволяє зменшити залежність між компонентами системи та спрощує подальше тестування й розширення функціоналу. Див. лістинг 2.6.

Лістинг 2.6 – Сервісний клас DbService як проміжна ланка між інтерфейсом та базою даних

```
public class DbService
{
    public List<ReminderEvent> GetEvents()
    {
        // робота з базою даних
    }

    public void AddEvent(ReminderEvent ev)
    {
        // SQL-операція додавання
    }

    public void UpdateEvent(ReminderEvent ev)
    {
        // SQL-операція оновлення
    }

    public void DeleteEvent(int id)
    {
        // SQL-операція видалення
    }
}
```

2.9. Розробка інтерфейсу користувача

Інтерфейс користувача реалізовано з використанням технології WPF, що забезпечує гнучкі можливості створення інтерактивних та адаптивних користувацьких інтерфейсів. Проєктування інтерфейсу здійснювалося з урахуванням принципів зручності та ергономіки. Див. лістинг 2.7 [13,14,26].

Лістинг 2.7 – XAML-опис головного вікна

```
<Button Content="➕ Додати подію"
                    Margin="0,10,0,0"
                    Height="40"
                    Click="Add_Click"/>
<Button Content="🗑 Видалити вибрану подію"
                    Margin="0,5,0,0"
                    Height="35"
                    Click="Delete_Click"/>

<Button Content="□ Видалити всі події"
                    Margin="0,5,0,0"
                    Height="35"
                    Click="DeleteAll_Click"/>

<Button Content="🔍 Фільтрація подій"
                    Margin="0,5,0,0"
                    Height="35"
                    Click="Filter_Click"/>
<Button Content="✏ Редагувати вибрану"
                    Margin="0,5,0,0"
                    Height="35"
                    Click="Edit_Click"/>
```

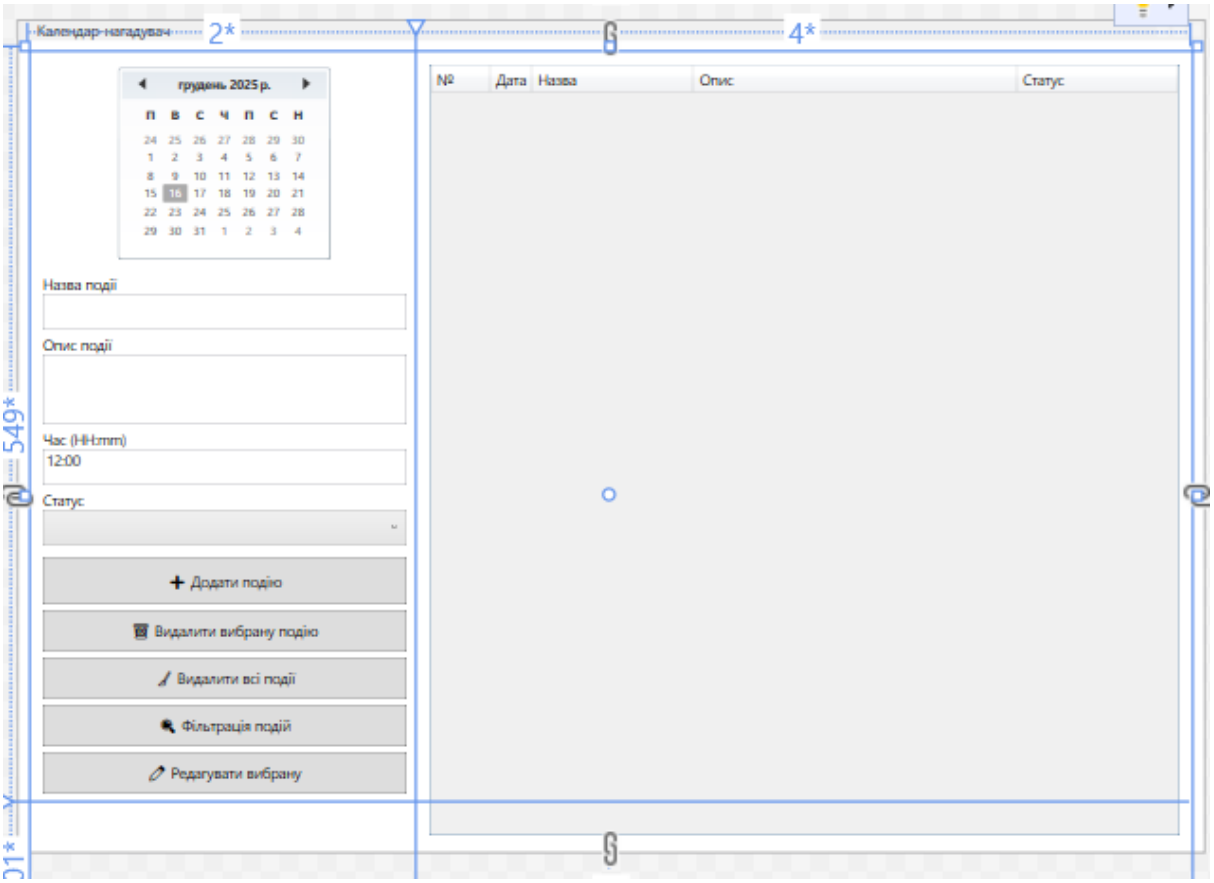

Рисунок 2.5 – Головне вікно застосунку

Використання механізмів прив’язки даних у середовищі WPF дозволяє автоматично оновлювати елементи користувацького інтерфейсу при зміні джерела даних без необхідності прямого втручання у програмний код представлення.

Дані подій відображаються у табличному вигляді з використанням елемента DataGrid, який прив’язується до колекції об’єктів предметної області. При додаванні, редагуванні або видаленні подій оновлення інтерфейсу відбувається автоматично. Див. рисунок 2.5, 2.6.

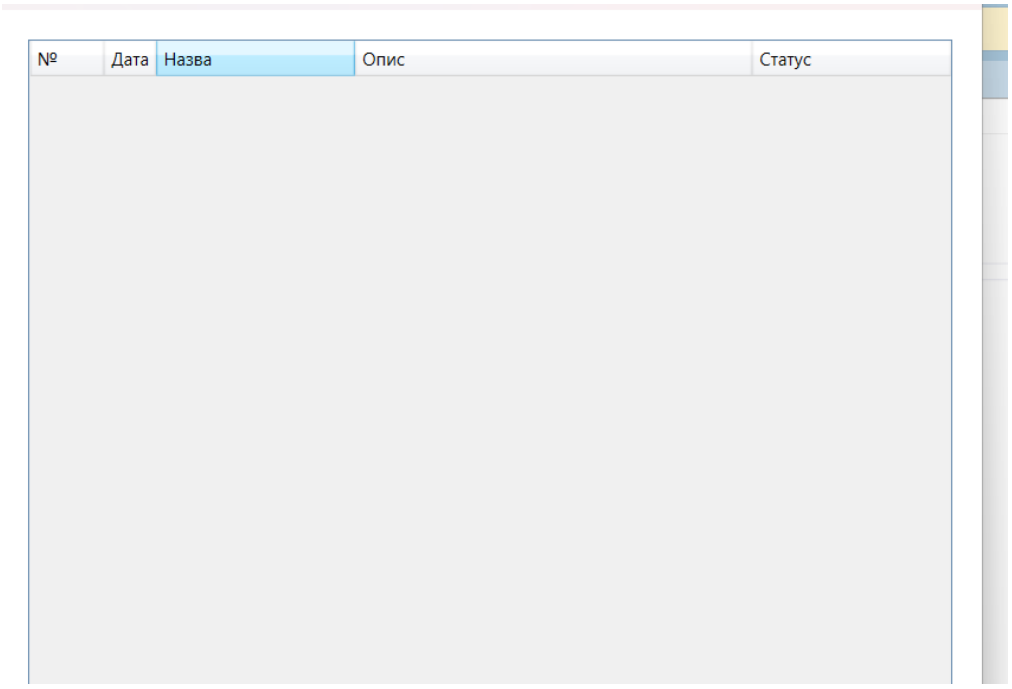

Рисунок 2.6 – Відображення даних подій у користувацькому інтерфейсі з використанням механізмів прив’язки даних WPF

3. ТЕСТУВАННЯ, ВПРОВАДЖЕННЯ ТА ПІДТРИМКА

У цьому розділі розглядаються аспекти тестування розробленої програмної платформи, її розгортання та умови подальшої експлуатації. Розглядаються типи тестувань, проведені для перевірки правильності функціонування системи, а також специфіка впровадження програмного забезпечення в локальне середовище.

3.1. Тестування програмної системи

Тестування програмного забезпечення було проведено для перевірки точності реалізації функціональних вимог, оцінювання стабільності взаємодії компонентів, таких як користувацький інтерфейс WPF, сервісний шар доступу до даних (DbService) та локальна база даних SQLite, а також для аналізу надійності роботи механізму нагадувань. Оскільки це рішення призначене для практичного використання кінцевим користувачем і забезпечує повний цикл операцій з подіями (авторизація, створення, перегляд, редагування, видалення, фільтрація, відображення статусів і отримання нагадувань у визначений час), у ході тестування було застосовано сценарний підхід. Цей метод дозволяє оцінювати функціональність системи всебічно – від введення даних у користувацький інтерфейс до їхнього збереження в базі даних і подальшого відображення [10,11].

Особливістю настільних застосунків WPF є те, що значна частина функціонала реалізується через інтерактивні елементи інтерфейсу. Це включає обробку натискань кнопок, роботу з формами введення, виділення рядків у таблицях DataGrid, відкриття додаткових вікон (наприклад, для авторизації, редагування або фільтрації) та демонстрацію системних повідомлень. Тому тестування охоплювало не лише перевірку окремих методів, але й аналіз поведінки системи в межах типових сценаріїв користувача. У таких випадках правильність роботи визначали злагодженістю виконання кількох послідовних операцій. Важливо було переконатися, що система коректно реагує як на правильні дії користувача, так і на можливі помилки. Серед них: некоректний формат часу,

незаповнення обов’язкових полів, спроби редагування чи видалення події без її вибору, застосування фільтра без заданих параметрів або введення неправильних облікових даних.

Окремий акцент робився на перевірці збереження даних між сеансами використання. Події повинні залишатися в базі даних після перезапуску програми та правильно відображатися в інтерфейсі, що підтверджує коректність роботи з SQLite і коректну реалізацію логіки завантаження даних через сервісний шар.

3.1.1 Види та план тестування

У процесі виконання роботи визначено та реалізовано комплекс тестувань, який відповідає реальним характеристикам створеного програмного продукту. Основний акцент було зроблено на функціональному тестуванні, метою якого стала перевірка відповідності системи заявленим можливостям. До основних тестів належали перевірка сценаріїв входу через LoginWindow (успішна і неуспішна автентифікація), контроль переходу до головного вікна лише після успішної авторизації, тестування створення подій із різними параметрами, відображення подій у таблиці DataGrid, редагування окремих записів, видалення подій та перевірка збереження даних без змін для інших записів. Окремо аналізувалась можливість фільтрації подій за діапазоном дат. У зв’язку з використанням статусів подій, ретельно тестувалася коректність їхнього збереження та відображення після змін, що має значну вагу в контексті тайм-менеджменту [10,11].

З паралельним виконанням інтеграційного тестування вивчалися взаємодії різних компонентів системи, зокрема UI-шару (WPF/XAML), керуючої логіки (code-behind), сервісного шару DbService та бази даних SQLite. Головним завданням було пересвідчитись у коректності ініціалізації бази даних, що включала створення таблиць Users і Events та додавання початкового користувача (за потреби через метод Init). Було перевірено, чи правильно фіксуються операції запису (додавання та редагування) у БД, чи правильно повертається вибірка подій без спотворень і чи інтерфейс відображає актуальний стан даних після виконаних

операцій. Крім того, досліджувалася стійкість роботи системи при перезапуску: оновлені або збережені дані мали залишатися доступними одразу після повторного запуску програми, а у DataGrid повинні коректно відображатися раніше створені записи. Такі перевірки підтверджували придатність програмного продукту для практичного використання, а не лише в умовах поточного запуску [10,11].

Особливу увагу приділили негативному тестуванню, спрямованому на виявлення типових помилок користувачів та обробку некоректного вводу. Зокрема, система мала запобігати створенню подій без обраної дати або дозволяти лише коректно вказаний формат часу без виклику аварійного завершення роботи. Якщо подія не мала назви, її збереження блокувалося, адже цей параметр є обов'язковим. Недопустимими також виявилися сценарії редагування або видалення невиділених записів у DataGrid – система реагувала на це передбачуваним чином. Крім того, неправильні облікові дані не відкривали доступу до основного функціоналу. Всі ці дії сприяли покращенню зручності використання програмного рішення та підвищенню його захисту від помилок.

У процесі розширення функціоналу застосунку виконувалося регресійне тестування. До нових можливостей належали редагування подій, фільтрація, оновлення таблиці і налаштування формату дати. Регресійні перевірки передбачали повторне тестування основних сценаріїв після кожного розширення функціоналу з метою впевненості у збереженні працездатності вже існуючих компонентів системи. У цих перевірках відтворювались стандартні сценарії: авторизація, створення події, її відображення, редагування, видалення, перезапуск програми, контроль наявності даних після повторного входу.

Проводилося додаткове тестування відображення та форматів, зосереджене на календарних системах. Важливо було переконатися, що дата і час представлені у стандартному для українських користувачів форматі (день-місяць-рік), а також перевірити читабельність значень у таблицях та колонках, зокрема назви й опису. Хоч такі аспекти належать до нефункціонального тестування інтерфейсу, вони суттєво впливають на зручність роботи із застосунком.

Загальний план тестування мав структурований підхід – від перевірки базової функціональності системи до складніших сценаріїв. Спершу тестувався початковий запуск, включаючи наявність чи відсутність файлу бази даних, коректність ініціалізації та створення необхідної структури. Наступним етапом була перевірка авторизації, яка виступає ключовою «точкою входу» до функціоналу. Після цього тестувалися CRUD-операції з подіями, сам механізм фільтрації, а завершальною частиною стало тестування нагадувань, де акцент робився на точності збігу часу події з моментом оповіщення [10,11].

3.1.2 Розробка тестових сценаріїв

Для забезпечення повторюваності тестування та можливості перевірки коректності роботи системи після її змін було створено набір тестових сценаріїв. У цій роботі тестовий сценарій визначається як структурований опис взаємодії користувача із застосунком, що включає початкові умови, послідовність дій і очікуваний результат. Такий підхід дозволяє перевіряти систему в повному обсязі, охоплюючи цілісні користувацькі потоки, що є особливо важливим для WPF-додатків, де ключова логіка проявляється через події інтерфейсу [19,20].

Першою групою тестових сценаріїв стали перевірки, пов’язані з ініціалізацією програми та авторизацією користувачів. Одним із таких сценаріїв було тестування першого запуску програми за відсутності бази даних. Очікувалося, що додаток створить необхідну структуру таблиць, уникне помилок у роботі SQLite і відкриє вікно авторизації. Далі тестувалися сценарії входу: позитивний – із введенням правильних даних (наприклад, admin/admin) та негативний – із неправильним паролем, де система повинна вивести повідомлення про помилку і не дозволити доступ до головного вікна. Див. рисунок 3.1, 3.2, 3.3.

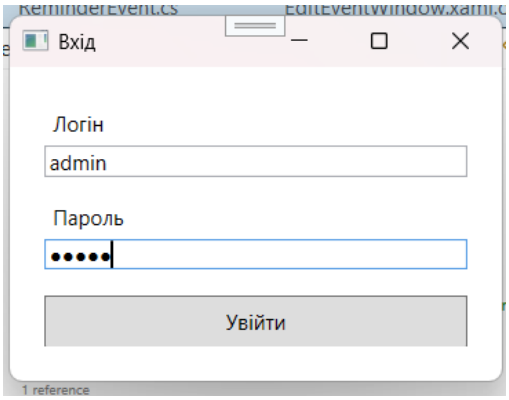

Рисунок 3.1 – Вікно авторизації користувача при запуску програмної системи

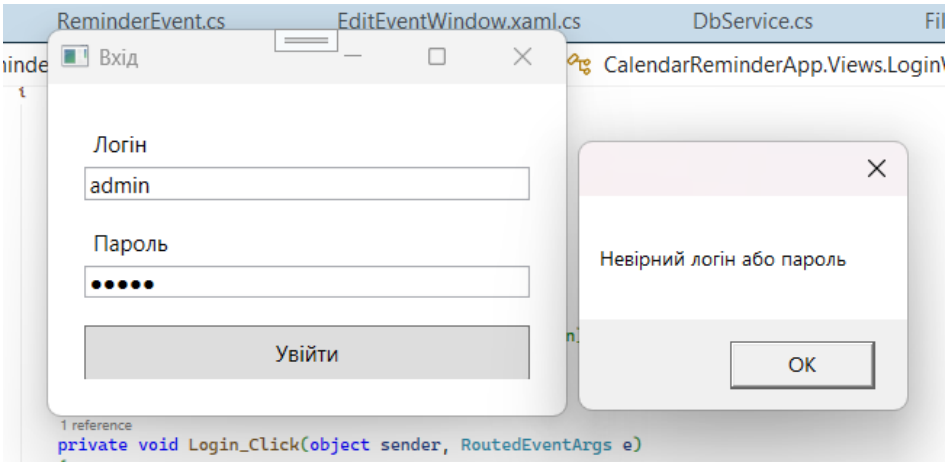

Рисунок 3.2 – Результат негативного тестування автентифікації користувача

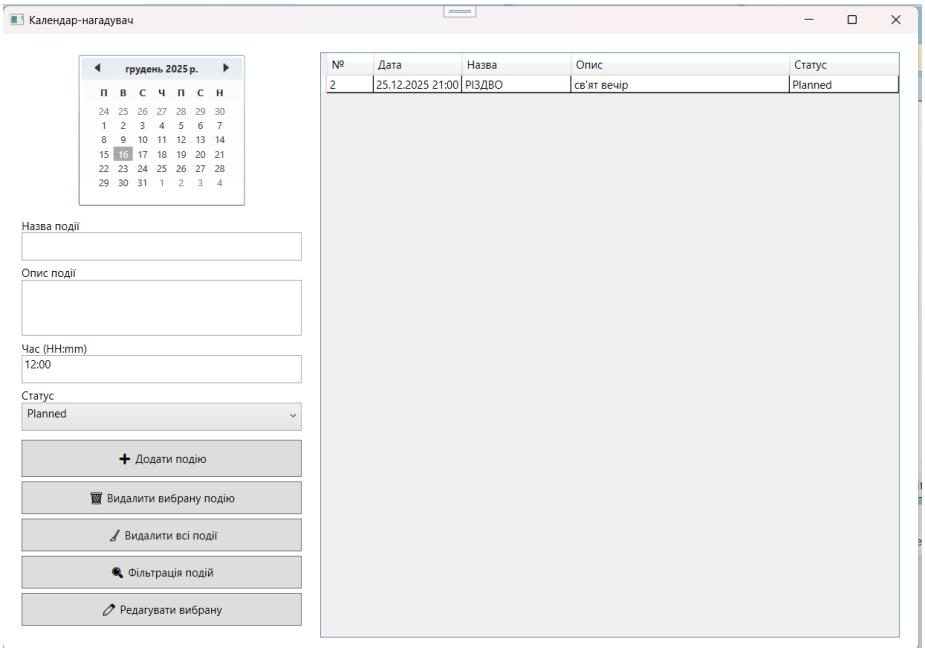

Рисунок 3.3 – Головне вікно застосунку після успішної авторизації

Другою ключовою групою тестів стали сценарії, пов’язані з операціями над подіями. Для перевірки додавання подій виконувались такі дії: вибір дати в календарі, введення часу у вказаному форматі, заповнення назви, опису й статусу. Результатом цього мав стати новий запис у DataGrid із відповідним збереженням даних у SQLite. Див. рисунок 3.4, 3.5.

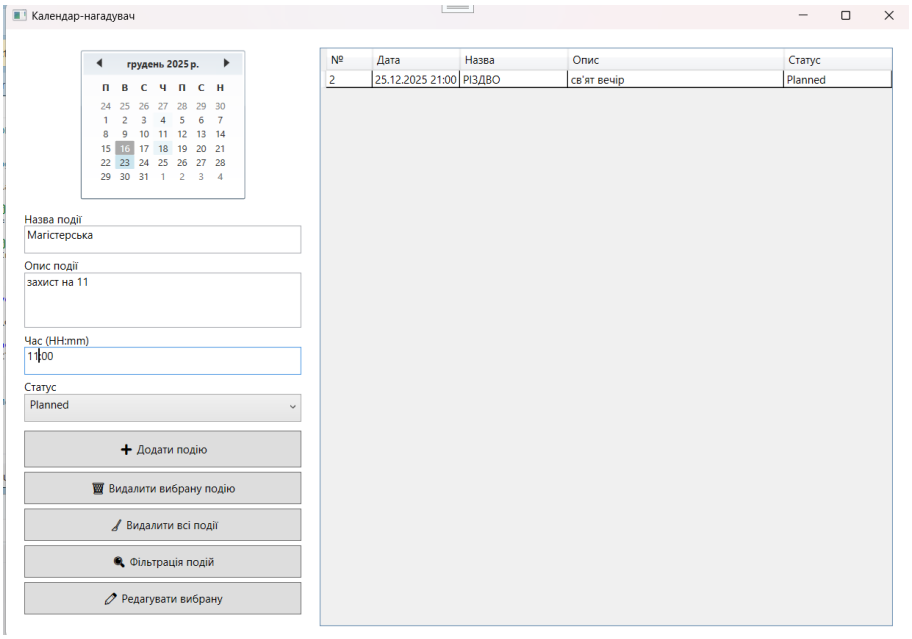

Рисунок 3.4 – Підготовка даних для додавання нової події (тестування операції створення)

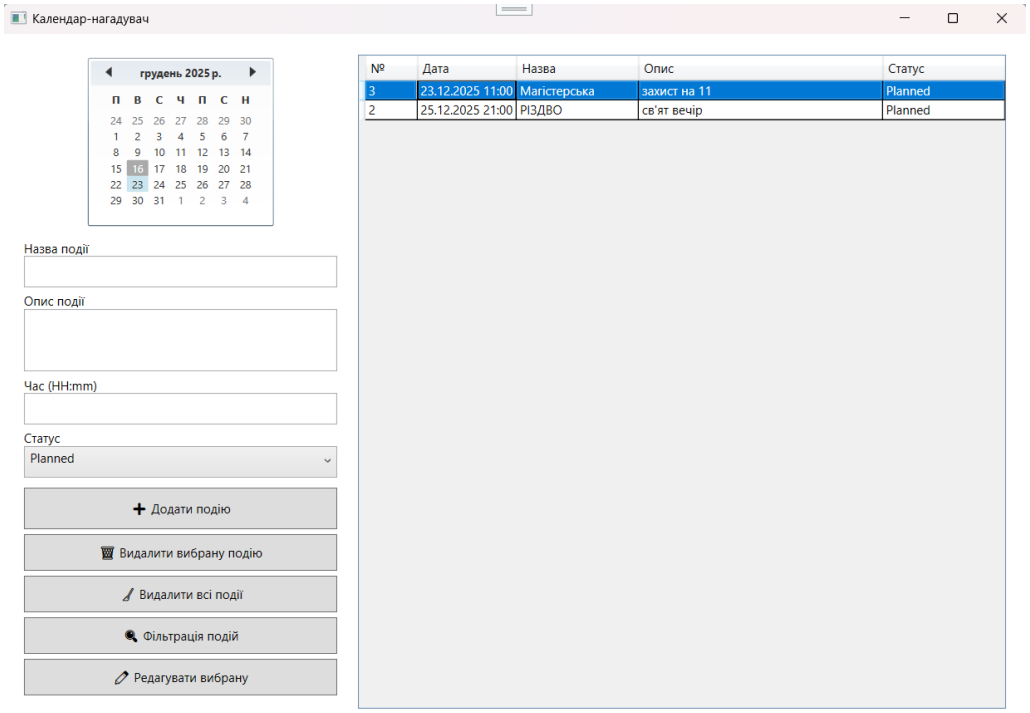

Рисунок 3.5 – Відображення доданої події в таблиці після збереження в базі даних

Для редагування подій перевірялося відкриття обраного запису у формі редагування, змінення атрибутів і відповідне оновлення даних у таблиці. Під час перевірки видалення тестувалося, чи видаляється саме обраний запис без впливу на інші дані. Крім того, враховувалися негативні випадки, наприклад, спроби редагування чи видалення за відсутності обраного рядка. У таких ситуаціях система має коректно повідомляти про помилку без аварійного завершення роботи програми. Див. рисунок 3.6 – 3.9.

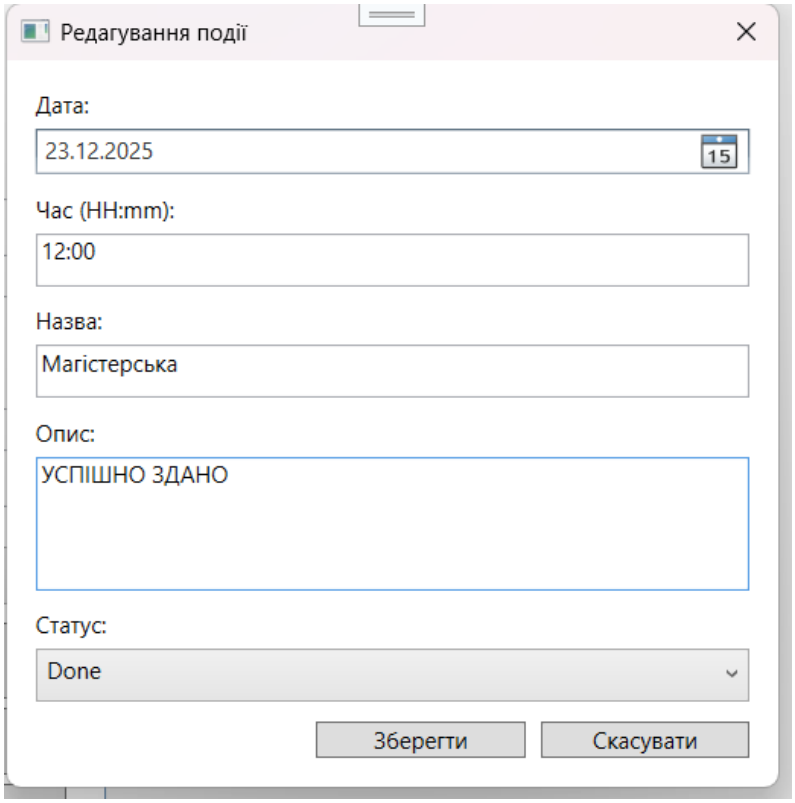

Рисунок 3.6 – Редагування вибраної події у процесі тестування

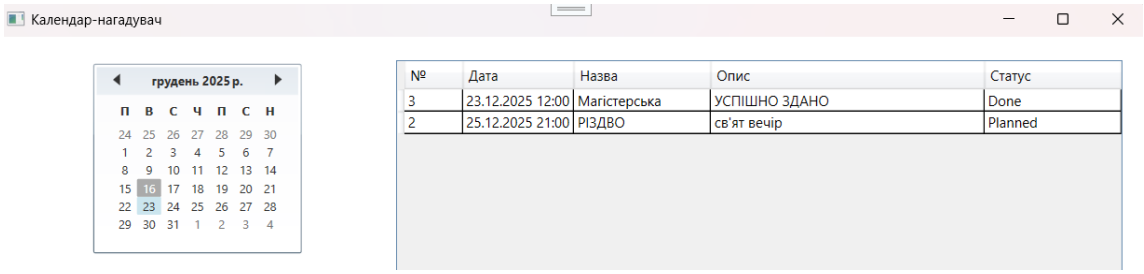

Рисунок 3.7 – Результат редагування вибраної події у процесі тестування

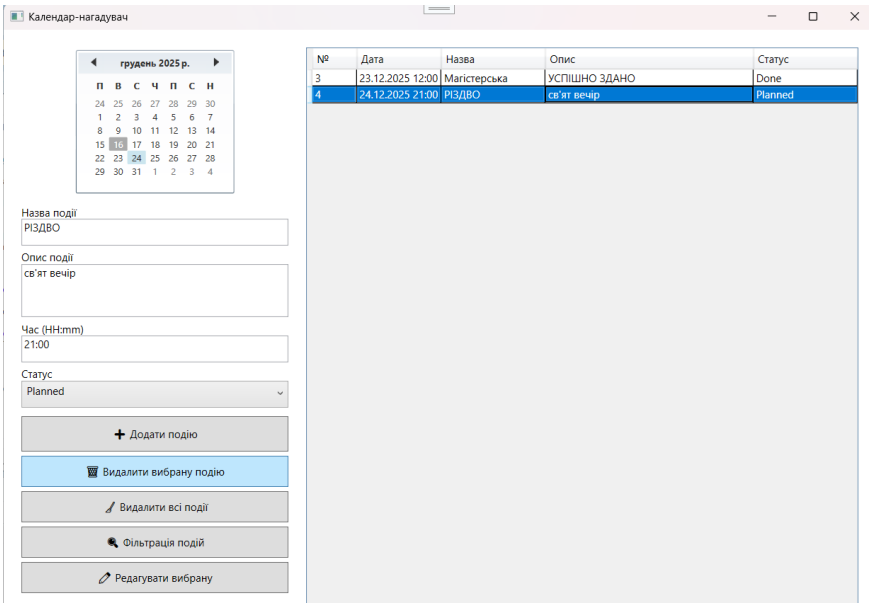

Рисунок 3.8 – Видалення вибраної події у процесі тестування

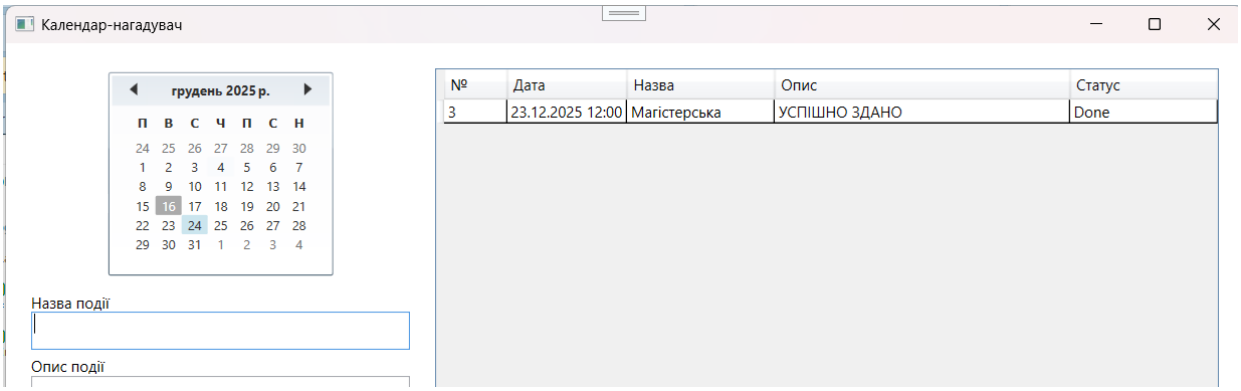

Рисунок 3.9 – Результат видалення вибраної події у процесі тестування

Окремо було розроблено сценарії для перевірки функціональності фільтрації. Для цього генерувалася множина подій на різні дати, а потім тестувався фільтр «від-до» через спеціальне вікно. Очікуваний результат – показ лише тих записів, які відповідають заданому часовому діапазону, що підтверджує правильну реалізацію логіки фільтрації та взаємодію між інтерфейсом користувача та сервісним шаром програми. Див. рисунок 3.10, 3.11.

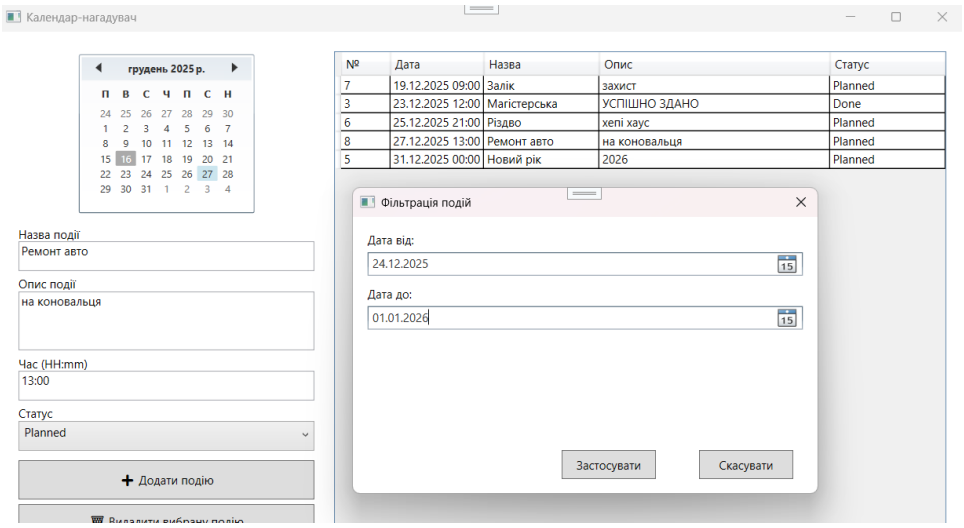

Рисунок 3.10 – Фільтрація подій з 24.12.2025 по 01.01.2026

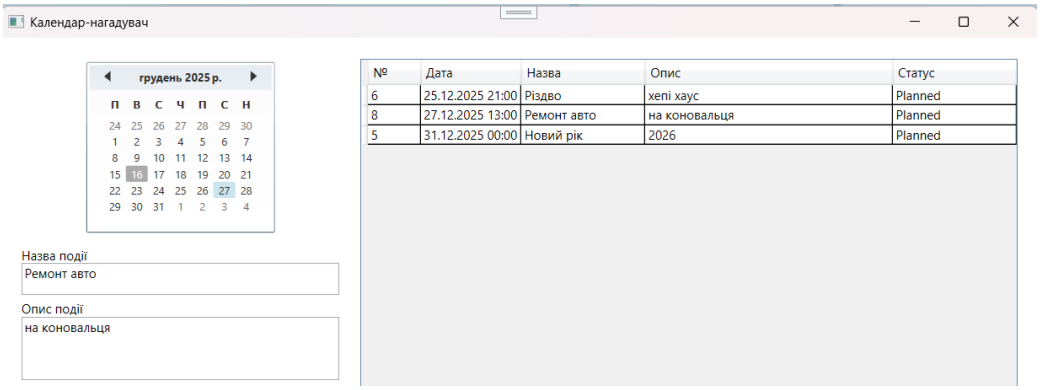

Рисунок 3.11 – Перевірка роботи механізму фільтрації подій за датою

Завершальна група сценаріїв присвячувалася тестуванню механізму нагадувань. Для цього створювали подію з найближчим часом нагадування (1–2 хвилини), що дозволяло швидко перевірити коректність роботи таймера. Очікуваним результатом було вчасне відображення повідомлення із текстом нагадування. Також перевірялися випадки, коли події з минулою датою або часом не повинні запускати таймери чи породжувати хибні сповіщення. Див. рисунок 3.12.

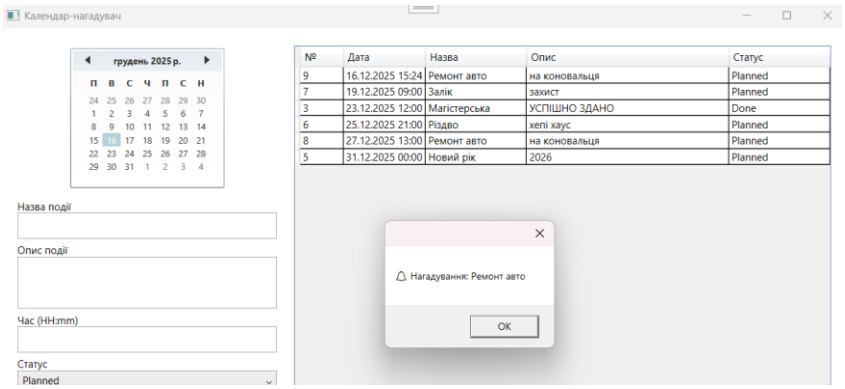

Рисунок 3.12 – Приклад спрацювання нагадування у визначений час

Розроблені тестові сценарії забезпечують покриття всього функціонального циклу програми і підтверджують її стабільну роботу в реальних умовах використання.

3.2. Розгортання програмної системи та системні вимоги

Проєкт розроблено з урахуванням автономного сценарію використання, що виключає потребу в мережевій чи серверній інфраструктурі. Такий формат оптимально підходить для настільних систем тайм-менеджменту, спрямованих на персональне користування, навчальні завдання або локальне застосування на одному комп’ютері. Автономна архітектура сприяє стабільності роботи, незалежності від доступу до інтернету та мінімізації ризиків, які можуть виникати під час передачі даних через мережу [16,29].

Застосунок реалізовано у вигляді традиційного настільного WPF-додатка на базі платформи .NET. Це дозволяє його встановлення як на комп’ютер розробника, так і на будь-який інший ПК із Windows. Для зберігання інформації використовується інтегрована локальна база даних SQLite, яка розташовується у файловій системі користувача. Використання такої СКБД виключає необхідність складної процедури налаштування серверів баз даних, скорочує зовнішні залежності та спрощує процес впровадження програми.

Однією з ключових переваг системи є прозорість її розгортання для кінцевого

користувача. Програма автоматично створює необхідні службові каталоги, ініціалізує базу даних та генерує таблиці для зберігання даних про користувачів і події. Це забезпечує можливість швидкого впровадження без потреби в залученні фахівців із адміністрування програмного забезпечення.

3.2.1 Процес розгортання програмної системи

Процес розгортання програмного забезпечення складається з кількох послідовних етапів, охоплюючи стадії розробки, тестування та подальшої експлуатації програми кінцевими користувачами. На початковому етапі розробки та налагодження програма запускається безпосередньо в середовищі Microsoft Visual Studio в режимі відлагодження. Це забезпечує швидку перевірку коректності роботи інтерфейсу, логіки обробки даних та реагування на дії користувача [16,29].

Після завершення розробки й тестування створюється виконуваний файл (.exe), який використовується для розгортання системи на інших комп'ютерах. Для цього достатньо скопіювати виконуваний файл та, за потреби, відповідні бібліотеки платформи .NET. Додаткові інсталяційні пакети для роботи з базою даних не потрібні, оскільки застосовується вбудована файлово орієнтована СКБД SQLite.

Перший запуск програмної системи супроводжується автоматичною ініціалізацією середовища зберігання даних. Відповідальний за це сервісний клас DbService створює спеціальну директорію у системному каталозі користувача (AppData), де генерується файл бази даних під назвою events.db. Якщо файл бази або потрібні таблиці відсутні, програма автоматично формує необхідну структуру, включно з таблицями для зберігання облікових записів користувачів і календарних подій. Завдяки цьому весь процес налаштування середовища є повністю автоматизованим і не потребує втручання з боку користувача [16,29].

Завершивши ініціалізацію, система відкриває вікно авторизації (LoginWindow), яке дозволяє здійснити базовий контроль доступу до функціональних можливостей програми. Для переходу до основного вікна (MainWindow) користувач має ввести правильні облікові дані. У разі успішної

перевірки відкривається головний екран із повним функціоналом управління подіями. Такий механізм забезпечує обмеження доступу до інформації та базовий рівень захисту даних без необхідності підключення до зовнішніх сервісів автентифікації.

Процес розгортання не вимагає встановлення серверного програмного забезпечення, додаткових компонентів баз даних чи складних налаштувань конфігураційних файлів. Єдиною необхідною умовою є сумісність із платформою .NET, що полегшує впровадження системи в навчальних закладах, офісах або для індивідуального використання. Див. рисунок 3.13.

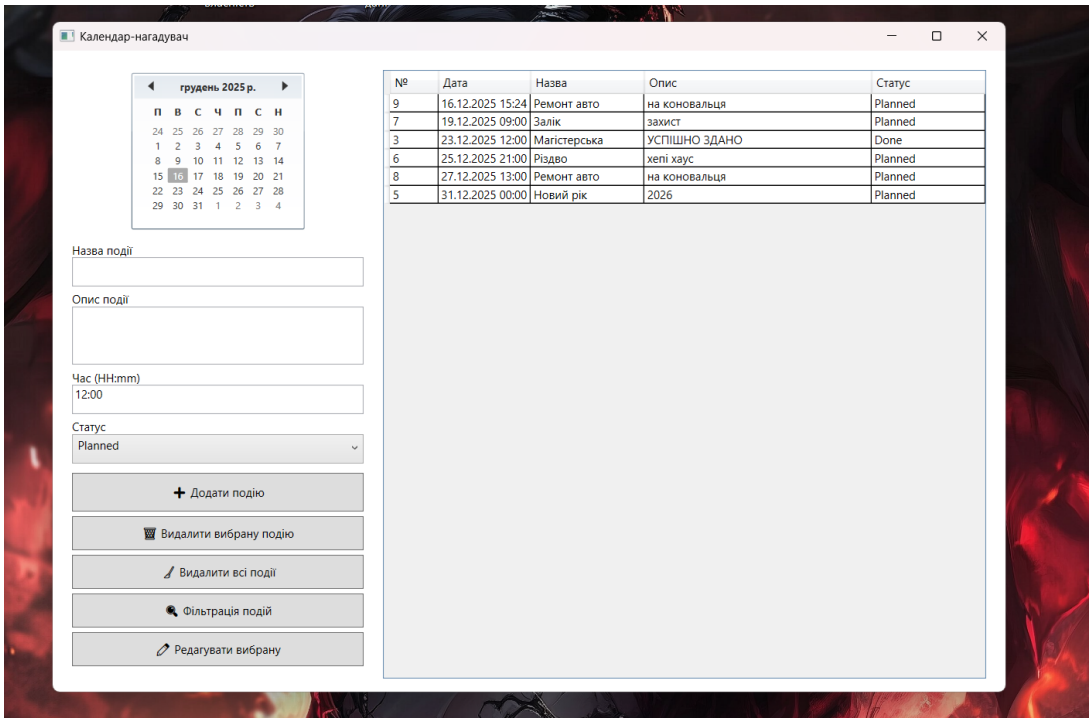

Рисунок 3.13 – Запуск програмної системи після розгортання на локальному комп’ютері

3.2.2 Системні вимоги до програмної системи

Системні вимоги до розробленої програмної системи є відносно невисокими, що дає змогу використовувати її на більшості сучасних персональних комп’ютерів. Оскільки застосунок не виконує ресурсоємних обчислень, не застосовує складних

графічних ефектів і працює з локальною базою даних невеликого обсягу, його стабільна робота забезпечується навіть на пристроях середнього рівня продуктивності [11].

Мінімальні системні вимоги включають:

- операційну систему: Windows 10 або Windows 11;
- програмну платформу: .NET з підтримкою технології WPF;
- процесор: двоядерний із тактовою частотою від 1,6 ГГц;
- оперативну пам’ять: щонайменше 4 ГБ;
- вільне місце на диску: від 100 МБ для зберігання виконуваного файлу та бази даних;
- роздільну здатність екрана: мінімум 1366×768.

Для підвищення комфорту роботи з інтерфейсом, особливо при відображенні великої кількості подій у таблиці, рекомендовані наступні характеристики:

- операційна система: Windows 11;
- процесор: чотириядерний;
- оперативна пам’ять: 8 ГБ або більше;
- роздільна здатність екрана: Full HD (1920×1080).

Застосування SQLite як вбудованої системи керування базами даних дозволяє суттєво знизити навантаження на систему та забезпечує швидкий доступ до даних навіть за умов обробки значної кількості подій. Усі операції виконуються локально, що позитивно впливає на продуктивність і стабільність роботи застосунку.

3.2.3 Особливості впровадження та експлуатації

Після встановлення програмна система готова до використання без додаткових налаштувань чи спеціальної підготовки користувача. Усі дані зберігаються локально в базі даних SQLite, що мінімізує ризик витоку інформації і полегшує контроль над їх збереженням. Такий підхід особливо зручний для персональних додатків з тайм-менеджменту, де конфіденційність даних є

визначальним фактором.

Для перенесення даних на інший комп’ютер або створення резервної копії достатньо скопіювати файл events.db у потрібну директорію нового пристрою. Це гарантує простий і надійний спосіб резервного копіювання без необхідності в використанні додаткових інструментів.

Додаток не вимагає спеціальних технічних знань для його використання. Інтерфейс розроблений таким чином, щоб бути зручним та інтуїтивно зрозумілим: основні функції виконуються за допомогою кнопок, а результати дій супроводжуються повідомленнями системи. Завдяки цьому програма підходить як для освітніх цілей, так і для особистого або професійного планування часу.

3.3. Верифікація програмної системи

Проведення верифікації програмної системи мало на меті підтвердити відповідність створеного програмного продукту визначеним вимогам, поставленій меті та завданням магістерської роботи. На відміну від тестування, яке орієнтоване на виявлення помилок і перевірку коректності виконання окремих функцій, верифікація спрямована на визначення того, чи вирішує система заявлені завдання та чи відповідає вона функціональним і нефункціональним вимогам [11,12].

У процесі роботи здійснювався порівняльний аналіз між початковими вимогами, розробленими на етапі проєктування, та фактично реалізованими можливостями системи. Основну увагу приділяли перевірці повноти функціоналу, правильності архітектурних рішень, зручності взаємодії користувача із системою, а також стабільності її роботи у типових сценаріях використання.

Результати підтвердили досягнення основної мети розробки — створення автономної настільної програмної платформи для керування подіями та тайм-менеджменту з підтримкою збереження даних і механізму нагадувань. Реалізований WPF-застосунок дозволяє проходити авторизацію, створювати події, переглядати їх у табличному форматі, редагувати, видаляти, фільтрувати за діапазонами дат і отримувати нагадування у визначений час. Усі ці можливості

відповідають поставленим завданням.

Перевірка функціональних вимог засвідчила, що система коректно реалізує базові операції CRUD над подіями. Додавання нових записів супроводжується перевіркою введених даних, що знижує ризик помилок або неповної інформації у базі. Редагування та видалення подій дозволяються виключно для обраних записів, скорочуючи ймовірність випадкового видалення важливих даних. Функція фільтрації за датами забезпечує швидкий доступ до релевантної інформації — ключова властивість для програм тайм-менеджменту [11,12].

Особлива увага приділялася збереженню та цілісності даних. Застосування локальної бази даних SQLite забезпечує надійне зберігання інформації між сеансами роботи програми. Події, створені раніше, автоматично завантажуються при наступному запуску програми, що свідчить про правильність реалізації сервісного шару доступу до даних і ініціалізації бази.

Механізм авторизації успішно відповідає вимогам базового контролю доступу. Вікно авторизації блокує можливість переходу до основного функціоналу без введення правильних облікових даних. У разі неправильного вводу з’являється відповідне повідомлення, що забезпечує елементарну інформаційну безпеку для локального застосунку.

Також було перевірено коректність роботи нагадувань — одного з основних компонентів системи тайм-менеджменту. Нагадування реалізовані на основі таймерів WPF та активуються під час активного сеансу роботи програми. У визначений час користувач отримує повідомлення з текстом події, що допомагає своєчасно реагувати на заплановані задачі. Такий підхід відповідає очікуванням щодо автономності застосунку.

Архітектурна перевірка підтвердила раціональність використання моделі, схожої на MVC. Системна логіка інкапсульована в окремих моделях, а логіка доступу до даних винесена в сервісний шар, тоді як взаємодія з користувачем реалізована через WPF-інтерфейс. Таке структурування спрощує подальшу підтримку коду та сприяє легшому розширенню функціональності без суттєвих змін.

Під час верифікації особливу увагу було приділено нефункціональним характеристикам системи, серед яких зручність використання і стабільність роботи. Інтерфейс застосунку відзначається інтуїтивною зрозумілістю: усі ключові операції виконуються за допомогою логічних елементів керування, а їх результати супроводжуються відповідними системними повідомленнями. У ході тестування критичних збоїв або неконтрольованих завершень роботи програми не виявлено, що свідчить про належний рівень надійності реалізації.

Отже, результати верифікації підтверджують, що створена програмна система відповідає поставленим цілям і завданням магістерської роботи, а також заявленим функціональним і архітектурним вимогам. Програмна платформа демонструє працездатність, стабільність та готовність до практичного використання в умовах автономного настільного середовища. Її архітектурні рішення забезпечують можливість подальшого розширення функціоналу, включаючи додавання нових типів подій, удосконалення механізмів повторюваності, покращення процесу авторизації або інтеграцію з іншими сервісами, що лише підкреслює перспективність обраних підходів [11,12].

4. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

У розділі розглядаються питання охорони праці та безпеки життєдіяльності при роботі з програмною платформою в умовах надзвичайних ситуацій.

4.1. Охорона праці

Під час створення високорівневої програмної платформи для управління подіями та тайм-менеджментом на базі WPF особливу увагу приділили дотриманню вимог із охорони праці, виробничої санітарії, електробезпеки та пожежної безпеки. Діяльність програмного розробника передбачає тривалий час роботи за комп’ютером, що супроводжується значним зоровим, статичним та психоемоційним навантаженням. Тому забезпечення безпечних умов праці залишається ключовим фактором для збереження здоров’я і підтримання ефективності.

Нормативно-правову основу з охорони праці, на яку спиралися під час виконання магістерської роботи, складають такі документи: Закон України «Про охорону праці», Кодекс законів про працю України, Закон України «Про пожежну безпеку», Правила пожежної безпеки в Україні (НАПБ А.01.001-2014), ДСТУ ISO 45001:2019 «Системи управління охороною здоров’я та безпекою праці», ДБН В.2.5-28:2018 «Природне і штучне освітлення», а також чинні державні стандарти та нормативи у сферах ергономіки і безпеки праці.

Робоче місце програмного розробника організовано за сучасними стандартами ергономіки та безпеки. Приміщення обладнане достатнім природним та штучним освітленням відповідно до вимог ДБН В.2.5-28:2018, що забезпечує комфортну роботу з монітором без утворення відблисків чи тіней, які можуть шкодити зору.

Меблі робочого місця підібрано з урахуванням антропометричних характеристик користувача. Крісла мають регулювання висоти сидіння та нахилу спинки, що сприяє правильній осанці та мінімізує статичне навантаження на

опорно-руховий апарат. Монітор комп’ютера розташовано на оптимальній відстані 50–70 см від очей, із верхньою межею екрана на рівні очей або трохи нижче, що відповідає сучасним рекомендаціям для комфортної роботи з відеодисплейними терміналами.

Розробка програмного забезпечення потребує тривалої взаємодії з програмним кодом, документацією та графічними інтерфейсами, що створює значне навантаження на зір і психоемоційний стан. Для мінімізації негативного впливу на органи зору застосовувалися оптимальні налаштування монітора, такі як яскравість, контрастність і колірна температура, а також використовувалися темні або нейтральні теми у середовищах розробки.

Робочий процес організовувався з урахуванням режимів роботи та відпочинку. Протягом години безперервної роботи за комп’ютером передбачалися заплановані перерви для виконання гімнастики для очей і зміни положення тіла. Такий режим допомагає зменшити ризик надмірного навантаження на зір, нервового перенапруження та запобігти професійним захворюванням, обумовленим тривалим використанням комп’ютерного обладнання.

У процесі виконання магістерської роботи особливу увагу приділяли питанням електробезпеки. Все комп’ютерне обладнання та периферійні пристрої забезпечувалися справною ізоляцією та підключалися до електромережі із захисним заземленням. Використання несправних кабелів, подовжувачів і розеток було категорично заборонено, а перед початком роботи проводився огляд обладнання для виявлення можливих пошкоджень.

Персональний комп’ютер живився через сертифікований блок живлення з інтегрованими засобами захисту від перенапруги та короткого замикання. Це знижувало ризики ураження електричним струмом, пожеж і поломки апаратури.

Забезпечення пожежної безпеки відповідало вимогам чинних Правил пожежної безпеки в Україні. Приміщення для роботи було забезпечене первинними засобами пожежогасіння, зокрема порошковим вогнегасником. Розміщення електрообладнання виконувалось із дотриманням нормативів відстаней, а перевантаження електромережі виключалося.

Легкозаймисті матеріали не розміщувалися поруч із комп'ютерною технікою, а експлуатація електроприладів здійснювалася згідно з правилами безпеки. Після завершення роботи всі пристрої відключалися від мережі, що значно знижувало ризик виникнення пожежонебезпечних ситуацій.

Вимоги охорони праці були враховані на етапі розробки програмного продукту. Інтерфейс застосунку, створеного на основі WPF, розроблено таким чином, щоб бути зрозумілим, добре структурованим і вільним від зайвого нагромадження візуальних елементів. Використання збалансованої кольорової гами, чіткої типографіки та інтуїтивної навігації допомагає знизити когнітивне навантаження на користувача.

Програмне забезпечення не включає різких візуальних ефектів чи мерехтінь, які можуть негативно позначитися на зорі користувача. Такий підхід повністю відповідає сучасним стандартам безпечного проєктування програмних інтерфейсів, забезпечуючи комфорт і безпеку роботи з додатком.

4.2. Вплив радіоактивного та світлового випромінювання на надійність роботи програмної платформи під час НС та заходи захисту.

У сучасних реаліях зростає ризик виникнення надзвичайних ситуацій техногенного та воєнного характеру, які супроводжуються впливом небезпечних фізичних факторів, таких як радіоактивне та світлове випромінювання. Ці чинники не лише серйозно загрожують здоров'ю людей, але й здатні значно впливати на функціонування технічних і інформаційних систем, зокрема програмних платформ, що використовуються для організації процесів, зберігання та обробки даних [31].

Програмна платформа, створена в межах магістерської роботи для управління подіями та тайм-менеджментом, працює на персональних комп'ютерах і серверному обладнанні. Ці електронні технічні засоби є чутливими до впливу іонізуючого та світлового випромінювання. У випадку надзвичайної ситуації такі фактори можуть спричинити відмову апаратних компонентів і, як наслідок, викликати збої у роботі програмного забезпечення [32].

Іонізуюче випромінювання може виникати внаслідок аварій на об’єктах із радіаційною небезпекою, застосування ядерної зброї чи пошкодження ядерних установок. Головна загроза такого випромінювання для електронних систем полягає в його здатності змінювати електричні характеристики напівпровідникових компонентів, що лежать в основі роботи комп’ютерної техніки [31].

Під дією радіації можуть виникати збої у роботі процесорів, оперативної пам’яті, пристроїв зберігання даних та мережевого обладнання. Наслідками стає спотворення інформації, втрата даних, помилки у виконанні програмного коду або навіть повна відмова системи. Для програмної платформи тайм-менеджменту це надзвичайно критично, адже порушується цілісність бази даних, доступність сервісів і правильність роботи ключових механізмів планування та нагадувань [32].

Світлове випромінювання, що виникає, зокрема, під час вибухів або пожеж значної інтенсивності, відзначається високою температурою та сильним світловим потоком. Його вплив може призводити до перегріву компонентів комп’ютерної техніки, пошкодження кабельної ізоляції, екранів моніторів та корпусів електронних пристроїв [31].

Перегрів створює несприятливі умови для роботи апаратного забезпечення, що часто спричиняє аварійне завершення роботи програмного забезпечення, втрату користувацьких сесій і некоректне збереження даних. Отже, світлове випромінювання впливає на надійність програмного забезпечення через погіршення стану або вихід з ладу апаратних засобів [32].

Для зменшення впливу радіаційного та світлового випромінювання на функціонування програмної платформи важливо запроваджувати комплекс організаційних, технічних і програмних заходів захисту. Організаційні заходи передбачають розміщення серверного й комп’ютерного обладнання в спеціально підготовлених приміщеннях з обмеженим доступом, використання конструкцій для захисту та дотримання норм цивільного захисту [31].

Серед технічних заходів рекомендується застосування захищених корпусів, систем примусового охолодження, стабілізаторів напруги й джерел безперебійного

живлення. Ці засоби допомагають знизити ризик виходу з ладу апаратного обладнання під впливом небезпечних чинників та забезпечують коректне завершення роботи програмного забезпечення [32].

На рівні програмної платформи доцільне використання механізмів резервного копіювання даних, автоматичного відновлення після збоїв і контролю цілісності інформації. Регулярне резервне копіювання баз даних MySQL та конфігураційних файлів дозволяє мінімізувати втрати даних у разі пошкодження обладнання під час надзвичайної ситуації [31].

Ключовим елементом безпеки програмної платформи в умовах надзвичайної ситуації є розробка заходів для забезпечення безперервності роботи інформаційних систем. Це включає віддалене зберігання резервних копій, можливість перенесення роботи платформи на альтернативне обладнання, а також дотримання інструкцій цивільного захисту для персоналу на випадок непередбачуваних обставин [32].

Комплексний підхід до захисту програмно-технічних засобів дозволяє зменшити шкідливий вплив радіаційного й світлового випромінювання на програмну платформу та забезпечити її стабільну роботу навіть за умов надзвичайних ситуацій.

ВИСНОВКИ

У процесі виконання магістерської кваліфікаційної роботи було розв’язано технічну задачу розробки високорівневої програмної платформи для управління подіями та тайм-менеджментом на основі технології Windows Presentation Foundation (WPF). Розроблене програмне забезпечення реалізовано у форматі автономного настільного додатка, який дозволяє ефективно планувати події, контролювати час і своєчасно інформувати користувачів завдяки системі нагадувань.

У ході дослідження проведено аналіз предметної області управління часом та існуючих програмних рішень. На основі отриманих даних сформульовано функціональні і нефункціональні вимоги до системи. Обґрунтовано вибір мови програмування C#, платформи .NET, технології WPF для побудови користувацького інтерфейсу і локальної бази даних SQLite для зберігання даних. Аналіз підтвердив доцільність використання обраного технологічного стеку для створення автономних настільних додатків із зручним інтерфейсом і структурованою архітектурою.

У результаті розробки створено додаток з MVC-подібною архітектурою, яка розділяє відповідальність між моделлю даних, інтерфейсом користувача і логікою керування. Модель предметної області реалізована у вигляді класів для подій, додано сервісний шар доступу до даних і автоматично створюється локальна база SQLite. Такий підхід дозволив зменшити залежність компонентів системи між собою, полегшити її підтримку і створити основу для масштабування функціоналу.

Практичним результатом роботи є створення працездатного програмного застосунку, який реалізує такі функціональні можливості:

- автентифікація користувачів;
- створення, редагування або видалення подій;
- збереження інформації в локальній SQLite базі даних;
- відображення подій у таблиці з використанням механізмів прив’язки даних WPF;

- фільтрація подій за датами;
- формування нагадувань в установлені терміни.

Якість отриманих результатів підтверджується стабільною роботою програми, коректною обробкою дій користувачів без критичних помилок під час тестування. Кількісні показники демонструють можливість обробляти значну кількість подій без помітного впливу на продуктивність завдяки використанню легкої локальної бази SQLite та оптимізації запитів до даних.

Достовірність результатів забезпечується застосуванням перевіреного технологічного стеку .NET, параметризованих SQL-запитів і сценарного тестування базових функцій програми із врахуванням правильних та помилкових варіантів взаємодії.

Практичне значення роботи полягає в можливості використання цього додатка як готового інструменту для особистого, навчального чи професійного тайм-менеджменту. Окрім того, система може бути навчальним прикладом з реалізації WPF-додатків із об’єктно-орієнтованою структурою, сервісним доступом до даних та архітектурою типу MVC.

Наукова новизна роботи полягає у практичній реалізації автономної платформи управління часом, яка поєднує сучасний графічний інтерфейс, локальне збереження даних та систему нагадувань без залучення зовнішніх серверних компонентів.

Перспективи розвитку системи передбачають додавання функцій повторюваних подій, категоризації та пріоритетності завдань, формування статистики виконання, а також інтеграцію з хмарними сервісами чи мобільними платформами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Sommerville I. Software Engineering. 10th ed. – Harlow : Pearson Education, 2020. – 792 p.
2. Pressman R. S., Maxim B. R. Software Engineering: A Practitioner’s Approach. – New York : McGraw-Hill, 2020. – 976 p.
3. Fowler M. Patterns of Enterprise Application Architecture. – Boston : Addison-Wesley, 2020. – 560 p.
4. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. – Addison-Wesley, 2020. – 395 p.
5. Martin R. C. Clean Code: A Handbook of Agile Software Craftsmanship. – Pearson Education, 2020. – 464 p.
6. Nielsen J. Usability Engineering. – San Francisco : Morgan Kaufmann, 2020. – 362 p.
7. Shneiderman B., Plaisant C. Designing the User Interface. – Pearson Education, 2021. – 600 p.
8. Troelsen A., Japikse P. Pro C# 10 with .NET 6. – New York : Apress, 2022. – 1360 p.
9. Petzold C. Programming Windows with C#. – Microsoft Press, 2021. – 1120 p.
10. Myers G. J., Sandler C., Badgett T. The Art of Software Testing. – Wiley, 2021. – 256 p.
11. ISO/IEC 25010:2011. Systems and software Quality Requirements and Evaluation.
12. ISO/IEC 12207. Software Life Cycle Processes.
13. Microsoft Docs. Windows Presentation Foundation (WPF) Overview [Електронний ресурс]. – 2025. – Режим доступу: https://learn.microsoft.com/dotnet/desktop/wpf/ (дата звернення: 05.10.2025).

14. Microsoft Docs. Data Binding in WPF [Електронний ресурс]. – 2025. – Режим доступу: https://learn.microsoft.com/dotnet/desktop/wpf/data/ (дата звернення: 07.10.2025).

15. Microsoft Docs. Commands and Events in WPF [Електронний ресурс]. – 2025. Режим доступу: https://learn.microsoft.com/dotnet/desktop/wpf/advanced/ (дата звернення: 10.10.2025).

16. Microsoft Docs. .NET Desktop Development Guide [Електронний ресурс]. – 2025. – Режим доступу: https://learn.microsoft.com/dotnet/desktop/ (дата звернення: 12.10.2025).

17. Microsoft Docs. Secure Coding Guidelines for .NET [Електронний ресурс]. – 2025. – Режим доступу: https://learn.microsoft.com/dotnet/standard/security/ (дата звернення: 15.10.2025).

18. SQLite Consortium. SQLite Documentation [Електронний ресурс]. – 2025. – Режим доступу: https://www.sqlite.org/docs.html (дата звернення: 18.10.2025).

19. Microsoft Learn. Using SQLite with .NET [Електронний ресурс]. – 2025. – Режим доступу: https://learn.microsoft.com/dotnet/standard/data/sqlite/ (дата звернення: 20.10.2025).

20. Microsoft Learn. Visual Studio 2022 Documentation [Електронний ресурс]. – 2025. – Режим доступу: https://learn.microsoft.com/visualstudio/ (дата звернення: 25.10.2025).

21. OECD. Skills for a Digital World [Електронний ресурс]. – 2025. – Режим доступу: https://www.oecd.org/education/ (дата звернення: 28.10.2025).

22. European Commission. Digital Education Action Plan 2021–2027 [Електронний ресурс]. – 2025. – Режим доступу: https://education.ec.europa.eu/focus-topics/digital-education/action-plan (дата звернення: 02.11.2025).

23. Allen D. Getting Things Done Methodology [Електронний ресурс]. – 2025. – Режим доступу: https://gettingthingsdone.com/ (дата звернення: 05.11.2025).

24. Covey S. R. Time Management Concepts [Електронний ресурс]. – 2025. – Режим доступу: https://www.franklincovey.com/ (дата звернення: 08.11.2025).

25. Microsoft Docs. DispatcherTimer Class [Електронний ресурс]. – 2025. – Режим доступу: https://learn.microsoft.com/dotnet/api/system.windows.threading.dispatchertimer (дата звернення: 12.11.2025).

26. Microsoft Docs. WPF DataGrid Control [Електронний ресурс]. – 2025. – Режим доступу: https://learn.microsoft.com/dotnet/desktop/wpf/controls/datagrid (дата звернення: 15.11.2025).

27. Microsoft Docs. Application Security Best Practices [Електронний ресурс]. – 2025. – Режим доступу: https://learn.microsoft.com/security/ (дата звернення: 18.11.2025).

28. WPF Community. Best Practices for WPF Applications [Електронний ресурс]. – 2025. – Режим доступу: https://github.com/dotnet/wpf (дата звернення: 22.11.2025).

29. Microsoft Docs. .NET Deployment Guide [Електронний ресурс]. – 2025. – Режим доступу: https://learn.microsoft.com/dotnet/core/deploying/ (дата звернення: 05.12.2025).

30. Методичні вказівки до виконання кваліфікаційної роботи магістра для здобувачів спеціальності 121 – Інженерія програмного забезпечення, всіх форм навчання / укладачі: Михалик Д.М., Цуприк Г.Б., Бревус В.М., Мудрик І.Я. – Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2024. – 44 с. URL: https://elartu.tntu.edu.ua/handle/lib/50316 (дата звернення: 13.10.2025).

31. Стручок В.С. Навчальний посібник «Техноекологія та цивільна безпека. Частина «Цивільна безпека»». – Тернопіль: ФОП Паляниця В. А., 156 с.

32. Методичні вказівки до виконання кваліфікаційної роботи магістра для здобувачів спеціальності 121 – Інженерія програмного забезпечення, всіх форм навчання / укладачі: Михалик Д.М., Цуприк Г.Б., Бревус В.М., Мудрик І.Я. – Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2024. – 44 с. (https://elartu.tntu.edu.ua/handle/lib/50316).

33. Methods of constructing algorithms for comparative test statistical verification of mathematical models of bioobject responses to low-intensity stimuli / Bohdan Yavorskyy, Evhenia Yavorska, Halyna Tsupryk, Roman Kinash // Scientific Journal of TNTU. — Tern.: TNTU, 2023. — Vol 112. — No 4. — P. 82–90.

34. Олянін, Д., Цуприк, Г. (2025)Transformer Neural Networks in Industry 4.0 / Д. Олянін, Г. Цуприк, Т. Говорущенко, О. Багрій-Заяць, І. Андрущак // Computer Information Technologies in Industry 4.0: proceedings of the 3rd International Workshop (CITI-2025), Ternopil, Ukraine, 11–12 June 2025. – Ternopil : Ternopil Ivan Puluj National Technical University, 2025 (Scopus) https://ceur-ws.org/Vol-4057/

35. Tsupryk, H., Olianin, D. (2025). Vydobuvannia danyh z tekstu vykorystovuiuchy transformerni neironni merezhi [Data extraction from text using Transformer Neural Networks]. Information Technology: Computer Science, Software Engineering and Cyber Security, 125–130, DOI: https://doi.org/10.32782/IT/2025-2-13

36. ОЛЯНІН D., & ЦУПРИК Н. (2025). Огляд ролі трансформерних нейроних мереж у видобувані інформації із неструктурованих даних. Measuring and computing devices in technological processes, 82(2), 360–364. https://doi.org/10.31891/2219-9365-2025-82-52

37. Tsupryk H. LLM-based Extraction from Resumes / D. Olianin, H. Tsupryk // Advanced Technologies in Scientific Research: collection of scientific papers with proceedings of the 1st International Scientific and Practical Conference, Rotterdam, Netherlands, 20–22 August 2025. – International Scientific Unity, 2025. – 72-76

ДОДАТКИ

ДОДАТОК А

Апробація результатів

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ІВАНА ПУЛЮЯ

М А Т Е Р І А Л И

XIII НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

17-18 грудня 2025 року

ТЕРНОПІЛЬ
2025

УДК 004.67
В. Масловський; Г. Цуприк, к. т. н., доц.
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

РОЗРОБЛЕННЯ ВИСОКОРІВНЕВОЇ ПЛАТФОРМИ ДЛЯ КЕРУВАННЯ ПОДІЯМИ ТА ТАЙМ-МЕНЕДЖМЕНТОМ НА БАЗІ WPF

UDC 004.67
V. Maslovskyy; H. Tsupryk, PhD., Assoc.Prof.

DEVELOPMENT OF A HIGH-LEVEL PLATFORM FOR EVENT MANAGEMENT AND TIME MANAGEMENT BASED ON WPF

Сучасні платформи управління подіями є важливим інструментом для організації діяльності користувачів, оскільки дозволяють структурувати інформацію та оптимізувати робочі процеси. Зростання обсягів цифрових завдань підсилює потребу у високорівневих системах планування, які поєднують зручність, інтерактивність і надійність. Використання WPF, C#, MySQL дає змогу створити гнучку платформу для роботи з подіями та нагадуваннями. Зростання інформаційних потоків створює потребу у швидкій обробці даних, тому важливо впроваджувати механізми оптимізації роботи із великими масивами інформації, що підвищує продуктивність системи [1–3].

Метою роботи є розробка програмної платформи для управління подіями та тайм-менеджментом з оптимізованим інтерфейсом і функціями планування. У дослідженні розглянуто принципи побудови WPF-інтерфейсів, організації подій, створено базу даних та реалізовано систему авторизації [4].

В умовах збільшення кількості даних ключовим аспектом стає ефективна їх обробка, що забезпечує стабільність і точність роботи функціональних модулів.

Для забезпечення стабільності застосовано валідацію даних, захист користувачів і розмежування доступу. Архітектура MVC підвищила масштабованість та гнучкість системи.

Тестування підтвердило ефективність роботи з подіями різних типів, підтримку нагадувань, фільтрації та групування. Реалізований функціонал покращує організацію часу і може застосовуватися в навчальних та корпоративних середовищах.

Платформа демонструє практичну цінність використання WPF, C#, MySQL та MVC для створення сучасних систем управління подіями, забезпечуючи баланс між зручністю, безпекою та функціональністю [1, 2].

Література

1. Олянін, Д., Цуприк, Г. (2025) Transformer Neural Networks in Industry 4.0 / Д. Олянін, Г. Цуприк, Т. Говорущенко, О. Багрій-Заяць, І. Андрущак // Computer Information Technologies in Industry 4.0: proceedings of the 3rd International Workshop (CITI-2025), Ternopil, Ukraine, 11–12 June 2025. – Ternopil : Ternopil Ivan Puluj National Technical University, 2025 (Scopus)
2. Tsupryk, H., Olianin, D. (2025). Vydobuvannia danyh z tekstu vykorystovuiuchy transformerni neironni merezhi [Data extraction from text using Transformer Neural Networks]. Information Technology: Computer Science, Software Engineering and Cyber Security, 125–130, DOI: https://doi.org/10.32782/IT/2025-2-13
3. Tsupryk, H., Olianin, D. (2025). Overview of transformers role indata mining from unstructured data. International Scientific-technical journal«Measuring and computing devices in technological processes»2025, Issue 2, 125–130, DOI: 10.31891/2219-9365-2025-82-52
4. Tsupryk H. LLM-based Extraction from Resumes / D. Olianin, H. Tsupryk // Advanced Technologies in Scientific Research: collection of scientific papers with proceedings of the 1st International Scientific and Practical Conference, Rotterdam, Netherlands, 20–22 August 2025. – International Scientific Unity, 2025. – 72–76

ДОДАТОК Б

Фрагменти програмного коду програмної платформи

Лістинг Б.1 – Підключення до локальної бази даних SQLite

Цей кодовий приклад ілюструє, як налаштувати підключення настільного WPF-додатка до локальної реляційної бази даних SQLite. Для цього використовується стандартний .NET-провайдер Microsoft.Data.Sqlite, причому файл бази даних створюється автоматично в каталозі користувача вашої системи.

```
using CalendarReminderApp.Models;
using Microsoft.Data.Sqlite;
using System;
using System.Collections.Generic;
using System.IO;
using System.Windows;
namespace CalendarReminderApp.Services
{
    public class DbService
    {
        private readonly string _dbPath;

        public DbService(string fileName)
        {
            string dataDir = Path.Combine(

Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData)
,
                "CalendarReminderApp"
            );

            if (!Directory.Exists(dataDir))
                Directory.CreateDirectory(dataDir);

            _dbPath = Path.Combine(dataDir, fileName);

            MessageBox.Show("SQLite DB path:\n" + _dbPath);
        }

        private SqliteConnection Open()
        {
            Return new SqliteConnection($"Data Source={_dbPath}");
        }
```

Лістинг Б.2 – Ініціалізація структури бази даних

Цей код відповідає за автоматичне формування структури локальної бази даних при першому запуску програми. У процесі виконання створюються таблиці для зберігання інформації про користувачів і події, а також додається початковий тестовий обліковий запис для перевірки роботи системи.

```
public void Init()
 {
     using var con = Open();
     con.Open();

     var cmd = con.CreateCommand();
     cmd.CommandText = @"
     CREATE TABLE IF NOT EXISTS Users(
         Id INTEGER PRIMARY KEY AUTOINCREMENT,
         Login TEXT UNIQUE,
         Password TEXT
     );

     CREATE TABLE IF NOT EXISTS Events(
         Id INTEGER PRIMARY KEY AUTOINCREMENT,
         EventDateTime TEXT,
         Title TEXT,
         Description TEXT,
         Status INTEGER
     );

     INSERT OR IGNORE INTO Users(Login, Password)
     VALUES ('admin','admin');
     ";

     cmd.ExecuteNonQuery();
 }
```

Лістинг Б.3 – Модель події ReminderEvent

У цьому лістингу представлено клас-модель події, який об'єднує ключові атрибути календарної події. Цей клас слугує базовою одиницею даних для виконання всіх операцій із керування подіями

```
using System;

namespace CalendarReminderApp.Models
{
    public enum EventStatus
    {
        Planned,
        Done,
        Canceled
    }

    public class ReminderEvent
    {
        public int Id { get; set; }
        public DateTime EventDateTime { get; set; }
        public string Title { get; set; }
        public string Description { get; set; }
        public EventStatus Status { get; set; }
    }
}
```

Лістинг Б.4 – Перевірка автентифікації користувача

Цей кодовий фрагмент відповідає за перевірку облікових даних користувача. Його методи застосовуються у вікні авторизації для обмеження доступу до ключових функціональних можливостей програми.

```
        public bool CheckLogin(string login, string password)
        {
            using var con = Open();
            con.Open();

            var cmd = con.CreateCommand();
            cmd.CommandText =
                "SELECT COUNT(*) FROM Users WHERE Login=$l AND
Password=$p;";
            cmd.Parameters.AddWithValue("$l", login);
            cmd.Parameters.AddWithValue("$p", password);

            return Convert.ToInt32(cmd.ExecuteScalar()) > 0;
        }
```

Лістинг Б.5 – Отримання списку подій із бази даних

Наведений приклад ілюструє вибірку всіх подій із бази даних SQLite та

перетворення отриманих даних у колекцію об'єктів класу ReminderEvent, які згодом використовуються для відображення в інтерфейсі користувача.

```
        public List<ReminderEvent> GetEvents()
        {
            var list = new List<ReminderEvent>();

            using var con = Open();
            con.Open();

            var cmd = con.CreateCommand();
            cmd.CommandText = "SELECT * FROM Events ORDER BY
EventDateTime;";

            using var r = cmd.ExecuteReader();
            while (r.Read())
            {
                list.Add(new ReminderEvent
                {
                    Id = r.GetInt32(0),
                    EventDateTime = DateTime.Parse(r.GetString(1)),
                    Title = r.GetString(2),
                    Description = r.GetString(3),
                    Status = (EventStatus)r.GetInt32(4)
                });
            }

            return list;
        }
```

Лістинг Б.6 – Додавання події до бази даних

Цей метод здійснює додавання нової події до локальної бази даних. Інформація передається через параметризований SQL-запит, що забезпечує підвищений рівень безпеки та точність обробки даних.

```
        public void AddEvent(ReminderEvent ev)
        {
            using var con = Open();
            con.Open();

            var cmd = con.CreateCommand();
            cmd.CommandText = @"
            INSERT INTO Events(EventDateTime, Title, Description,
Statu
```

```
            VALUES($dt, $t, $d, $s);";

            cmd.Parameters.AddWithValue("$dt",
ev.EventDateTime.ToString("O"));
            cmd.Parameters.AddWithValue("$t", ev.Title);
            cmd.Parameters.AddWithValue("$d", ev.Description);
            cmd.Parameters.AddWithValue("$s", (int)ev.Status);

            cmd.ExecuteNonQuery();
        }
```

Лістинг Б.7 – Логіка додавання події в головному вікні

Цей кодовий фрагмент ілюструє обробку дій користувача при додаванні нової події через графічний інтерфейс головного вікна застосунку. У методі перевіряється правильність введених даних, створюється об’єкт предметної області ReminderEvent і передається до сервісного шару для подальшого збереження у базі даних. Після цього оновлюється список подій та активується механізм нагадувань.

```
            private void Add_Click(object sender, RoutedEventArgs e)
            {
                if (CalendarControl.SelectedDate == null)
                {
                    MessageBox.Show("Оберіть дату");
                    return;
                }

                if (!TimeSpan.TryParse(TimeBox.Text, out var time))
                {
                    MessageBox.Show("Час у форматі HH:mm");
                    return;
                }

                var ev = new ReminderEvent
                {
                    EventDateTime                                    =
CalendarControl.SelectedDate.Value.Date + time,
                    Title = TitleBox.Text,
                    Description = DescBox.Text,
                    Status = (EventStatus)StatusBox.SelectedItem
                };

                _db.AddEvent(ev);
                LoadEvents();
                StartTimers();
            }
```

Лістинг Б.8 – Реалізація механізму нагадувань

Цей кодовий фрагмент відповідає за створення таймерів для нагадувань користувачу у вказаний час. Основою реалізації є використання класу DispatcherTimer з платформи WPF.

```
private void StartTimers()
  {
      foreach (var ev in _db.GetEvents())
      {
          var delay = ev.EventDateTime - DateTime.Now;

          if (delay.TotalSeconds > 0)
          {
              var timer = new DispatcherTimer
              {
                  Interval = delay
              };

              timer.Tick += (s, e) =>
              {
                  timer.Stop();
                  MessageBox.Show($"🔔Нагадування:{ev.Title}");
              };

              timer.Start();
          }
      }
  }
```