

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ
Циклова комісія інформаційних технологій

КУРСОВИЙ ПРОЕКТ
(РОБОТА)

Програмування
(назва дисципліни)

на тему: Створення календар з нагадуванням про події

Студента (ки) IV курсу 40-ІС групи
спеціальності 151 Автоматизація та
комп'ютерно-інтегровані технології
спеціалізації 5.151.1 Обслуговування
інтелектуальних інтегрованих систем
Островський Р.Д.

Керівник: викладач *Васильєв М.В.*

Національна шкала _____

Кількість балів: ____ Оцінка: ECTS ____

Члени комісії: _____ (Васильєв М.В.)
(підпис)

_____ (_____)
(підпис)

_____ (_____)
(підпис)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ

РОЗГЛЯНУТО

на засіданні циклової комісії
Голова циклової комісії
Марина ВЕЛИЧКО
«24» жовтня 2024 р.

ЗАТВЕРДЖЕНО

Зав. відділення
Олеся ТВЕРДОХЛІБОВА
«25» жовтня 2024 р.

Завдання

на курсовий проєкт студенту гр. 40-ІС
спеціальність/освітньо-професійна програма 151 Автоматизація та
комп'ютерно-інтегровані технології/5.151.1 Обслуговування
інтелектуальних інтегрованих систем

Островського Руслана Дмитровича

Тема: Створення календар з нагадуванням про події.
Термін виконання роботи з 12.11.2024 р. по 31.03.2025 р.
Вхідні дані для проектування: (Варіант №10)

Розробка програмного продукту для нагадування про події. Додаток надає можливість збереження на певну дату, редагувати та видаляти події, можливість позначення подій по важливості і терміновості, нагадування про подію повідомленням в системі. Для зберігання інформації програма використовує базу даних.

Література та посібники для проектування:

1. Head-First Python, 2nd edition: Paul Barry. - Sebastopol, California, U.S.: O'Reilly Media, 2016. – 622 с.
2. Think Python: How to Think Like a Computer Scientist, 2nd edition: Allen B. Downey. - Sebastopol, California, U.S.: O'Reilly Media, 2015. – 292 с.
3. Clean Code: A Handbook of Agile Software Craftsmanship: Robert C. Martin. - London, England: Pearson, 2008. – 464 с.
4. Python.org: веб-сайт. URL: <https://www.python.org> (дата звернення: 01.09.2024)
5. Python Tutorial: веб-сайт. URL: <https://www.w3schools.com/python/> (дата звернення: 01.09.2024)
6. Learn to become a modern Python developer: веб-сайт. URL: <https://roadmap.sh/python/> (дата звернення: 01.09.2024)

Зміст розрахунково-пояснювальної записки

№ п/п	Зміст	Планований термін виконання	Фактичний термін виконання	%
1.	Вступ	30.11.2024	30.12.2024	5
2.	Аналіз задачі, засобів та методів її вирішення	13.12.2024	13.12.2024	15
3.	Проектування загального алгоритму роботи програми	18.12.2024	18.12.2024	40
4.	Розробка програмного забезпечення	02.03.2025	02.03.2025	80
5.	Керівництво користувача	09.03.2025	09.03.2025	90
6.	Висновки	15.03.2025	15.03.2025	95
7.	Список використаних джерел	30.03.2025	30.03.2025	100

Керівник курсової роботи _____ Микола ВАСИЛЬЄВ

Завдання до курсової роботи

Одержав(ла) студент(ка) гр. 40-ІС _____ (_____)

Дата «12» листопада 2024 р.

РЕФЕРАТ

Пояснювальна записка містить сторінки, рисунки, таблиці, використану літературу та додатки.

Об'єктом розробки є програма для створення календар з нагадуванням про події.

Мета розробки – створення програмного забезпечення для зручного планування та управління подіями.

У процесі роботи проведено розробку блок-схеми алгоритму роботи програми та окремих функцій, створено програму, яка реалізує цей алгоритм.

Основні конструктивні та техніко-економічні показники:

висока надійність, зручність у користуванні, оптимізація процесу управління подіями.

Розроблений застосунок може використовуватися для підвищення ефективності планування та нагадування про важливі події.

Python, Visual Studio, календар, подія, нагадування, функція, метод, змінна, параметр.

ПЕРЕЛІК СКОРОЧЕНЬ

МП – Мова програмування;

ПЗ – Програмне забезпечення;

БД – База даних.

ЗМІСТ

РЕФЕРАТ.....	4
ПЕРЕЛІК СКОРОЧЕНЬ	5
1. ВСТУП.....	7
2. АНАЛІЗ ЗАДАЧІ, ЗАСОБІВ ТА МЕТОДІВ ЇЇ ВИРІШЕННЯ	8
2.1 Python	8
2.2 Історія створення мови програмування Python	9
2.3 Середовище розробки Visual Studio	10
2.4 База даних SQL Server	11
3. ПРОЕКТУВАННЯ ЗАГАЛЬНОГО АЛГОРИТМУ РОБОТИ ПРОГРАМИ	13
4. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	20
4.1 Імпорт бібліотек.....	20
4.2 Опис класу ReminderApp та підключення до бази даних	22
4.3 Опис створення графічного інтерфейсу	23
4.4 Створення таблиці в базі даних для зберігання подій.....	27
4.5 Додавання події до бази даних	28
4.6 Відображення подій	29
4.7 Видалення подій.....	30
4.8 Оновлення списку подій	30
4.9 Редагування події.....	31
4.10 Збереження змін у події.....	32
4.11 Завершення події.....	34
4.12 Щоденна перевірка подій	35
4.13 Приховування вікна та робота з іконкою в треї.....	35
4.14 Запуск програми	37
5. КЕРІВНИЦТВО КОРИСТУВАЧА	38
6. ВИСНОВКИ.....	51
7. СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	52
ДОДАТКИ	53

					5.151.1.40-КП							
Змін.	Арк.	№ докум.	Підпис	Дата								
Розроб.		Островський Р.Д.			Пояснювальна записка			Літ.	Арк.	Аркуші		
Перевір.		Васильєв М.В.								6	63	
Т. контр.								ХПФК група 40-ІС				
Н. Контр.												
Затверд.												

1. ВСТУП

У сучасному світі, де інформаційні технології проникають у всі сфери життя, важливим аспектом стає ефективне управління часом та організація подій. Кількість завдань, зустрічей та інших важливих справ постійно зростає, що вимагає наявності зручних та функціональних інструментів для планування.

З метою полегшення процесу управління подіями та нагадування про важливі дати, у даному проекті буде розроблено програмний продукт для нагадування про події. Цей додаток надасть користувачам можливість зберігати події на певні дати, редагувати та видаляти їх, а також позначати події за рівнем важливості та терміновості. Крім того, програма забезпечить автоматичні нагадування у вигляді повідомлень у системі, що допоможе користувачам вчасно виконувати заплановані справи.

Актуальність цього проекту зумовлена необхідністю зручного та надійного інструменту для керування подіями, особливо в умовах інтенсивного робочого ритму. Використання бази даних для збереження інформації гарантує надійність та швидкість доступу до даних, а функція календаря з нагадуванням зробить додаток ще більш корисним для користувачів.

Метою даного проекту є створення зручного та ефективного застосунку, який дозволить автоматизувати процес нагадувань про події, що значно покращить організацію часу користувачів та допоможе їм не забувати про важливі справи.

					5.151.1.40-КП	Лист
						7
Изм.	Лист	№ докум.	Підпис	Дата		

2. АНАЛІЗ ЗАДАЧІ, ЗАСОБІВ ТА МЕТОДІВ ЇЇ ВИРІШЕННЯ

Розробка програмного продукту для нагадування про події здійснюватиметься за допомогою мови програмування Python у середовищі Visual Studio. Це середовище розробки забезпечує широкий набір інструментів для зручної роботи з кодом, налагодження та тестування програм. Python був обраний через свою гнучкість, зручний синтаксис та велику кількість бібліотек, що дозволяють ефективно реалізовувати функціонал програми.

Для зберігання інформації про події буде використовуватися SQL Server Management Studio (SSMS). Ця система керування базами даних надає стабільне середовище для збереження та обробки інформації. База даних міститиме відомості про події, їхні дати, рівень важливості, терміновість та нагадування. Робота з базою даних буде здійснюватися за допомогою бібліотеки pyodbc, що забезпечить зручний доступ до SQL Server.

Також у програмі буде реалізовано систему сповіщень, яка надсилатиме нагадування про заплановані події безпосередньо у вигляді повідомлень. Це зробить додаток зручним для користувачів, які потребують ефективного інструменту для керування подіями та завданнями.

2.1 Python

Python — це високорівнева мова програмування, яка широко використовується у сфері розробки програмного забезпечення, автоматизації процесів та роботи з базами даних. Завдяки динамічній типізації та простому синтаксису, Python є ідеальним вибором для створення зручних застосунків.

Основні переваги Python:

- Легкість у вивченні та використанні завдяки зрозумілому синтаксису.
- Наявність великої кількості бібліотек для роботи з базами даних, обробки даних та створення інтерфейсів.
- Кросплатформність — можливість запуску програм на різних операційних системах без значних змін у коді.

					5.151.1.40-КП	Лист
Изм.	Лист	№ докум.	Підпис	Дата		8

- Використання в провідних компаніях, таких як Google, Microsoft, NASA, Netflix, що підтверджує його надійність та ефективність.

Python дозволяє створювати як прості, так і складні додатки з високою продуктивністю, що робить його чудовим вибором для реалізації програмного продукту з нагадуванням про події.



Рис.2.1.1 – Логотип Python

2.2 Історія створення мови програмування Python

Мова програмування Python була розроблена Гвідо ван Россумом у 1991 році під час його роботи в дослідницькому центрі CWI (Centrum Wiskunde & Informatica) в Нідерландах. Основною метою створення нової мови було забезпечення простоти написання коду, зручності синтаксису та можливості швидкої розробки програм.

При створенні Python ван Россум надихався мовою ABC, яка була орієнтована на навчання програмуванню. Водночас він прагнув усунути її недоліки, такі як обмеженість можливостей і складність розширення. Також на розробку Python вплинули такі мови, як C, Modula-3 і Lisp.

Цікаво, що назва Python не має жодного відношення до змій. Гвідо ван Россум вирішив назвати свою мову саме так, оскільки був фанатом британського комедійного шоу "Monty Python's Flying Circus", що виходило у 70-х роках.

З моменту створення Python активно розвивається завдяки великій спільноті програмістів по всьому світу. Його розвиток регулюється спеціальними документами PEP (Python Enhancement Proposal), які містять

					5.151.1.40-КП	Лист
						9
Изм.	Лист	№ докум.	Підпис	Дата		

пропозиції щодо вдосконалення мови. Завдяки відкритій природі розробки Python регулярно оновлюється, а нові версії виходять із покращеннями в продуктивності, безпеці та функціональності.

2.3 Середовище розробки Visual Studio

Для розробки програмного продукту буде використано середовище Visual Studio, яке є потужним інструментом для створення програм різної складності. Visual Studio підтримує велику кількість мов програмування, включаючи Python, що робить його зручним для розробки застосунків.

Основні переваги Visual Studio для Python-розробки:

- Інтеграція з Python – середовище має вбудовану підтримку Python через Python Tools for Visual Studio (PTVS), що дозволяє легко працювати з кодом.
- Зручний редактор коду – підсвічування синтаксису, автодоповнення та аналіз коду значно спрощують процес розробки.
- Вбудовані інструменти для налагодження – можливість крокового виконання коду, відстеження змінних та виявлення помилок.
- Підтримка роботи з базами даних – завдяки інтеграції з SQL Server Management Studio, Visual Studio дозволяє зручно взаємодіяти з базою даних.
- Гнучке налаштування середовища – можна встановлювати додаткові модулі та розширення для покращення продуктивності роботи.

Visual Studio забезпечує комфортну розробку та тестування програмного забезпечення, що робить його чудовим вибором для створення застосунку для нагадувань про події.

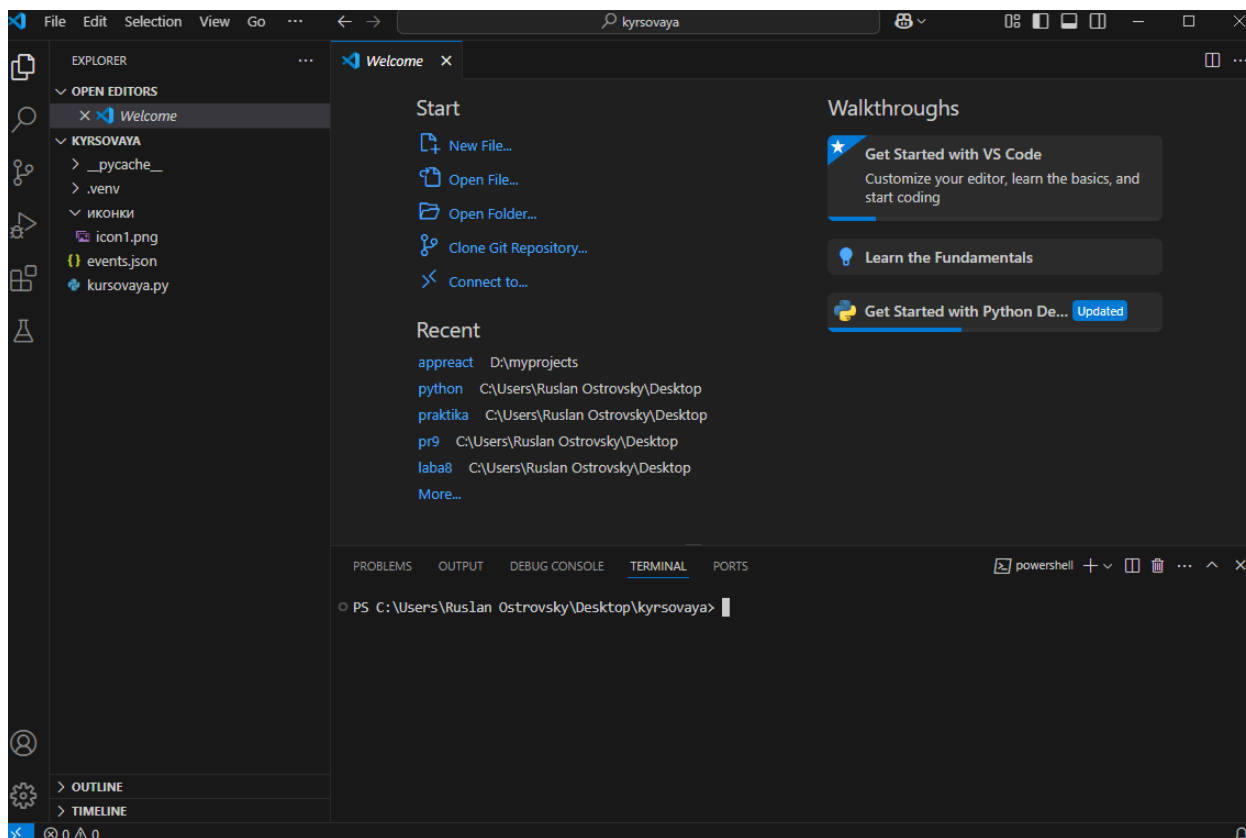


Рис.2.3.1 – Середовище розробки Visual Studio

2.4 База даних SQL Server

Для зберігання та обробки інформації у програмному продукті буде використовуватися SQL Server, керування яким здійснюватиметься через SQL Server Management Studio (SSMS). Це потужна система керування реляційними базами даних (RDBMS), яка забезпечує надійне збереження, швидкий доступ та ефективне управління даними.

Основні переваги використання SQL Server:

- Надійність та безпека – SQL Server підтримує розширені механізми аутентифікації, резервного копіювання та захисту даних.
- Висока продуктивність – ефективно обробляє великі обсяги інформації та оптимізує виконання запитів.
- Зручність у використанні – SQL Server Management Studio надає інтуїтивно зрозумілий інтерфейс для роботи з базами даних.

- Підтримка складних запитів – можливість використання SQL-запитів, тригерів, збережених процедур, що дозволяє реалізувати складну бізнес-логіку.
- Легка інтеграція з Python – завдяки бібліотеці pyodbc забезпечується зручна взаємодія між програмою та базою даних.

Завдяки використанню SQL Server програма матиме можливість зберігати велику кількість подій, швидко виконувати запити та надавати користувачам зручний доступ до їхніх нагадувань

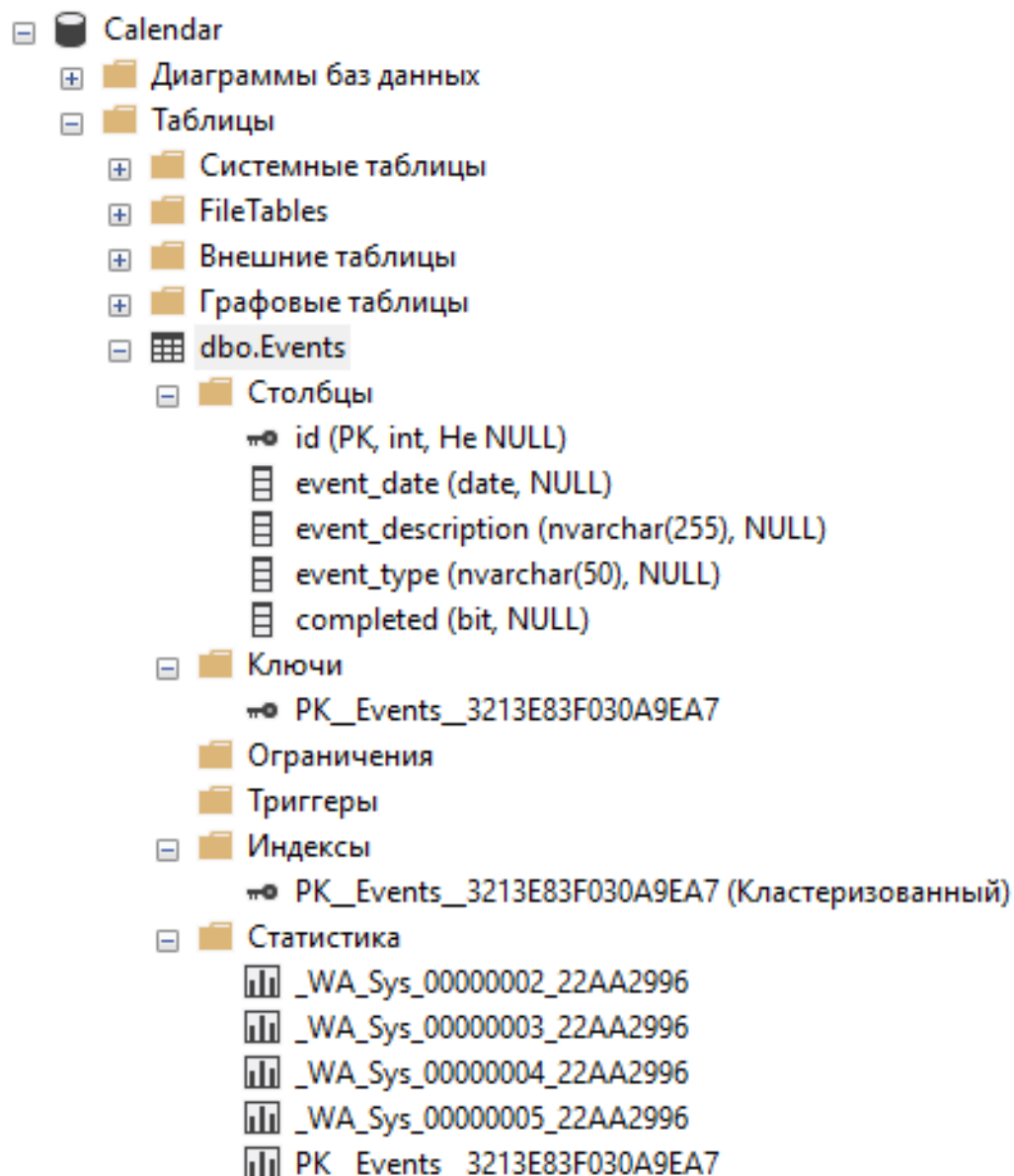


Рис.2.4.1 – База даних SQL Server

3. ПРОЕКТУВАННЯ ЗАГАЛЬНОГО АЛГОРИТМУ РОБОТИ ПРОГРАМИ

Проектування алгоритму роботи програми було зосереджено на розробці основних функцій для взаємодії з календарем. Першим кроком було розробити методи для додавання, редагування, перегляду та видалення подій. Для кожної події потрібно було визначити ключову інформацію, таку як дата, опис, та можливі додаткові параметри (наприклад, пріоритет), і зберігати її для подальшого використання.

Особливу увагу було приділено розробці методів для зручного перегляду всіх подій або окремих, з фільтрацією за датою, категорією чи іншими параметрами. Для зручності користувача були додані функції, що дозволяють відображати події у вигляді списку або в календарному вигляді.

Головний алгоритм роботи програми включає цикл, що реагує на дії користувача, і відповідно до вибору користувача запускає необхідні функції. Інтерфейс програми побудований таким чином, щоб дозволити користувачу легко додавати нові події в календар, редагувати або видаляти існуючі. Кожен крок чітко відображається користувачеві через інтерфейс програми, де кожна дія має своє підтвердження або помилку.

Для полегшення взаємодії з програмою всі дані зберігаються в спеціально створеному форматі, що дозволяє швидко відкривати та змінювати інформацію, а також забезпечує збереження даних при закритті програми, щоб користувач міг продовжити свою роботу без втрати важливої інформації.

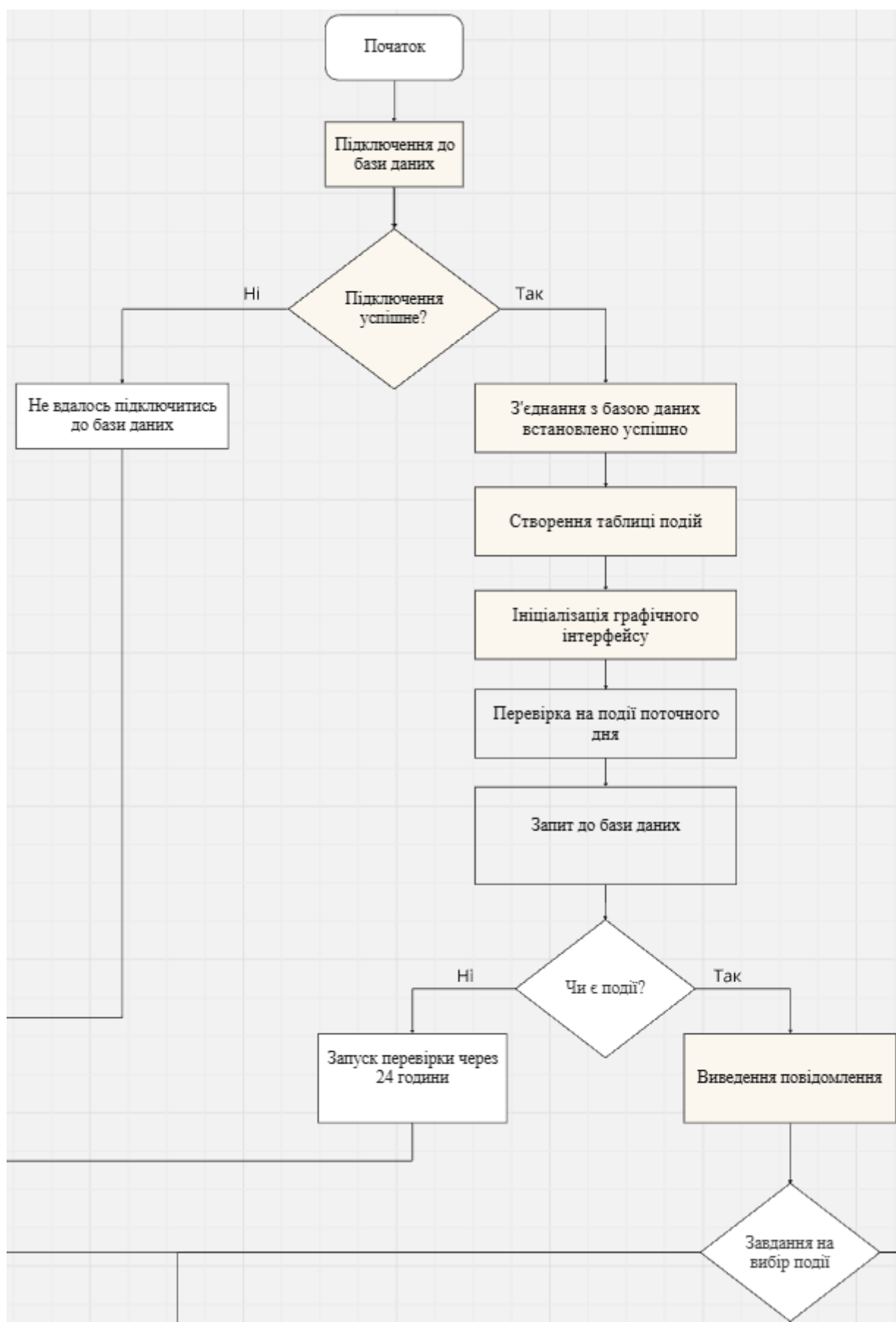


Рис.3.1 – Алгоритм роботи застосунку від початку до завдань

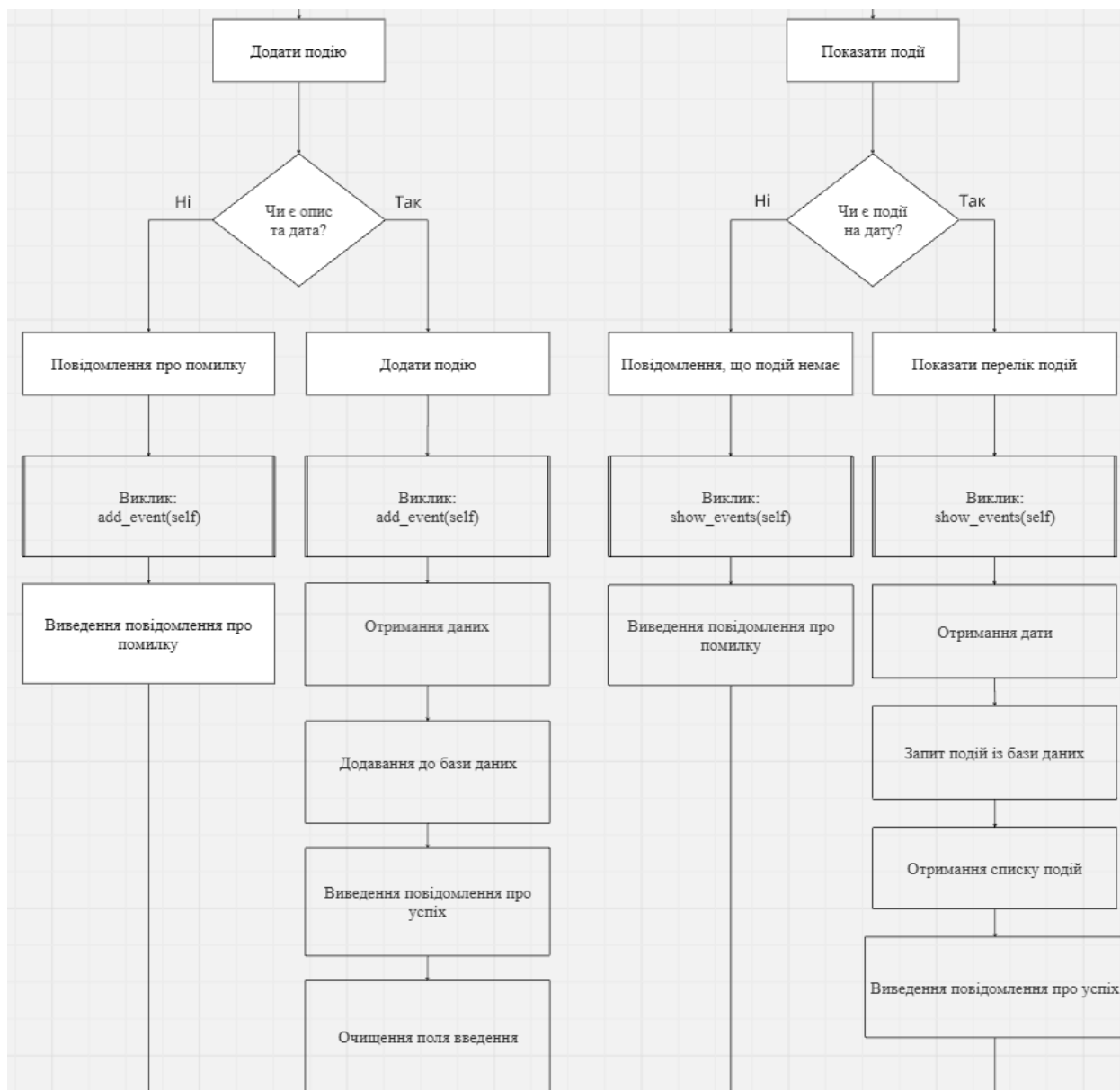


Рис.3.2 – Алгоритм роботи завдань “Додати подію” та “Показати події”

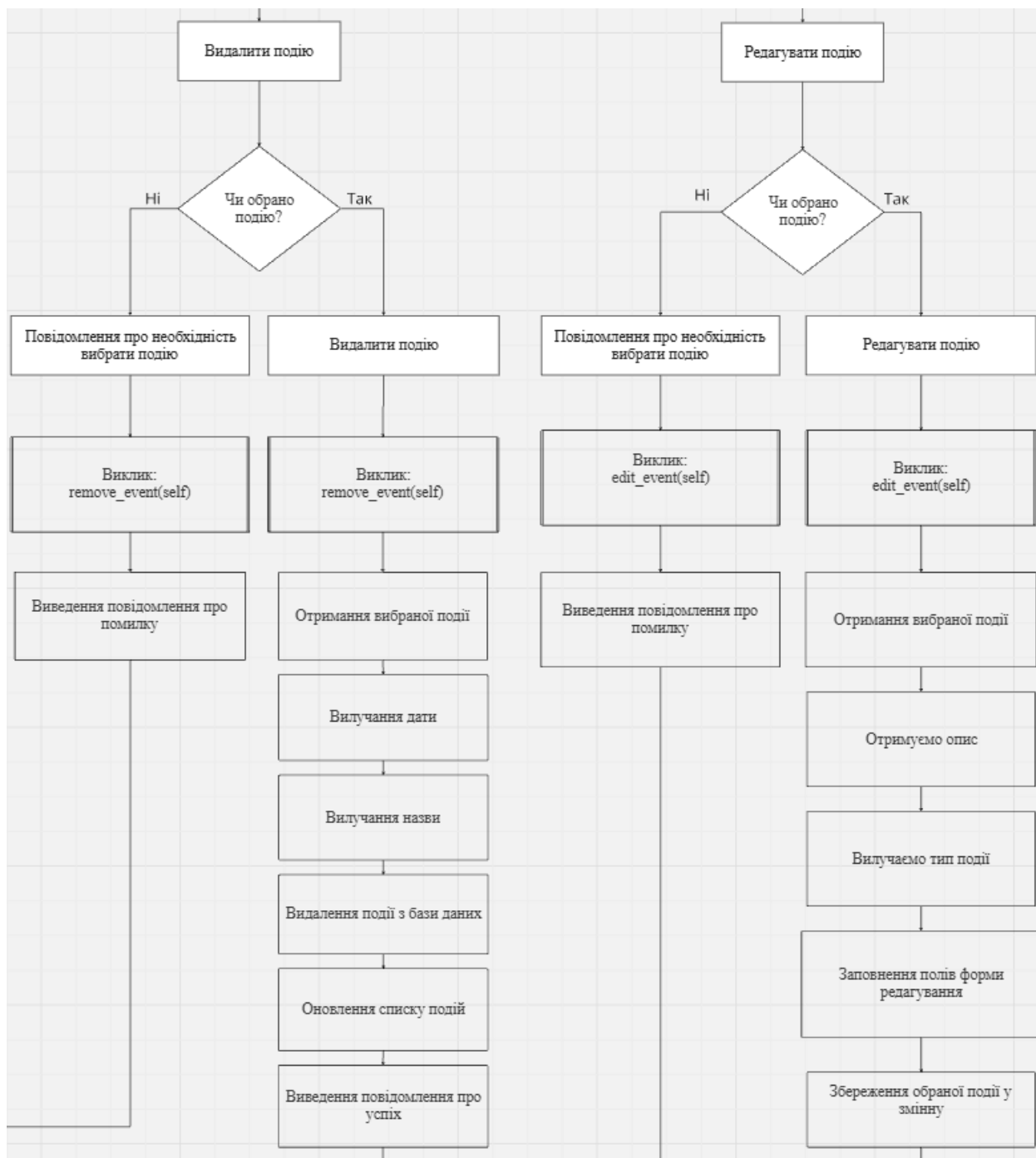


Рис.3.3 – Алгоритм роботи завдань “Видалити подію” та “Редагувати подію”

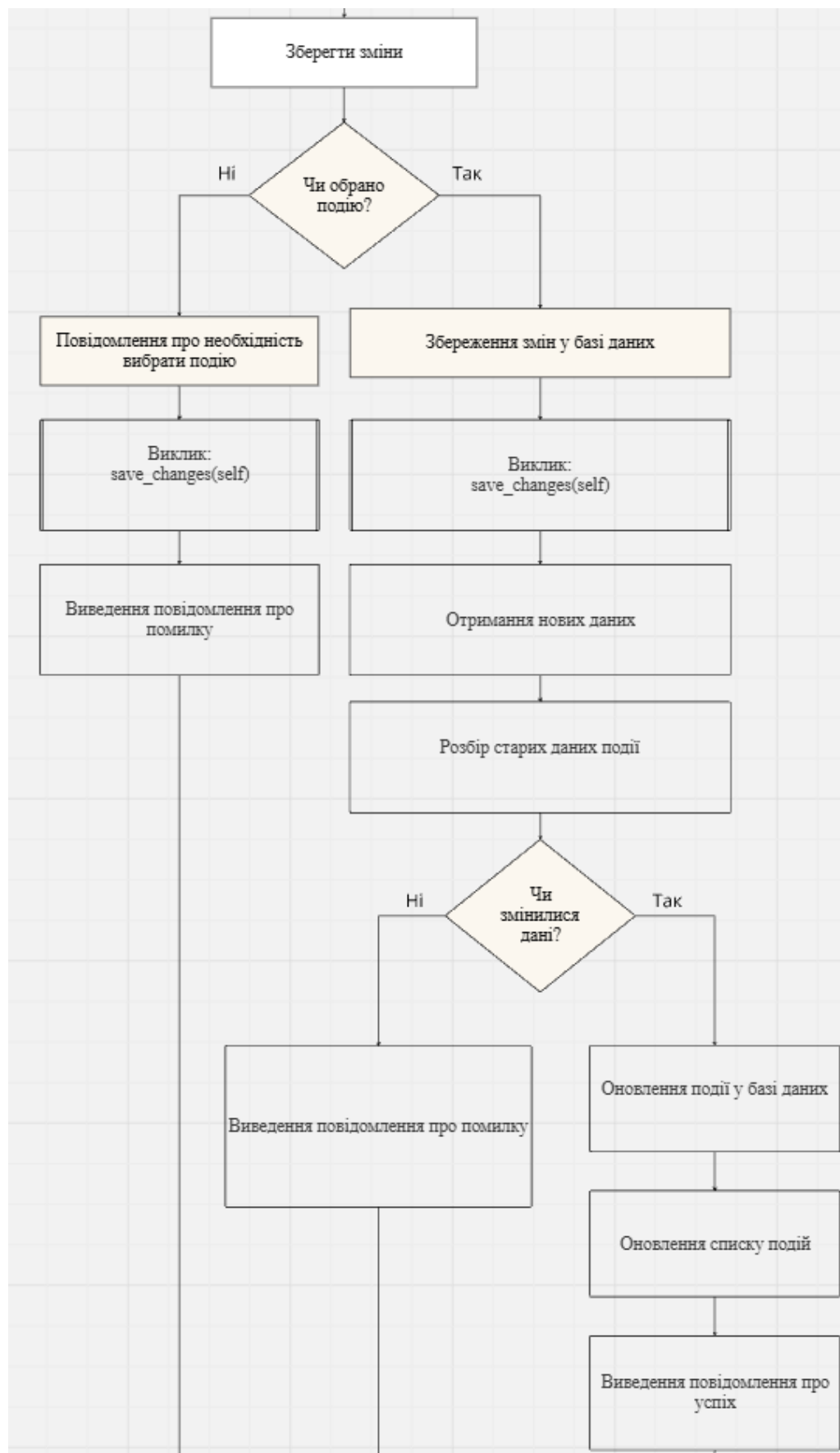


Рис.3.4 – Алгоритм роботи завдань “Зберегти зміни” та “Розбір старих даних подій”

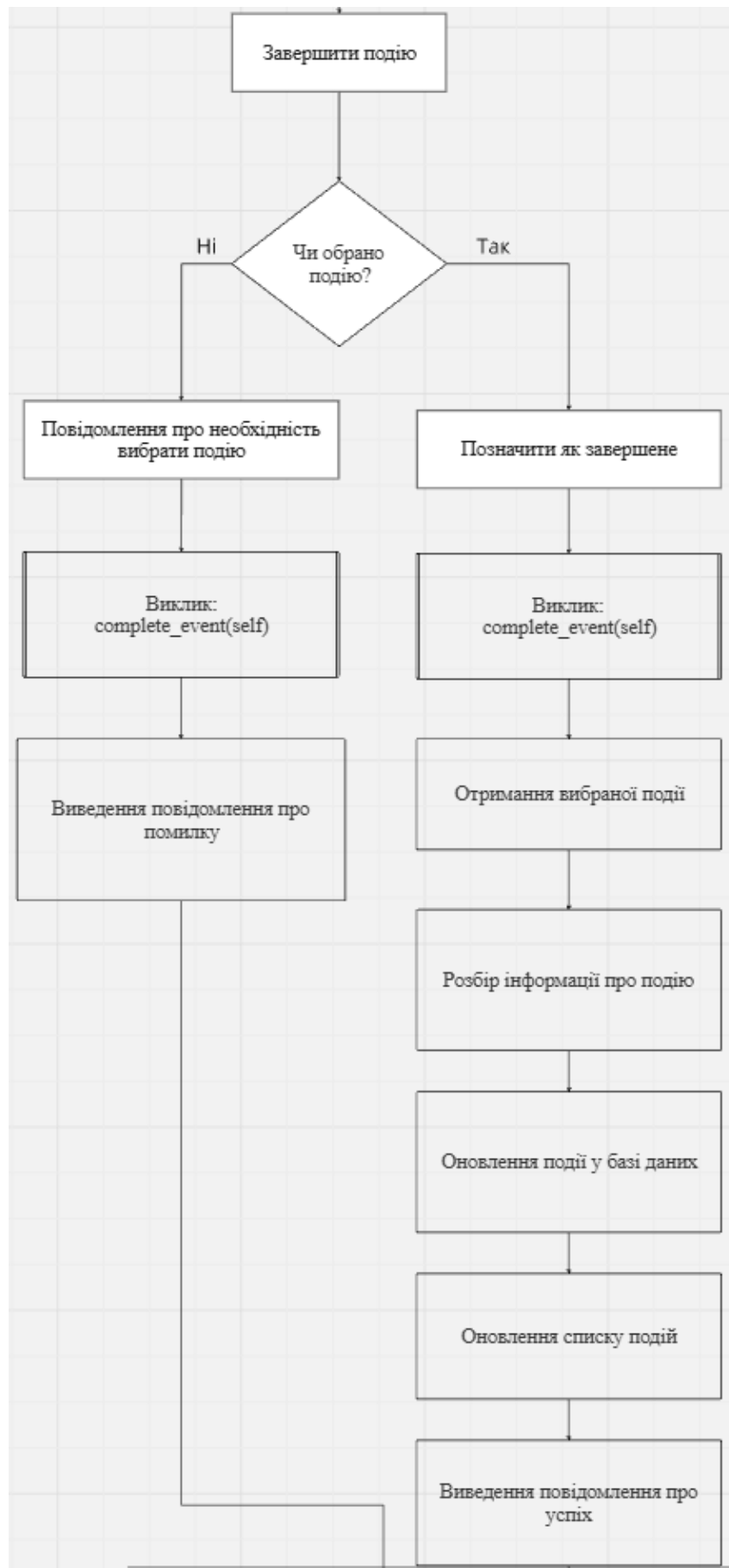


Рис.3.5 – Алгоритм роботи завдання “Завершити подію”



Рис.3.6 – Алгоритм роботи завдань “Приховування вікна” та “Створення іконки”

4. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка програмного забезпечення є важливим етапом у створенні функціональних і ефективних додатків. У цьому розділі описано процес розробки програмного забезпечення для застосунку "Календар з нагадуваннями", який включає використання таких технологій, як Python, tkinter для графічного інтерфейсу, pyodbc для роботи з базою даних SQL Server, та pystray для створення іконки в системному треї.

4.1 Імпорт бібліотек

На початковому етапі розробки програми було імпортовано кілька бібліотек, які забезпечують функціональність для створення графічного інтерфейсу користувача, роботи з базою даних, а також з додатковими можливостями, такими як сповіщення через системний трей і звукові ефекти.

```
import tkinter as tk
from tkinter import messagebox, ttk
from tkcalendar import Calendar
import datetime
import pyodbc
import winsound
from pystray import Icon, MenuItem, Menu
from PIL import Image, ImageDraw
import threading
```

Рис.4.1.1 – Імпорт бібліотек

1. `import tkinter as tk` — бібліотека `tkinter` є стандартним інструментом для створення графічного інтерфейсу користувача (GUI) в Python. Вона надає елементи управління, такі як кнопки, текстові поля, вікна та інші компоненти для взаємодії з користувачем. Тут імпортується основний модуль `tkinter` під псевдонімом `tk`.

2. `from tkinter import messagebox, ttk`:

`Messagebox` — використовується для відображення вікон з повідомленнями (наприклад, для виведення сповіщень або помилок).

ttk — це додатковий набір віджетів для tkinter, який дозволяє використовувати більш сучасні та стильні елементи інтерфейсу, наприклад, для створення комбо-боксів, прогрес-барів тощо.

3. `from tkcalendar import Calendar` — бібліотека tkcalendar додає віджет Calendar, який дозволяє користувачам вибирати дату з календаря у графічному інтерфейсі. Це зручно для додавання подій у календар та їх редагування.

4. `import datetime` — модуль datetime використовується для роботи з датами та часом у Python. Він надає різноманітні функції для отримання поточної дати, часу, а також для маніпуляцій з датами (додавання чи віднімання днів, порівняння дат тощо).

5. `import pyodbc` — бібліотека pyodbc дозволяє підключатися до баз даних через ODBC (Open Database Connectivity). Вона використовується для взаємодії з базою даних, зокрема для виконання SQL-запитів (наприклад, додавання, редагування чи видалення подій).

6. `import winsound` — бібліотека winsound дозволяє генерувати звукові сигнали або відтворювати звукові файли на системах Windows. Вона може бути використана для реалізації звукових сповіщень, наприклад, коли приходить нагадування про подію.

7. `from pystray import Icon, MenuItem, Menu` — бібліотека pystray дозволяє створювати іконки в системному треї (маленька іконка в правому нижньому куті екрану). Це зручно для додатків, які працюють у фоновому режимі і хочуть інформувати користувача про події без необхідності відкривати головне вікно програми.

8. `from PIL import Image, ImageDraw` — бібліотека PIL (Python Imaging Library) або її оновлена версія Pillow надає функції для роботи з зображеннями. У даному випадку, Image і ImageDraw використовуються для створення і редагування зображень (наприклад, для створення іконок для системного трея).

9. `import threading` — модуль `threading` дозволяє створювати потоки в Python, що дозволяє виконувати декілька задач одночасно. Це корисно, наприклад, для виконання фонових задач, таких як обробка сповіщень, без блокування основного інтерфейсу користувача.

4.2 Опис класу `ReminderApp` та підключення до бази даних

Клас `ReminderApp` є основною частиною програми для створення календаря з нагадуваннями. Цей клас відповідає за ініціалізацію графічного інтерфейсу користувача, обробку взаємодії з користувачем, а також за підключення до бази даних, де зберігаються всі події та нагадування.

```
class ReminderApp:
    def __init__(self, root):
        self.root = root
        self.root.protocol("WM_DELETE_WINDOW", self.hide_window)
        self.root.title("Календар з нагадуваннями")
        self.root.geometry("600x700")
        self.root.resizable(False, False)
        self.is_dark_mode = False

        try:
            self.conn = pyodbc.connect('DRIVER={ODBC Driver 17 for SQL Server};'
                                      'SERVER=DESKTOP-QQAOEK4;'
                                      'DATABASE=Calendar;'
                                      'Trusted_Connection=yes;'
                                      'Encrypt=yes;'
                                      'TrustServerCertificate=yes;')

            self.cursor = self.conn.cursor()
            print("Підключення з базою даних встановлено успішно.")
        except pyodbc.Error as e:
            messagebox.showerror("Помилка підключення", f"Не вдалось підключитись до бази даних: {e}")
            self.root.quit()

        self.create_table()

        self.root.configure(bg="#d4e8d4")
```

Рис.4.2.1 – Ініціалізація класу та підключення до бази даних

При створенні об'єкта класу `ReminderApp` ініціалізуються основні параметри програми, зокрема:

Вікно програми: встановлюється заголовок вікна, його розміри та обмеження на зміну розміру. Для вікна задається фоновий колір та параметри дизайну.

Обробка закриття вікна: за допомогою методу `protocol("WM_DELETE_WINDOW", self.hide_window)` організовано оброблення події закриття вікна, що дозволяє не просто закрити програму, а

виконати певні дії, наприклад, приховати вікно або зробити вихід з програми через системний трей.

Темна тема: клас має змінну `is_dark_mode`, яка визначає, чи використовується темна тема в інтерфейсі (ця функціональність може бути розширена в подальшому).

Для зберігання подій та нагадувань клас `ReminderApp` підключається до бази даних SQL Server через бібліотеку `pyodbc`. Підключення здійснюється за допомогою `Windows Authentication`, що дозволяє використовувати системні облікові записи для доступу до бази даних.

У випадку успішного підключення створюється курсор, який використовується для виконання SQL-запитів. Якщо підключення не вдається, виводиться повідомлення про помилку, і програма завершується.

За допомогою методу `create_table()` клас перевіряє наявність таблиці для зберігання подій в базі даних. Якщо таблиця не існує, вона створюється, що забезпечує коректну роботу з даними.

Програма має приємний та простий інтерфейс, який відповідає вимогам дизайну. Встановлюється світло-зелений фон, що робить інтерфейс більш привабливим для користувачів.

4.3 Опис створення графічного інтерфейсу

У цьому пункті реалізовано графічний інтерфейс користувача для додатка "Календар з нагадуваннями", що складається з кількох основних елементів управління. За допомогою бібліотеки `tkinter` було створено вікно програми з календарем, полем для введення подій, компонентами для вибору типу події та кнопками.

```

header = tk.Label(root, text="📅 Мій Календар", font=("Helvetica", 20, "bold"), bg="#4CAF50", fg="white")
header.pack(fill="x", pady=10)

self.cal_frame = tk.Frame(root, bg="#d4e8d4")
self.cal_frame.pack(pady=10)
self.cal = Calendar(
    self.cal_frame,
    selectmode="day",
    date_pattern="yyyy-mm-dd",
    locale="uk",
    background="lightblue",
    foreground="black",
    bordercolor="lightblue"
)
self.cal.pack()

self.today_date = datetime.date.today()
self.cal.calevent_create(self.today_date, "Сьогодні", "today")

self.event_frame = tk.Frame(root, bg="#d4e8d4")
self.event_frame.pack(pady=10)

self.event_label = tk.Label(self.event_frame, text="Опис події:", font=("Helvetica", 12), bg="#d4e8d4")
self.event_label.grid(row=0, column=0, padx=5, pady=5)

self.event_entry = ttk.Entry(self.event_frame, width=30, font=("Helvetica", 12))
self.event_entry.grid(row=0, column=1, padx=5)
self.event_entry.insert(0, "Введіть опис події")

self.event_type_label = tk.Label(self.event_frame, text="Тип події:", font=("Helvetica", 12), bg="#d4e8d4")
self.event_type_label.grid(row=1, column=0, padx=5, pady=5)

self.event_type = ttk.Combobox(self.event_frame, values=["Звичайна", "Важлива", "Термінова"], font=("Helvetica", 12))
self.event_type.grid(row=1, column=1, padx=5)
self.event_type.set("Звичайна")

```

Рис.4.3.1 – Графічний інтерфейс введення подій та вибору подій

Створено заголовок для вікна програми за допомогою компонента Label. Заголовок містить текст "📅 Мій Календар", шрифт "Helvetica" розміру 20 та стиль "bold". Колір фону встановлений на зелений (#4CAF50), а колір тексту — білий. Заголовок займає всю ширину вікна й має відступи зверху (pady=10).

Для відображення календаря створено рамку Frame, в яку додано віджет календаря Calendar з бібліотеки tkcalendar. Календар має такі параметри:

- Вибір дати здійснюється по дням (selectmode="day").
- Формат дати — "yyyy-mm-dd".
- Локалізація українська (locale="uk").
- Колір фону та тексту — блакитний для фону і чорний для тексту.

Визначається поточна дата за допомогою модуля datetime. Для цієї дати встановлюється подія з позначкою "Сьогодні", яка відображається у календарі за допомогою методу `calevent_create`.

Під календарем розташовано поле для введення опису подій. Створено ще одну рамку `Frame`, у якій розміщено текстове поле для введення (`Entry`) та лейбл для позначення цього поля як "Опис події". Поле для введення має початкове значення "Введіть опис події".

Для вибору типу події створено комбінований список (`Combobox`) з трьома варіантами: "Звичайна", "Важлива", "Термінова". За допомогою цього компонента користувач може вказати важливість події. Значення за замовчуванням — "Звичайна".

```
self.button_frame = tk.Frame(root, bg="#d4e8d4")
self.button_frame.pack(pady=10)

self.add_button = ttk.Button(self.button_frame, text="⊕ Додати подію", command=self.add_event)
self.add_button.grid(row=0, column=0, padx=10, pady=5)

self.show_button = ttk.Button(self.button_frame, text="📅 Показати події", command=self.show_events)
self.show_button.grid(row=0, column=1, padx=10, pady=5)

self.remove_button = ttk.Button(self.button_frame, text="✗ Видалити подію", command=self.remove_event)
self.remove_button.grid(row=0, column=2, padx=10, pady=5)

self.edit_button = ttk.Button(self.button_frame, text="✎ Редагувати подію", command=self.edit_event)
self.edit_button.grid(row=1, column=0, padx=10, pady=5)

self.complete_button = ttk.Button(self.button_frame, text="✅ Завершити подію", command=self.complete_event)
self.complete_button.grid(row=1, column=1, padx=10, pady=5)

self.save_button = ttk.Button(self.button_frame, text="💾 Зберегти зміни", command=self.save_changes)
self.save_button.grid(row=1, column=2, padx=10, pady=5)

self.list_frame = tk.Frame(root, bg="#e6ffe6", bd=2, relief="groove")
self.list_frame.pack(fill="both", expand=True, padx=10, pady=10)

self.list_label = tk.Label(self.list_frame, text="Список подій:", bg="#e6ffe6", font=("Helvetica", 14))
self.list_label.pack(anchor="nw", padx=5, pady=5)

self.event_list = tk.Listbox(
    self.list_frame,
    font=("Helvetica", 12),
    bg="#f9fff9",
    selectbackground="#b3ffb3",
    selectforeground="black",
    height=10
)
self.event_list.pack(fill="both", expand=True, padx=5, pady=5)

self.root.after(1000, self.daily_check)
```

Рис.4.3.2 – Графічний інтерфейс кнопок, списку подій та звукового сповіщення

У цьому розділі створюються інтерактивні елементи для роботи з подіями в календарі. Спочатку створюється рамка для кнопок, що дозволяють користувачеві взаємодіяти з подіями. Кнопки мають зрозумілі іконки та підписи для основних дій:

- Кнопка "Додати подію" викликає метод `add_event`, що дозволяє додавати нову подію до календаря.
- Кнопка "Показати події" відкриває метод `show_events`, який дає можливість переглядати список усіх подій.
- Кнопка "Видалити подію" дозволяє користувачеві вибрати подію зі списку та видалити її за допомогою методу `remove_event`.
- Кнопка "Редагувати подію" викликає метод `edit_event`, що дозволяє редагувати вже існуючі події.
- Кнопка "Завершити подію" дозволяє позначити подію як завершену через метод `complete_event`.
- Кнопка "Зберегти зміни" зберігає зміни, зроблені в події, після редагування чи додавання, за допомогою методу `save_changes`.

Усі ці кнопки організовані в рядки та стовпці за допомогою методів компоновки, що дозволяє чітко розподілити їх у графічному інтерфейсі.

Далі додається список подій у вигляді віджета `Listbox`. Це дозволяє користувачеві бачити всі події в календарі та вибирати їх для подальших операцій, таких як редагування або видалення. Для зручності використовується кольорове оформлення: елементи списку мають світло-зелений фон, а вибрані елементи підсвічуються іншим кольором для покращення візуального сприйняття.

Також реалізовано звукове сповіщення за допомогою функції, яка запускається кожну секунду. Ця функція перевіряє наявність подій, які потребують нагадування або інших дій, що допомагає своєчасно інформувати користувача.

Ці елементи забезпечують повну інтерактивність додатку і дозволяють користувачеві зручно працювати з подіями в календарі.

4.4 Створення таблиці в базі даних для зберігання подій

У цьому фрагменті реалізовано метод `create_table`, який перевіряє наявність таблиці в базі даних і, якщо її немає, створює нову таблицю для зберігання подій. Це важливий крок для забезпечення коректної роботи програми з базою даних.

```
def create_table(self):
    self.cursor.execute("""
    IF NOT EXISTS (SELECT * FROM sysobjects WHERE name='Events' AND xtype='U')
    CREATE TABLE Events (
        id INT PRIMARY KEY IDENTITY(1,1),
        event_date DATE,
        event_description NVARCHAR(255),
        event_type NVARCHAR(50),
        completed BIT
    )
    """)
    self.conn.commit()
```

Рис.4.4.1 – Створення таблиці в базі даних

Спочатку за допомогою SQL-запиту перевіряється, чи існує таблиця з назвою "Events" у базі даних. Для цього використовується системна таблиця `sysobjects`, в якій зберігаються метадані всіх об'єктів бази даних, таких як таблиці, індекси та інші об'єкти.

Якщо таблиця "Events" не існує, то створюється нова таблиця з такими полями:

- `id`: ідентифікатор події, що є первинним ключем і автоматично збільшується для кожного нового запису.
- `event_date`: дата події.
- `event_description`: опис події (рядок довжиною до 255 символів).
- `event_type`: тип події, що визначає її категорію (наприклад, "звичайна", "важлива", "термінова").
- `completed`: статус завершення події (тип BIT з двома значеннями: 0 для незавершених і 1 для завершених).

Після виконання SQL-запиту зміни фіксуються за допомогою методу `commit`, що дозволяє зберегти створену таблицю в базі даних.

Цей метод гарантує, що програма буде мати необхідну структуру для зберігання подій, навіть якщо таблиця ще не була створена в базі даних, і забезпечує подальшу роботу з подіями в календарі.

4.5 Додавання події до бази даних

Метод `add_event` відповідає за додавання нової події до бази даних після того, як користувач вибере дату та введе опис події.

```
def add_event(self):
    date = self.cal.get_date()
    event = self.event_entry.get()
    event_type = self.event_type.get()

    if date and event and event.strip() != "":
        self.cursor.execute("""
            INSERT INTO Events (event_date, event_description, event_type, completed)
            VALUES (?, ?, ?, ?)
        """, (date, event, event_type, False))
        self.conn.commit()

        messagebox.showinfo("Успіх", f"Подія '{event}' додана на {date}!")
        self.event_entry.delete(0, tk.END)
        self.update_event_list()
    else:
        messagebox.showerror("Помилка", "Будь ласка, виберіть дату та введіть подію.")
```

Рис.4.5.1 – Додавання події до бази даних

Отримується вибрана користувачем дата (`self.cal.get_date()`), введений опис події (`self.event_entry.get()`) і вибраний тип події (`self.event_type.get()`).

Перевіряється, чи заповнені всі необхідні поля, і чи не є введений опис порожнім або стандартним текстом.

Якщо дані коректні, виконується SQL-запит `INSERT INTO Events`, який додає новий запис у таблицю `Events` з переданими значеннями:

- `event_date` – вибрана дата події.
- `event_description` – опис події.
- `event_type` – тип події.
- `completed` – статус події, який за замовчуванням встановлюється як `False` (незавершена подія).

Після успішного додавання викликається `commit()`, щоб зберегти зміни в базі даних.

Відображається інформаційне повідомлення про успішне додавання події.

Очищується поле введення події для зручності користувача.

Викликається метод `update_event_list()`, який оновлює список подій у графічному інтерфейсі.

Якщо якісь дані не введені, з'являється повідомлення про помилку.

Цей метод забезпечує інтерактивне додавання подій у календар та їхнє збереження в базі даних.

4.6 Відображення подій

Метод `show_events` використовується для виведення подій, які були заплановані на вибрану користувачем дату.

```
def show_events(self):
    date = self.cal.get_date()
    self.cursor.execute("SELECT event_description, event_type, completed FROM Events WHERE event_date = ?", (date,))
    events = self.cursor.fetchall()
    if events:
        event_details = "\n".join([f"{event[0]} ({event[1]}) {'✓' if event[2] else 'X'}" for event in events])
        messagebox.showinfo("Події", f"Події на {date}:\n{event_details}")
    else:
        messagebox.showinfo("Події", f"На {date} немає подій.")
```

Рис.4.6.1 – Відображення подій

Отримується дата, вибрана в календарі (`self.cal.get_date()`).

Виконується SQL-запит `SELECT event_description, event_type, completed FROM Events WHERE event_date = ?`, який отримує з бази даних опис подій, їхній тип та статус виконання для зазначеної дати.

Якщо знайдені події, формується список, у якому кожен запис містить:

- Опис події (`event[0]`).
- Тип події (`event[1]`).
- Статус виконання (✓, якщо подія завершена, і ✗, якщо ні).

Сформований список подій виводиться у вигляді інформаційного вікна `messagebox.showinfo`.

Якщо жодної події на цю дату немає, виводиться повідомлення про їхню відсутність.

Цей метод забезпечує зручний перегляд запланованих подій, що зберігаються в базі даних, без необхідності переглядати їх вручну.

4.7 Видалення подій

Метод `remove_event` дозволяє користувачеві видаляти події зі списку.

```
def remove_event(self):
    selected_event = self.event_list.curselection()
    if selected_event:
        event_info = self.event_list.get(selected_event)
        date = event_info.split(":")[0].strip()
        event_name = event_info.split(":")[1].split("(")[0].strip()

        self.cursor.execute("DELETE FROM Events WHERE event_date = ? AND event_description = ?", (date, event_name))
        self.conn.commit()

        self.update_event_list()
        messagebox.showinfo("Успіх", f"Подія '{event_name}' була видалена.")
    else:
        messagebox.showerror("Помилка", "Будь ласка, виберіть подію для видалення.")
```

Рис.4.7.1 – Видалення подій

Отримується вибрана подія зі списку `self.event_list.curselection()`.

Якщо подія обрана, зчитується її інформація, а саме:

Дата події (відокремлюється від рядка за допомогою `split(":")[0].strip()`).

Назва події (`split(":")[1].split("(")[0].strip()`).

Виконується SQL-запит `DELETE FROM Events WHERE event_date = ? AND event_description = ?`, який видаляє подію з бази даних.

Оновлюється список подій (`self.update_event_list()`).

Виводиться інформаційне повідомлення про успішне видалення події.

Якщо жодну подію не вибрано, користувач отримує повідомлення про помилку.

Цей метод дозволяє легко видаляти непотрібні або застарілі події, зберігаючи список актуальним.

4.8 Оновлення списку подій

Метод `update_event_list` відповідає за оновлення відображення списку подій у графічному інтерфейсі.

```
def update_event_list(self):
    self.event_list.delete(0, tk.END)
    self.cursor.execute("SELECT event_date, event_description, event_type, completed FROM Events ORDER BY event_date")
    events = self.cursor.fetchall()
    for event in events:
        date, description, event_type, completed = event
        color = "lightgreen" if event_type == "Звичайна" else "yellow" if event_type == "Важлива" else "red"
        self.event_list.insert(tk.END, f"{date}: {description} ({event_type}) {'✓' if completed else '✗'}")
        self.event_list.itemconfig(tk.END, {'bg': color})
```

Рис.4.8.1 – Оновлення списку подій

Очищається список подій у `self.event_list.delete(0, tk.END)`, щоб уникнути дублювання даних.

Виконується SQL-запит `SELECT event_date, event_description, event_type, completed FROM Events ORDER BY event_date`, який отримує всі події з бази даних, впорядковані за датою.

Отримані події додаються до списку:

- Для кожної події отримуються її дата, опис, тип і статус виконання.
- Визначається колір фону рядка на основі типу події:
 - Звичайна подія – зелений (lightgreen).
 - Важлива подія – жовтий (yellow).
 - Термінова подія – червоний (red).
- Подія додається до списку у форматі дата: опис (тип) ✓ / ✗ (галочка, якщо виконано).
- Використовується `self.event_list.itemconfig(tk.END, {'bg': color})`, щоб встановити відповідний фон для кожного запису.

Цей метод забезпечує актуальне відображення подій після їх додавання, видалення або змін.

4.9 Редагування події

Метод `edit_event` дозволяє користувачеві вибрати подію зі списку та змінити її опис або тип.

```
def edit_event(self):
    selected_event = self.event_list.curselection()
    if selected_event:
        event_info = self.event_list.get(selected_event)
        description = event_info.split(":")[1].split("(")[0].strip()
        event_type = event_info.split("(")[1].split(")")[0].strip()

        self.event_entry.delete(0, tk.END)
        self.event_entry.insert(0, description)
        self.event_type.set(event_type)

        self.selected_event = event_info
    else:
        messagebox.showerror("Помилка", "Будь ласка, виберіть подію для редагування.")
```

Рис.4.9.1 – Редагування події

Використовується `self.event_list.curselection()`, щоб отримати індекс вибраного елемента у списку подій.

Якщо користувач вибрав подію, вона отримується через `self.event_list.get(selected_event)`.

Зі строкового представлення події виділяється:

- Опис події (`description`), який знаходиться після дати та до дужок.
- Тип події (`event_type`), який міститься у дужках.

Поле введення очищується (`self.event_entry.delete(0, tk.END)`) і заповнюється вибраним описом.

У випадяючому списку вибирається відповідний тип події (`self.event_type.set(event_type)`).

Збережена у змінну `self.selected_event` інформація про вибрану подію допоможе при подальшому оновленні її у базі даних.

Якщо жодна подія не була вибрана, виводиться повідомлення про помилку.

Цей метод дозволяє користувачеві швидко внести зміни до події перед її оновленням у базі.

4.10 Збереження змін у події

Метод `save_changes` використовується для збереження оновлених даних вибраної події.


```

def save_changes(self):
    new_description = self.event_entry.get()
    new_event_type = self.event_type.get()

    if hasattr(self, 'selected_event') and self.selected_event:
        old_description = self.selected_event.split(":")[1].split(" ")[0].strip()
        old_event_type = self.selected_event.split(" ")[1].split(" ")[0].strip()
        date = self.selected_event.split(":")[0].strip()

        if new_description != old_description or new_event_type != old_event_type:
            self.cursor.execute("UPDATE Events SET event_description = ?, event_type = ? WHERE event_date = ? AND event_description = ?",
                                (new_description, new_event_type, date, old_description))
            self.conn.commit()

            self.update_event_list()
            messagebox.showinfo("Успіх", f"Подія успішно оновлена!")
        else:
            messagebox.showinfo("Інформація", "Опис події та тип не змінились.")
    else:
        messagebox.showerror("Помилка", "Будь ласка, спочатку виберіть подію для редагування.")

```

Рис.4.10.1 – Збереження змін у події

Отримуються нові значення для опису (new_description) та типу події (new_event_type) з відповідних полів введення.

Перевіряється, чи існує змінна self.selected_event, яка містить дані про подію, що редагується.

Якщо подія була вибрана, розбивається її строкове представлення, щоб отримати:

- Початковий опис події (old_description).
- Початковий тип події (old_event_type).
- Дату події (date).

Якщо новий опис або тип події відрізняється від початкових значень, виконується SQL-запит для оновлення відповідного запису в таблиці Events.

Викликається self.update_event_list(), щоб оновити відображення подій у списку.

Якщо зміни були успішно внесені, відображається повідомлення про оновлення.

Якщо користувач не змінив опис або тип події, виводиться інформаційне повідомлення.

Якщо жодна подія не була вибрана для редагування, виводиться повідомлення про помилку.

Цей метод завершує процес редагування події, дозволяючи оновити її дані у базі та в списку подій.

4.11 Завершення події

Метод `complete_event` використовується для позначення вибраної події як завершенної.

```
def complete_event(self):
    selected_event = self.event_list.curselection()
    if selected_event:
        event_info = self.event_list.get(selected_event)
        date = event_info.split(":")[0].strip()
        event_name = event_info.split(":")[1].split("(")[0].strip()

        self.cursor.execute("UPDATE Events SET completed = 1 WHERE event_date = ? AND event_description = ?", (date, event_name))
        self.conn.commit()

        self.update_event_list()
        messagebox.showinfo("Успіх", f"Подія '{event_name}' була позначена як завершена.")
    else:
        messagebox.showerror("Помилка", "Будь ласка, виберіть подію для завершення.")
```

Рис.4.11.1 – Завершення події

Отримується індекс вибраного запису у списку `self.event_list` за допомогою `curselection()`.

Якщо подія була вибрана, зчитується її строкове представлення (`event_info`).

Розбивається строка, щоб отримати:

- Дату події (`date`).
- Опис події (`event_name`).

Виконується SQL-запит для оновлення поля `completed`, встановлюючи його значення в 1, що означає завершення події.

Викликається метод `self.update_event_list()`, щоб оновити список подій та відобразити зміну.

Виводиться повідомлення про успішне завершення події.

Якщо жодна подія не була вибрана, користувач отримує повідомлення про помилку.

Цей метод дозволяє відзначати виконані події, покращуючи управління завданнями в додатку.

4.12 Щоденна перевірка подій

Метод `daily_check` призначений для автоматичного нагадування про незавершені події, які заплановані на поточний день.

```
def daily_check(self):
    today = datetime.date.today()
    self.cursor.execute("SELECT event_description FROM Events WHERE event_date = ? AND completed = 0", (today,))
    events = self.cursor.fetchall()
    for event in events:
        messagebox.showinfo("Нагадування", f"Сьогодні: {event[0]}")

    self.root.after(86400000, self.daily_check)
```

Рис.4.12.1 – Щоденна перевірка подій

Отримується поточна дата за допомогою `datetime.date.today()`.

Виконується SQL-запит, який вибирає всі незавершені події (`completed = 0`) із таблиці `Events`, що призначені на сьогоднішній день.

Якщо знайдені події, вони виводяться у вигляді повідомлень `messagebox.showinfo()`, нагадуючи користувачеві про заплановані завдання.

Метод `self.root.after(86400000, self.daily_check)` встановлює автоматичний запуск цього методу через 24 години (86400000 мілісекунд), забезпечуючи щоденну перевірку подій.

Ця функція дозволяє користувачеві не забувати про важливі події, надаючи щоденні нагадування.

4.13 Приховування вікна та робота з іконкою в треї

Даний фрагмент коду дозволяє програмі працювати у фоновому режимі, мінімізуючи головне вікно та створюючи іконку в системному треї.

Приховування головного вікна методом `hide_window(self)` ховає головне вікно програми за допомогою `self.root.withdraw()`, роблячи його невидимим. Після цього створюється іконка за допомогою `self.create_icon()`, яка запускається у фоновому режимі (`icon.run()`).

```

def hide_window(self):
    self.root.withdraw()

    icon = self.create_icon()
    icon.run()

def create_icon(self):
    try:
        icon_image = Image.open("C:/Users/Ruslan Ostrovsky/Desktop/kyrsovaya/иконки/icon1.png")
        icon_image = icon_image.resize((64, 64))
    except Exception as e:
        messagebox.showerror("Помилка", f"Не вдалось завантажити іконку: {e}")
        return None

    icon = Icon("ReminderApp", icon_image, menu=Menu(
        MenuItem("Показати", self.show_window),
        MenuItem("Вийти", self.quit_app)
    ))

    icon.tooltip = "Нагадування"
    return icon

def show_window(self, icon):
    self.root.deiconify()
    icon.stop()

def quit_app(self, icon):
    self.conn.close()
    icon.stop()
    self.root.quit()

```

Рис.4.13.1 – Приховування вікна та робота з іконкою в треї

Створення іконки методом create_icon(self):

Завантажує зображення іконки з вказаного шляху "C:/Users/Ruslan Ostrovsky/Desktop/kyrsovaya/иконки/icon1.png".

Змінює розмір зображення до 64x64 пікселів.

Якщо завантаження не вдається, виводить помилку через messagebox.showerror().

Створює іконку в системному треї за допомогою pystray.Icon.

Додає до іконки контекстне меню з двома пунктами:

- "Показати" – викликає метод show_window(), який повертає вікно.
- "Вийти" – викликає quit_app(), що завершує роботу програми.

Встановлює підказку для іконки "Нагадування".

Показ головного вікна методом show_window(self, icon) робить головне вікно знову видимим (self.root.deiconify()) і зупиняє іконку (icon.stop()).

Завершення роботи програми методом quit_app(self, icon):

Закриває підключення до бази даних (self.conn.close()).

Зупиняє іконку в треї (icon.stop()).

Завершує роботу програми (self.root.quit()).

4.14 Запуск програми

Останній блок коду відповідає за запуск програми.

```
if __name__ == "__main__":  
    root = tk.Tk()  
    app = ReminderApp(root)  
    root.mainloop()
```

Рис.4.14.1 – Запуск програми

Створює головне вікно Tkinter (root = tk.Tk()).

Створює екземпляр класу ReminderApp, передаючи йому root.

Запускає головний цикл програми (root.mainloop()), який обробляє події та підтримує роботу інтерфейсу.

Цей код дозволяє програмі працювати у фоновому режимі, що зручно для нагадувань без зайвого завантаження екрану.

5. КЕРІВНИЦТВО КОРИСТУВАЧА

Даний застосунок розроблений для планування подій і нагадувань. Він дозволяє користувачам додавати, редагувати, видаляти та переглядати події, а також відзначати їх як завершені. Програма має зручний інтерфейс і підтримує роботу з базою даних для збереження подій.

Для того щоб запустити застосунок, потрібно запустити файл з кодом kursova.ua.py. Після запуску відкриється головне вікно програми, де можна працювати з подіями.

Програма має вбудовану систему автоматичних нагадувань.

1. Щоранку програма перевіряє базу даних і шукає незавершені події.
2. Якщо на сьогодні заплановані незавершені події, з'явиться вікно з нагадуванням.
3. У повідомленні буде вказано список подій, які потрібно виконати.

Ця функція працює у фоновому режимі та активується автоматично.

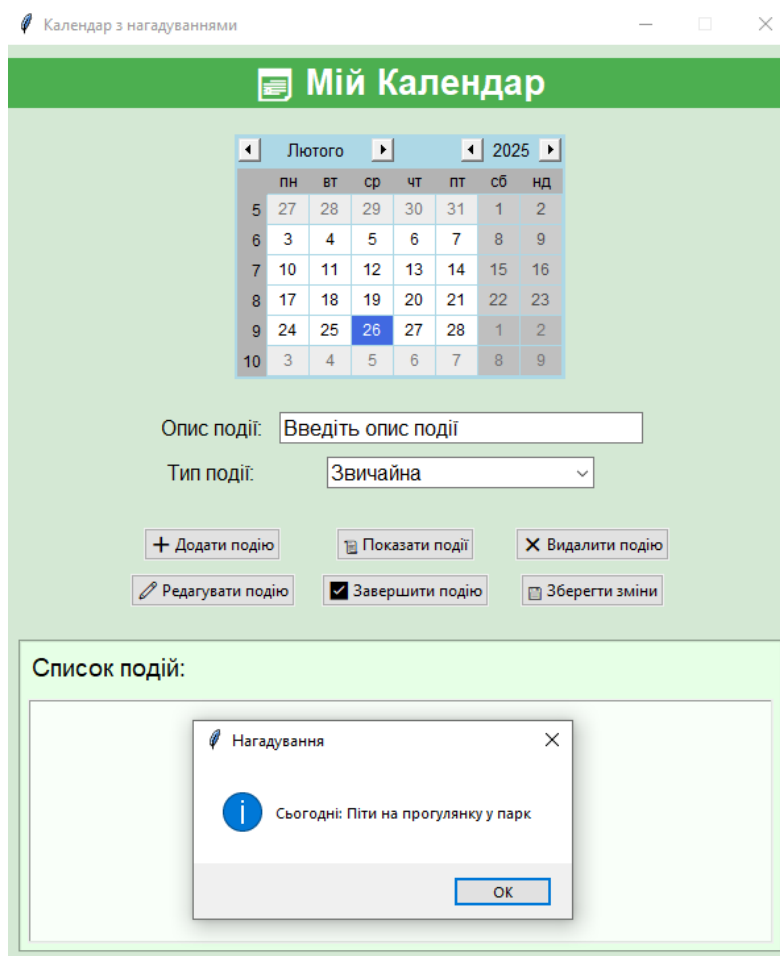


Рис.5.1 – Інтерфейс відкритого застосунку та нагадування події

Щоб створити нову подію, потрібно виконати наступні кроки:

1. Потрібно обрати дату у календарі – ця дата буде зафіксована як день події.
2. Потрібно ввести опис події у відповідне текстове поле (наприклад, "Зустріч з друзями").
3. Потрібно вибрати тип події зі списку:
 - Звичайна – стандартна подія.
 - Важлива – подія, яка має вищий пріоритет.
 - Критична – надзвичайно важлива подія, що потребує негайної уваги.
4. Потрібно натиснути кнопку "+ Додати подію" – після цього подія збережеться у базі даних.

Якщо все пройшло успішно, з'явиться повідомлення про додавання події.

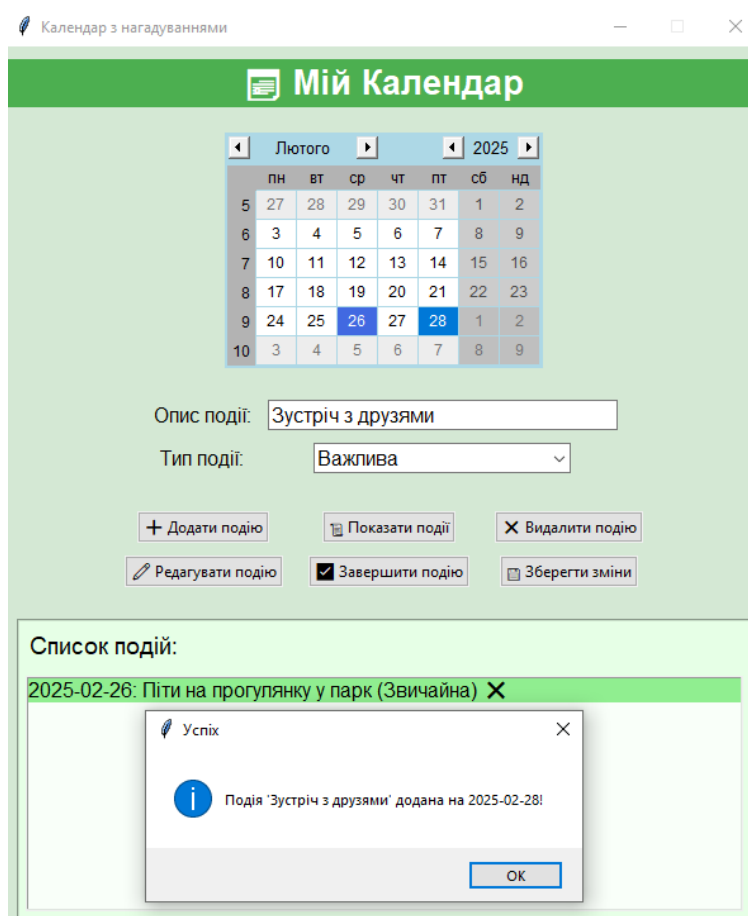


Рис.5.2 – Повідомлення про успішне додавання нової події

Примітка: якщо не вказати дату або опис події, програма повідомить про помилку.

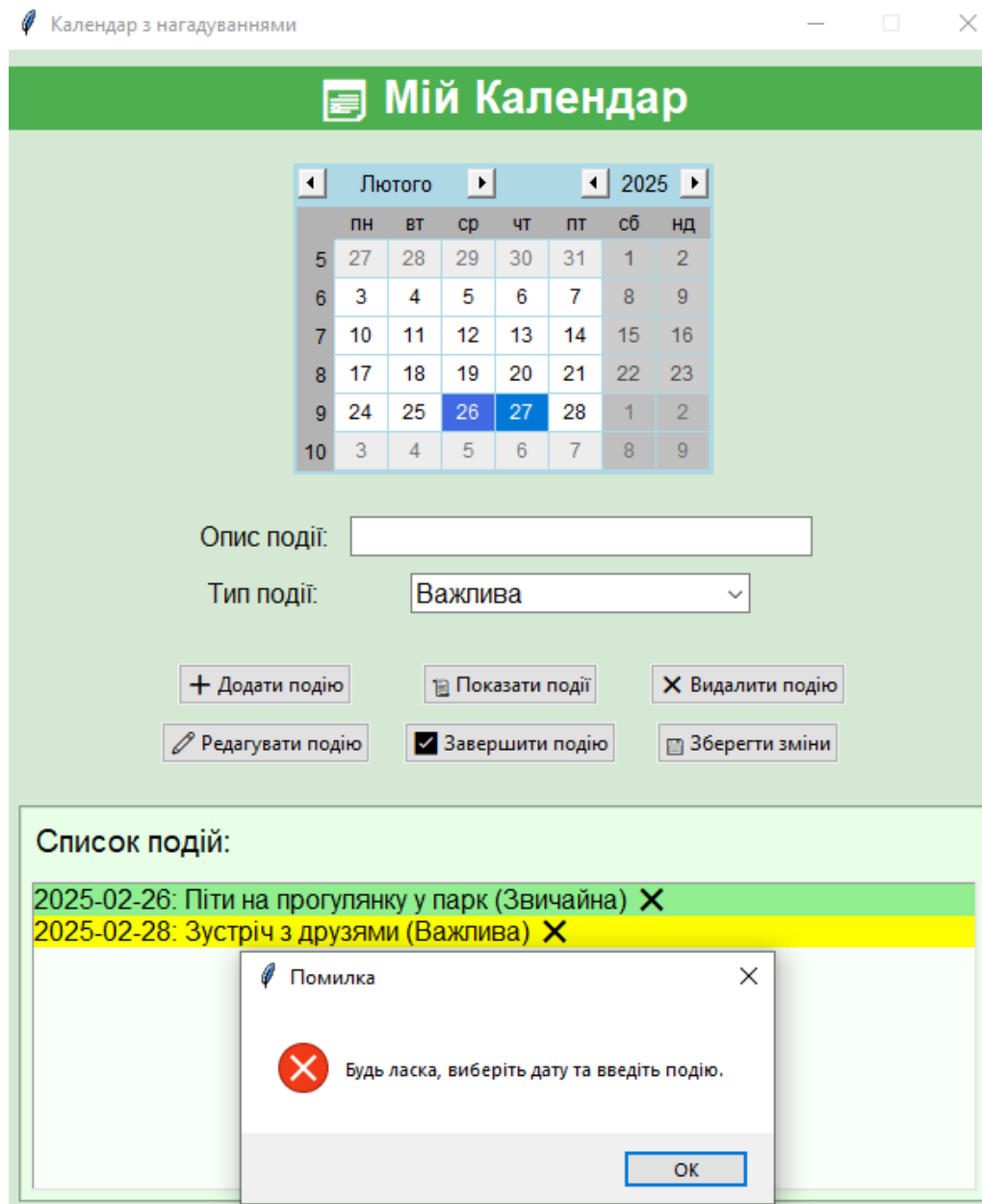



Рис.5.3 – Повідомлення про те, що потрібно обрати дату та ввести подію

Щоб переглянути список запланованих подій на конкретну дату:

1. Потрібно обрати дату у календарі.
2. Потрібно натиснути кнопку "  Показати події".
3. У спливаючому вікні з'явиться список подій, запланованих на вибраний день.

4. Для кожної події відображається:

- Опис події.
- Тип події.
- Статус виконання (✓ виконано або ✗ не виконано).

Якщо подій на вибрану дату немає, програма повідомить про це.

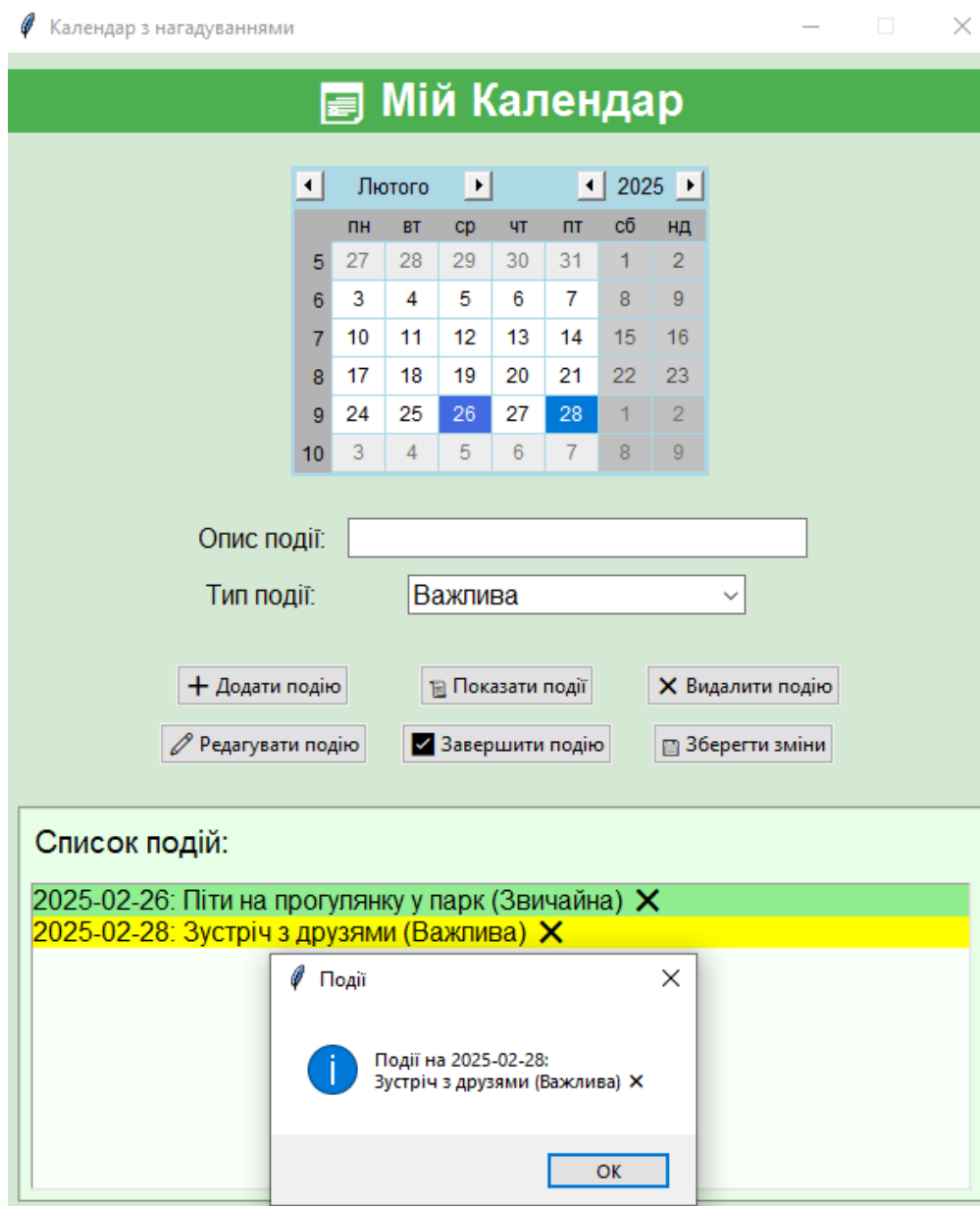


Рис.5.4 – Повідомлення про список подій запланований на конкретну дату

Примітка: якщо подій на вибрану дату немає, програма повідомить про це.

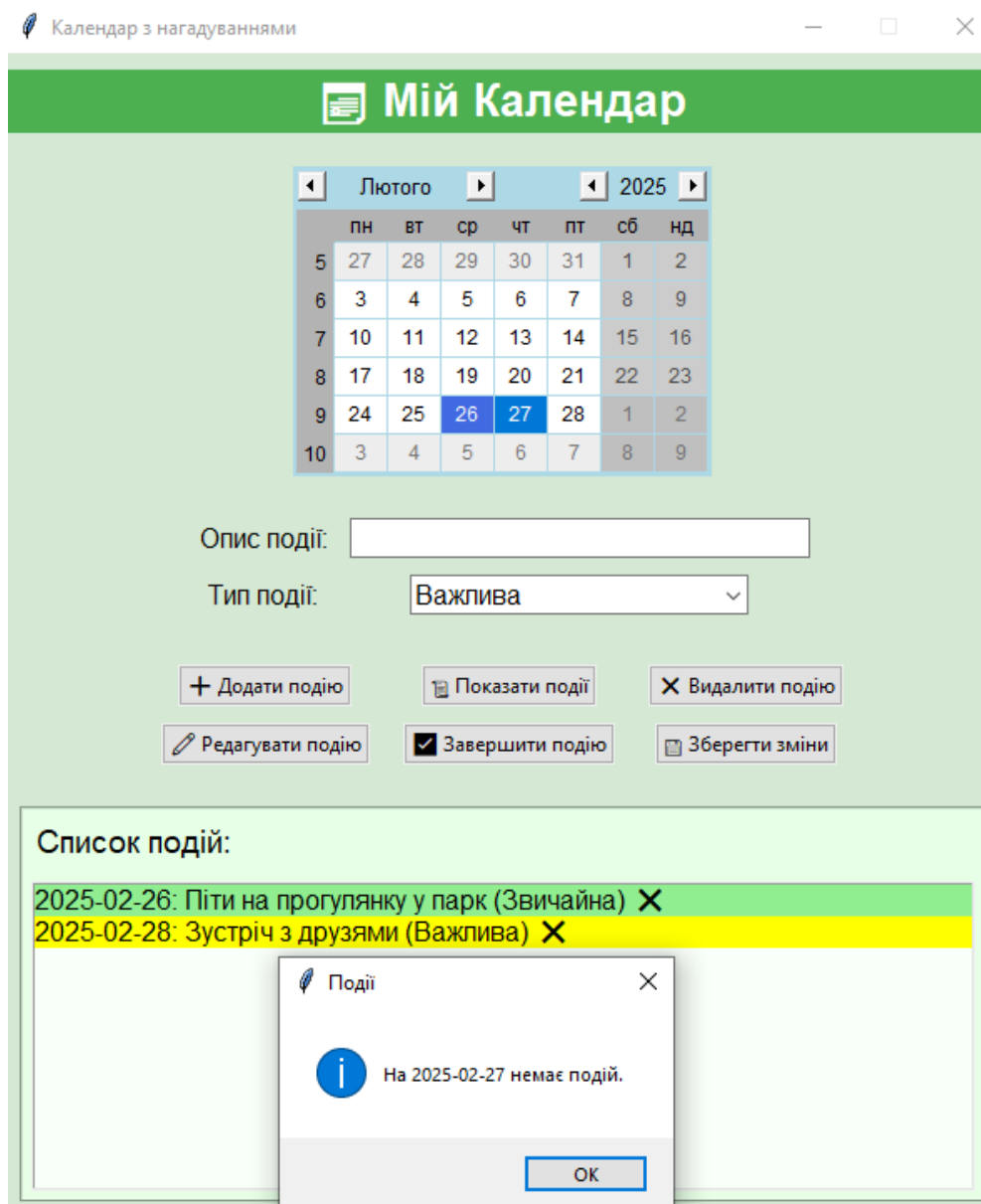




Рис.5.5 – Повідомлення про те, що немає подій на обраний день
Якщо потрібно змінити інформацію про подію:

1. Потрібно вибрати подію у списку.
2. Потрібно натиснути кнопку "  Редагувати подію".
3. Опис події та її тип з'являться у відповідних полях введення.
4. Потрібно внести необхідні зміни.
5. Потрібно натиснути кнопку "  Зберегти зміни", щоб оновити дані.
6. Програма повідомить про успішне редагування.

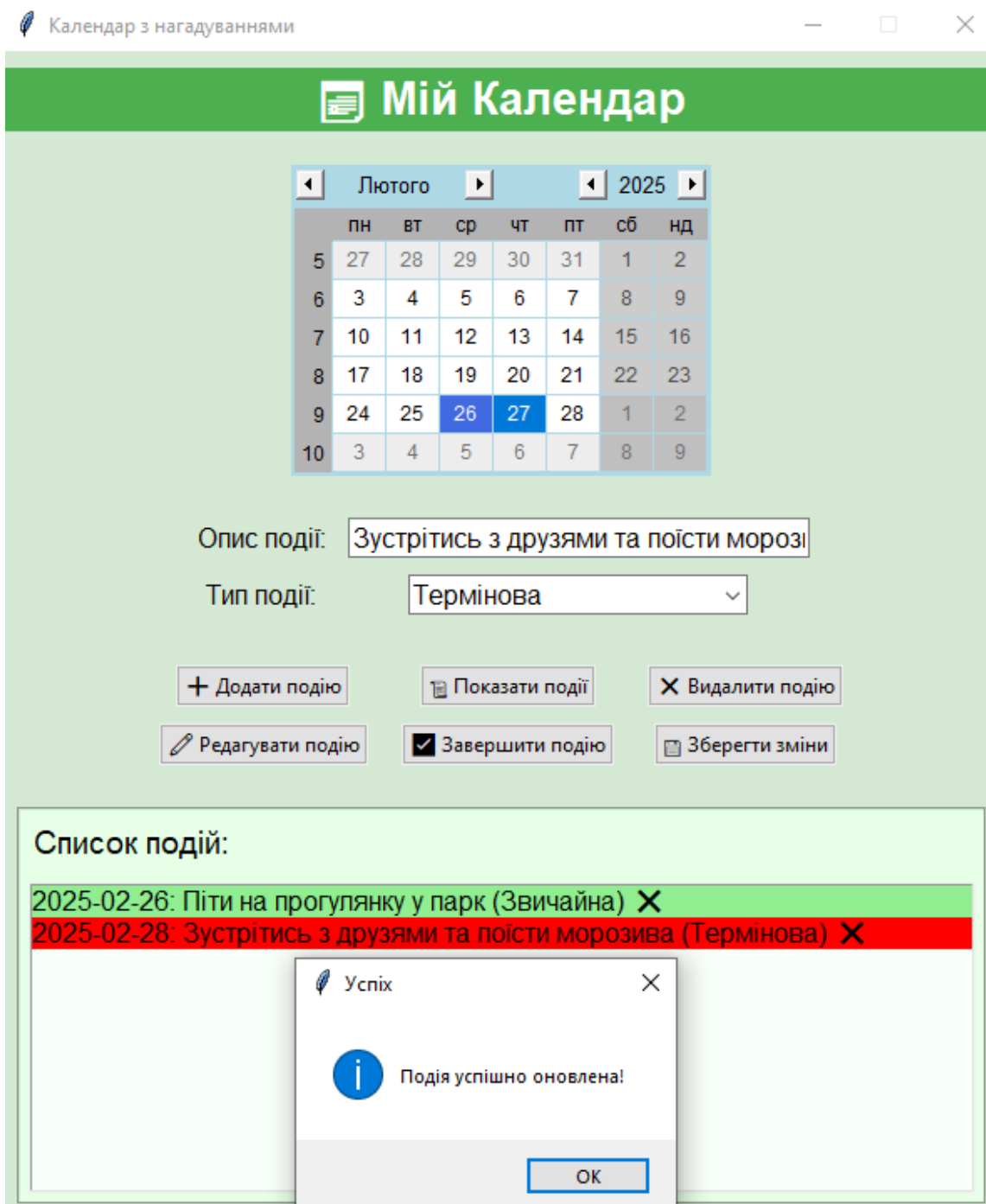


Рис.5.6 – Повідомлення про успішне оновлення події

Примітка: якщо не вибрати подію для редагування, програма повідомить про помилку.

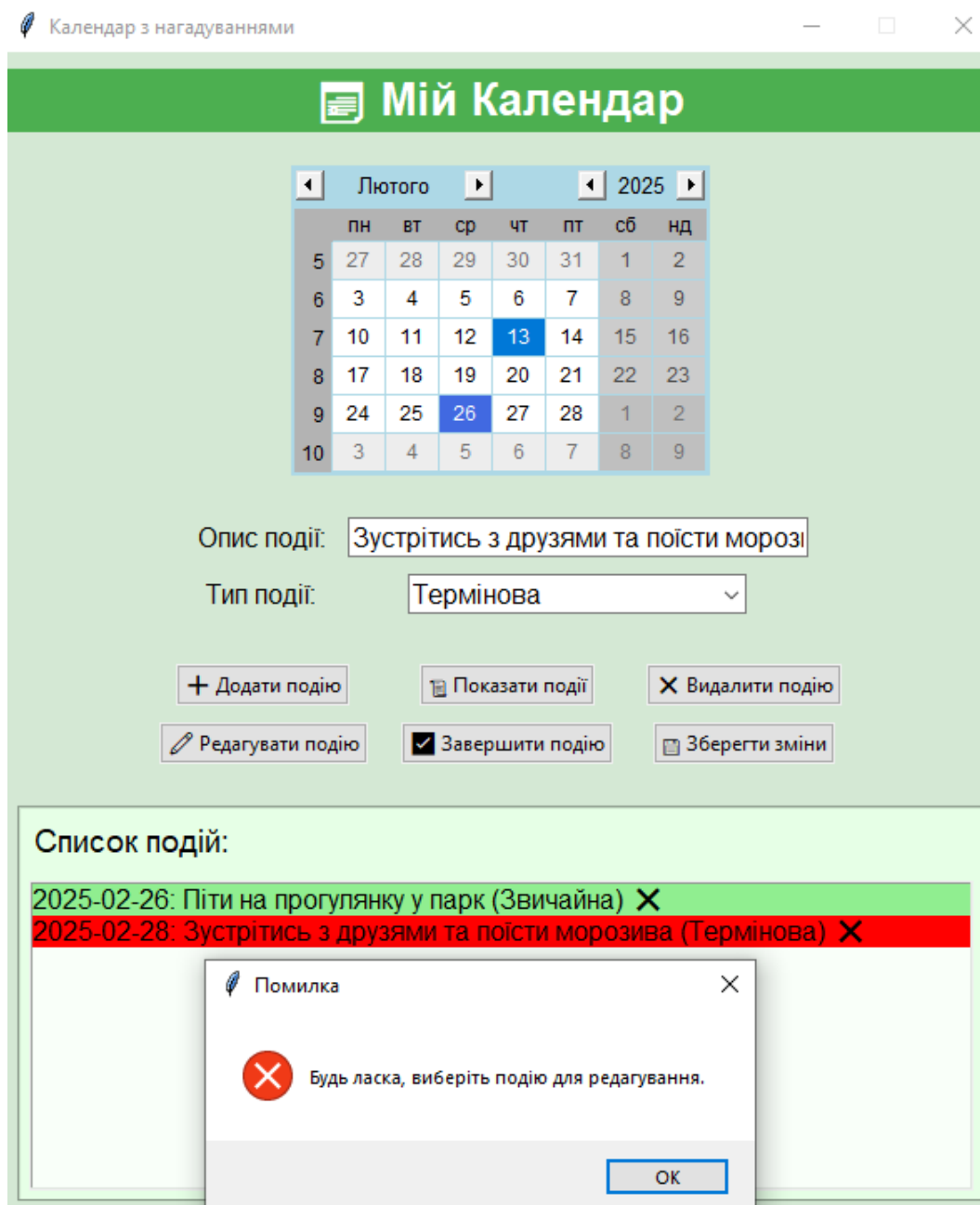


Рис.5.7 – Повідомлення про те, що потрібно обрати подію для редагування

Якщо подія більше не потрібна:

1. Потрібно вибрати подію у списку.
2. Потрібно натиснути кнопку "✕ Видалити подію".
3. Подія буде видалена з бази даних.
4. Програма оновить список подій і повідомить про успішне видалення.

Мій Календар

◀ Лютого ▶

◀ 2025 ▶

	пн	вт	ср	чт	пт	сб	нд
5	27	28	29	30	31	1	2
6	3	4	5	6	7	8	9
7	10	11	12	13	14	15	16
8	17	18	19	20	21	22	23
9	24	25	26	27	28	1	2
10	3	4	5	6	7	8	9

Опис події:

Зустрітись з друзями та поїсти морозів

Тип події:

Термінова

+ Додати подію

Показати події

✕ Видалити подію

Редагувати подію

☒ Завершити подію

Зберегти зміни

Список подій:

2025-02-26: Піти на прогулянку у парк (Звичайна) ✕

Успіх

і

Подія 'Зустрітись з друзями та поїсти морозива' була видалена.

ОК

Рис.5.8 – Повідомлення про успішне видалення події

Примітка: якщо подію не вибрати, тоді з'явиться повідомлення про помилку.

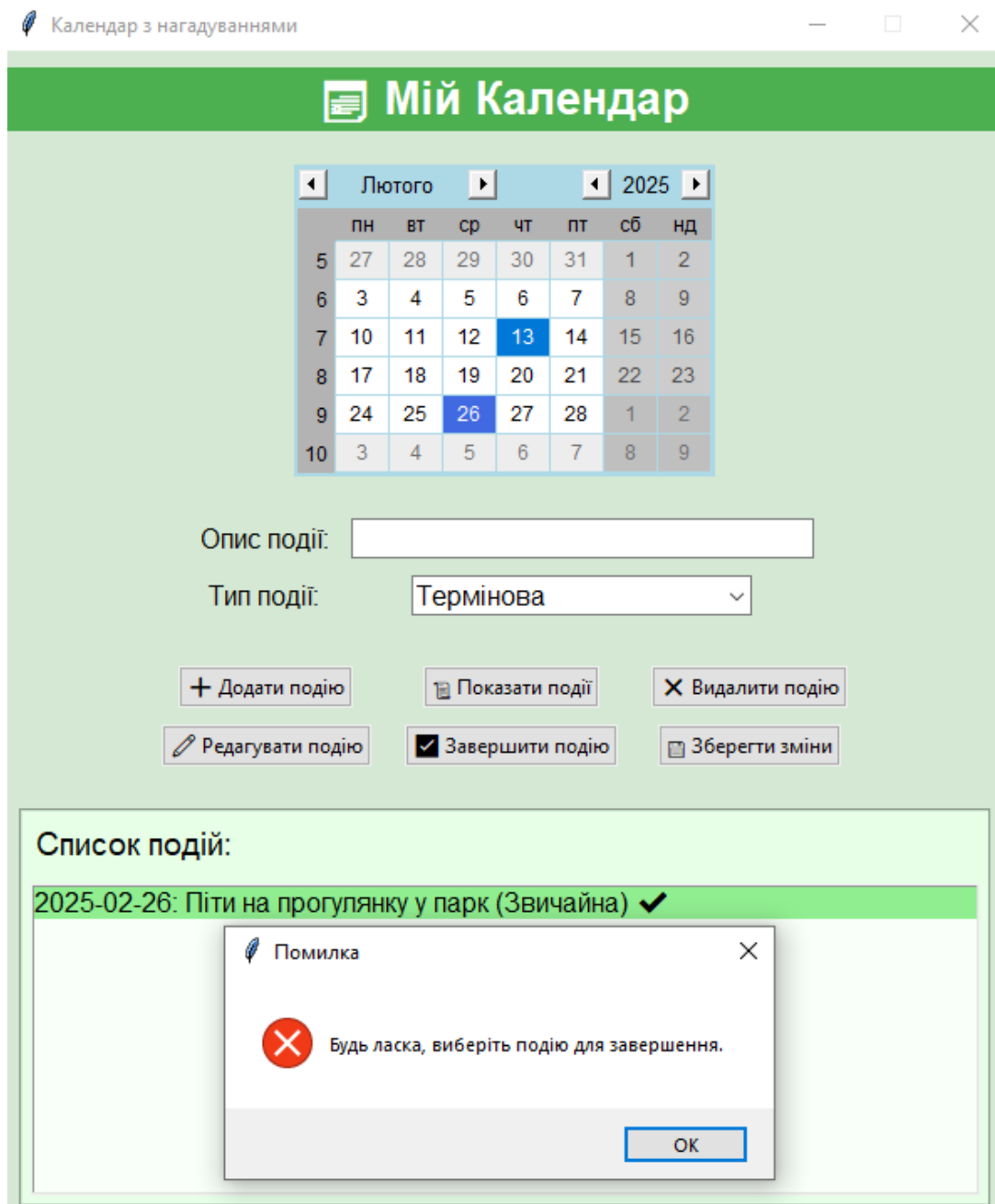


Рис.5.11 – Повідомлення про те, що потрібно обрати подію для її завершення

Щоб програма не займала місце на екрані, її можна згорнути в трей. Щоб приховати програму в трей:

1. Потрібно натиснути кнопку закриття (X) – програма не завершиться, а сховається у трей.
2. У системному треї з'явиться іконка програми.

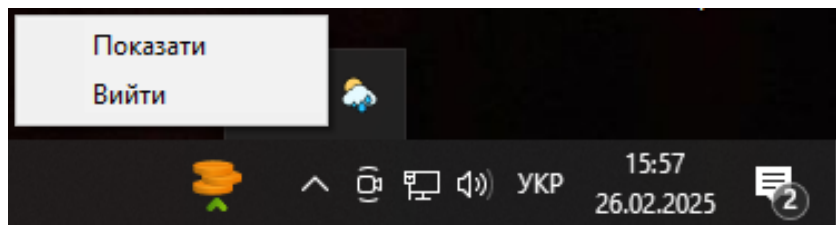


Рис.5.12 – Програма з’явилась у системному треї як іконка після приховування

Щоб відновити вікно програми календарю:

1. Потрібно натиснути правою кнопкою миші на іконку в треї.
2. Потрібно вибрати "Показати" – головне вікно програми знову відкриється.

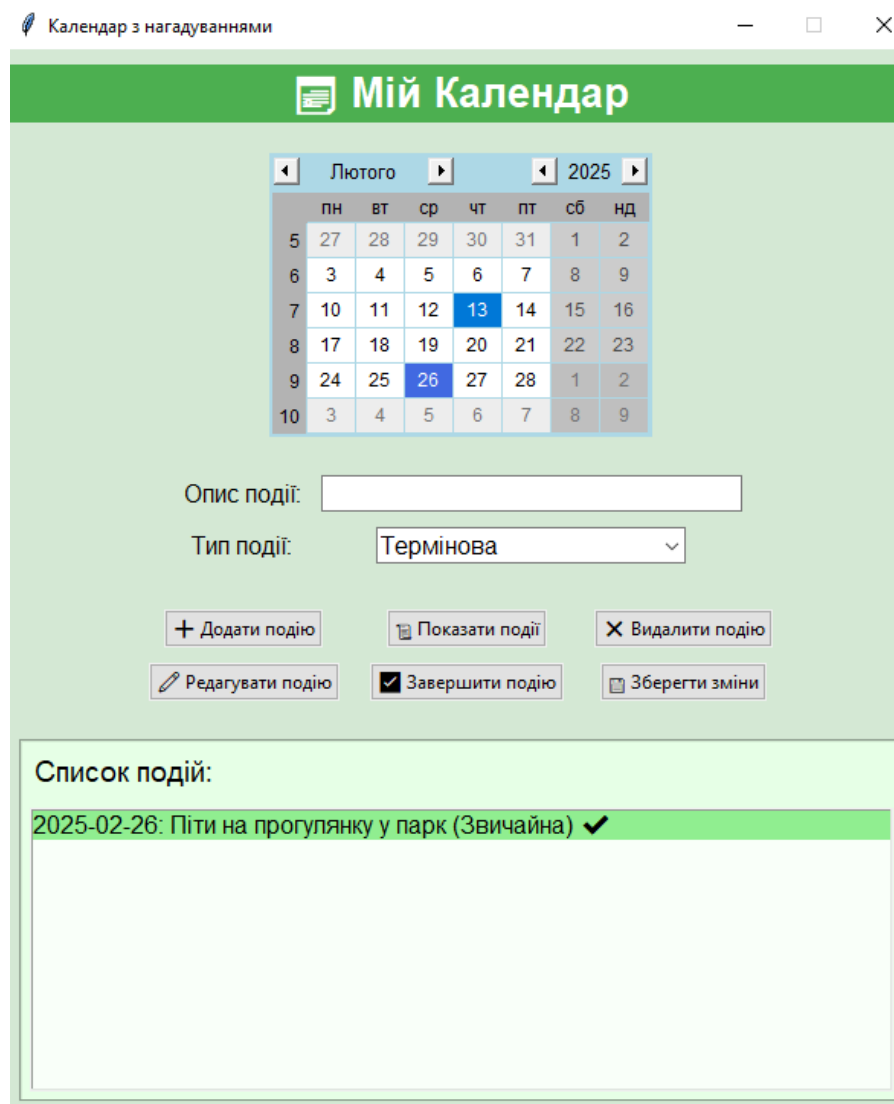


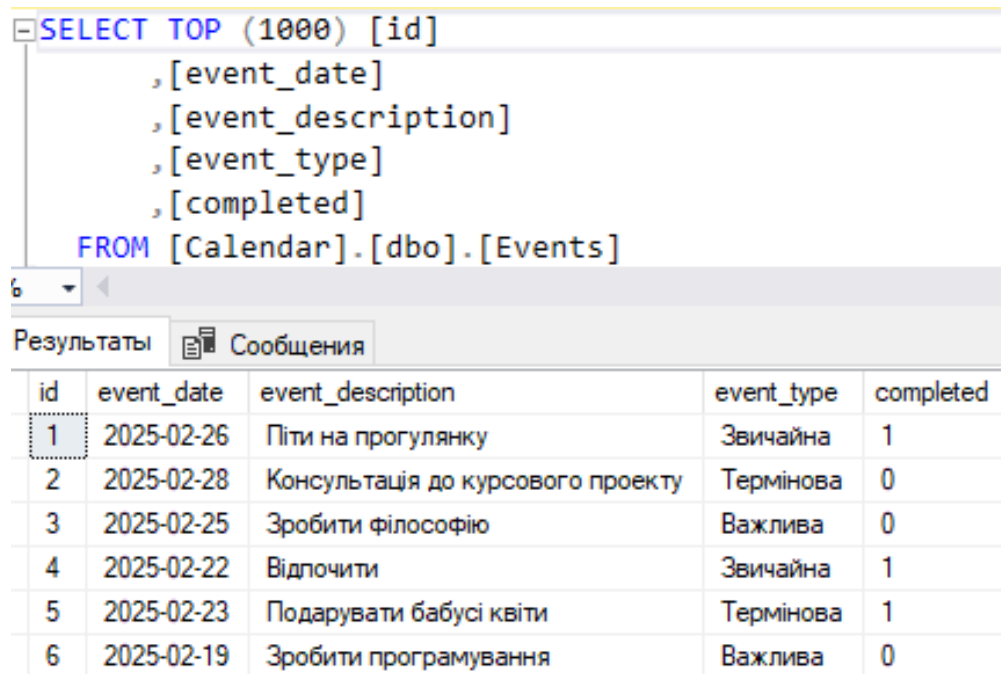
Рис.5.13 – Програма відновилась після використання іконки в треї

Щоб вийти з програми календарю:

1. Потрібно натиснути правою кнопкою миші на іконку в треї.
2. Потрібно вибрати "Вийти".
3. Програма закриється, а база даних буде збережена.

Для перегляду подій безпосередньо в базі даних можна використовувати SQL-запит: `SELECT TOP (1000) [id], [event_date], [event_description], [event_type], [completed] FROM [Calendar].[dbo].[Events];`

- id – унікальний ідентифікатор події
- event_date – дата події
- event_description – опис події
- event_type – тип події
- completed – статус завершення події (0 – незавершена, 1 – завершена)



id	event_date	event_description	event_type	completed
1	2025-02-26	Піти на прогулянку	Звичайна	1
2	2025-02-28	Консультація до курсового проекту	Термінова	0
3	2025-02-25	Зробити філософію	Важлива	0
4	2025-02-22	Відпочити	Звичайна	1
5	2025-02-23	Подарувати бабусі квіти	Термінова	1
6	2025-02-19	Зробити програмування	Важлива	0

Рис.5.14 – Перегляд збережених подій у базі даних

6. ВИСНОВКИ

У ході виконання курсової роботи було розроблено програму «Створення календар з нагадуванням про події.», яка дозволяє зберігати, редагувати, видаляти та переглядати заплановані події. Реалізовано функції додавання подій, їх категоризації, позначення завершених завдань, а також автоматичне нагадування про незавершені події.

Розробка програми включала створення графічного інтерфейсу за допомогою бібліотеки Tkinter, використання бази даних Microsoft SQL Server для зберігання інформації про події та інтеграцію програми з системним треем для зручного доступу.

Основні результати роботи:

1. Розроблено інтуїтивно зрозумілий інтерфейс, що дозволяє користувачам легко взаємодіяти з програмою.
2. Реалізовано збереження подій у базі даних, що забезпечує довготривале зберігання інформації.
3. Реалізовано функціонал нагадувань, який щодня перевіряє наявність незавершених подій та повідомляє про них користувача.
4. Забезпечено можливість редагування та видалення подій, що дозволяє гнучко керувати списком запланованих справ.
5. Додано функцію системного троя, що дозволяє приховувати програму та викликати її у зручний момент без потреби повторного запуску.

У результаті виконаної роботи вдалося створити зручний та ефективний інструмент для управління подіями, який може використовуватися як для особистих, так і для робочих цілей. Подальший розвиток проєкту може включати розширення функціоналу, додавання підтримки мобільних пристроїв та інтеграцію з іншими календарними сервісами.

7. СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Head-First Python, 2nd edition: Paul Barry. - Sebastopol, California, U.S.: O'Reilly Media, 2016. – 622 с.
2. Think Python: How to Think Like a Computer Scientist, 2nd edition: Allen B. Downey. - Sebastopol, California, U.S.: O'Reilly Media, 2015. – 292 с.
3. Clean Code: A Handbook of Agile Software Craftsmanship: Robert C. Martin. - London, England: Pearson, 2008. – 464 с.
4. Python.org: веб-сайт. URL: <https://www.python.org>
5. Python Tutorial: веб-сайт. URL: <https://www.w3schools.com/python/>
6. Learn to become a modern Python developer: веб-сайт. URL: <https://roadmap.sh/python/>

ДОДАТКИ

Додаток А

```
import tkinter as tk
from tkinter import messagebox, ttk
from tkcalendar import Calendar
import datetime
import pyodbc
import winsound
from pystray import Icon, MenuItem, Menu
from PIL import Image, ImageDraw
import threading

class ReminderApp:
    def __init__(self, root):
        self.root = root
        self.root.protocol("WM_DELETE_WINDOW", self.hide_window)
        self.root.title("Календар з нагадуваннями")
        self.root.geometry("600x700")
        self.root.resizable(False, False)
        self.is_dark_mode = False

        try:
            self.conn = pyodbc.connect('DRIVER={ODBC Driver 17 for SQL
Server};'

                                     'SERVER=DESKTOP-QQAOEK4;'
                                     'DATABASE=Calendar;'
                                     'Trusted_Connection=yes;'
                                     'Encrypt=yes;'
                                     'TrustServerCertificate=yes;')

            self.cursor = self.conn.cursor()
```

					5.151.1.40-КП	Лист
						53
Изм.	Лист	№ докум.	Подпис	Дата		

```

print("З'єднання з базою даних встановлено успішно.")
except pyodbc.Error as e:
    messagebox.showerror("Помилка підключення", f"Не вдалось
підключитись до бази даних: {e}")
    self.root.quit()

self.create_table()

self.root.configure(bg="#d4e8d4")

header = tk.Label(root, text="📅 Мій Календар", font=("Helvetica",
20, "bold"), bg="#4CAF50", fg="white")
header.pack(fill="x", pady=10)

self.cal_frame = tk.Frame(root, bg="#d4e8d4")
self.cal_frame.pack(pady=10)
self.cal = Calendar(
    self.cal_frame,
    selectmode="day",
    date_pattern="yyyy-mm-dd",
    locale="uk",
    background="lightblue",
    foreground="black",
    bordercolor="lightblue"
)
self.cal.pack()

self.today_date = datetime.date.today()
self.cal.calevent_create(self.today_date, "Сьогодні", "today")

```

```

self.event_frame = tk.Frame(root, bg="#d4e8d4")
self.event_frame.pack(pady=10)

self.event_label = tk.Label(self.event_frame, text="Опис події:",
font=("Helvetica", 12), bg="#d4e8d4")
self.event_label.grid(row=0, column=0, padx=5, pady=5)

self.event_entry = ttk.Entry(self.event_frame, width=30,
font=("Helvetica", 12))
self.event_entry.grid(row=0, column=1, padx=5)
self.event_entry.insert(0, "Введіть опис події")

self.event_type_label = tk.Label(self.event_frame, text="Тип події:",
font=("Helvetica", 12), bg="#d4e8d4")
self.event_type_label.grid(row=1, column=0, padx=5, pady=5)

self.event_type = ttk.Combobox(self.event_frame,
values=["Звичайна", "Важлива", "Термінова"], font=("Helvetica", 12))
self.event_type.grid(row=1, column=1, padx=5)
self.event_type.set("Звичайна")

self.button_frame = tk.Frame(root, bg="#d4e8d4")
self.button_frame.pack(pady=10)

self.add_button = ttk.Button(self.button_frame, text="✚ Додати
подію", command=self.add_event)
self.add_button.grid(row=0, column=0, padx=10, pady=5)

```

```
self.show_button = ttk.Button(self.button_frame, text="📄 Показати  
події", command=self.show_events)
```

```
self.show_button.grid(row=0, column=1, padx=10, pady=5)
```

```
self.remove_button = ttk.Button(self.button_frame, text="✕  
Видалити подію", command=self.remove_event)
```

```
self.remove_button.grid(row=0, column=2, padx=10, pady=5)
```

```
self.edit_button = ttk.Button(self.button_frame, text="✎ Редагувати  
подію", command=self.edit_event)
```

```
self.edit_button.grid(row=1, column=0, padx=10, pady=5)
```

```
self.complete_button = ttk.Button(self.button_frame, text="✅  
Завершити подію", command=self.complete_event)
```

```
self.complete_button.grid(row=1, column=1, padx=10, pady=5)
```

```
self.save_button = ttk.Button(self.button_frame, text="💾 Зберегти  
зміни", command=self.save_changes)
```

```
self.save_button.grid(row=1, column=2, padx=10, pady=5)
```

```
self.list_frame = tk.Frame(root, bg="#e6ffe6", bd=2, relief="groove")
```

```
self.list_frame.pack(fill="both", expand=True, padx=10, pady=10)
```

```
self.list_label = tk.Label(self.list_frame, text="Список подій:",  
bg="#e6ffe6", font=("Helvetica", 14))
```

```
self.list_label.pack(anchor="nw", padx=5, pady=5)
```

```
self.event_list = tk.Listbox(  
self.list_frame,
```



```

        font=("Helvetica", 12),
        bg="#f9fff9",
        selectbackground="#b3ffb3",
        selectforeground="black",
        height=10
    )
    self.event_list.pack(fill="both", expand=True, padx=5, pady=5)

    self.root.after(1000, self.daily_check)

def create_table(self):
    self.cursor.execute("""
        IF NOT EXISTS (SELECT * FROM sysobjects WHERE
name='Events' AND xtype='U')
        CREATE TABLE Events (
            id INT PRIMARY KEY IDENTITY(1,1),
            event_date DATE,
            event_description NVARCHAR(255),
            event_type NVARCHAR(50),
            completed BIT
        )
    """)
    self.conn.commit()

def add_event(self):
    date = self.cal.get_date()
    event = self.event_entry.get()
    event_type = self.event_type.get()

    if date and event and event.strip() != "":

```

```

self.cursor.execute("""
    INSERT INTO Events (event_date, event_description, event_type,
completed)
    VALUES (?, ?, ?, ?)
    """, (date, event, event_type, False))
self.conn.commit()

messagebox.showinfo("Успіх", f"Подія '{event}' додана на
{date}!")

self.event_entry.delete(0, tk.END)
self.update_event_list()
else:
    messagebox.showerror("Помилка", "Будь ласка, виберіть дату та
введіть подію.")

def show_events(self):
    date = self.cal.get_date()
    self.cursor.execute("SELECT event_description, event_type, completed
FROM Events WHERE event_date = ?", (date,))
    events = self.cursor.fetchall()
    if events:
        event_details = "\n".join([f'{event[0]} ({event[1]}) {'✓' if event[2]
else '✗'}" for event in events])
        messagebox.showinfo("Події", f"Події на
{date}:\n{event_details}")
    else:
        messagebox.showinfo("Події", f"На {date} немає подій.")

def remove_event(self):

```

```

selected_event = self.event_list.curselection()
if selected_event:
    event_info = self.event_list.get(selected_event)
    date = event_info.split(":")[0].strip()
    event_name = event_info.split(":")[1].split("(")[0].strip()

    self.cursor.execute("DELETE FROM Events WHERE event_date =
? AND event_description = ?", (date, event_name))
    self.conn.commit()

    self.update_event_list()
    messagebox.showinfo("Успіх", f"Подія '{event_name}' була
видалена.")
else:
    messagebox.showerror("Помилка", "Будь ласка, виберіть подію
для видалення.")

def update_event_list(self):
    self.event_list.delete(0, tk.END)
    self.cursor.execute("SELECT event_date, event_description,
event_type, completed FROM Events ORDER BY event_date")
    events = self.cursor.fetchall()
    for event in events:
        date, description, event_type, completed = event
        color = "lightgreen" if event_type == "Звичайна" else "yellow" if
event_type == "Важлива" else "red"
        self.event_list.insert(tk.END, f"{date}: {description} ({event_type})
{'✓' if completed else '✗'}")
        self.event_list.itemconfig(tk.END, {'bg': color})

```

```

def edit_event(self):
    selected_event = self.event_list.curselection()
    if selected_event:
        event_info = self.event_list.get(selected_event)
        description = event_info.split(":")[1].split("(")[0].strip()
        event_type = event_info.split("(")[1].split(")")[0].strip()

        self.event_entry.delete(0, tk.END)
        self.event_entry.insert(0, description)
        self.event_type.set(event_type)

        self.selected_event = event_info
    else:
        messagebox.showerror("Помилка", "Будь ласка, виберіть подію
для редагування.")

def save_changes(self):
    new_description = self.event_entry.get()
    new_event_type = self.event_type.get()

    if hasattr(self, 'selected_event') and self.selected_event:
        old_description = self.selected_event.split(":")[1].split("(")[0].strip()
        old_event_type = self.selected_event.split("(")[1].split(")")[0].strip()
        date = self.selected_event.split(":")[0].strip()

        if new_description != old_description or new_event_type !=
old_event_type:
            self.cursor.execute("UPDATE Events SET event_description = ?,
event_type = ? WHERE event_date = ? AND event_description = ?",

```

```

        (new_description, new_event_type, date,
old_description))
        self.conn.commit()

        self.update_event_list()
        messagebox.showinfo("Успіх", f"Подія успішно оновлена!")
    else:
        messagebox.showinfo("Інформація", "Опис події та тип не
змінились.")
    else:
        messagebox.showerror("Помилка", "Будь ласка, спочатку
виберіть подію для редагування.")

def complete_event(self):
    selected_event = self.event_list.curselection()
    if selected_event:
        event_info = self.event_list.get(selected_event)
        date = event_info.split(":")[0].strip()
        event_name = event_info.split(":")[1].split("(")[0].strip()

        self.cursor.execute("UPDATE Events SET completed = 1 WHERE
event_date = ? AND event_description = ?", (date, event_name))
        self.conn.commit()

        self.update_event_list()
        messagebox.showinfo("Успіх", f"Подія '{event_name}' була
позначена як завершена.")
    else:
        messagebox.showerror("Помилка", "Будь ласка, виберіть подію
для завершення.")

```

```

def daily_check(self):
    today = datetime.date.today()
    self.cursor.execute("SELECT event_description FROM Events
WHERE event_date = ? AND completed = 0", (today,))
    events = self.cursor.fetchall()
    for event in events:
        messagebox.showinfo("Нагадування", f"Сьогодні: {event[0]}")

self.root.after(86400000, self.daily_check)

def hide_window(self):
    self.root.withdraw()

    icon = self.create_icon()
    icon.run()

def create_icon(self):
    try:
        icon_image = Image.open("C:/Users/Ruslan
Ostrovsky/Desktop/kyrsovaaya/иконки/icon1.png")
        icon_image = icon_image.resize((64, 64))
    except Exception as e:
        messagebox.showerror("Помилка", f"Не вдалось завантажити
іконку: {e}")
        return None

    icon = Icon("ReminderApp", icon_image, menu=Menu(
        MenuItem("Показати", self.show_window),
        MenuItem("Вийти", self.quit_app)

```

```
))
```

```
icon.tooltip = "Нагадування"  
return icon
```

```
def show_window(self, icon):  
    self.root.deiconify()  
    icon.stop()
```

```
def quit_app(self, icon):  
    self.conn.close()  
    icon.stop()  
    self.root.quit()
```

```
if __name__ == "__main__":  
    root = tk.Tk()  
    app = ReminderApp(root)  
    root.mainloop()
```