

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

КУРСОВИЙ ПРОЕКТ  
(РОБОТА)

Програмування  
(назва дисципліни)

на тему: **Розробка програмного додатку для інформування користувачів**

Студента (ки) IV курсу 40-ІС групи  
спеціальності 151 Автоматизація та  
комп'ютерно-інтегровані технології  
спеціалізації 5.151.1 Обслуговування  
інтелектуальних інтегрованих систем  
**Федулов І.Ю.**

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

**Федулова Іллі Юрійовича**

Тема: Розробка програмного додатку для інформування користувачів.  
Термін виконання роботи з 12.11.2024 р. по 31.03.2025 р.  
Вхідні дані для проектування: (Варіант №16)

Розробка програмного додатку для інформування користувачів про події. Додаток повинен мати список користувачів і канали зв'язку з ними та можливість відправити повідомлення за цими каналами. 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 59, рисунки 43, використану літературу 6 та додатки 3.

Об'єктом розробки є програма для інформування користувачів.

Мета розробки – створення програмного забезпечення для зручного планування та управління подіями, а також для ефективного інформування користувачів про важливі події через різні канали зв'язку.

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 та інформування користувачів.

Розроблений застосунок може використовуватися для підвищення ефективності планування та нагадування про важливі події, що дозволяє користувачам не пропускати значущі моменти і своєчасно отримувати необхідну інформацію через вибрані канали зв'язку (електронну пошту, SMS або інші).

Python, Visual Studio, SQL Server, Tkinter, pyodbc, інформування.

## **ПЕРЕЛІК СКОРОЧЕНЬ**

МП – Мова програмування;

ПЗ – Програмне забезпечення;

БД – База даних;

API – Інтерфейс програмування додатків.

## ЗМІСТ

РЕФЕРАТ.....	4
ПЕРЕЛІК СКОРОЧЕНЬ.....	5
1. ВСТУП .....	7
2. АНАЛІЗ ЗАДАЧІ, ЗАСОБІВ ТА МЕТОДІВ ЇЇ ВИРІШЕННЯ.....	8
2.1 Python.....	9
2.2 Історія створення мови програмування Python .....	11
2.4 База даних SQL Server.....	13
3. ПРОЕКТУВАННЯ ЗАГАЛЬНОГО АЛГОРИТМУ РОБОТИ ПРОГРАМИ.....	15
3.1 Загальна структура програми .....	15
3.2 Алгоритм роботи програми.....	15
4. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ .....	22
4.1 Створення бази даних.....	22
4.2 Метод з'єднання з базою даних .....	22
4.3 Метод перевірки таблиць .....	23
4.4 Метод виконання SQL-запитів.....	24
4.5 Розробка графічного інтерфейсу користувача та взаємодія з базою даних .....	25
4.6 Імпорти та клас .....	26
4.7 Клас EventNotifierGUI .....	27
4.8 Метод всіх елементів інтерфейсу.....	28
4.9 Метод завантаження даних користувачів .....	29
4.10 Метод відправки повідомлень .....	29
4.11 Методи додавання та видалення користувачів.....	30
4.12 Метод створення, впливаючого вікна та збереження події.....	31
4.13 Метод перевірки подій .....	32
4.14 Метод відправки нагадувань та показу візуальних повідомлень.....	33
4.15 Створення та запуск основної програми з підключенням до бази даних .....	34
5. КЕРІВНИЦТВО КОРИСТУВАЧА .....	36
6. ВИСНОВКИ .....	45
7. СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	46
ДОДАТКИ.....	47

					<i>5.151.1.40-КП</i>			
Змін.	Арк.	№ докум.	Підпис	Дата				
Розроб.		Федулов І.Ю.			Пояснювальна записка	Літ.	Арк.	Аркуші
Перевір.		Васильєв М.В.					6	59
Т. контр.						ХПФК група 40-ІС		
Н. Контр.								
Затверд.								

## 1. ВСТУП

Останнім часом інформаційні технології стали невід'ємною частиною повсякденного життя, і їх застосування охоплює всі сфери діяльності. Усе більша кількість компаній і організацій використовує програмне забезпечення для оптимізації комунікацій, підвищення ефективності взаємодії з клієнтами, співробітниками та партнерами. Одним із ключових аспектів є інформування користувачів про важливу інформацію, що дозволяє своєчасно донести необхідні дані та забезпечити належний рівень комунікації.

Метою цього проекту є розробка програмного додатку для інформування користувачів через різні канали зв'язку. Додаток буде надавати можливість зберігати дані про користувачів, визначати канали зв'язку з ними та надсилати повідомлення через ці канали. Програма забезпечить автоматичне відправлення повідомлень користувачам, що значно спростить процес комунікації та допоможе своєчасно доставляти важливу інформацію.

Актуальність цього проекту зумовлена потребою в ефективному управлінні комунікаціями, що особливо важливо для великих організацій, де інформація повинна бути доставлена до багатьох осіб швидко і без помилок. Використання бази даних для збереження інформації дозволить забезпечити надійний доступ до даних і зручне управління ними.

Метою даного проекту є створення зручного та ефективного інструменту для автоматизації процесу інформування користувачів, що допоможе покращити організацію комунікацій і забезпечить своєчасне доставлення важливих повідомлень.

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

## 2. АНАЛІЗ ЗАДАЧІ, ЗАСОБІВ ТА МЕТОДІВ ЇЇ ВИРІШЕННЯ

В умовах постійного розвитку інформаційних технологій одним із важливих аспектів у сучасному світі є забезпечення ефективної комунікації між користувачами. Зокрема, у великих організаціях, де кількість співробітників або користувачів постійно зростає, важливо забезпечити своєчасне інформування через різні канали зв'язку. Під час виконання даного проекту основною задачею є розробка програмного забезпечення, яке дозволить автоматизувати процес інформування користувачів через канали зв'язку.

Задача полягає у створенні програмного додатку, який буде зберігати інформацію про користувачів, їх канали зв'язку та надавати можливість відправлення повідомлень через ці канали. Оскільки ефективність комунікацій є важливою складовою в роботі організацій, створення такого інструменту дозволить оптимізувати процеси взаємодії між користувачами та забезпечить своєчасне та точне інформування.

Основними вимогами до програмного забезпечення є:

- Збереження та організація даних про користувачів.
- Підтримка кількох каналів зв'язку (електронна пошта, SMS, месенджери тощо).
- Можливість відправлення повідомлень на визначені канали зв'язку.
- Надійність і швидкість доступу до даних.
- Інтерфейс, що дозволяє зручно управляти користувачами та їх даними.

Засоби вирішення задачі:

Для вирішення поставленої задачі будуть використані сучасні програмні засоби та технології:

- Мова програмування Python: вона є зручним інструментом для розробки таких додатків завдяки великій кількості бібліотек, зокрема для роботи з базами даних, управління користувачами та канали зв'язку.

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



- Tkinter: бібліотека для створення графічного інтерфейсу користувача (GUI), що забезпечить зручне взаємодія користувача з додатком.
- База даних SQL Server: для зберігання інформації про користувачів, їх канали зв'язку та відправлені повідомлення. Використання реляційної бази даних забезпечить надійність та швидкість доступу до даних.
- pyodbc: бібліотека для роботи з SQL Server через Python, що дозволить зручно виконувати операції з базою даних.

Методи вирішення задачі:

Для реалізації програмного забезпечення будуть використані наступні методи:

1. Модульне програмування: розробка окремих модулів для управління базою даних, інтерфейсом користувача та процесом відправлення повідомлень.
2. Інтерфейс користувача: створення графічного інтерфейсу, що забезпечить простоту взаємодії з додатком та зручне управління даними.
3. Автоматизація процесу інформування: реалізація механізму автоматичного відправлення повідомлень користувачам через вибрані канали зв'язку.
4. Оптимізація доступу до даних: використання реляційної бази даних дозволить швидко здійснювати запити та маніпулювати інформацією про користувачів і повідомлення.

## 2.1 Python

Python є однією з найпопулярніших мов програмування на сьогоднішній день. Вона відома своєю простотою та зручністю для початківців, а також потужністю та гнучкістю, що дозволяє її використовувати для різноманітних завдань. Python активно застосовується у розробці програмного забезпечення, аналізі даних, автоматизації процесів, створенні веб-додатків, а також в інших сферах, таких як машинне навчання, штучний інтелект та багато іншого.

Однією з основних причин вибору Python для цього проекту є його зручність у розробці та великий набір бібліотек, які значно спрощують виконання завдань, таких як робота з базами даних, побудова графічного інтерфейсу та інтеграція з іншими технологіями.

Основні переваги Python для реалізації проекту:

1. Простота синтаксису: python має дуже чистий і зрозумілий синтаксис, що дозволяє швидко освоїти мову і зосередитись на вирішенні завдань, а не на складних деталях реалізації. Це особливо важливо під час розробки програм, де важлива швидка і ефективна реалізація ітерацій.

2. Широкий набір бібліотек: python має багатий набір бібліотек, які дозволяють легко вирішувати конкретні завдання без необхідності розробляти складні алгоритми з нуля. Наприклад, для роботи з базами даних буде використано бібліотеку pyodbc, що спрощує процес взаємодії з SQL Server. Для створення графічного інтерфейсу користувача буде використана бібліотека Tkinter.

3. Міжплатформність: python є кросплатформною мовою, що означає, що програми, написані на Python, можуть працювати на різних операційних системах (Windows, Linux, macOS) без значних змін у коді. Це дозволяє забезпечити широку сумісність програми, що є важливим для подальшого використання програмного продукту на різних пристроях.

4. Швидка розробка: завдяки своїй простоті та багатим бібліотекам Python дозволяє скоротити час розробки програмного забезпечення, що особливо важливо в умовах обмежених термінів. У поєднанні з можливістю швидко тестувати та відлагоджувати програму, Python є оптимальним вибором для розробки.

5. Гнучкість та потужність: python є достатньо потужним для вирішення складних завдань. Зокрема, за допомогою бібліотек для роботи з базами даних (наприклад, pyodbc) можна ефективно взаємодіяти з SQL Server, а бібліотеки для роботи з мережевими протоколами дозволяють

реалізувати функціонал для відправки повідомлень через різні канали зв'язку (SMS, електронна пошта, месенджери).

6. Підтримка об'єктно-орієнтованого програмування (ООП): python підтримує принципи об'єктно-орієнтованого програмування, що дозволяє структурувати код і зробити його більш зручним для подальшого розширення та супроводження. Це особливо важливо при розробці програмного забезпечення, яке повинно бути масштабованим і підтримувати додаткові функціональні можливості в майбутньому.



Рис.2.1.1 – Логотип Python

## 2.2 Історія створення мови програмування Python

Мова програмування **Python** була створена у кінці 1980-х років голландським програмістом **Гвідо ван Россумом** (Guido van Rossum). Офіційний реліз першої версії відбувся **20 лютого 1991 року**. Основною метою розробки було створення простої, читабельної та гнучкої мови програмування, яка б забезпечувала високу продуктивність і зручність використання.

Гвідо ван Россум працював у Центрі математики та інформатики (CWI) в Нідерландах і був залучений до розробки мови програмування **ABC**. Ця мова мала простий синтаксис, але мала певні обмеження, які заважали її широкому поширенню. Надихнувшись її ідеями та бажаючи створити більш потужний та універсальний інструмент, ван Россум розпочав роботу над новою мовою.

Python отримав свою назву не на честь змії, а завдяки британському комедійному шоу "**Monty Python's Flying Circus**", яке було популярним у той час. Ван Россум хотів, щоб його мова була простою, зрозумілою та, водночас, приносила задоволення від програмування.

Основні етапи розвитку Python:

#### 1. Python 1.0 (1991 рік)

- Перша офіційна версія Python 1.0 була випущена у 1991 році.
- Вона вже містила основні особливості, які відрізняли її від інших мов:
  - динамічну типізацію,
  - автоматичне керування пам'яттю (збирання сміття),
  - модульність (підтримку бібліотек),
  - простий та читабельний синтаксис.

#### 2. Python 2.x (2000 рік)

- У 2000 році вийшла версія Python 2.0, яка принесла важливі нововведення:
  - підтримку спискових виразів (list comprehensions),
  - покращене збирання сміття,
  - нові можливості роботи з Unicode.
- Python 2.x став дуже популярним, і хоча його офіційна підтримка завершилася у 2020 році, багато компаній та програмістів використовували його протягом двох десятиліть.

#### 3. Python 3.x (2008 – сьогодні)

- У 2008 році була випущена версія Python 3.0, яка внесла значні зміни:
  - покращена підтримка Unicode,
  - новий підхід до розділення цілочисельного та дійсного ділення (/ та //),
  - оновлення стандартної бібліотеки.

- Python 3.x продовжує активно розвиватися, додаючи нові можливості та покращуючи продуктивність.

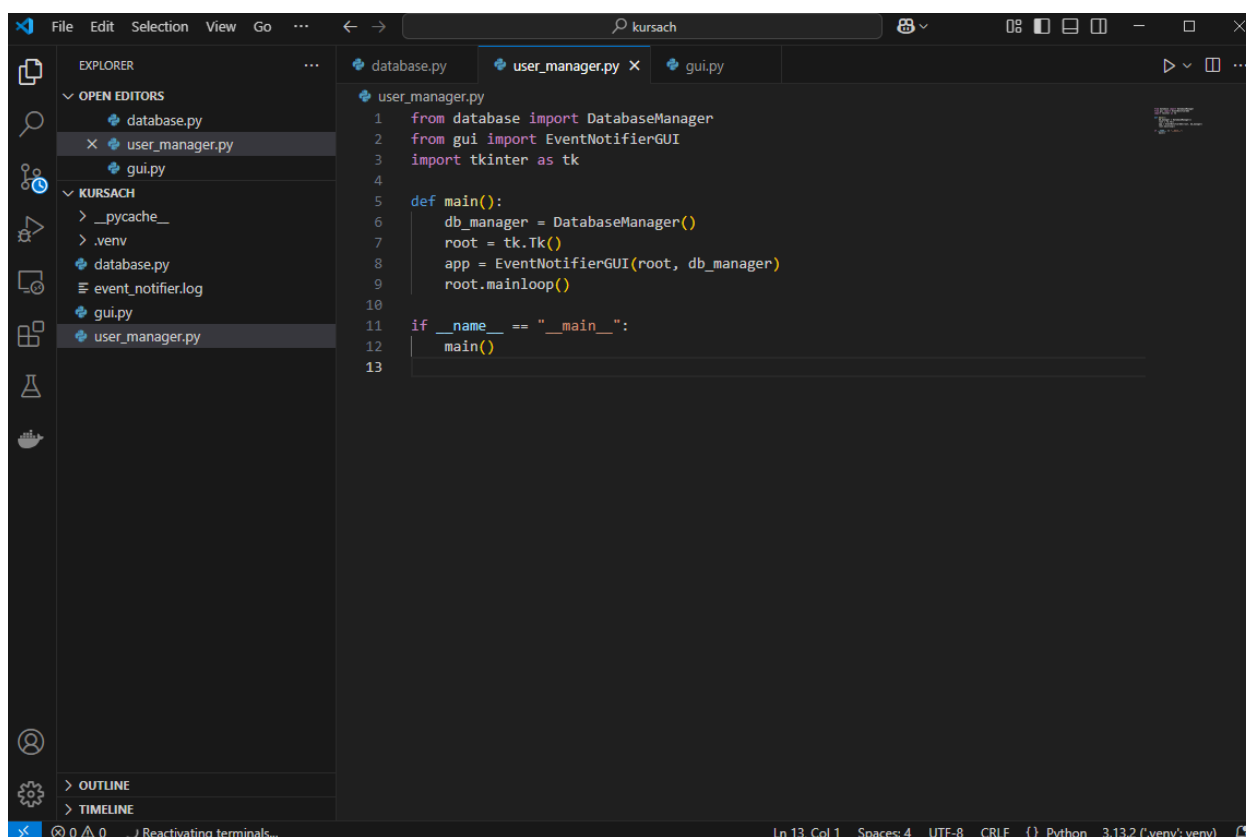


Рис.2.3.1 – Середовище розробки Visual Studio

## 2.4 База даних SQL Server

SQL Server — це реляційна система управління базами даних (СУБД), розроблена компанією Microsoft. Вона використовується для зберігання, управління та обробки великих обсягів даних у різних додатках, від корпоративних систем до веб-додатків та мобільних сервісів.

SQL Server був створений у 1989 році компаніями Microsoft, Sybase і Ashton-Tate. Перші версії були розроблені для OS/2, а згодом Microsoft почала розвивати його як продукт для Windows.

Офіційно Microsoft SQL Server 6.0 (1995 р.) став першою повністю самостійною версією компанії Microsoft. Відтоді SQL Server постійно оновлювався, отримуючи нові функції, покращену продуктивність та безпеку.

					5.151.1.40-КП	Лист
Изм.	Лист	№ докум.	Підпис	Дата		13

Сучасні версії, такі як **SQL Server 2019** та **SQL Server 2022**, підтримують роботу з великими даними, штучним інтелектом та хмарними технологіями.

Microsoft SQL Server є потужною, надійною та масштабованою базою даних, яка підходить для реалізації різних інформаційних систем. Використання SQL Server у нашому проекті забезпечить ефективне управління даними користувачів, повідомленнями та каналами зв'язку.

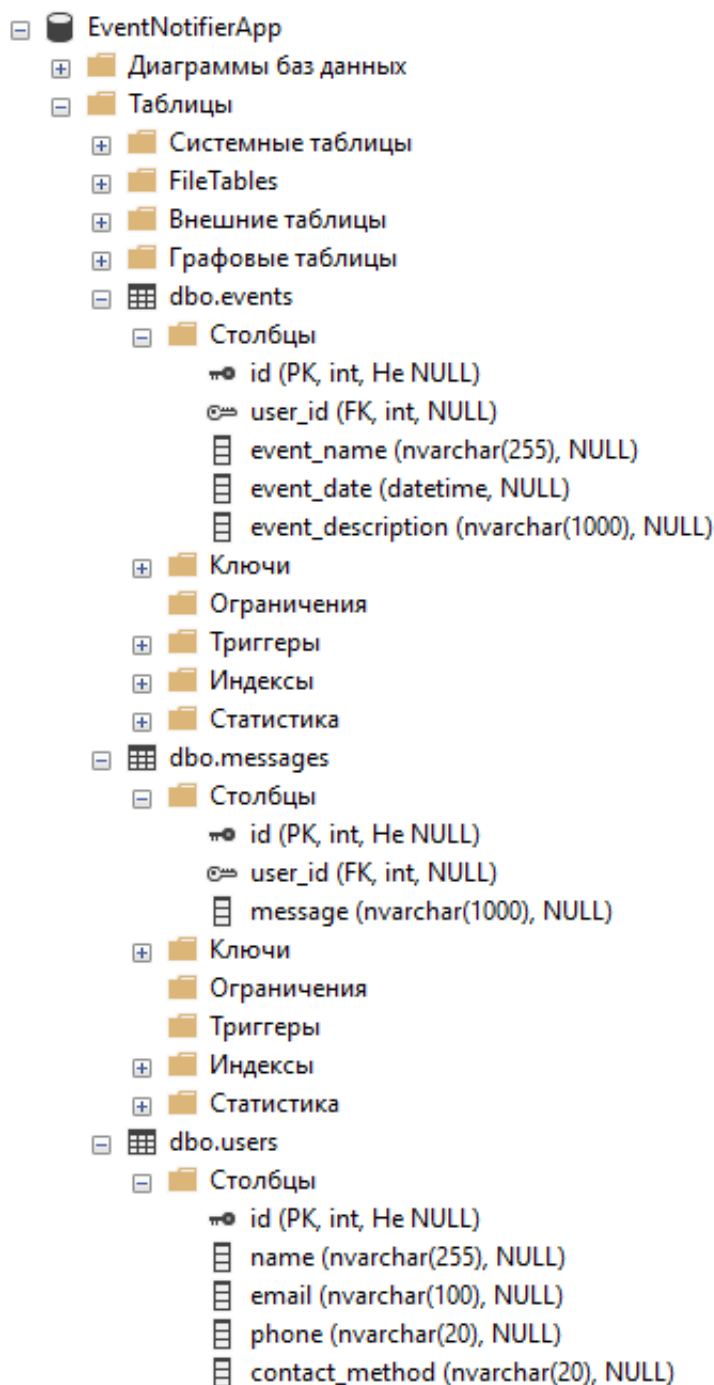


Рис.2.4.1 – База даних EventNotifierApp SQL Server

### 3. ПРОЕКТУВАННЯ ЗАГАЛЬНОГО АЛГОРИТМУ РОБОТИ ПРОГРАМИ

#### 3.1 Загальна структура програми

Розроблений програмний застосунок призначений для інформування користувачів шляхом надсилання повідомлень через різні канали зв'язку. Його основна функціональність включає:

- збереження списку користувачів у базі даних,
- керування каналами зв'язку (електронна пошта, месенджери тощо),
- можливість відправлення повідомлень користувачам,
- перегляд історії повідомлень та їх статусів.

Програма складається з трьох основних модулів:

1. Модуль управління базою даних (database.py) – збереження користувачів, контактної інформації та історії повідомлень.
2. Модуль запуску програми (user\_manager.py) – взаємодія з іншими модулями та запуск програми.
3. Модуль інтерфейсу (gui.py) – графічний інтерфейс для взаємодії з користувачем.

#### 3.2 Алгоритм роботи програми

Робота застосунку базується на наступному алгоритмі:

Запуск програми:

- Ініціалізація підключення до бази даних SQL Server.
- Завантаження списку користувачів та їх каналів зв'язку.
- Відображення головного вікна інтерфейсу.

Робота з користувачами:

- Додавання нового користувача до бази даних.
- Редагування або видалення існуючих користувачів.
- Прив'язка каналів зв'язку (email, Telegram, SMS тощо).

Формування та відправлення повідомлення:

- Вибір користувача або групи користувачів.
- Визначення каналу зв'язку.

					5.151.1.40-КП	Лист
						15
Изм.	Лист	№ докум.	Підпис	Дата		

- Введення тексту повідомлення.
- Збереження статусу повідомлення у базі даних.

Перегляд історії повідомлень:

- Відображення списку надісланих повідомлень.

Завершення роботи програми:

- Закриття підключення до бази даних.
- Завершення виконання процесів.



Рис.3.2.1 – Алгоритм роботи застосунку від початку до дій користувача



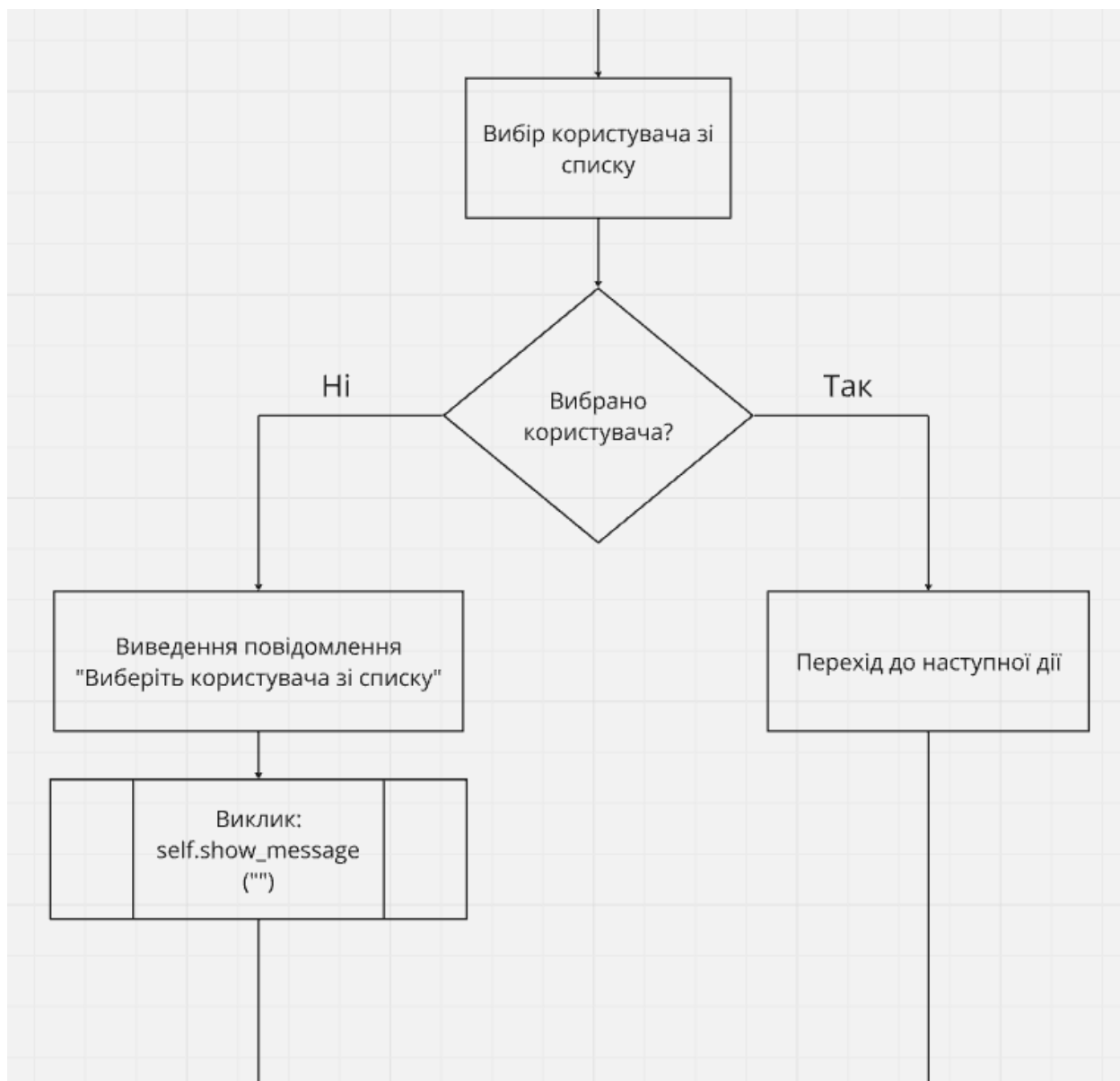


Рис.3.2.2 – Алгоритм роботи завдання “Вибір користувача зі списку”

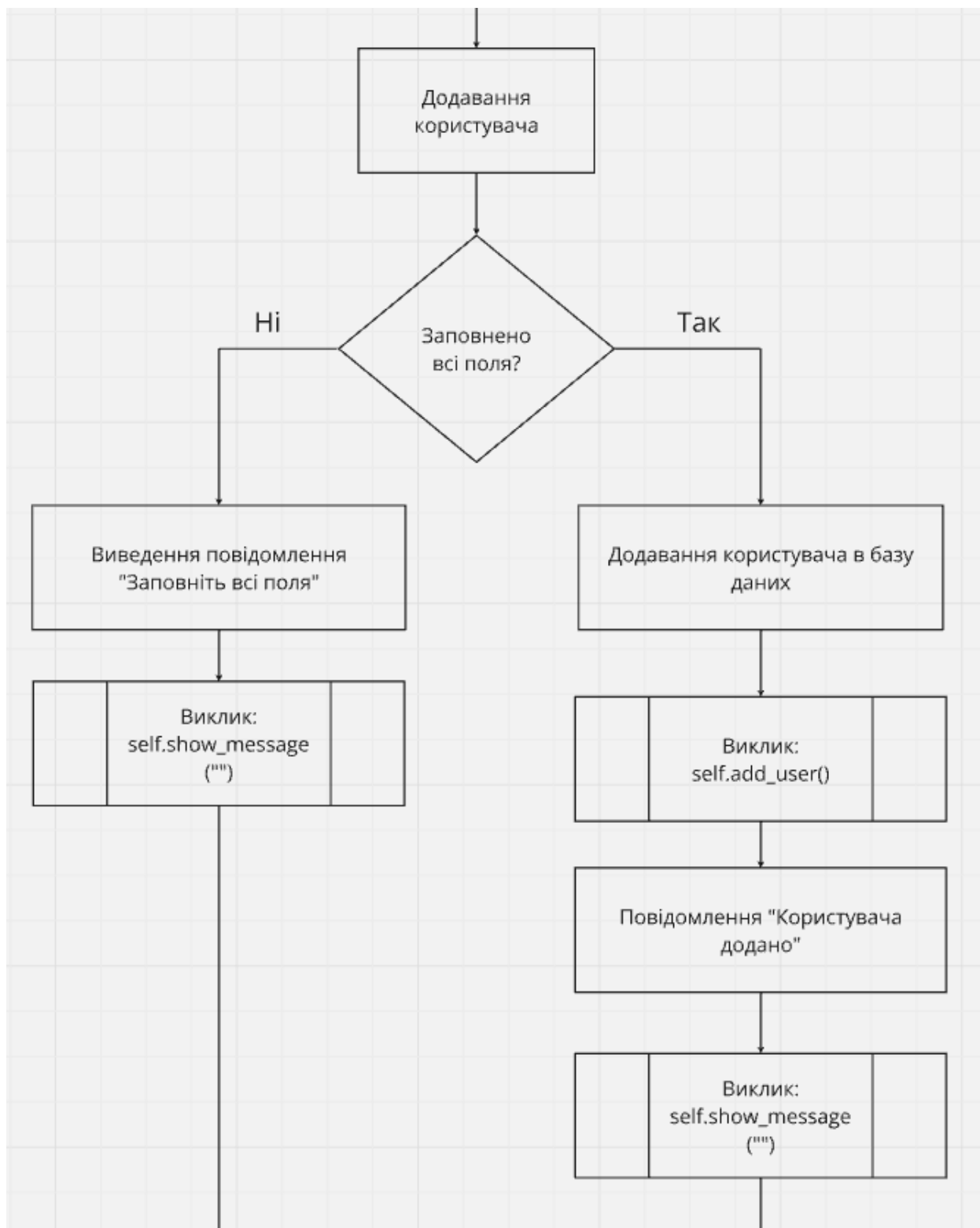


Рис.3.2.3 – Алгоритм роботи завдання “Додавання користувача”

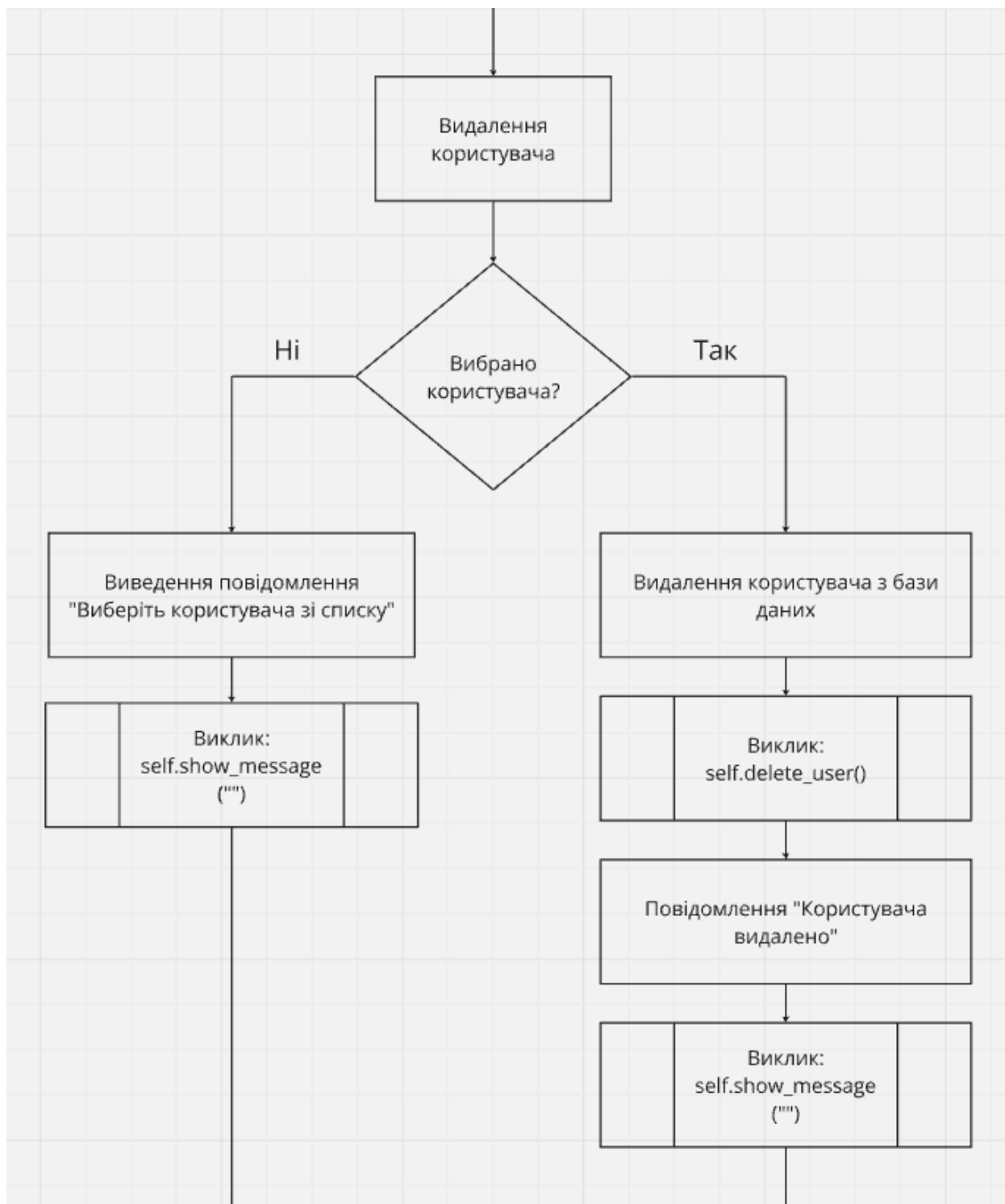


Рис.3.2.4 – Алгоритм роботи завдання “Видалення користувача”

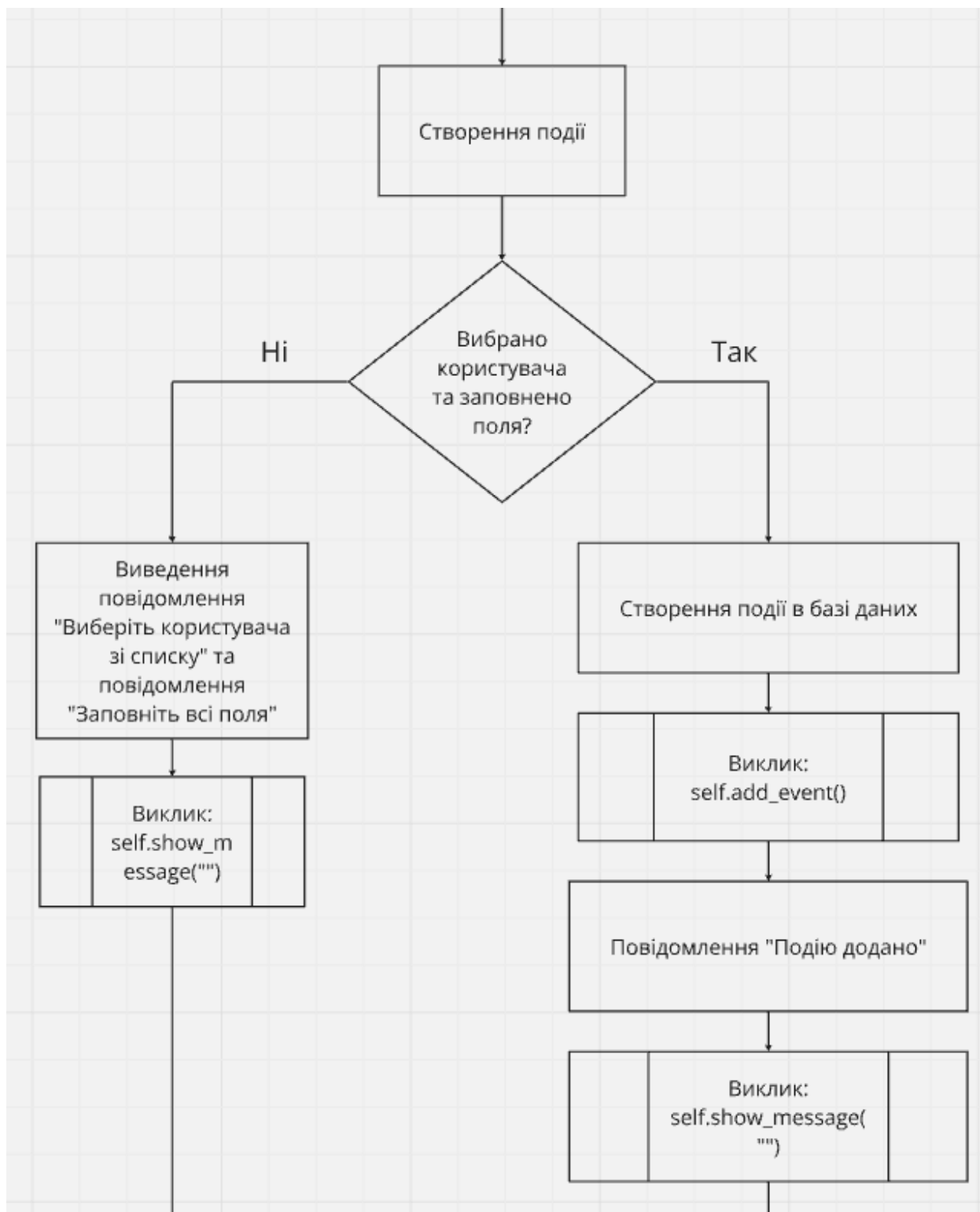


Рис.3.2.5 – Алгоритм роботи завдання “Створення події”

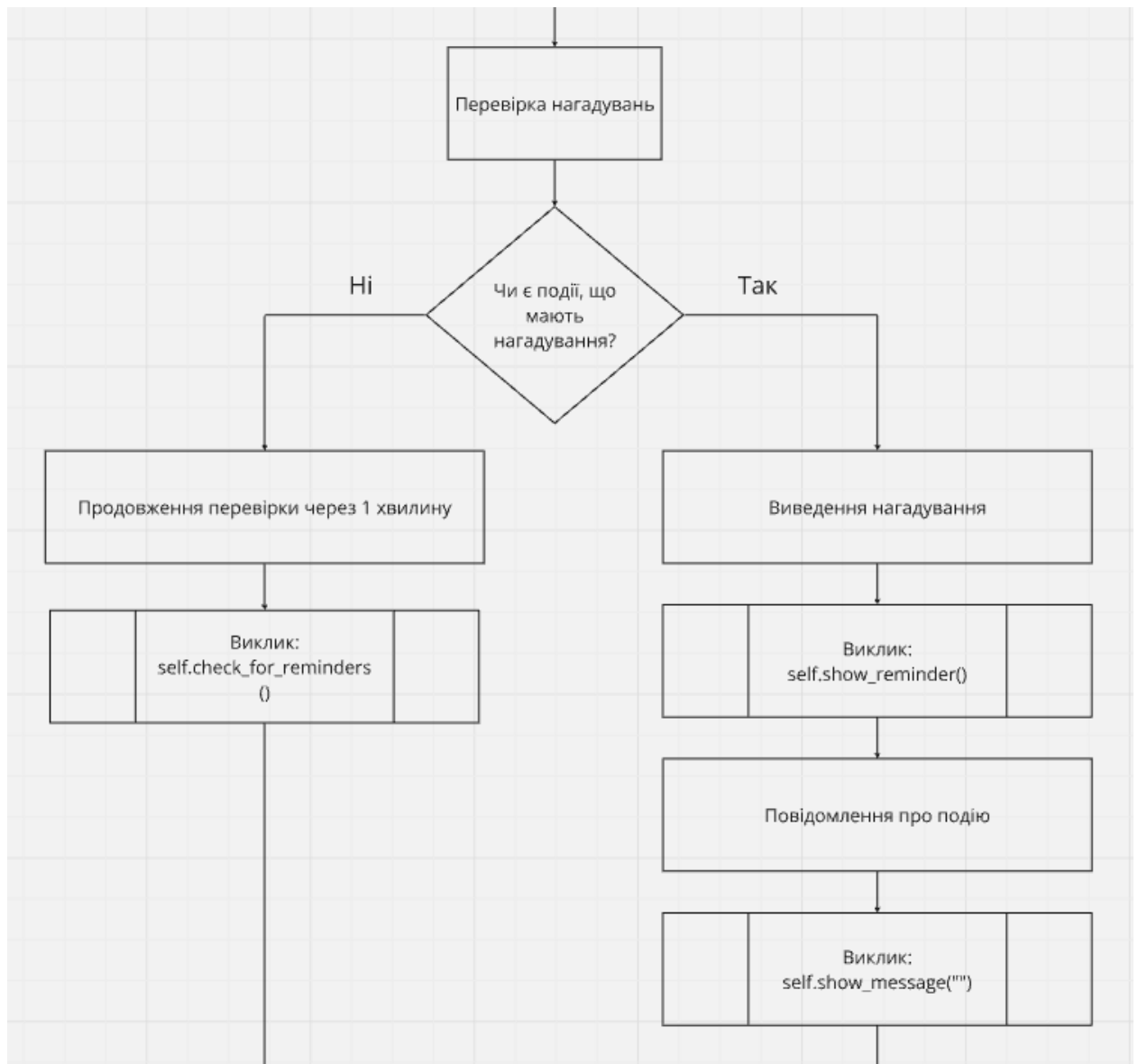


Рис.3.2.6 – Алгоритм роботи завдання “Перевірка нагадувань”

Проектування алгоритму є важливим етапом розробки програмного забезпечення, що дозволяє структурувати логіку роботи та оптимізувати процеси. Запропонована схема забезпечує гнучкість, надійність та зручність у використанні, а також гарантує ефективне інформування користувачів через різні канали зв’язку.

## 4. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка програмного забезпечення для інформування користувачів включає кілька основних етапів: створення бази даних, реалізацію основних модулів та розробку графічного інтерфейсу.

### 4.1 Створення бази даних

База даних є ключовим компонентом програмного додатку, оскільки вона відповідає за збереження інформації про користувачів, повідомлення та події. Для реалізації роботи з базою даних у даному проекті використовується SQL Server, а взаємодія з нею здійснюється за допомогою бібліотеки pyodbc.

Бібліотека pyodbc використовується для підключення Python-додатку до SQL Server та виконання SQL-запитів.

Клас DatabaseManager відповідає за підключення до бази даних, створення необхідних таблиць та виконання SQL-запитів.

Даний клас інкапсулює основні операції роботи з базою даних, що забезпечує зручність її використання в програмному коді.

### 4.2 Метод з'єднання з базою даних

Призначення методу `__init__`:

- Встановлює з'єднання з SQL Server через ODBC Driver 17.
- Підключається до бази даних EventNotifierApp.
- Використовує захищене підключення (Trusted Connection).
- Якщо підключення успішне, створюється об'єкт cursor, який дозволяє виконувати SQL-запити.
- Викликає метод `create_tables()`, який перевіряє наявність необхідних таблиць та створює їх у разі потреби.
- Якщо підключення не вдалося, програма генерує виняток і повідомляє про помилку.

```

database.py > DatabaseManager > create_tables
1  import pyodbc
2
3  class DatabaseManager:
4
5      def __init__(self):
6          try:
7              self.conn = pyodbc.connect(
8                  'DRIVER={ODBC Driver 17 for SQL Server};'
9                  'SERVER=DESKTOP-;'
10                 'DATABASE=EventNotifierApp;'
11                 'Trusted_Connection=yes;'
12                 'Encrypt=yes;'
13                 'TrustServerCertificate=yes;'
14             )
15             self.cursor = self.conn.cursor()
16             self.create_tables()
17         except pyodbc.Error as e:
18             raise Exception(f"Помилка підключення до бази даних: {str(e)}")

```

Рис.4.2.1 – Метод \_\_init\_\_

### 4.3 Метод перевірки таблиць

Призначення методу create\_tables:

- Метод перевіряє, чи існують таблиці users, messages, events, і створює їх, якщо вони відсутні.
- Використовується SQL-запит IF NOT EXISTS, щоб уникнути дублювання таблиць.
- users – містить дані про користувачів (ім'я, email, телефон, спосіб зв'язку).
- messages – зберігає повідомлення для користувачів. Має зовнішній ключ user\_id, який посилається на користувача у таблиці users.
- events – містить записи про події, збережені користувачами. Також пов'язана з users через зовнішній ключ user\_id.
- Усі запити виконуються у циклі, щоб кожна таблиця перевірялася та створювалася окремо.
- Викликається commit(), щоб зберегти зміни у базі.

```

20 def create_tables(self):
21     queries = [
22         '''IF NOT EXISTS (SELECT * FROM sysobjects WHERE name='users' AND xtype='U')
23         CREATE TABLE users (
24             id INT IDENTITY(1,1) PRIMARY KEY,
25             name NVARCHAR(100),
26             email NVARCHAR(100),
27             phone NVARCHAR(20),
28             contact_method NVARCHAR(20)
29         )''',
30         '''IF NOT EXISTS (SELECT * FROM sysobjects WHERE name='messages' AND xtype='U')
31         CREATE TABLE messages (
32             id INT IDENTITY(1,1) PRIMARY KEY,
33             user_id INT,
34             message NVARCHAR(1000),
35             FOREIGN KEY (user_id) REFERENCES users(id)
36         )''',
37         '''IF NOT EXISTS (SELECT * FROM sysobjects WHERE name='events' AND xtype='U')
38         CREATE TABLE events (
39             id INT IDENTITY(1,1) PRIMARY KEY,
40             user_id INT,
41             event_date DATETIME,
42             event_description NVARCHAR(1000),
43             FOREIGN KEY (user_id) REFERENCES users(id)
44         )'''
45     ]
46     for query in queries:
47         self.cursor.execute(query)
48     self.conn.commit()

```

Рис.4.3.1 – Метод create\_tables

#### 4.4 Метод виконання SQL-запитів

Призначення методу execute\_query:

- Загальний метод для виконання SQL-запитів.
- query – SQL-запит, який потрібно виконати.
- params – параметри, які передаються у запит.
- fetch=False – якщо встановлено True, метод поверне результат виконання запиту (fetchall()).
- Використовується контекстний менеджер with, який автоматично закриває курсор після виконання операції.



```

50     def execute_query(self, query, params=(), fetch=False):
51         """Выполнить SQL-запрос."""
52         with self.conn.cursor() as cursor:
53             cursor.execute(query, params)
54             if fetch:
55                 return cursor.fetchall()
56             self.conn.commit()
57

```

Рис.4.4.1 – Метод execute\_query

Розроблений модуль для роботи з базою даних забезпечує:

1. Безпечне підключення до SQL Server.
2. Автоматичне створення таблиць, якщо вони відсутні.
3. Гнучке виконання SQL-запитів через метод execute\_query.
4. Інтеграцію з іншими модулями додатку для роботи з користувачами, повідомленнями та подіями.

Ця реалізація гарантує надійність та ефективність збереження та обробки даних у програмному додатку для інформування користувачів.

#### 4.5 Розробка графічного інтерфейсу користувача та взаємодія з базою даних

1. Створення інтерфейсу програми: розробив візуальну частину програми, використовуючи бібліотеку Tkinter. Це включає:

- Головне вікно з кнопками для різних дій (наприклад, додавання, видалення користувачів, відправка повідомлень).
- Використання віджетів Tkinter (кнопки, мітки, текстові поля) для створення інтерфейсу, де користувач може взаємодіяти з програмою.
- Вбудований календар (за допомогою tkcalendar) для вибору дати події.

2. Завантаження та управління користувачами: налаштував механізм для завантаження даних користувачів з бази даних і відображення їх в інтерфейсі. Користувачі можуть бути додані або видалені через інтерфейс програми.

- Для цього використовувався Listbox для відображення списку користувачів.

- Для роботи з даними була використана база даних SQL Server, до якої ми підключалися через pyodbc.

3. Відправка повідомлень користувачеві: в програмі реалізована можливість відправки повідомлень конкретним користувачам через графічний інтерфейс:

- Користувач вибирає зі списку потрібного отримувача і вводить текст повідомлення.

- Повідомлення надсилається в базу даних і зберігається для обраного користувача.

4. Створення та управління подіями: реалізував функціонал для створення подій, які прив'язуються до користувачів:

- Користувач вибирає дату з календаря і створює подію з назвою, часом та описом.

- Ці події зберігаються в базі даних і можуть бути використані для подальшого інформування користувачів.

5. Потокова перевірка нагадувань: в окремому потоці реалізована перевірка подій на їх наближення (в межах 15 хвилин). Коли настає час події, користувачу надсилається нагадування.

- Це нагадування виводиться в консоль і відображається в графічному інтерфейсі як спливаюче вікно.

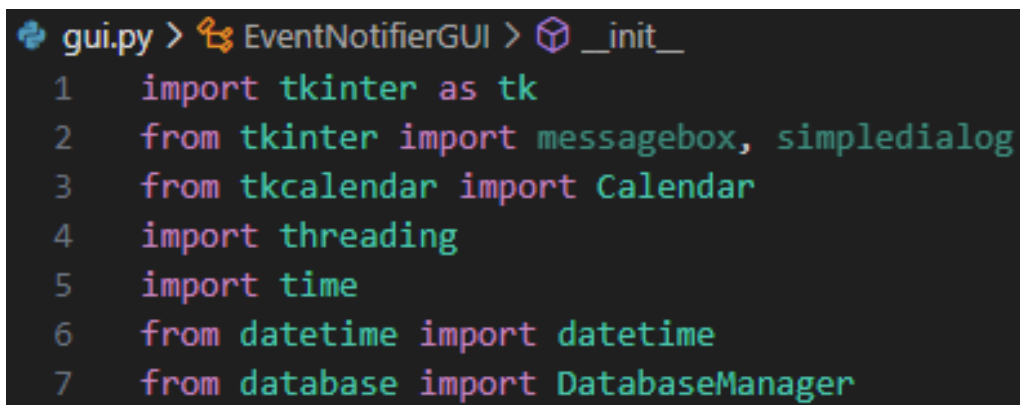
6. Використання багатозадачності: для того, щоб програма могла працювати з нагадуваннями без блокування інтерфейсу, використовував багатозадачність за допомогою бібліотеки threading, що дозволяє перевіряти нагадування у фоновому режимі, не заважаючи роботі користувача з інтерфейсом.

#### 4.6 Імпорти та клас

Імпорти та клас:

					5.151.1.40-КП	Лист
						26
Изм.	Лист	№ докум.	Підпис	Дата		

1. tkinter — бібліотека для створення графічного інтерфейсу користувача (GUI) в Python.
  - tk — стандартний модуль для роботи з GUI.
  - messagebox — використовують для відображення стандартних діалогових вікон.
  - simpledialog — для введення даних через прості діалогові вікна.
2. tkcalendar — використовується для створення календаря в інтерфейсі для вибору дати події.
3. threading — використовується для створення фонових потоків, щоб виконувати перевірку нагадувань у фоновому режимі.
4. time — використовується для встановлення затримок між перевірками подій (кожну хвилину).
5. datetime — для роботи з датами та часом, перевірка та обробка подій.
6. DatabaseManager — клас, який відповідає за взаємодію з базою даних (підключення, запити тощо).



```
gui.py > EventNotifierGUI > __init__
1  import tkinter as tk
2  from tkinter import messagebox, simpledialog
3  from tkcalendar import Calendar
4  import threading
5  import time
6  from datetime import datetime
7  from database import DatabaseManager
```

Рис.4.6.1 – Імпорти та клас

#### 4.7 Клас EventNotifierGUI

Клас EventNotifierGUI – цей клас визначає GUI для програми і реалізує основні функціональні можливості.

`__init__` — конструктор класу:

- root — це основне вікно програми, створене через `tk.Tk()`.
- db\_manager — об'єкт для роботи з базою даних.

- Налаштування заголовку вікна та розміру.
- Виклик методів `create_gui` для створення інтерфейсу та `load_users` для завантаження даних про користувачів із бази.
- Створення потоку для перевірки нагадувань, який працює у фоновому режимі та викликає метод `check_reminders`.

```

9  class EventNotifierGUI:
10     """Основний клас інтерфейсу програми."""
11
12     def __init__(self, root, db_manager):
13         self.root = root
14         self.db = db_manager
15         self.root.title("Система інформування користувачів")
16         self.root.geometry("850x850")
17         self.root.configure(bg="#e6f7e6")
18
19         self.create_gui()
20         self.load_users()
21
22         reminder_thread = threading.Thread(target=self.check_reminders)
23         reminder_thread.daemon = True
24         reminder_thread.start()

```

Рис.4.7.1 – Ініціалізація та налаштування інтерфейсу

#### 4.8 Метод всіх елементів інтерфейсу

`create_gui` — метод для створення всіх елементів інтерфейсу:

- Використовуються віджети, такі як `Label`, `Listbox`, `Text`, `Button` для різних компонентів (список користувачів, введення повідомлення, кнопки для додавання/видалення користувачів, створення подій).
- Календар, на якому користувач може вибрати дату події.
- Кожна кнопка викликає відповідний метод для додавання користувача, видалення користувача, або створення події.

```

26 def create_gui(self):
27     """Створення графічного інтерфейсу."""
28     tk.Label(self.root, text="Повідомлення для користувачів",
29             font=("Arial", 16), bg="#e6f7e6", fg="#333").pack(pady=10)
30
31     self.users_listbox = tk.Listbox(self.root, height=6, width=45, font=("Arial", 12))
32     self.users_listbox.pack(pady=10)
33
34     tk.Label(self.root, text="Введіть повідомлення", bg="#e6f7e6", fg="#333").pack()
35     self.message_entry = tk.Text(self.root, font=("Arial", 12), width=45, height=10)
36     self.message_entry.pack(pady=10)
37
38     tk.Button(self.root, text="Відправити повідомлення", font=("Arial", 12),
39             bg="#4CAF50", fg="#fff", command=self.send_message).pack(pady=20)
40
41     buttons_frame = tk.Frame(self.root, bg="#e6f7e6")
42     buttons_frame.pack(pady=20)
43
44     tk.Button(buttons_frame, text="Додати користувача", font=("Arial", 12),
45             bg="#2196F3", fg="#fff", command=self.add_user).pack(side=tk.LEFT, padx=10)
46
47     tk.Button(buttons_frame, text="Видалити користувача", font=("Arial", 12),
48             bg="#F44336", fg="#fff", command=self.delete_user).pack(side=tk.LEFT, padx=10)
49
50     # Кнопка для створення події
51     tk.Button(self.root, text="Створити подію", font=("Arial", 12),
52             bg="#FFC107", fg="#fff", command=self.create_event).pack(pady=20)

```

Рис.4.8.1 – Створення графічного інтерфейсу

## 4.9 Метод завантаження даних користувачів

load\_users — метод для завантаження даних користувачів з бази даних:

- Очистка списку користувачів в GUI.
- Виконується запит до бази даних для отримання користувачів.
- Дані кожного користувача виводяться в форматі: "Ім'я (метод зв'язку: email/телефон)".
- Збереження даних про користувачів у словнику self.user\_data для подальшого використання (ідентифікація користувачів).

```

59 def load_users(self):
60     """Завантаження списку користувачів."""
61     self.users_listbox.delete(0, tk.END)
62     users = self.db.execute_query("SELECT id, name, contact_method, email, phone FROM users", fetch=True)
63     self.user_data = {} # Зберігаємо ID користувачів у словник
64     for user in users:
65         user_id, name, method, email, phone = user
66         contact_info = email if method == "email" else phone
67         display_text = f"{name} ({method}: {contact_info})"
68         self.users_listbox.insert(tk.END, display_text)
69         self.user_data[display_text] = user_id

```

Рис.4.9.1 – Завантаження користувачів із бази даних

## 4.10 Метод відправки повідомлень

send\_message — метод для відправки повідомлення обраному користувачу:

- Перевіряється, чи користувач вибраний в списку.
- Береться текст повідомлення з текстового поля.
- Якщо повідомлення не введено, виводиться повідомлення про помилку.
- Відправка повідомлення до бази даних для збереження.

```
71 def send_message(self):
72     """Відправка повідомлення вибраному користувачу."""
73     selected = self.users_listbox.curselection()
74     if not selected:
75         self.show_custom_message("Помилка", "Виберіть користувача зі списку.", "red")
76         return
77
78     user_text = self.users_listbox.get(selected[0])
79     user_id = self.user_data[user_text]
80
81     message = self.message_entry.get("1.0", tk.END).strip()
82     if not message:
83         self.show_custom_message("Помилка", "Введіть повідомлення.", "red")
84         return
85
86     self.db.execute_query("INSERT INTO messages (user_id, message) VALUES (?, ?)", (user_id, message))
87     self.show_custom_message("Готово", "Повідомлення відправлено.", "green")
88     self.message_entry.delete("1.0", tk.END)
```

Рис.4.10.1 – Відправка повідомлення

#### 4.11 Методи додавання та видалення користувачів

##### Додавання та видалення користувачів

Методи add\_user і delete\_user відповідають за додавання та видалення користувачів з бази даних через відповідні вікна для введення даних або вибору користувача для видалення.

```

90 def add_user(self):
91     """Додавання користувача."""
92     def save_user():
93         name, email, phone = name_entry.get(), email_entry.get(), phone_entry.get()
94         contact_method = contact_var.get()
95
96         if not name or (contact_method == "email" and not email) or (contact_method == "phone" and not phone):
97             self.show_custom_message("Помилка", "Заповніть всі поля.", "red")
98             return
99
100         self.db.execute_query("INSERT INTO users (name, email, phone, contact_method) VALUES (?, ?, ?, ?)",
101                                (name, email if contact_method == "email" else None,
102                                 phone if contact_method == "phone" else None, contact_method))
103         self.show_custom_message("Готово", "Користувача додано.", "green")
104         new_user_window.destroy()
105         self.load_users()
106
107     new_user_window = tk.Toplevel(self.root)
108     new_user_window.title("Додати користувача")
109     new_user_window.geometry("400x250")
110     new_user_window.configure(bg="#dff0d8")
111
112     tk.Label(new_user_window, text="Ім'я", bg="#dff0d8").pack()
113     name_entry = tk.Entry(new_user_window, width=25, font=("Arial", 12))
114     name_entry.pack()
115
116     tk.Label(new_user_window, text="Електронна пошта", bg="#dff0d8").pack()
117     email_entry = tk.Entry(new_user_window, width=25, font=("Arial", 12))
118     email_entry.pack()
119
120     tk.Label(new_user_window, text="Номер телефону", bg="#dff0d8").pack()
121     phone_entry = tk.Entry(new_user_window, width=25, font=("Arial", 12))
122     phone_entry.pack()
123
124     contact_var = tk.StringVar(value="email")
125     tk.Radiobutton(new_user_window, text="Email", variable=contact_var, value="email", bg="#dff0d8").pack()
126     tk.Radiobutton(new_user_window, text="Телефон", variable=contact_var, value="phone", bg="#dff0d8").pack()
127
128     tk.Button(new_user_window, text="Зберегти", font=("Arial", 12), bg="#4CAF50", fg="white",
129               command=save_user).pack(pady=10)

```

Рис.4.11.1 – Додавання користувача

```

132 def delete_user(self):
133     """Видалення користувача."""
134     selected = self.users_listbox.curselection()
135     if not selected:
136         self.show_custom_message("Помилка", "Виберіть користувача зі списку.", "red")
137         return
138
139     user_text = self.users_listbox.get(selected[0])
140     user_id = self.user_data[user_text]
141
142     self.db.execute_query("DELETE FROM users WHERE id=?", (user_id,))
143     self.show_custom_message("Готово", "Користувача видалено.", "green")
144     self.load_users()

```

Рис.4.11.2 – Видалення користувача

## 4.12 Метод створення, впливаючого вікна та збереження події

create\_event — метод для створення події:

- Перевірка на вибір користувача.
- Вибір дати з календаря.
- Перевірка правильності формату дати.

```

146 def create_event(self):
147     """Створення події для користувача на вибрану дату з описом."""
148     selected = self.users_listbox.curselection()
149     if not selected:
150         self.show_custom_message("Помилка", "Виберіть користувача зі списку.", "red")
151         return
152
153     user_text = self.users_listbox.get(selected[0])
154     user_id = self.user_data[user_text]
155
156     # Вибір дати події з календаря
157     event_date_str = self.calendar.get_date()
158     try:
159         event_date = datetime.strptime(event_date_str, "%Y-%m-%d")
160     except ValueError:
161         self.show_custom_message("Помилка", "Невірний формат дати.", "red")
162         return

```

Рис.4.12.1 – Створення події

```

164 # Створення нового вікна для введення назви події та часу
165 event_window = tk.Toplevel(self.root)
166 event_window.title("Створення події")
167 event_window.geometry("500x425")
168 event_window.configure(bg="#dfff0d8")
169
170 tk.Label(event_window, text="Назва події:", font=("Arial", 12), bg="#dfff0d8").pack(pady=10)
171 event_name_entry = tk.Entry(event_window, font=("Arial", 12), width=40)
172 event_name_entry.pack(pady=10)
173
174 tk.Label(event_window, text="Час події:", font=("Arial", 12), bg="#dfff0d8").pack(pady=10)
175 time_entry = tk.Entry(event_window, font=("Arial", 12), width=40)
176 time_entry.pack(pady=10)
177
178 tk.Label(event_window, text="Введіть опис події:", font=("Arial", 12), bg="#dfff0d8").pack(pady=10)
179 event_text = tk.Text(event_window, font=("Arial", 12), width=40, height=6)
180 event_text.pack(pady=10)

```

Рис.4.12.2 – Створення вікна для введення події, часу та опису події

```

182 def save_event():
183     event_name = event_name_entry.get().strip()
184     event_time = time_entry.get().strip()
185     event_description = event_text.get("1.0", tk.END).strip()
186
187     if not event_name or not event_time or not event_description:
188         self.show_custom_message("Помилка", "Заповніть всі поля.", "red")
189         return
190
191     event_datetime_str = f"{event_date_str} {event_time}"
192     try:
193         event_datetime = datetime.strptime(event_datetime_str, "%Y-%m-%d %H:%M")
194     except ValueError:
195         self.show_custom_message("Помилка", "Невірний формат часу.", "red")
196         return
197
198     self.db.execute_query("INSERT INTO events (user_id, event_date, event_name, event_description) VALUES (?, ?, ?, ?)",
199                          (user_id, event_datetime, event_name, event_description))
200     self.show_custom_message("Готово", "Подію додано.", "green")
201     event_window.destroy()
202
203     tk.Button(event_window, text="Зберегти", font=("Arial", 12), bg="#4CAF50", fg="white",
204              command=save_event).pack(pady=10)

```

Рис.4.12.3 – Збереження вікна для створення події

## 4.13 Метод перевірки подій

					5.151.1.40-КП	Лист
Изм.	Лист	№ докум.	Підпис	Дата		32



check\_reminders — метод, який перевіряє події на наближення до них (15 хвилин до події).

- Для кожної події перевіряється, чи вона відбудеться найближчим часом.
- Якщо так, викликається метод send\_reminder для відправки нагадування.

```
207 def check_reminders(self):
208     """Перевірка нагадувань та виведення їх у консоль."""
209     while True:
210         # Отримуємо всі події
211         events = self.db.execute_query("SELECT id, user_id, event_date, event_description FROM events", fetch=True)
212         for event in events:
213             event_id, user_id, event_date, event_description = event
214             # Якщо подія сьогодні або в найближчі 15 хвилин
215             if (event_date - datetime.now()).total_seconds() <= 900 and (event_date - datetime.now()).total_seconds() > 0:
216                 # Відправляємо нагадування
217                 self.send_reminder(user_id, event_description)
218             time.sleep(60) # Перевірка раз на хвилину
```

Рис.4.13.1 – Метод check\_reminders

#### 4.14 Метод відправки нагадувань та показу візуальних повідомлень

send\_reminder — метод для відправки нагадувань користувачу через консоль і повідомлення в інтерфейсі.

```
220 def send_reminder(self, user_id, event_description):
221     """Відправка нагадування користувачу."""
222     user = self.db.execute_query("SELECT name, contact_method, email, phone FROM users WHERE id=?", (user_id,), fetch=True)[0]
223     name, contact_method, email, phone = user
224     message = f"Нагадування для {name}: {event_description}"
225
226     # Виводимо нагадування в консоль
227     print(f"Нагадування: {message}")
228
229     # Для подальшої реалізації: можна додати відображення нагадування в інтерфейсі
230     self.show_custom_message("Нагадування", message, "blue")
```

Рис.4.14.1 – Метод send\_reminder

Метод show\_custom\_message: це метод класу, який відповідає за показ візуального повідомлення у новому вікні.

```

232 def show_custom_message(self, title, message, color):
233     """Виведення кастомного повідомлення в кольоровому вікні."""
234     msg_box = tk.Toplevel(self.root)
235     msg_box.title(title)
236     msg_box.geometry("400x200")
237     msg_box.configure(bg=color)
238
239     label = tk.Label(msg_box, text=message, font=("Arial", 14), bg=color, fg="white", wraplength=350)
240     label.pack(expand=True, fill=tk.BOTH, pady=20)
241
242     button = tk.Button(msg_box, text="Закрити", font=("Arial", 12), bg="gray", fg="white", command=msg_box.destroy)
243     button.pack(pady=10)

```

Рис.4.14.2 – Метод show\_custom\_message

```

245 # Створення екземпляра GUI та підключення до бази даних
246 if __name__ == "__main__":
247     root = tk.Tk()
248     db_manager = DatabaseManager() # Ініціалізуємо об'єкт для роботи з базою даних
249     app = EventNotifierGUI(root, db_manager)
250     root.mainloop()

```

Рис.4.14.3 – Екземпляр та підключення до бази даних

Код реалізує систему інформування користувачів через графічний інтерфейс, дозволяючи додавати користувачів, відправляти повідомлення, створювати події та отримувати нагадування. Основна логіка реалізована через методи класу EventNotifierGUI, які взаємодіють з базою даних через об'єкт db\_manager.

#### 4.15 Створення та запуск основної програми з підключенням до бази даних

Здійснюється ініціалізація та запуск основної програми з використанням графічного інтерфейсу та підключенням до бази даних.

```

user_manager.py > ...
1  from database import DatabaseManager
2  from gui import EventNotifierGUI
3  import tkinter as tk
4
5  def main():
6      db_manager = DatabaseManager()
7      root = tk.Tk()
8      app = EventNotifierGUI(root, db_manager)
9      root.mainloop()
10
11 if __name__ == "__main__":
12     main()

```

Рис.4.15.1 – Графічний інтерфейс введення подій та вибору подій

### 1. Імпорт необхідних модулів:

- Спочатку імпортуються два важливих компоненти:
  - `DatabaseManager` з модуля `database`, який буде відповідати за роботу з базою даних.
  - `EventNotifierGUI` з модуля `gui`, який реалізує графічний інтерфейс програми.
- Також імпортується модуль `tkinter`, який використовується для створення графічного інтерфейсу.

### 2. Функція `main()`:

- В функції `main()` створюється об'єкт `db_manager` класу `DatabaseManager`, який відповідатиме за взаємодію з базою даних.
- Далі ініціалізується головне вікно програми через `tk.Tk()`, яке є основним елементом графічного інтерфейсу.
- Створюється об'єкт `app` класу `EventNotifierGUI`, передаючи йому в якості параметрів головне вікно `root` та об'єкт для роботи з базою даних `db_manager`.
- Викликається метод `root.mainloop()`, який запускає цикл подій для обробки взаємодії користувача з графічним інтерфейсом.

### 3. Перевірка запуску програми:

- Останній блок коду `if __name__ == "__main__":` перевіряє, чи є цей файл основним при виконанні програми. Якщо так, то запускається функція `main()`, що забезпечує ініціалізацію і запуск додатка.

## 5. КЕРІВНИЦТВО КОРИСТУВАЧА

Процес використання програми для інформування користувачів про події. Програма складається з графічного інтерфейсу, що надає можливість користувачам взаємодіяти з додатком для перегляду, додавання та керування подіями. Також передбачена можливість відправлення повідомлень користувачам через систему сповіщень.

### 1. Запуск програми:

- Для запуску програми користувач повинен запустити файл програми `user_manager.py`, де ініціалізується графічний інтерфейс і здійснюється підключення до бази даних. Після цього відкриється головне вікно програми.

### 2. Головне вікно:

Головне вікно містить наступні елементи:

- Список користувачів: тут відображається список всіх користувачів, для яких можна надіслати повідомлення.
- Канали зв'язку: у цьому розділі можна вибрати канал зв'язку для сповіщення користувачів при додаванні користувача (наприклад, через email або інші засоби зв'язку).
- Кнопка для відправлення повідомлень: користувач може натискати на кнопку для відправлення повідомлень обраним користувачам.

Система інформування користувачів

### Повідомлення для користувачів

Введіть повідомлення

Відправити повідомлення

Додати користувача    Видалити користувача

Створити подію

Виберіть дату події

Березня		2025					
пн	вт	ср	чт	пт	сб	нд	
9	24	25	26	27	28	1	2
10	3	4	5	6	7	8	9
11	10	11	12	13	14	15	16
12	17	18	19	20	21	22	23
13	24	25	26	27	28	29	30
14	31	1	2	3	4	5	6

Рис.5.1 – Інтерфейс відкритого застосунку

Додати користувача

Ім'я

Федулов Ілля

Електронна пошта

fedylovillya@gmail.com

Номер телефону

☒ Email

☐ Телефон

Зберегти

Рис.5.2 – Додавання нового користувача та вибір зв'язку з користувачем

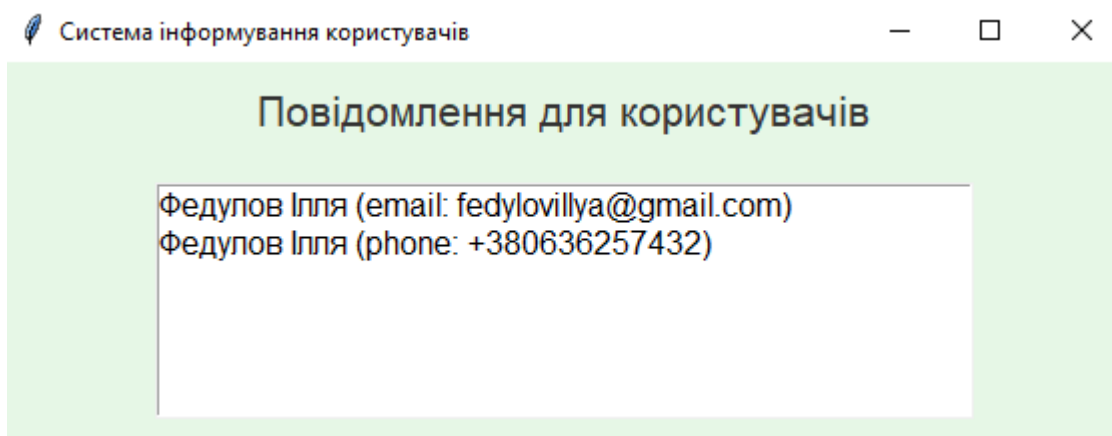


Рис.5.3 – Вивід створених користувачів

### 3. Додавання нових подій:

- Користувач може додавати нові події за допомогою інтерфейсу, заповнивши відповідні поля (наприклад, дата, час, опис події).
- Після заповнення форми для додавання події, інформація зберігається в базі даних і стає доступною для перегляду та сповіщення користувачів.

Рис.5.4 – Створення нової події

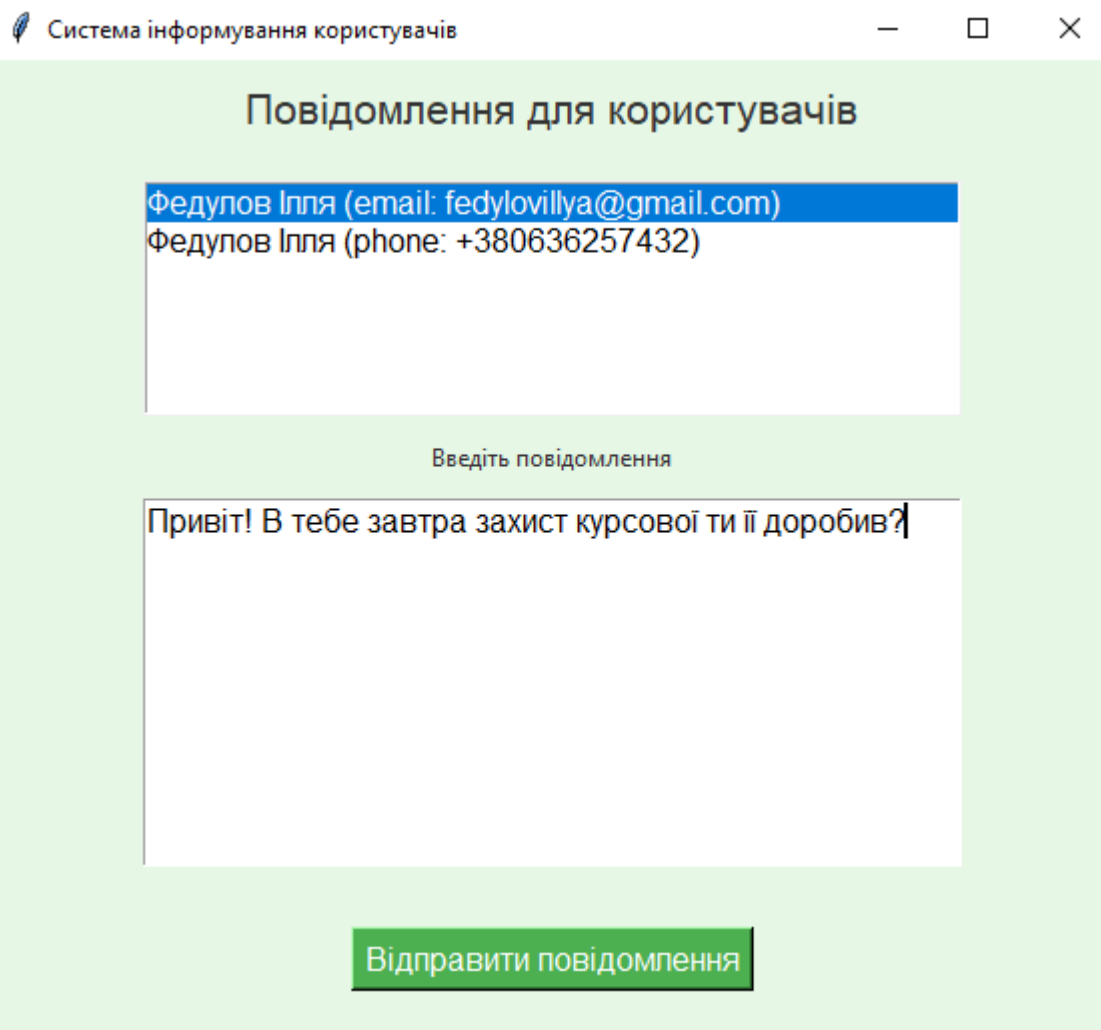


Рис.5.5 – Повідомлення до користувача

#### 4. Виведення повідомлень:

- Програма надає можливість відправлення кастомізованих повідомлень через спливаючі вікна, що з'являються на екрані користувача. Повідомлення можуть бути різних кольорів, щоб привертати увагу користувачів.
- У кожному вікні є кнопка для закриття сповіщення.

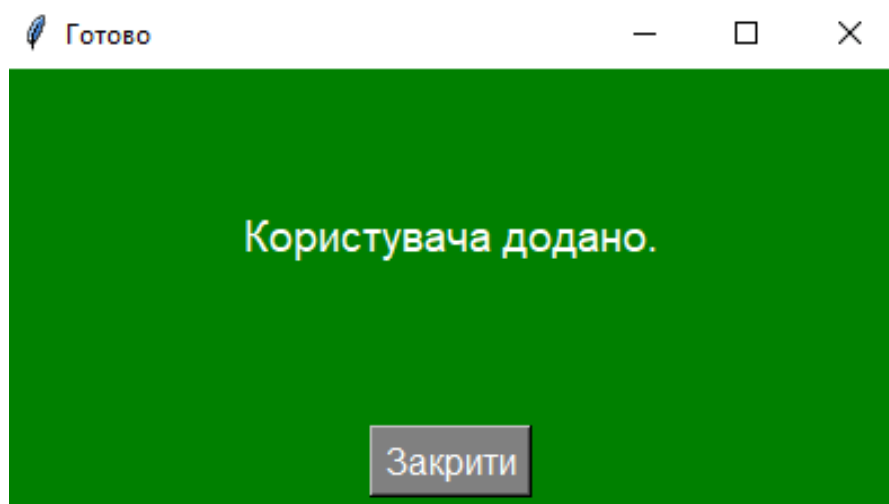


Рис.5.6 – Повідомлення про додавання користувача

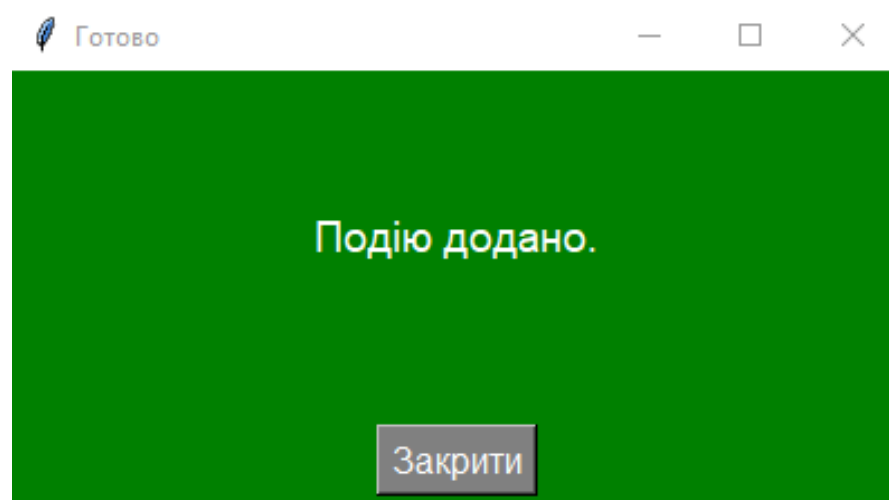


Рис.5.7 – Повідомлення про додавання події

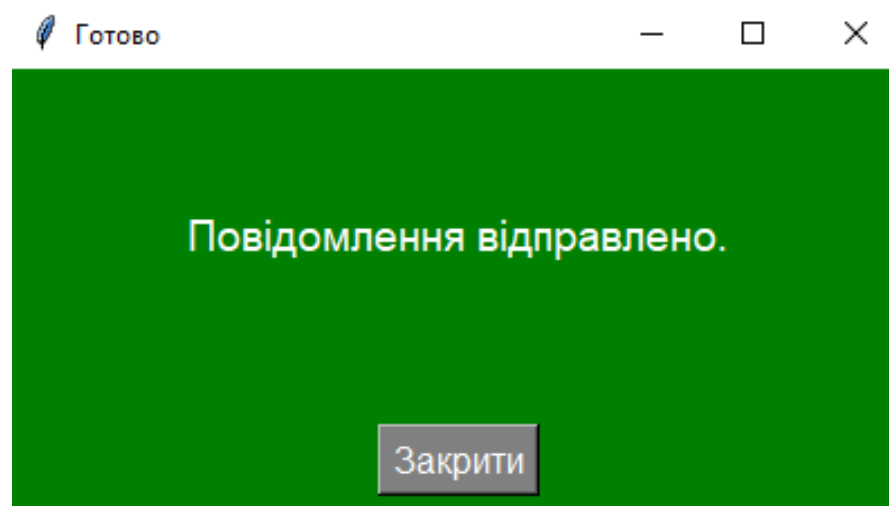


Рис.5.8 – Повідомлення про відправлене повідомлення



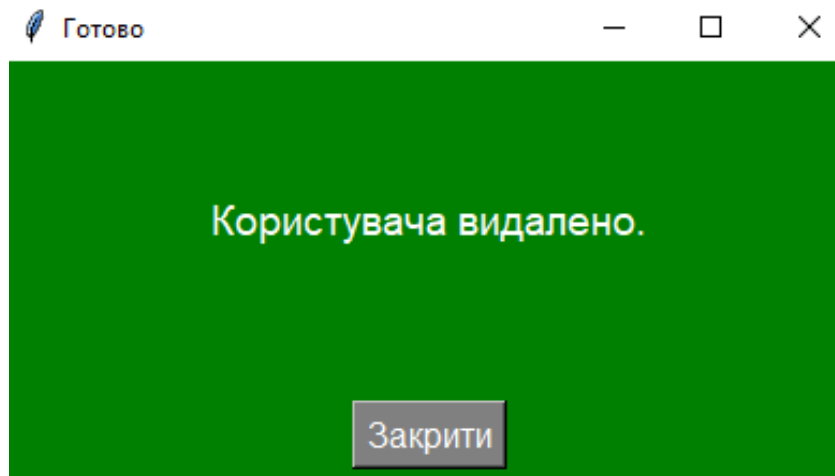


Рис.5.9 – Повідомлення про видалення користувача

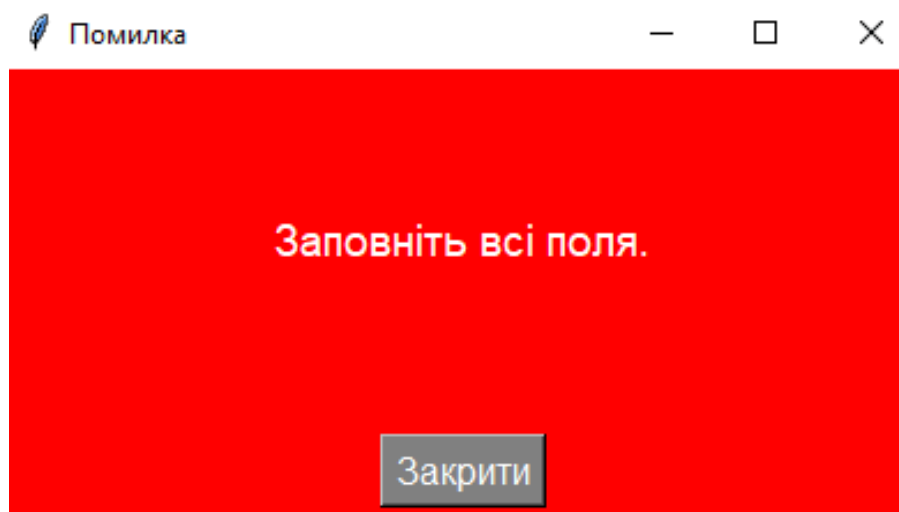


Рис.5.10 – Повідомлення про необхідність заповнити всі поля

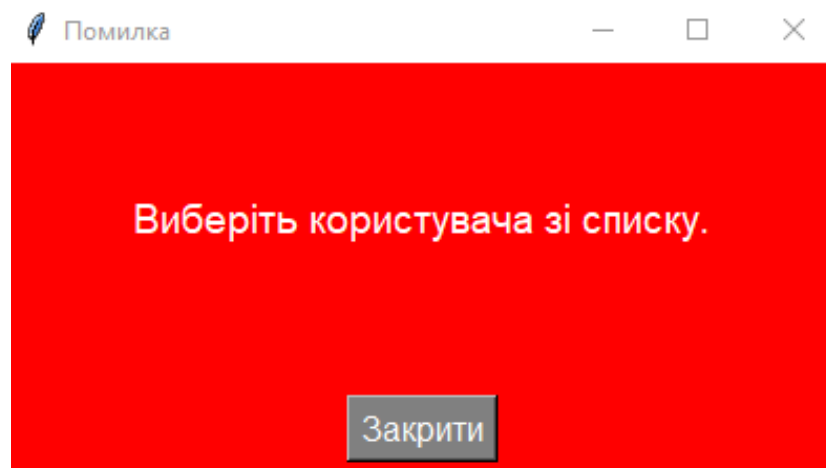


Рис.5.11 – Повідомлення про необхідність обрати користувача зі списку

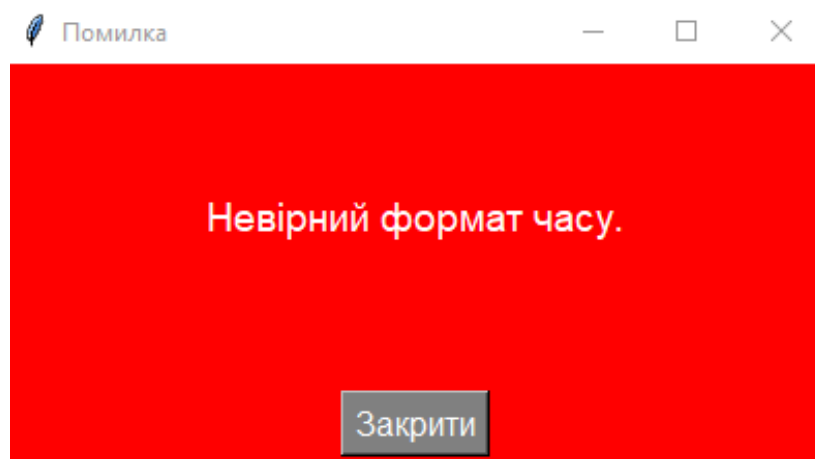


Рис.5.12 – Повідомлення про необхідність правильно написати формат часу

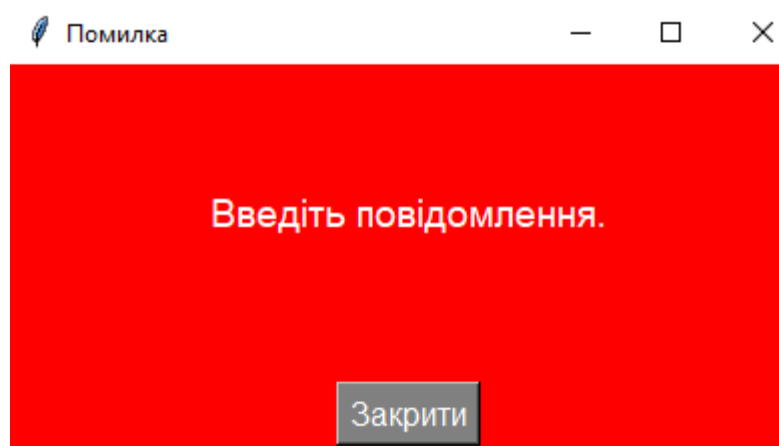


Рис.5.13 – Повідомлення про необхідність ввести повідомлення

## 5. Перегляд інформації про події:

- Користувач може переглядати список подій, що зберігаються в базі даних.
- Інтерфейс дозволяє фільтрувати події за різними критеріями (наприклад, по даті або типу події).

SELECT TOP (1000) [id]  
 ,[user\_id]  
 ,[event\_name]  
 ,[event\_date]  
 ,[event\_description]  
 FROM [EventNotifierApp].[dbo].[events]

110 %

Результаты    Сообщения

	id	user_id	event_name	event_date	event_description
1	1	2	Айкідо	2025-03-18 12:30:00.000	Сходити на айкідо. Не запізнитись!
2	2	1	Курсова робота	2025-03-19 17:30:00.000	Зробити курсову роботу до вечора. Терміново!!!! ...

Рис.5.14 – Перегляд айді, користувача айді, події, дня, часу та опису події

```
SELECT TOP (1000) [id]
      ,[user_id]
      ,[message]
FROM [EventNotifierApp].[dbo].[messages]
```

10 %

Результаты   Сообщения

	id	user_id	message
1	1	1	Привіт! В тебе завтра захист курсової ти її доробив?

Рис.5.15– Перегляд айді, користувача айді та повідомлення

```
SELECT TOP (1000) [id]
      , [name]
      , [email]
      , [phone]
      , [contact_method]
FROM [EventNotifierApp].[dbo].[users]
```

110 %

Результаты   Сообщения

	id	name	email	phone	contact_method
1	1	Федулов Илья	fedylovillya@gmail.com	NULL	email
2	2	Федулов Илья	NULL	+380636257432	phone

Рис.5.16– Перегляд айді, ім'я користувача та метод зв'язку

## 6. Закриття програми:

- Для закриття програми користувач може натискати кнопку "Закрити" в головному вікні або закрити вікно через стандартні засоби операційної системи.

## 7. Помилки та обробка виключень:

- Якщо при роботі з базою даних або при додаванні нових подій виникають помилки, програма відображає повідомлення про помилку у вигляді спливаючих вікон, що містять опис помилки.
- Програма обробляє основні помилки при підключенні до бази даних і забезпечує користувача необхідною інформацією для вирішення проблем.

- Програма обробляє основні помилки при підключенні до бази даних і забезпечує користувача необхідною інформацією для вирішення проблем.

#### 8. Особливості інтерфейсу:

- Інтерфейс користувача простий та зрозумілий, з усіма основними функціями, доступними на головному екрані.
- Кожна функціональність має чітке і зрозуміле позначення, що дозволяє користувачеві швидко освоїтися в програмі.

Програма забезпечує зручну систему для управління подіями та сповіщеннями, дозволяючи користувачам ефективно управляти інформацією та взаємодіяти з іншими користувачами через зручний графічний інтерфейс.

					5.151.1.40-КП	Лист
						44
Изм.	Лист	№ докум.	Підпис	Дата		

## 6. ВИСНОВКИ

Метою заданого курсового проекту було створення програмного додатку для інформування користувачів. Роботу було розділено на наступні етапи для чіткого розуміння поставленої задачі:

- Аналіз задачі та постановка цілей завдання;
- Проектування архітектури додатку та бази даних;
- Розробка програмного алгоритму роботи додатку;
- Створення графічного інтерфейсу та реалізація функціоналу;
- Тестування та налагодження програми.

Робота розпочалася з аналізу основних вимог до додатку та визначення функцій, які необхідно реалізувати, таких як управління користувачами, подіями та каналами зв'язку. Було обрано мову програмування Python, оскільки вона забезпечує високу гнучкість і швидкість розробки, а також підтримує численні бібліотеки для роботи з графічними інтерфейсами та базами даних.

Програма включає функцію відправки повідомлень користувачам, що робить її корисною для компаній, що займаються розсилкою інформації. Крім того, було реалізовано кастомізовані повідомлення, які дозволяють виводити різні типи інформаційних вікон для покращення взаємодії з користувачем.

Таким чином, в результаті виконання курсової роботи, я навчився проектувати програмні додатки, працювати з базами даних, створювати графічні інтерфейси та інтегрувати різні компоненти програми. Результатом виконаної роботи став функціональний додаток для інформування користувачів, який може бути використаний в різних сферах для полегшення управління інформацією та комунікацією.

## 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 pyodbc
```

```
class DatabaseManager:
```

```
    def __init__(self):
```

```
        try:
```

```
            self.conn = pyodbc.connect(
```

```
                'DRIVER={ODBC Driver 17 for SQL Server};'
```

```
                'SERVER=DESKTOP-;'
```

```
                'DATABASE=EventNotifierApp;'
```

```
                'Trusted_Connection=yes;'
```

```
                'Encrypt=yes;'
```

```
                'TrustServerCertificate=yes;'
```

```
            )
```

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

```
            self.create_tables()
```

```
        except pyodbc.Error as e:
```

```
            raise Exception(f"Помилка підключення до бази даних: {str(e)}")
```

```
    def create_tables(self):
```

```
        queries = [
```

```
            '''IF NOT EXISTS (SELECT * FROM sysobjects WHERE
```

```
name='users' AND xtype='U')
```

```
            CREATE TABLE users (
```

```
                id INT IDENTITY(1,1) PRIMARY KEY,
```

```
                name NVARCHAR(100),
```

```
                email NVARCHAR(100),
```

```
                phone NVARCHAR(20),
```

					5.151.1.40-КП	Лист
						47
Изм.	Лист	№ докум.	Підпис	Дата		

```

        contact_method NVARCHAR(20)
    )",
    ""IF NOT EXISTS (SELECT * FROM sysobjects WHERE
name='messages' AND xtype='U')
        CREATE TABLE messages (
            id INT IDENTITY(1,1) PRIMARY KEY,
            user_id INT,
            message NVARCHAR(1000),
            FOREIGN KEY (user_id) REFERENCES users(id)
        )",
    ""IF NOT EXISTS (SELECT * FROM sysobjects WHERE
name='events' AND xtype='U')
        CREATE TABLE events (
            id INT IDENTITY(1,1) PRIMARY KEY,
            user_id INT,
            event_date DATETIME,
            event_description NVARCHAR(1000),
            FOREIGN KEY (user_id) REFERENCES users(id)
        )"
]

for query in queries:
    self.cursor.execute(query)
    self.conn.commit()

def execute_query(self, query, params=(), fetch=False):
    """"Выполнити SQL-запит.""""
    with self.conn.cursor() as cursor:
        cursor.execute(query, params)
        if fetch:
            return cursor.fetchall()
    self.conn.commit()

```



```

import tkinter as tk
from tkinter import messagebox, simpledialog
from tkcalendar import Calendar
import threading
import time
from datetime import datetime
from database import DatabaseManager

class EventNotifierGUI:
    """Основний клас інтерфейсу програми."""

    def __init__(self, root, db_manager):
        self.root = root
        self.db = db_manager
        self.root.title("Система інформування користувачів")
        self.root.geometry("850x850")
        self.root.configure(bg="#e6f7e6")

        self.create_gui()
        self.load_users()

        reminder_thread = threading.Thread(target=self.check_reminders)
        reminder_thread.daemon = True
        reminder_thread.start()

    def create_gui(self):
        """Створення графічного інтерфейсу."""
        tk.Label(self.root, text="Повідомлення для користувачів",
                  font=("Arial", 16), bg="#e6f7e6", fg="#333").pack(pady=10)

```

```

self.users_listbox = tk.Listbox(self.root, height=6, width=45,
font=("Arial", 12))
self.users_listbox.pack(pady=10)

tk.Label(self.root, text="Введіть повідомлення", bg="#e6f7e6",
fg="#333").pack()

self.message_entry = tk.Text(self.root, font=("Arial", 12), width=45,
height=10)
self.message_entry.pack(pady=10)

tk.Button(self.root, text="Відправити повідомлення", font=("Arial",
12),
bg="#4CAF50", fg="#fff",
command=self.send_message).pack(pady=20)

buttons_frame = tk.Frame(self.root, bg="#e6f7e6")
buttons_frame.pack(pady=20)

tk.Button(buttons_frame, text="Додати користувача", font=("Arial",
12),
bg="#2196F3", fg="#fff",
command=self.add_user).pack(side=tk.LEFT, padx=10)

tk.Button(buttons_frame, text="Видалити користувача",
font=("Arial", 12),
bg="#F44336", fg="#fff",
command=self.delete_user).pack(side=tk.LEFT, padx=10)

```

# Кнопка для створення події

					5.151.1.40-КП	Лист
						50
Изм.	Лист	№ докум.	Підпис	Дата		

```

tk.Button(self.root, text="Створити подію", font=("Arial", 12),
          bg="#FFC107", fg="#fff",
          command=self.create_event).pack(pady=20)

# Календар для вибору дати події
tk.Label(self.root, text="Виберіть дату події", bg="#e6f7e6",
          fg="#333").pack()

self.calendar = Calendar(self.root, selectmode='day',
                          date_pattern='yyyy-mm-dd', locale='uk')
self.calendar.pack(pady=20)

def load_users(self):
    """Завантаження списку користувачів."""
    self.users_listbox.delete(0, tk.END)
    users = self.db.execute_query("SELECT id, name, contact_method,
email, phone FROM users", fetch=True)
    self.user_data = { } # Зберігаємо ID користувачів у словник
    for user in users:
        user_id, name, method, email, phone = user
        contact_info = email if method == "email" else phone
        display_text = f"{name} ({method}: {contact_info})"
        self.users_listbox.insert(tk.END, display_text)
        self.user_data[display_text] = user_id

def send_message(self):
    """Відправка повідомлення вибраному користувачу."""
    selected = self.users_listbox.curselection()
    if not selected:
        self.show_custom_message("Помилка", "Виберіть користувача зі
списку.", "red")

```

```

        return

    user_text = self.users_listbox.get(selected[0])
    user_id = self.user_data[user_text]

    message = self.message_entry.get("1.0", tk.END).strip()
    if not message:
        self.show_custom_message("Помилка", "Введіть повідомлення.",
        "red")
        return

    self.db.execute_query("INSERT INTO messages (user_id, message)
VALUES (?, ?)", (user_id, message))
    self.show_custom_message("Готово", "Повідомлення відправлено.",
    "green")
    self.message_entry.delete("1.0", tk.END)

def add_user(self):
    """Додавання користувача."""
    def save_user():
        name, email, phone = name_entry.get(), email_entry.get(),
phone_entry.get()
        contact_method = contact_var.get()

        if not name or (contact_method == "email" and not email) or
(contact_method == "phone" and not phone):
            self.show_custom_message("Помилка", "Заповніть всі поля.",
            "red")
            return

```

```

        self.db.execute_query("INSERT INTO users (name, email, phone,
contact_method) VALUES (?, ?, ?, ?)",
                                (name, email if contact_method == "email" else None,
                                phone if contact_method == "phone" else None,
contact_method))

        self.show_custom_message("Готово", "Користувача додано.",
"green")

        new_user_window.destroy()

        self.load_users()


new_user_window = tk.Toplevel(self.root)
new_user_window.title("Додати користувача")
new_user_window.geometry("400x250")
new_user_window.configure(bg="#dff0d8")


tk.Label(new_user_window, text="Ім'я", bg="#dff0d8").pack()
name_entry = tk.Entry(new_user_window, width=25, font=("Arial",
12))
name_entry.pack()


tk.Label(new_user_window, text="Електронна пошта",
bg="#dff0d8").pack()
email_entry = tk.Entry(new_user_window, width=25, font=("Arial",
12))
email_entry.pack()


tk.Label(new_user_window, text="Номер телефону",
bg="#dff0d8").pack()
phone_entry = tk.Entry(new_user_window, width=25, font=("Arial",
12))

```

```

phone_entry.pack()

contact_var = tk.StringVar(value="email")
tk.Radiobutton(new_user_window, text="Email", variable=contact_var,
value="email", bg="#dff0d8").pack()
tk.Radiobutton(new_user_window, text="Телефон",
variable=contact_var, value="phone", bg="#dff0d8").pack()

tk.Button(new_user_window, text="Зберегти", font=("Arial", 12),
bg="#4CAF50", fg="white",
command=save_user).pack(pady=10)

def delete_user(self):
    """Видалення користувача."""
    selected = self.users_listbox.curselection()
    if not selected:
        self.show_custom_message("Помилка", "Виберіть користувача зі
списку.", "red")
        return

    user_text = self.users_listbox.get(selected[0])
    user_id = self.user_data[user_text]

    self.db.execute_query("DELETE FROM users WHERE id=?",
(user_id,))
    self.show_custom_message("Готово", "Користувача видалено.",
"green")
    self.load_users()

```

```

def create_event(self):
    """Створення події для користувача на вибрану дату з описом."""
    selected = self.users_listbox.curselection()
    if not selected:
        self.show_custom_message("Помилка", "Виберіть користувача зі списку.", "red")
        return

    user_text = self.users_listbox.get(selected[0])
    user_id = self.user_data[user_text]

    # Вибір дати події з календаря
    event_date_str = self.calendar.get_date()
    try:
        event_date = datetime.strptime(event_date_str, "%Y-%m-%d")
    except ValueError:
        self.show_custom_message("Помилка", "Невірний формат дати.", "red")
        return

    # Створення нового вікна для введення назви події та часу
    event_window = tk.Toplevel(self.root)
    event_window.title("Створення події")
    event_window.geometry("500x425")
    event_window.configure(bg="#dff0d8")

    tk.Label(event_window, text="Назва події:", font=("Arial", 12),
    bg="#dff0d8").pack(pady=10)
    event_name_entry = tk.Entry(event_window, font=("Arial", 12),
    width=40)

```

```

event_name_entry.pack(pady=10)

tk.Label(event_window, text="Час події:", font=("Arial", 12),
bg="#dff0d8").pack(pady=10)

time_entry = tk.Entry(event_window, font=("Arial", 12), width=40)
time_entry.pack(pady=10)

tk.Label(event_window, text="Введіть опис події:", font=("Arial",
12), bg="#dff0d8").pack(pady=10)

event_text = tk.Text(event_window, font=("Arial", 12), width=40,
height=6)
event_text.pack(pady=10)

def save_event():

    event_name = event_name_entry.get().strip()
    event_time = time_entry.get().strip()
    event_description = event_text.get("1.0", tk.END).strip()

    if not event_name or not event_time or not event_description:
        self.show_custom_message("Помилка", "Заповніть всі поля.",
"red")

        return

    event_datetime_str = f"{event_date_str} {event_time}"
    try:
        event_datetime = datetime.strptime(event_datetime_str, "%Y-%m-
%d %H:%M")
    except ValueError:
        self.show_custom_message("Помилка", "Невірний формат
часу.", "red")

```



```

        return

        self.db.execute_query("INSERT INTO events (user_id, event_date,
event_name, event_description) VALUES (?, ?, ?, ?)",
                                (user_id, event_datetime, event_name,
event_description))

        self.show_custom_message("Готово", "Подію додано.", "green")
        event_window.destroy()

        tk.Button(event_window, text="Зберегти", font=("Arial", 12),
bg="#4CAF50", fg="white",
                    command=save_event).pack(pady=10)

def check_reminders(self):
    """Перевірка нагадувань та виведення їх у консоль."""
    while True:
        # Отримуємо всі події
        events = self.db.execute_query("SELECT id, user_id, event_date,
event_description FROM events", fetch=True)
        for event in events:
            event_id, user_id, event_date, event_description = event
            # Якщо подія сьогодні або в найближчі 15 хвилин
            if (event_date - datetime.now()).total_seconds() <= 900 and
(event_date - datetime.now()).total_seconds() > 0:
                # Відправляємо нагадування
                self.send_reminder(user_id, event_description)
            time.sleep(60) # Перевірка раз на хвилину

def send_reminder(self, user_id, event_description):
    """Відправка нагадування користувачу."""

```

```

        user = self.db.execute_query("SELECT name, contact_method, email,
phone FROM users WHERE id=?", (user_id,), fetch=True)[0]

        name, contact_method, email, phone = user

        message = f"Нагадування для {name}: {event_description}"

# Виводимо нагадування в консоль
print(f"Нагадування: {message}")

# Для подальшої реалізації: можна додати відображення
нагадування в інтерфейсі

self.show_custom_message("Нагадування", message, "blue")

def show_custom_message(self, title, message, color):
    """Виведення кастомного повідомлення в кольоровому вікні."""
    msg_box = tk.Toplevel(self.root)
    msg_box.title(title)
    msg_box.geometry("400x200")
    msg_box.configure(bg=color)

    label = tk.Label(msg_box, text=message, font=("Arial", 14), bg=color,
fg="white", wraplength=350)
    label.pack(expand=True, fill=tk.BOTH, pady=20)

    button = tk.Button(msg_box, text="Закрити", font=("Arial", 12),
bg="gray", fg="white", command=msg_box.destroy)
    button.pack(pady=10)

# Створення екземпляра GUI та підключення до бази даних
if __name__ == "__main__":
    root = tk.Tk()

```

```

db_manager = DatabaseManager() # Ініціалізуємо об'єкт для роботи з
базою даних
app = EventNotifierGUI(root, db_manager)
root.mainloop()

```

## Додаток В

```

from database import DatabaseManager
from gui import EventNotifierGUI
import tkinter as tk

def main():
    db_manager = DatabaseManager()
    root = tk.Tk()
    app = EventNotifierGUI(root, db_manager)
    root.mainloop()

if __name__ == "__main__":
    main()

```