

БАКАЛАВРСЬКА РОБОТА

БР.КІ-41.00.00.000 ПЗ

Група КІ-22-2

Нейлюк Максим

2026

Міністерство освіти і науки України
Івано-Франківський національний технічний університет нафти і газу
Факультет інформаційних технологій
Кафедра комп'ютерних систем і мереж

Нейлюк Максим Романович

УДК 004.4

БАКАЛАВРСЬКА РОБОТА

Розробка локального додатку-менеджера паролів засобами Python

Комп'ютерна інженерія

(назва освітньої програми)

123 - Комп'ютерна інженерія

(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня Нейлюк М.Р.
(підпис, прізвище та ініціали здобувача)

Науковий керівник Бабчук С.М., к. т. н., доцент
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту

Завідувач кафедри КСМ

д.т.н., професор /С. І. Мельничук/
(посада) (підпис) (дата) (прізвище та ініціали)

Івано-Франківськ – 2026 рік

Івано-Франківський національний технічний університет нафти і газу
Факультет Інформаційних технологій
Кафедра Комп'ютерних систем і мереж
Освітній рівень бакалавр
Спеціальність 123 – Комп'ютерна інженерія

ЗАТВЕРДЖУЮ

Завідувач кафедри КСМ

Мельничук С. І.

" 21 " квітня 2026 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Нейлюку Максиму Романовичу

(прізвище, ім'я, по батькові)

1. Тема роботи Розробка локального додатку-менеджера паролів засобами Python.
- керівник роботи Бабчук Сергій Миронович, к. т. н., доцент
затверджені наказом закладу вищої освіти від " 21 " квітня 2026 року № 180/7
2. Термін подання студентом роботи 11 червня 2026 року
3. Вихідні дані до роботи Методичні вказівки, технічна література
4. Зміст пояснювальної записки: 1. Аналіз предметної області та технологій реалізації. 2. Розробка клієнт-серверного додатка менеджера паролів. 3. Перевірка працездатності розробленого додатка.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Нормоконтроль		
Антиплагіат		

6. Дата видачі завдання – 02 лютого 2026 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Термін виконання етапів роботи	Примітка
1	Постановка завдання та вибір літератури	<i>Лютий 2026</i>	виконав
2	Аналіз аналогів та вибір необхідних технологій реалізації	<i>Березень 2026</i>	виконав
3	Розробка локального додатку-менеджера паролів засобами Python	<i>Квітень 2026</i>	виконав
4	Перевірка працездатності розробленого локального додатку-менеджера паролів	<i>Травень 2026</i>	виконав
5	Оформлення пояснювальної записки	<i>Червень 2026</i>	виконав

Студент _____ Нейлюк М.Р.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Бабчук С.М.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Метою даної бакалаврської роботи є розробка локального додатку-менеджера паролів засобами Python.

У першому розділі проведено аналіз предметної області та аналогів. Розглянуто та обрано бібліотеки мови програмування Python, що дозволяють створювати графічний інтерфейс та взаємодіяти із базою даних та проведено аналіз технологій захищеного клієнт-серверного з'єднання.

У другому розділі розроблено структуру графічного інтерфейсу додатку, вона включає сторінки: автентифікації, реєстрації, генерації паролів, додавання паролів до сховища, перегляду збережених паролів та налаштування. Розроблено захищену систему взаємодії із сервером, що дозволяє користувачу проходити автентифікацію, реєструватись, додавати, змінювати, видаляти та отримувати паролі. Створено реляційну базу даних, що зберігає основну інформацію щодо клієнтів та сервера, його логіку взаємодії із клієнтом та базою даних.

У третьому розділі проведено перевірку працездатності розробленого локального додатку-менеджера паролів засобами Python, його основних функцій: додавання, зміна, видалення, та отримання паролів від сервера.

Ключові слова: python, менеджер паролів, база даних, сервер, графічний інтерфейс, безпечне з'єднання, локальна мережа.

ANNOTATION

The purpose of this bachelor's thesis is to develop a local password manager application using Python.

The first section analyzes the subject area and reviews analogues. Python programming language libraries are considered and selected, which allow you to create a graphical interface and interact with the database and analyze secure client-server connection technologies.

In another section, the structure of the graphical interface of the application is designed, it contains pages: authentication, registration, password generation, adding passwords to the repository, viewing saved passwords and settings. A secure system for interacting with the server is developed, which allows the user to authenticate, register, add, change, delete and retrieve passwords. A relational database is designed and created, which stores basic information about clients and the server, its logic for interacting with the client and the database.

In the third section, the performance of the developed local password manager application using Python was tested, as well as its main functions: adding, changing, deleting, and retrieving passwords from the server.

Keywords: python, password manager, database, server, graphical interface, secure connection, local network.

ЗМІСТ

ПЕРЕЛІК ОСНОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ	4
ВСТУП	5
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ТЕХНОЛОГІЙ РЕАЛІЗАЦІЇ	6
1.1 Аналіз локальних та хмарних менеджерів паролів	6
1.2 Аналіз бібліотеки створення графічного інтерфейсу додатка	8
1.3 Вибір технології захищеного клієнт-серверного з'єднання	10
1.4 Аналіз бібліотеки взаємодії із базою даних	14
Висновок до розділу	15
2 РОЗРОБКА ЛОКАЛЬНОГО ДОДАТКУ-МЕНЕДЖЕРА ПАРОЛІВ ЗАСОБАМИ PYTHON	16
2.1 Розробка структури додатка	16
2.2 Розробка блоку реєстрації та автентифікації додатка	22
2.3 Розробка блоків генерації та додавання паролів	25
2.4 Розробка блоків отримання паролів та налаштування додатка	32
2.5 Розробка блоків зв'язку із сервером	39
2.6 Розробка модуля серверної взаємодії з клієнтом та базою даних	42
Висновок до розділу	51
3 ПЕРЕВІРКА ПРАЦЕЗДАТНОСТІ РОЗРОБЛЕНОГО ДОДАТКА	52
3.1 Перевірка працездатності клієнтської частини додатка	52
3.2 Перевірка працездатності серверної частини додатка	62
Висновок до розділу	63
ВИСНОВКИ	64
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ	65
ДОДАТКИ	
БІБЛІОГРАФІЧНА ДОВІДКА	

					БР.КІ-41.00.00.000 ПЗ					
Змн	Лист	№ докум.	Підпис	Дата						
Розроб.		Нейлюк М. Р.			Розробка локального додатку-менеджера паролів засобами Python			Лім.	Арк.	Аркушів
Перевір.		Бабчук С.М.						Н	3	67
Реценз.		Заячук Я.І.						ІФНТУНГ, КІ-22-2		
Н. Контр.		Лазорів А.М.								
Затверд.		Мельничук С.І.								

ПЕРЕЛІК ОСНОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

БД (*База даних*) – організована сукупність даних, що зберігається для подальшого пошуку, обробки та керування;

СУБД (*Система управління базами даних*) – програмне забезпечення для створення, зберігання, зміни та керування базами даних;

СА (*Certification Authority*) – довірений сертифікат, який використовується для перевірки ідентичності;

ДЕК (*Data Encryption Key*) – випадково згенерований ключ для шифрування даних у БД;

IP (*Internet Protocol*) – мережевий протокол, який реалізує адресацію та маршрутизацію пакетів даних між пристроями в мережі;

КЕК (*Key Encryption Key*) – ключ, що створюється з пароля користувача, для шифрування та розшифрування DEK;

mTLS (*Mutual Transport Layer Security*) – механізм взаємної автентифікації, коли і клієнт, і сервер підтверджують свої сертифікати під час встановлення захищеного з'єднання;

SQL (*Structured Query Language*) – мова запитів для створення, зміни та отримання даних із реляційних баз даних;

SSL (*Secure Sockets Layer*) – криптографічний протокол для захисту мережових з'єднань, що був замінений TLS;

TCP (*Transmission Control Protocol*) – транспортний протокол, що гарантує доставку пакетів та вимагає підключення;

TLS (*Transport Layer Security*) – сучасний криптографічний протокол, який дозволяє шифрування, цілісність і автентифікацію даних під час передачі мережею;

UDP (*User Datagram Protocol*) – транспортний протокол без встановлення з'єднання, який передає дані без гарантії доставки та порядку пакетів.

					БР.КІ-41.00.00.000 ПЗ	Арк.
Змн.	Арк.	окум.	Підпис	Дат		4

ВСТУП

Актуальність обраної теми. У сучасному цифровому світі кожна людина використовує десятки онлайн-сервісів: електронну пошту, соціальні мережі, банківські додатки, хмарні сховища, освітні платформи та інші вебресурси. Для забезпечення безпеки особистих даних важливо використовувати надійні та унікальні паролі для кожного облікового запису. Проте запам'ятати велику кількість складних паролів, що містять літери верхнього та нижнього регістру, цифри та спеціальні символи, досить важко, тому багато користувачів застосовують однакові або занадто прості паролі. Такий підхід суттєво підвищує ризик несанкціонованого доступу до конфіденційної інформації та може призвести до втрати важливих даних або фінансових збитків.

Об'єктом є процес забезпечення безпечного зберігання облікових даних користувачів для доступу до інформаційних систем.

Предметом дослідження є засоби мови програмування Python для розробки додатків.

Мета та завдання роботи. Метою роботи є розробка локального додатку-менеджера паролів засобами Python.

Для досягнення мети поставлено такі завдання:

- провести аналіз предметної області та технологій реалізації;
- створити структуру локального додатку та схему бази даних;
- розробити графічний інтерфейс для взаємодії користувача з додатком;
- реалізувати систему автентифікації та реєстрації користувачів;
- створити систему додавання, зміни, отримання та видалення паролів;
- провести перевірку працездатності розробленого менеджера паролів.

Методи дослідження. Використано метод порівняльного аналізу для дослідження технологій та засобів розробки додатків.

Практичне значення отриманих результатів. Практичне значення полягає у розробці локального додатку-менеджера паролів засобами Python.

					БР.КІ-41.00.00.000 ПЗ	Арк.
Змн.	Арк.	окум.	Підпис	Дат		5

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ТЕХНОЛОГІЙ РЕАЛІЗАЦІЇ

1.1 Аналіз локальних та хмарних менеджерів паролів

Менеджери паролів є спеціалізованими програмними засобами, призначеними для безпечного зберігання, генерації та керування обліковими даними користувачів. Основними функціями таких систем є створення складних унікальних паролів, їх зберігання, а також захист від несанкціонованого доступу. Залежно від архітектури, менеджери паролів поділяються на локальні та хмарні.

Одним із найпоширеніших локальних рішень є KeePassXC (рис.1.1).

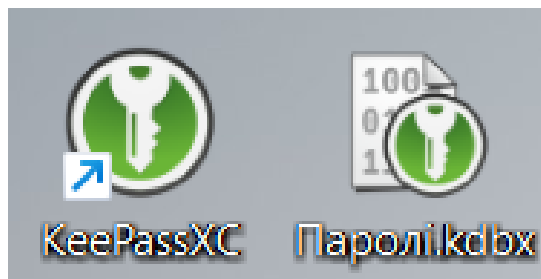


Рисунок 1.1 – Додаток KeePassXC та створена база даних “Паролі.kdbx”

Він працює за принципом зберігання всіх даних у локальному зашифрованому файлі бази даних формату .kdbx. Доступ до цього файлу захищається головним паролем або ключовим файлом. Усі операції – додавання, редагування та генерація паролів – виконуються без підключення до інтернету.

Хмарним прикладом є LastPass for Desktop, який працює за принципом централізованого зберігання даних у хмарному сховищі.

Додаток надає багато можливостей керування обліковим записом та паролями. Для роботи із менеджером користувач створює обліковий запис, після чого його паролі синхронізуються між пристроями через сервери компанії. Дані шифруються на стороні клієнта, однак для роботи системи необхідне постійне інтернет-з’єднання (рис.1.2).

					БР.КІ-41.00.00.000 ПЗ	Арк.
Змн.	Арк.	окум.	Підпис	Дат		6

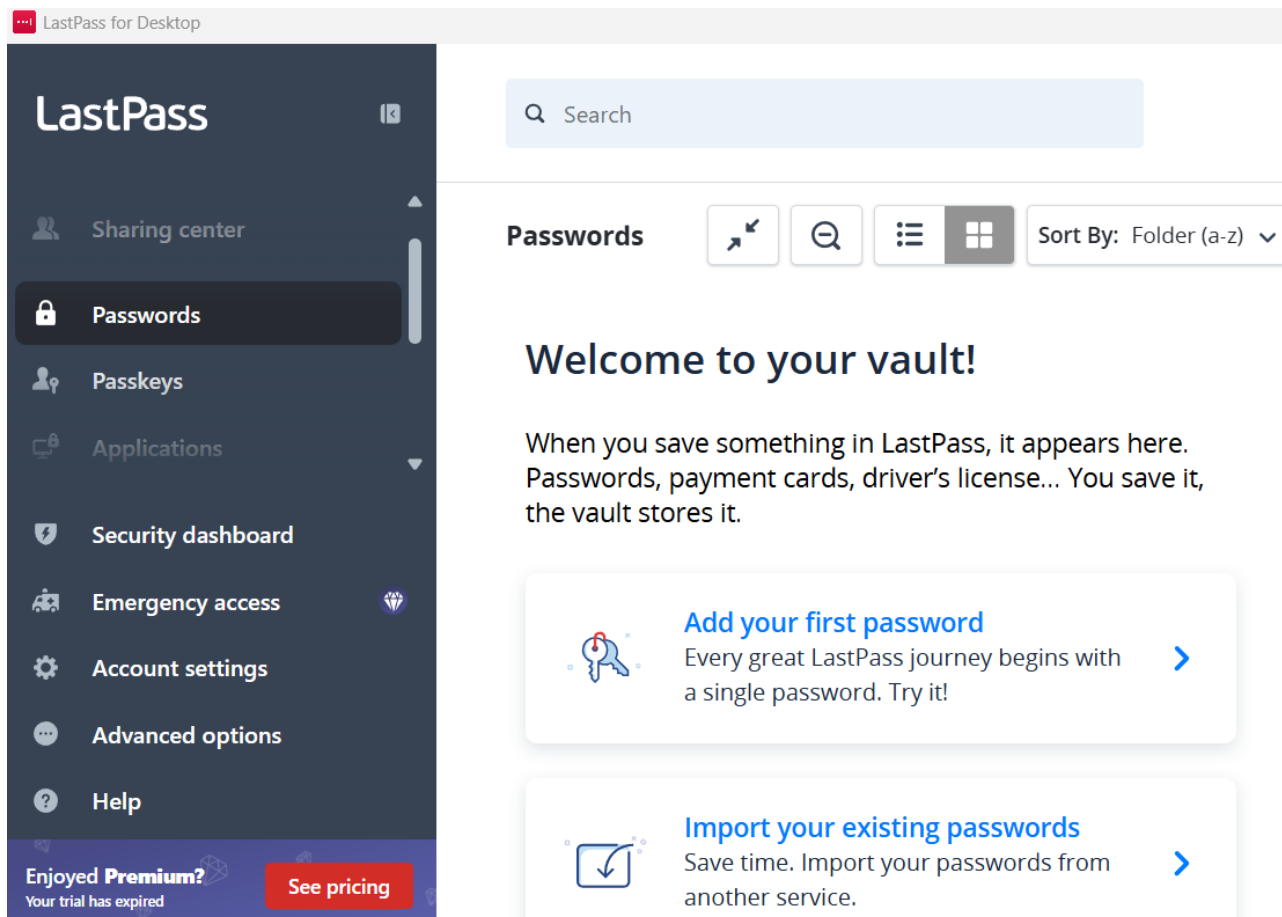


Рисунок 1.2 – Головна сторінка додатку LastPass for Desktop

Основною перевагою хмарного підходу є зручність синхронізації, але недоліком є залежність від віддаленої інфраструктури та ризики, пов'язані з компрометацією хмарних сервісів.

У таблиці 1.1 наведено порівняння локального менеджера KeePassXC із хмарним LastPass за типом системи, способом зберігання даних, доступом до інтернету, багатокористувацької роботи, рівнем контролю над даними безпекою доступу та основним призначенням.

Таблиця 1.1 – Порівняння менеджерів KeePassXC та LastPass із розроблювальним додатком

Характеристика	KeePassXC	LastPass for Desktop
Тип системи	Локальний	Хмарний
Зберігання даних	Локальний файл (.kdbx)	Хмарні сервери
Доступ до інтернету	Не потрібен	Обов'язковий
Багатокористувацька робота	Обмежена	Так

Рівень контролю над даними	Високий	Середній
Безпека доступу	Головний пароль, шифрування	Клієнтське шифрування, хмарний сервіс
Основне призначення	Особисте використання	Масове використання з синхронізацією

Локальний менеджер KeePassXC не дозволяє багатокористувацьку роботу, та не надає централізоване керування доступом через сервер.

Хмарний менеджер LastPass дозволяє багатокористувацький доступ та підтримку централізованого керування даними, проте залежить від віддаленої інфраструктури.

У зв'язку з цим, в розроблювальній системі потрібно реалізувати можливість доступу для багатьох користувачів, за допомогою центрального керування доступу зі сторони сервера, не встановлюючи віддалене інтернет-з'єднання. Така організація робить менеджер можливим для використання в межах локальної мережі малих організацій або домашніх середовищ.

1.2 Аналіз бібліотеки створення графічного інтерфейсу додатка

Для реалізації клієнтської частини менеджера паролів було обрано бібліотеку CustomTkinter, яка є розширенням стандартної бібліотеки Tkinter для мови програмування Python. Дана бібліотека надає сучасний набір графічних компонентів та дозволяє створювати інтерфейси з дизайном, наближеним до сучасних настільних застосунків.

Кожен елемент інтерфейсу представлений окремим віджетом, який може реагувати на дії користувача, такі як натискання кнопок, введення тексту або вибір пунктів меню.

Віджети можуть бути організовані за допомогою менеджерів компоновання, що дають можливість розташовувати елементи у вигляді таблиць, блоків або вільних координат. Під час взаємодії користувача з інтерфейсом

відповідні події передаються до функцій обробників, які виконують необхідну логіку програми, наприклад надсилання запиту до сервера, отримання даних або оновлення інформації на екрані.

Нижче наведено приклад програмної реалізації головного вікна за допомогою бібліотеки CustomTkinter.

```
import customtkinter as ctk
ctk.set_appearance_mode("dark")
ctk.set_default_color_theme("blue")
app = ctk.CTk()
app.title("Головна сторінка")
app.geometry("600x400")
title_label = ctk.CtkLabel(app, text="Сторінка", font=("Arial", 24, "bold"))
title_label.pack(pady=20)
generate_button = ctk.CTkButton( app, text=" Нова кнопка" )
generate_button.pack(pady=10)
app.mainloop()
```

У наведеному прикладі створюється головне вікно програми, текстовий заголовок та кнопка для взаємодії. Аналогічним чином реалізуються інші елементи користувацького інтерфейсу.

У розроблюваному менеджері паролів бібліотека CustomTkinter використовуватиметься для створення вікон реєстрації та автентифікації користувачів, відображення списку збережених паролів, форм додавання та редагування записів.

Аргументувати вибір доповнення для створення графічного інтерфейсу можна провівши порівняння обраного рішення із популярними бібліотеками, серед яких Tkinter, PyQt, PySide та Kivy.

У таблиці 1.2 зображене порівняння бібліотек створення графічного інтерфейсу мовою програмування Python за зовнішнім виглядом, швидкістю розробки, використанням системних ресурсів та сумісністю із настільними застосунками.

					БР.КІ-41.00.00.000 ПЗ	Арк.
Змн.	Арк.	окум.	Підпис	Дат		9

Таблиця 1.2 – Порівняння бібліотек Tkinter, PyQt, PySide, Kivy та CustomTkinter

Характеристика	Tkinter	CustomTkinter	PyQt/PySide	Kivy
Сучасний зовнішній вигляд	Низький	Високий	Високий	Високий
Швидкість розробки	Висока	Висока	Середня	Середня
Використання системних ресурсів	Низьке	Низьке	Середнє	Високе
Підходить для настільних застосунків	Так	Так	Так	Частково

Стандартний Tkinter є простим у використанні, однак його елементи керування мають застарілий зовнішній вигляд.

Бібліотеки PyQt та PySide дозволяють створювати професійні інтерфейси з великою кількістю компонентів, проте характеризуються складнішою архітектурою та вищими вимогами до ресурсів.

Kivy орієнтована переважно на мультиплатформні та сенсорні застосунки, що є надлишковим для розроблюваного проєкту.

Вибір CustomTkinter для реалізації клієнтської частини менеджера паролів обумовлений рядом переваг.

По-перше, бібліотека забезпечує сучасний зовнішній вигляд програми без необхідності використання складних фреймворків.

По-друге, вона повністю сумісна зі стандартним Tkinter, що спрощує розробку та супровід програмного коду.

По-третє, бібліотека має невеликі системні вимоги та не потребує значної кількості додаткових залежностей.

В результаті порівняння, бібліотека CustomTkinter є хорошим рішенням для реалізації графічного інтерфейсу додатку менеджера паролів.

1.3 Вибір технології захищеного клієнт-серверного з'єднання

Під час розробки клієнт-серверного менеджера паролів особливу увагу

необхідно приділити захисту мережевого обміну даними між клієнтською та серверною частинами системи. Для реалізації мережевої взаємодії в мові програмування Python використовуються бібліотеки socket та ssl, які забезпечують передачу даних і їх криптографічний захист.

Бібліотека socket є стандартним засобом Python для створення мережевих застосунків. Вона надає інтерфейс для обміну даними між процесами через мережу за допомогою протоколів TCP/IP або UDP.

Принцип роботи socket полягає у створенні двох кінцевих точок зв'язку: сервера та клієнта. Сервер відкриває мережевий порт та очікує вхідних підключень. Після отримання запиту від клієнта між сторонами встановлюється з'єднання, через яке здійснюється обмін повідомленнями.

Однак використання лише socket не забезпечує конфіденційності інформації, оскільки дані передаються мережею у відкритому вигляді.

Для усунення цього недоліку використовується бібліотека ssl, що використовує SSL.

SSL створює захищений канал поверх звичайного TCP-з'єднання. Перед початком передачі даних між сторонами виконується процедура TLS рукоштовування, під час якої відбувається перевірка сертифікатів, узгодження криптографічних алгоритмів та генерація сеансових ключів шифрування.

Протокол захисту передачі даних (TLS) буде використовуватись версії 1.2. Ця версія протоколу рекомендується для застосування Агентством національної безпеки Сполучених Штатів Америки [3].

В результаті всі дані, що передаються між клієнтом та сервером, шифруються. Це забезпечує конфіденційність інформації під час її передачі, контроль цілісності даних, захист від атак типу “людина посередині” та автентифікацію учасників з'єднання.

Для підвищення рівня безпеки у розроблюваній системі використовується механізм Mutual TLS (mTLS).

Основою роботи mTLS є використання власного центру сертифікації (Certificate Authority, CA). Центр сертифікації створює довірений сертифікат CA,

					БР.КІ-41.00.00.000 ПЗ	Арк.
Змн.	Арк.	окум.	Підпис	Дат		11

який використовується для підписування сертифікатів сервера та клієнтів.

Процес встановлення захищеного з'єднання складається з таких етапів:

- створюється кореневий сертифікат СА;
- на основі СА генерується сертифікат сервера;
- для кожного клієнта створюється власний сертифікат;
- сертифікати сервера та клієнтів підписуються сертифікатом СА;
- під час встановлення з'єднання сервер перевіряє сертифікат клієнта;
- клієнт перевіряє сертифікат сервера;
- якщо обидві перевірки успішні, створюється захищений TLS-канал.

На рисунку 1.3 зображений процес встановлення захищеного з'єднання між клієнтом та сервером із використанням mTLS.

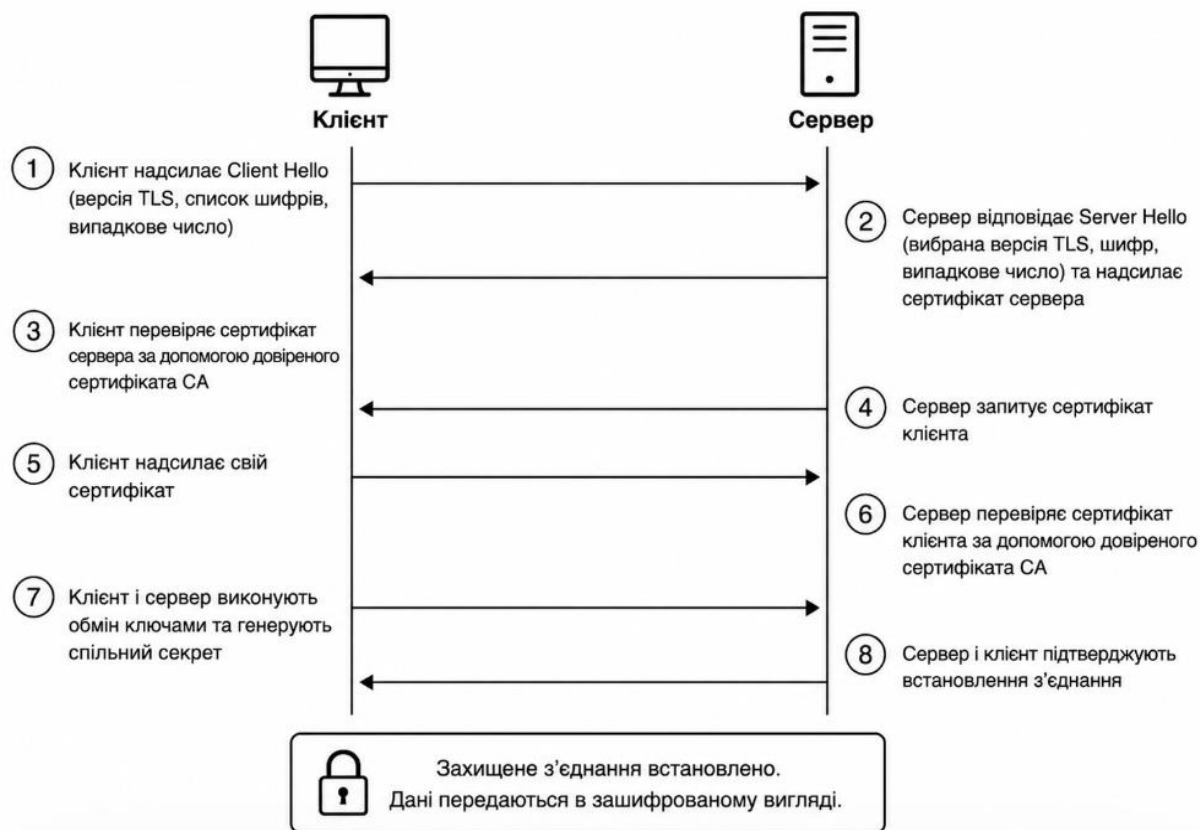


Рисунок 1.3 – Процес встановлення захищеного з'єднання між клієнтом та сервером за допомогою mTLS

Такий підхід дозволяє, підключатися до сервера лише авторизованим клієнтам, а самі клієнти, взаємодіють виключно з довіреним сервером.

Для забезпечення одночасної роботи декількох користувачів сервер використовує механізм багатопотоковості.

Потік є окремою послідовністю виконання інструкцій усередині процесу. При надходженні нового клієнтського підключення сервер створює окремий потік, який обслуговує конкретного користувача.

Спершу сервер запускає основний потік та починає прослуховування порту, далі клієнт встановлює з'єднання із сервером, тому для нового клієнта створюється окремий потік, який обробляє всі його запити, а інші клієнти обслуговуються паралельно власними потоками.

На рисунку 1.4 зображено схема роботи сервера що реалізує багатопотокову технологію.

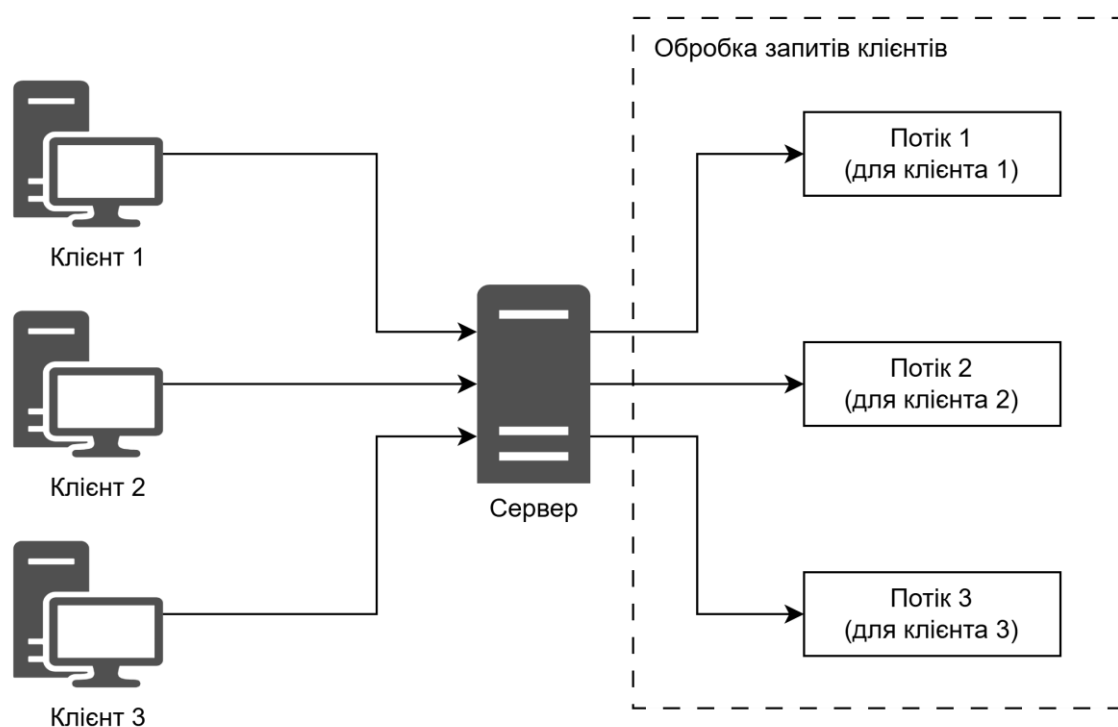


Рисунок 1.4 – Схема роботи багатопотокового сервера

Завдяки використанню багатопотоковості сервер здатний одночасно працювати із запитом одразу декількох користувачів без блокування основного процесу.

Таким чином, поєднання бібліотек socket та ssl, використання механізму

mTLS, та багатопотокової обробки запитів, дозволяє створити захищену клієнт-серверну взаємодію.

1.4 Аналіз бібліотеки взаємодії із базою даних

Для зберігання даних у розроблюваному менеджері паролів використовуватиметься база даних (БД), а для керування базою даних буде використовуватися система управління базами даних (СУБД).

В даний час популярними СУБД є SQLite, MySQL, PostgreSQL та MongoDB. Проведемо їх порівняння за типом, принципом роботи, споживанні ресурсів, масштабованості, швидкості читання та доцільності для локальних застосунків. Результати порівняння наведені у таблиці 1.3.

Таблиця 1.3 – Порівняння систем управління базами даних SQLite, MySQL, PostgreSQL та MongoDB

Характеристика	SQLite	MySQL	PostgreSQL	MongoDB
Тип СУБД	Реляційна	Реляційна	Реляційна	Документна
Окремий сервер	Ні	Так	Так	Так
Споживання ресурсів	Низьке	Середнє	Середнє	Високе
Масштабованість	Низька	Висока	Висока	Висока
Швидкість читання	Висока	Висока	Висока	Висока
Доцільність для локальних застосунків	Висока	Середня	Середня	Низька

У порівнянні з MySQL та PostgreSQL, SQLite не потребує запуску окремого сервісу бази даних, це значно спрощує розгортання системи. У свою чергу, MongoDB орієнтована на роботу з документно-орієнтованими структурами даних і є менш доцільною для проекту, який використовує чітко визначені реляційні зв'язки між користувачами та записами паролів.

Враховуючи переваги та недоліки запропонованих систем управління базами даних, для реалізації додатку оберемо SQLite, оскільки вона: на відміну від серверних систем, використовує файл як основне сховище інформації, має

високу швидкість виконання операцій читання, має низькі вимоги до апаратних ресурсів та інтегрована зі стандартною бібліотекою Python;

Основним недоліком SQLite є обмеження на одночасний запис даних. У певний момент часу лише один процес або потік може виконувати операцію запису до бази даних. При великій кількості одночасних запитів це може призводити до блокування файлу бази даних.

Проте для розроблюваного менеджера паролів дана проблема вирішується за допомогою серверної архітектури та механізму черги запитів. Усі клієнтські застосунки взаємодіють лише із сервером, а безпосередній доступ до SQLite має виключно серверна частина.

Після отримання запиту на зміну даних сервер додає його до внутрішньої черги операцій запису. Далі спеціальний потік обробки послідовно виконує ці запити один за одним. Такий підхід дозволяє уникнути конфліктів блокування бази даних, підтримувати одночасну роботу декількох клієнтів та спростити механізми синхронізації між потоками.

Висновок до розділу

У результаті аналізу менеджерів паролів було розглянуто локальний KeePassXC та хмарний LastPass for Desktop сервіси. На основі їх переваг і недоліків обрано клієнт-серверну архітектуру локального менеджера паролів, яка забезпечує централізоване зберігання даних та підтримку багатокористувацької роботи в локальній мережі.

В результаті проведеного аналізу було встановлено, що для реалізації графічного інтерфейсу доцільно використати бібліотеку CustomTkinter, яка дозволяє створювати настільні застосунки з мінімальними системними вимогами, а для захисту мережевої взаємодії доцільно використати бібліотеки socket та ssl із механізмом mTLS, що забезпечує безпечну передачу даних.

Для керування базою даних обрано СУБД SQLite, яка характеризується достатньою продуктивністю для локальних систем.

					БР.КІ-41.00.00.000 ПЗ	Арк.
Змн.	Арк.	окум.	Підпис	Дат		15

2 РОЗРОБКА КЛІЄНТ-СЕРВЕРНОГО ДОДАТКА МЕНЕДЖЕРА ПАРОЛІВ

2.1 Розробка структури додатка

Вхідними елементами системи є модулі реєстрації та аутентифікації. Модуль реєстрації дозволяє створювати нові облікові записи користувачів шляхом введення імені, електронної пошти та пароля.

Модуль аутентифікації забезпечує перевірку облікових даних та надає доступ до функціоналу системи лише авторизованим користувачам.

Після успішного входу в додаток, користувач повинен отримати доступ до основного блоку програми, яке містить модуль керування паролями, що дає можливість: генерації, додавання, перегляду, редагування та видалення паролів.

Окремим компонентом є модуль налаштувань додатка, який дозволяє змінювати параметри інтерфейсу та обліковий запис.

На рисунку 2.1 зображена структура додатку, його ключових елементів.



Рисунок 2.1 – Структура додатку

Наступним кроком визначимо структуру реляційної бази даних та її взаємопов'язаних таблиць. Вона складатиметься з трьох основних таблиць: users, passwords та tokens.

На рисунку 2.2 зображена розроблена схема бази даних.

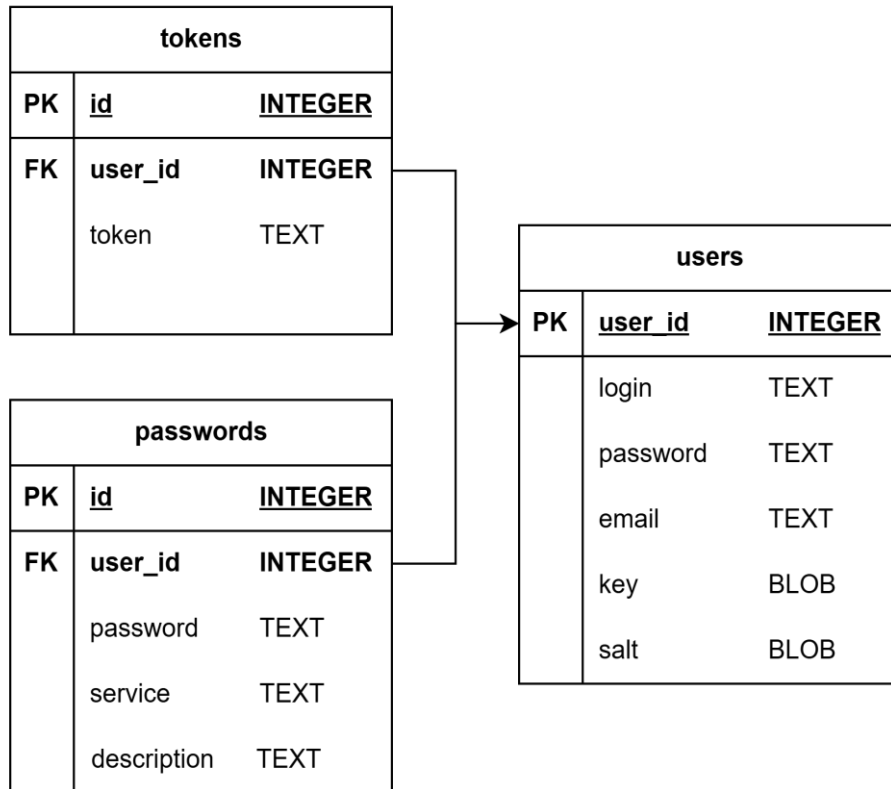


Рисунок 2.2 – Схема бази даних

Зв'язок між таблицями реалізовано за допомогою первинного ключа user_id типу INTEGER у таблиці users, на який посиляються зовнішні ключі user_id у таблицях tokens та passwords.

Завдяки налаштованому каскадному видаленню під час видалення запису користувача з таблиці users автоматично видаляються всі пов'язані записи в таблицях tokens і passwords. Така структура дає змогу зберігати кілька пов'язаних записів для одного користувача, що відповідає зв'язку типу “один до багатьох”.

У таблиці 2.1 наведений вміст таблиці users.

Таблиця 2.1 – Структура таблиці users

Ключ	Назва поля	Тип	Розмір	Призначення
ПК	user_id	INTEGER	11	Ідентифікатор користувача
-	login	TEXT	65 535	Ім'я
-	password	TEXT	65 535	Хешований пароль
-	email	TEXT	65 535	Електронна адреса
-	key	BLOB	65 535	Зашифрований ключ доступу до паролів
-	salt	BLOB	65 535	Сіль для розшифрування

Таблиця users призначена для зберігання інформації про користувачів системи. Вона містить унікальний ідентифікатор користувача (user_id), логін (login), зашифрований пароль (password), зашифровану електронну адресу (email), зашифрований ключ доступу до паролів та сіль, для розшифрування ключа.

У таблиці 2.2 наведений вміст таблиці tokens.

Таблиця 2.2 – Структура таблиці tokens

Ключ	Назва поля	Тип	Розмір	Призначення
ПК	id	INTEGER	11	Ідентифікатор токена
ЗК	user_id	INTEGER	11	Ідентифікатор користувача (до users)
-	token	TEXT	65 535	Токен

Таблиця tokens призначена для зберігання токенів автентифікації. Вона містить унікальний ідентифікатор запису (id), зовнішній ключ (user_id) та зашифроване значення токена (token).

Таблиця passwords (табл.2.3) використовується для зберігання паролів користувачів. Вона містить унікальний ідентифікатор запису (id), зовнішній ключ (user_id), який пов'язує пароль із конкретним користувачем, сам зашифрований пароль (password), назву сервісу (service) та зашифрований опис (description).

Таблиця 2.3 – Структура таблиці passwords

Ключ	Назва поля	Тип	Розмір	Призначення
ПК	id	INTEGER	11	Ідентифікатор пароля
ЗК	user_id	INTEGER	11	Ідентифікатор користувача (до users)
-	password	TEXT	65 535	Пароль
-	service	TEXT	65 535	Назва сервісу пароля
-	description	TEXT	65 535	Додатковий опис пароля

Визначимо можливості користувача під час входу до додатку за допомогою діаграми випадків використання (рис.2.3).

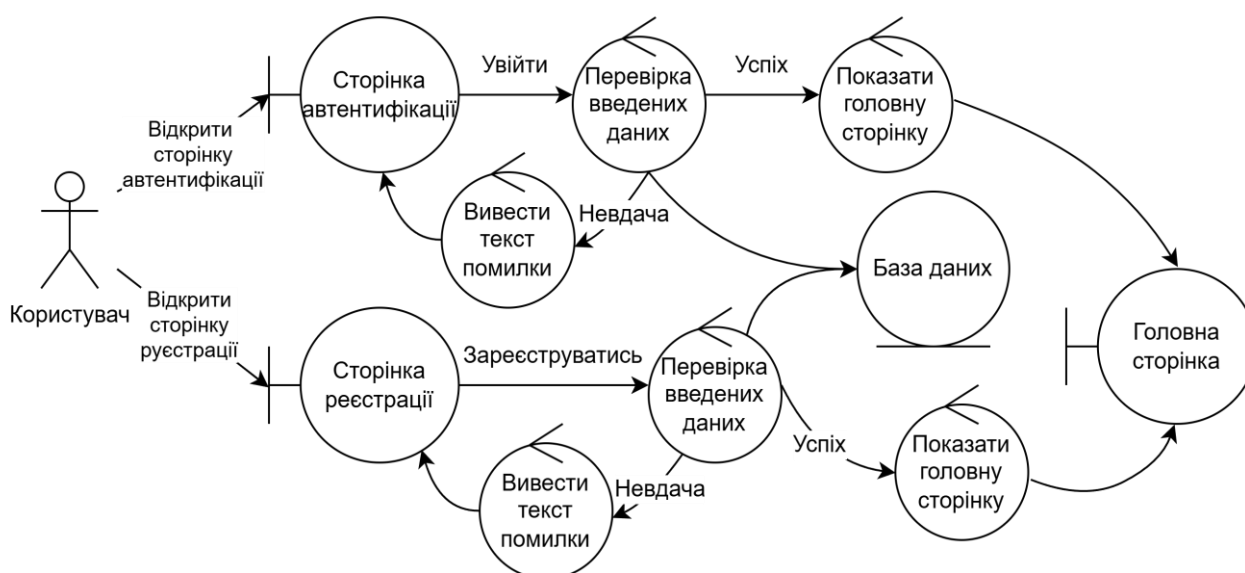


Рисунок 2.3 – Діаграма випадків використання входу користувача до додатку

Після запуску застосунку користувачу доступні сторінки автентифікації та реєстрації. На сторінці автентифікації після вибору “Увійти” система перевіряє введені дані. Якщо перевірка не пройдена, користувач повертається до сторінки автентифікації для повторного введення даних. У разі успішної перевірки система ідентифікує користувача та відкриває головну сторінку застосунку.

На сторінці реєстрації після обрання “Зареєструватись” система перевіряє коректність введенних даних. Якщо виявлено помилки, користувач повертається

до форми реєстрації для їх виправлення. Після успішної реєстрації створюється новий обліковий запис, і користувач автоматично переходить на головну сторінку застосунку.

Далі визначимо дії користувача після входу в систему (рис.2.4).

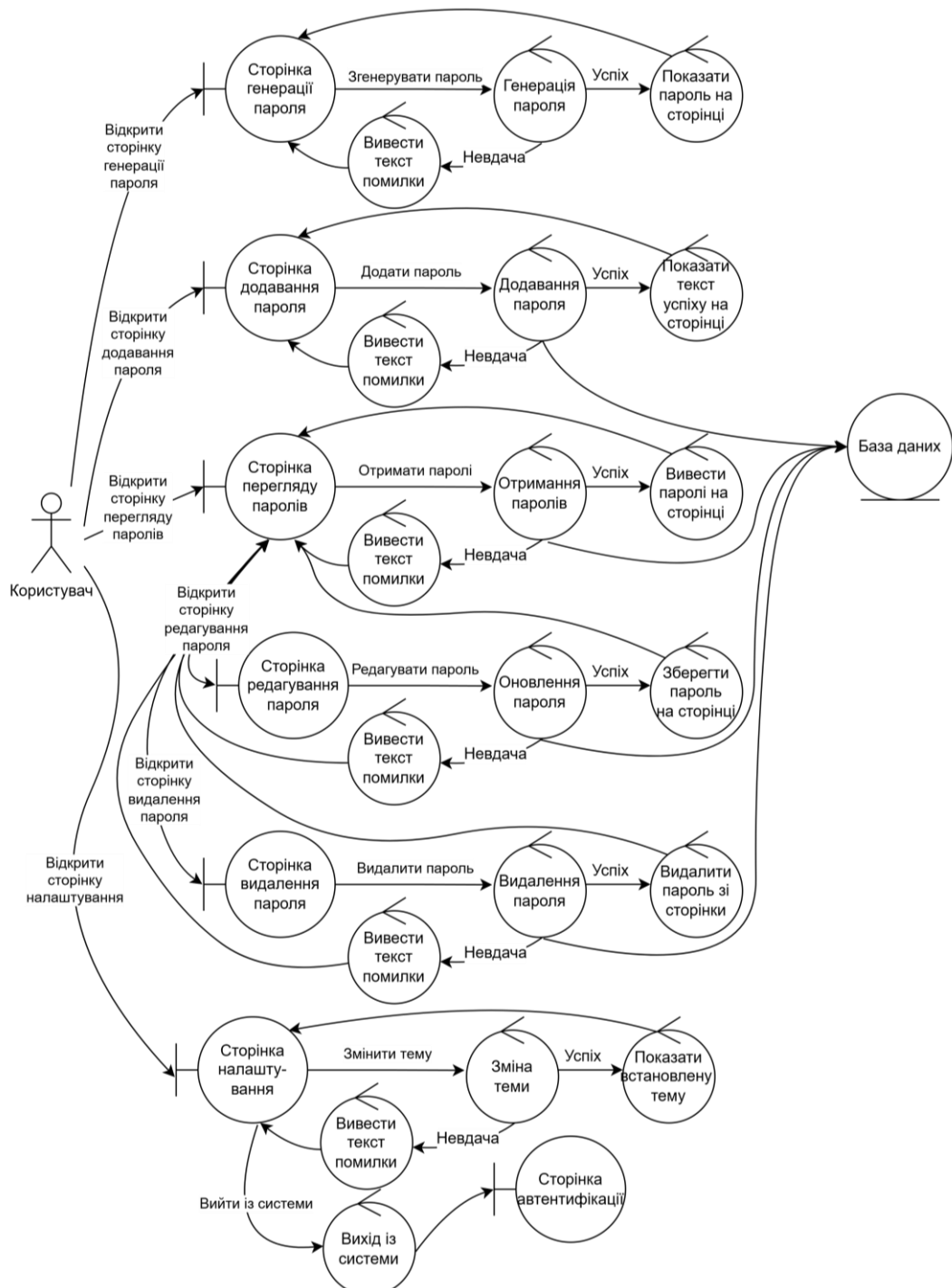


Рисунок 2.4 – Діаграма випадків використання додатку користувачем

Для генерування нового пароля користувачу необхідно перейти на

сторінку генерації пароля та запустити процес створення нового пароля. Система генерує випадкову комбінацію символів відповідно до встановлених правил безпеки та відображає результат на екрані. У випадку виникнення помилки користувач повертається на сторінку генерації для повторного виконання операції.

Для збереження нового пароля користувачу потрібно використати функцію додавання пароля. Після переходу на відповідну сторінку він вводить необхідні дані та надсилає їх до системи. Система виконує перевірку введеної інформації та зберігає її у базі даних. У разі успішного завершення операції користувачу відображається повідомлення про успішне додавання запису. Якщо під час обробки виникає помилка, система повертає користувача до форми введення даних.

Перегляд збережених паролів здійснюватиметься через сторінку перегляду паролів. Після відкриття сторінки система надсилає запит до бази даних, отримує необхідну інформацію та відображає список паролів користувачу.

Функція редагування паролів надає можливість змінювати власні записи. Користувач обирає потрібний пароль зі списку, вносить зміни та зберігає їх. Система оновлює відповідний запис у базі даних і відображає актуальні дані після успішного виконання операції.

Для видалення непотрібних записів передбачена функція видалення пароля. Користувач обирає пароль зі списку та підтверджує видалення. Система видаляє відповідний запис із бази даних та оновлює список доступних паролів.

Окрім основних функцій роботи з паролями, система містить розділ налаштувань. У ньому користувач може змінити тему оформлення інтерфейсу, а також вийти із системи.

Таким чином, розроблена структура дозволяє створити клієнт-серверний менеджер паролів.

2.2 Розробка блоку реєстрації та автентифікації додатка

Для забезпечення доступу користувачів до системи в клієнтському застосунку реалізовано два окремих класи: Authentication та Registration. Обидва класи успадковуються від компонента CTkFrame бібліотеки CustomTkinter та відповідають за взаємодію користувача із системою на етапі входу в додаток.

Код класу Authentication розміщений у додатку А пункті “Код реалізації класу Authentication”.

Клас Authentication реалізує сторінку автентифікації користувача. Основними елементами сторінки є поля введення логіна та пароля, кнопка входу в систему, а також кнопка переходу до сторінки реєстрації.

Після запуску сторінки виконується спроба встановлення з'єднання із сервером. У разі виникнення помилки відповідне повідомлення відображається користувачу.

Перед надсиланням облікових даних на сервер, виконується перевірка введеної користувачем інформації. Спочатку перевіряється, чи заповнені обидва обов'язкові поля: логін та пароль. Далі контролюється довжина логіна, яка повинна становити від 2 до 50 символів, а також довжина пароля, яка повинна знаходитися в межах від 4 до 25 символів. Якщо хоча б одна з перевірок завершується невдало, користувачу відображається відповідне повідомлення про помилку, а передача даних на сервер не виконується.

Основна взаємодія із сервером реалізована у функції authentication.

```
def authentication(self, login, password):  
    request = self.__app.__client.message(header="authentication",  
login=login, password=password)  
    result = self.__app.__client.send(request)  
    if result:  
        show_message(self.__message_label, result)  
        return  
    answer = self.__app.__client.get()
```

Принцип роботи функції полягає у формуванні повідомлення з типом

					БР.КІ-41.00.00.000 ПЗ	Арк.
Змн.	Арк.	окум.	Підпис	Дат		22

запиту “authentication”, логіном та паролем користувача. Після цього повідомлення передається серверу. Сервер перевіряє облікові дані у базі даних та повертає результат автентифікації. У випадку успішного входу клієнт отримує токен доступу, який використовується для подальшої взаємодії із сервером, після чого користувач перенаправляється до головного вікна програми.

На рисунку 2.5 зображена сторінка автентифікації користувача.

Автентифікація

Введь логін та пароль для підтвердження входу

Логін Пароль

Увійти

Якщо Ви немає акаунта - можете зареєструватись!

Зареєструватись

Рисунок 2.5 – Сторінка автентифікації користувача

Клас Registration реалізує сторінку створення нового облікового запису. Інтерфейс сторінки містить поля для введення логіна, пароля, електронної адреси та підтвердження пароля, а також кнопку реєстрації користувача.

Код класу Registration розміщений у додатку А пункті “Код реалізації класу Registration”.

Перед виконанням реєстрації здійснюється більш розширена перевірка введених даних. Насамперед контролюється заповнення всіх полів форми: логіна, пароля, електронної адреси та поля підтвердження пароля. Після цього перевіряється довжина логіна (від 2 до 50 символів) та пароля (від 4 до 25 символів). Наступним етапом є перевірка збігу введеного пароля та його

					БР.КІ-41.00.00.000 ПЗ	Арк.
Змн.	Арк.	окум.	Підпис	Дат		23

підтвердження. Якщо значення відрізняються, процес реєстрації припиняється та користувач отримує повідомлення про помилку.

Окрему увагу приділено перевірці електронної адреси. Спочатку контролюється її довжина, яка повинна становити від 6 до 254 символів. Після цього виконується перевірка формату за допомогою регулярного виразу. Така перевірка дозволяє переконатися, що адреса містить ім'я користувача, символ “@”, доменне ім'я та коректне доменне розширення. Лише після успішного проходження всіх перевірок формується запит на реєстрацію та надсилається серверу.

На рисунку 2.6 зображена сторінка реєстрації користувача.

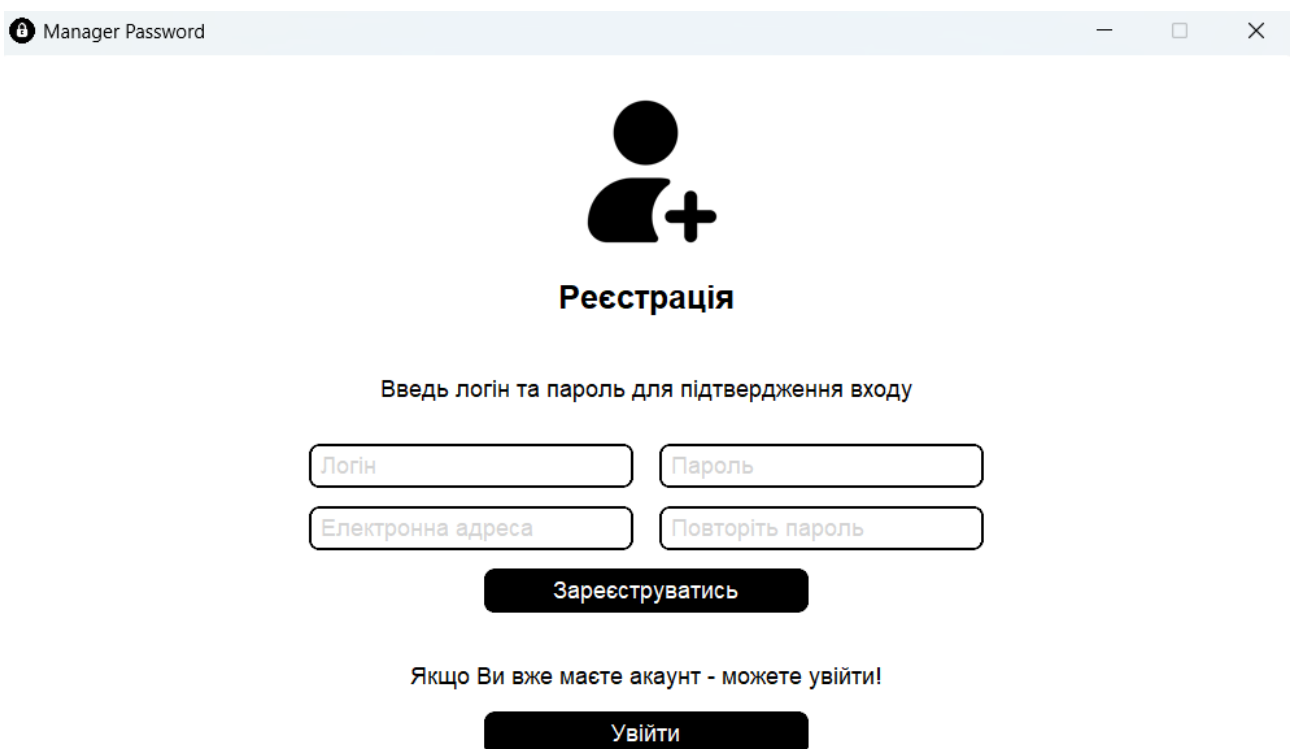


Рисунок 2.6 – Сторінка реєстрації користувача

Основний механізм взаємодії із сервером реалізований у функції registration.

```
def registration(self, login, password, email):  
    request = self.__app.__client.message(header="registration",  
    login=login, password=password, email=email)
```

					БР.КІ-41.00.00.000 ПЗ	Арк.
Змн.	Арк.	окум.	Підпис	Дат		24

```

result = self.__app.__client.send(request)
if result:
    show_message(self.__message_label, result)
    return
answer = self.__app.__client.get()

```

Функція створює мережевий запит типу “registration”, який містить дані нового користувача. Після надсилання повідомлення сервер перевіряє наявність користувача у базі даних та, якщо обліковий запис відсутній, створює новий. У разі успішної реєстрації сервер повертає токен доступу, який зберігається у клієнтському застосунку. Після цього користувач автоматично переходить до основного інтерфейсу програми без необхідності повторного входу.

В результаті, сторінки автентифікації та реєстрації реалізують початковий етап роботи користувача із системою.

2.3 Розробка блоків генерації та додавання паролів

Для розміщення основних сторінок та забезпечення навігації між ними використовується клас App. Він є головним класом клієнтського застосунку та успадковується від класу STk бібліотеки CustomTkinter. Він відповідає за ініціалізацію головного вікна програми, створення графічного інтерфейсу, керування сторінками застосунку та організацію взаємодії між окремими компонентами системи.

Код реалізації класу App, розміщений у додатку А пункті “Код реалізації класу App”.

Під час створення об'єкта класу налаштовуються параметри головного вікна, завантажуються теми оформлення, створюються сторінки авторизації, реєстрації та основні функціональні сторінки менеджера паролів. Крім того, у класі зберігаються службові дані поточного користувача, зокрема токен автентифікації та логін.

Для оформлення додатку використовується файл “theme.json” який містить

					БР.КІ-41.00.00.000 ПЗ	Арк.
Змн.	Арк.	окум.	Підпис	Дат		25

описані параметри теми, які застосовуватиметься до віджетів, його вміст розміщений у додатку Б пункті “Вміст файлу “theme.json””.

Також для коректної роботи, клас використовує функцію знаходження файлу проекту, вона розміщується нижче.

```
import os
import sys
def resource_path(relative_path):
    if hasattr(sys, "_MEIPASS"):
        base_path = sys._MEIPASS
    else:
        base_path = os.path.dirname(os.path.abspath(__file__))
        base_path = base_path[:-6]
    return os.path.join(base_path, relative_path)
```

Функція приймає відносний шлях до необхідного ресурсу та повертає його абсолютний шлях у файловій системі. На початку роботи виконується перевірка наявності атрибута `_MEIPASS` у модулі `sys`. Цей атрибут автоматично створюється програмою PyInstaller під час запуску зібраного виконуваного файлу та містить шлях до тимчасового каталогу, у який були розпаковані всі ресурси застосунку.

Якщо програма запущена у вигляді виконуваного файлу, функція використовує значення `_MEIPASS` як кореневий каталог ресурсів. Якщо ж застосунок запускається безпосередньо з вихідного коду Python, визначається каталог поточного файлу та формується шлях до кореневої директорії проекту.

Після визначення базового каталогу функція об'єднує його з переданим відносним шляхом за допомогою методу `os.path.join()` та повертає готовий абсолютний шлях до потрібного ресурсу.

Для виведення повідомлення, зокрема на всіх сторінках, використовується функція `show_message()`, її реалізація наведена нижче.

```
def show_message(master, message):
    master.configure(text=message)
```

					БР.КІ-41.00.00.000 ПЗ	Арк.
Змн.	Арк.	окум.	Підпис	Дат		26

Функція приймає два параметри: посилання на графічний елемент інтерфейс (master) та текст повідомлення (message) Після виклику функції виконується зміна властивості text переданого елемента за допомогою методу configure, внаслідок чого користувач бачить нове повідомлення на екрані.

Основним елементом взаємодії із сервером є об'єкт Client, який створюється під час ініціалізації програми:

```
self._client = Client()
```

Саме через цей об'єкт усі сторінки застосунку надсилають запити до сервера та отримують відповіді.

Однією з найважливіших функцій класу є метод get_passwords, який використовується для отримання списку паролів поточного користувача після пройденої автентифікації.

```
def get_passwords(self, is_show_message=True):
    request = self._client.message(header="get",
    token=self.__token)
    result = self._client.send(request)
    if not result:
        result = self._client.get()
    else:
        result = None
    self.__pages["RefugePage"].create_password_frames(result,
    is_show_message)
```

На першому етапі формується запит із заголовком “get” та токеном користувача, що використовується для підтвердження прав доступу.

Після формування повідомлення воно надсилається серверу через метод send. Сервер перевіряє отриманий токен, знаходить усі паролі, що належать користувачу, та повертає їх клієнту у вигляді відповіді.

Отримані дані передаються до сторінки сховища паролів, де автоматично створюються графічні елементи для відображення записів користувача.

Наступним кроком реалізуємо клас GeneratePasswordPage, що

					БР.КІ-41.00.00.000 ПЗ	Арк.
Змн.	Арк.	окум.	Підпис	Дат		27

відповідатиме за головну сторінку додатку. Він реалізує сторінку генерації паролів. Основним призначенням даного класу є створення складних паролів відповідно до обраних користувачем параметрів.

Код класу `GeneratePasswordPage` розміщений в додатку А пункті “Код реалізації класу `GeneratePasswordPage`”.

Інтерфейс сторінки складається з поля відображення пароля, повзунка вибору довжини, набору прапорців для вибору допустимих символів та кнопки генерації. Користувач може включати до пароля малі літери, великі літери, цифри та спеціальні символи.

На рисунку 2.7 зображена сторінка генерації пароля.

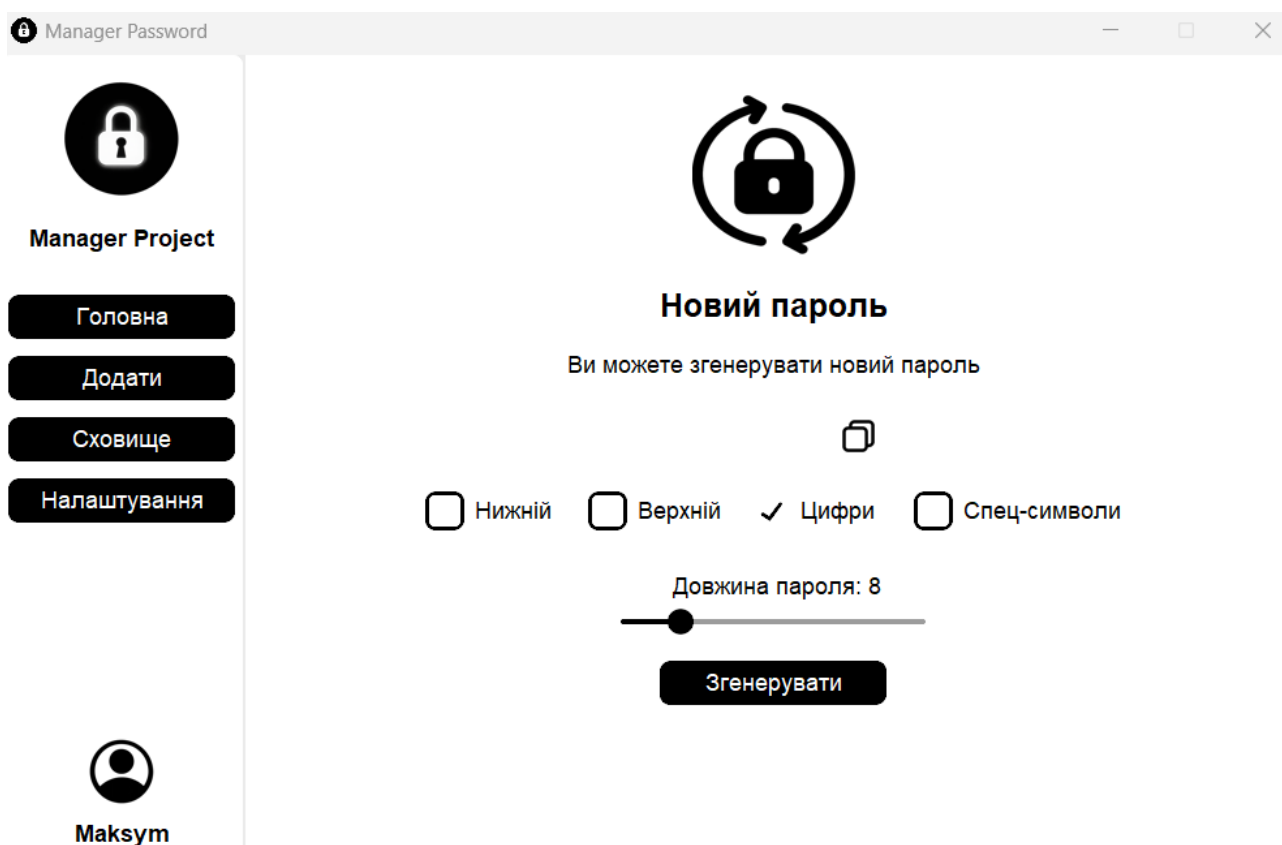


Рисунок 2.7 – Сторінка генерації пароля

Логіка створення пароля реалізована у функції `generate_password`, нижче зображена її реалізація.

```
def generate_password(self):
    self.__options["lower"] = self.__checkbox_frame_lower.get()
```

					БР.КІ-41.00.00.000 ПЗ	Арк.
Змн.	Арк.	окум.	Підпис	Дат		28

```

self.__options["upper"] = self.__checkbox_frame_upper.get()
self.__options["digits"] = self.__checkbox_frame_digits.get()
self.__options["punctuation"] =
self.__checkbox_frame_punctuation.get()
self.__chars = set_options(self.__options)
self.__length = int(self.__generate_slider.get())
self.__password = generate_password(self.__length,
self.__chars)

```

Під час виконання функції зчитуються значення всіх прапорців та формується набір допустимих символів. Далі визначається довжина майбутнього пароля відповідно до положення повзунка. На основі отриманих параметрів викликається функція генерації випадкового пароля, ця функція наведена нижче.

```

import string
import secrets
def set_options(options):
    chars = ""
    chars += string.ascii_lowercase if options['lower'] else ""
    chars += string.ascii_uppercase if options['upper'] else ""
    chars += string.digits if options['digits'] else ""
    chars += string.punctuation if options['punctuation'] else ""
    return chars
def generate_password(length, chars):
    try:
        password = "".join(secrets.choice(chars) for _ in
range(length))
    except:
        password = None
    return password

```

Функція set_options формує рядок доступних символів для генерації пароля залежно від параметрів, заданих користувачем. Якщо відповідний параметр увімкнений, до набору додаються:

- малі літери;
- великі літери
- цифри;

					БР.КІ-41.00.00.000 ПЗ	Арк.
Змн.	Арк.	окум.	Підпис	Дат		29

- спеціальні символи.

У результаті функція повертає об'єднаний набір символів, з якого буде створюватися пароль.

Функція `generate_password` відповідає за безпосередню генерацію пароля. Вона створює випадковий рядок заданої довжини, обираючи кожен символ за допомогою `secrets.choice`. Якщо виникає помилка (наприклад, порожній набір символів), функція повертає `None`.

Бібліотека `secrets` є більш безпечною альтернативою `random` для криптографічних задач, зокрема генерації паролів. Основна різниця полягає у рівні безпеки генерації випадкових значень. Бібліотека `random` використовує псевдовипадковий генератор, який не призначений для криптографічної стійкості. Це означає, що послідовність значень теоретично може бути передбачена при достатньому аналізі або знанні внутрішнього стану генератора.

Натомість `secrets` створена спеціально для задач безпеки. Вона використовує джерела випадковості операційної системи, які є криптографічно стійкими. Це робить згенеровані значення практично непередбачуваними, що особливо важливо для паролів, токенів та ключів доступу.

Таким чином, використання `secrets` у даному випадку забезпечує значно вищий рівень захисту даних користувача та відповідає вимогам безпечної генерації паролів.

Можливість додавання паролів реалізована в класі `AddPasswordPage`, що створює сторінку додавання нових паролів до сховища. Інтерфейс сторінки містить поля введення пароля та назви сервісу, текстове поле для опису пароля та кнопку додавання запису.

Код класу `AddPasswordPage` розміщений у додатку А пункті “Код реалізації класу `AddPasswordPage`”.

Перед надсиланням даних виконується перевірка їх коректності. Система контролює такі обмеження:

- пароль є обов'язковим полем;
- назва сервісу є обов'язковим полем;

					БР.КІ-41.00.00.000 ПЗ	Арк.
Змн.	Арк.	окум.	Підпис	Дат		30

- довжина назви сервісу повинна бути від 2 до 25 символів;
- довжина пароля повинна бути від 4 до 25 символів;
- довжина опису не повинна перевищувати 250 символів.

Після успішної перевірки викликається функція взаємодії із сервером `add_password`, фрагмент її коду зображений нижче.

```
def add_password(self, password, service, description):
    request = self.__app.__client.message(header="add",
    password=password, service=service, description=description,
    token=self.__app.get_token())
    result = self.__app.__client.send(request)
    if result:
        show_message(self.__message_label, result)
        return
    answer = self.__app.__client.get()
```

Функція формує мережевий запит типу “Add”, який містить пароль, назву сервісу, опис та токен авторизації користувача. Після цього сформований пакет даних надсилається серверу.

На сервері виконується перевірка токена користувача та пошук аналогічного запису в базі даних. Якщо пароль для зазначеного сервісу відсутній, сервер додає новий запис до SQLite та повертає повідомлення про успішне завершення операції. У випадку виявлення дубліката повертається повідомлення про помилку.

Після успішного додавання запису клієнт автоматично очищає поля введення та викликає метод `self.__app.get_passwords`. Цей виклик ініціює новий запит до сервера для отримання актуального списку паролів. Завдяки цьому вміст сховища одразу синхронізується з базою даних і користувач бачить оновлену інформацію без необхідності повторного входу в систему.

На рисунку 2.8 зображена сторінка додавання нового пароля до сховища.

					БР.КІ-41.00.00.000 ПЗ	Арк.
Змн.	Арк.	окум.	Підпис	Дат		31

Рисунок 2.8 – Сторінка додавання пароля

В результаті, клас `GeneratePasswordPage` відповідає за локальну генерацію паролів, а клас `AddPasswordPage` проводить їх перевірку, та додає до сховища.

2.4 Розробка блоків отримання паролів та налаштування додатка

Для перегляду доданих паролів, використовується клас `RefugePage`, що реалізує сторінку сховища в застосунку для керування збереженими паролями. Основним призначенням даного класу є відображення всіх паролів користувача, отриманих із віддаленого сервера, а також забезпечення пошуку та фільтрації цих записів.

Код класу `RefugePage` розміщений в додатку А пункті “Код реалізації класу `RefugePage`”.

Інтерфейс сторінки складається з заголовкової частини, яка містить логотип та опис, блоку пошуку та основної прокручуваної області, в якій відображаються елементи паролів. Кожен такий елемент представляє один запис

					БР.КІ-41.00.00.000 ПЗ	Арк.
Змн.	Арк.	окум.	Підпис	Дат		32

і містить інформацію про сервіс, пароль та додатковий опис.

Основна взаємодія із сервером реалізована через метод `create_password_frames`, який отримує відповідь від сервера та формує внутрішній стан сторінки. Отримані дані перевіряються на коректність, після чого, зберігаються у внутрішньому списку та використовуються для побудови графічних елементів інтерфейсу.

Нижче реалізовано ключовий етап обробки серверної відповіді: дані спочатку сортуються за назвою сервісу, після чого для кожного запису створюється відповідний графічний компонент.

```
if answer.get("header") == "get-respond" and answer.get("answer") and
answer.get("passwords"):
    self.__passwords = answer["passwords"]
    self.__passwords.sort(key=lambda x: x[1])
    for element in self.__passwords:
        frame = PasswordFrame(parent=self,
                               master=self.__refuge_frame, identifier=element[0],
                               password=element[1], service=element[2],
                               description=element[3], app=self.__app)
        self.__frames.append(frame)
```

Пошук паролів реалізує локальну фільтрацію без повторного звернення до сервера. Користувач вводить текст у поле пошуку, після чого викликається метод `filter_password_frames`, який перевіряє введені значення та або повертає повний список паролів, якщо користувач не ввів слово для пошуку, або передає запит на фільтрацію, якщо ввів.

Безпосередній пошук виконується у методі `find_passwords`, який здійснює порівняння введенного тексту з назвою сервісу кожного запису. Порівняння виконується у нечутливому до регістру режимі, що дозволяє знаходити збіги незалежно від написання. Результатом роботи є відфільтрований список записів, який передається до методу `render_password_frames`, де відбувається оновлення інтерфейсу.

Після отримання результатів фільтрації викликається метод оновлення

					БР.КІ-41.00.00.000 ПЗ	Арк.
Змн.	Арк.	окум.	Підпис	Дат		33

інтерфейсу, який видаляє попередні елементи та створює нові відповідно до відфільтрованого списку.

На рисунку 2.9 зображена сторінка перегляду збережених паролів.

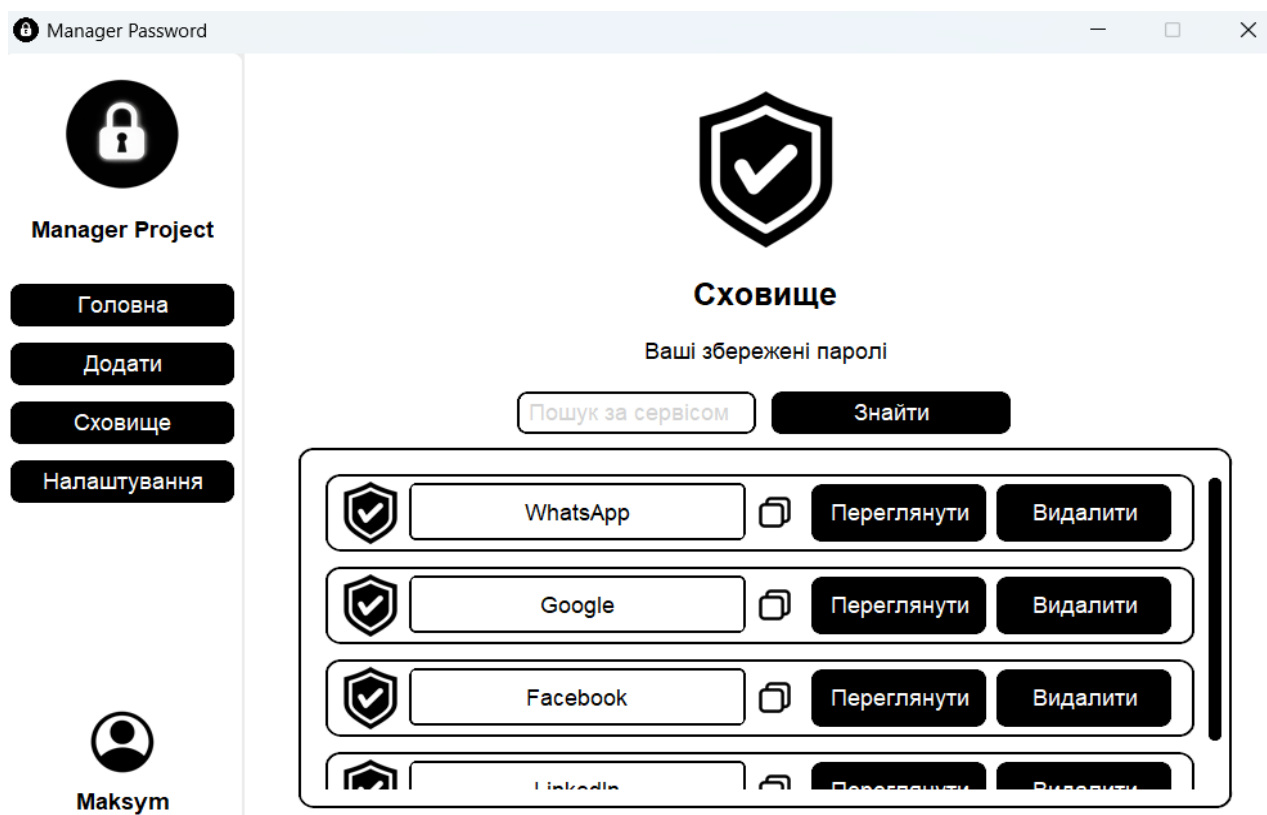


Рисунок 2.9 – Сторінка перегляду збережених паролів

Можливість зміни та видалення пароля дозволяє клас PasswordFrame, який і відображає кожен пароль на сторінці сховища.

Інтерфейс кожного елемента містить іконку сервісу, скорочену назву сервісу, а також кнопки для основних дій.

Код класу PasswordFrame розміщений в додатку А пункті “Код реалізації класу PasswordFrame”.

Основна логіка взаємодії користувача з паролем реалізована через набір методів, які відповідають за різні операції над записом.

Функція копіювання пароля дозволяє перенесення значення у буфер обміну та виведення повідомлення користувачу, її реалізація наведена нижче.

```
def copy_password(self):
    self.clipboard_clear()
    self.clipboard_append(self.__password)
    show_message(self.__parent._message_label,
        "Пароль скопійований до буфера обміну")
```

Окремим функціональним блоком є перегляд та редагування пароля. Для цього створюється модальне вікно `STkToplevel`, у якому відображаються поточні дані запису. Користувач може змінити пароль або опис та зберегти зміни. Перед збереженням виконується перевірка введених даних, включаючи обов’язковість пароля, допустиму довжину та обмеження для опису.

На рисунку 2.10 зображена сторінка перегляду та редагування пароля.

Рисунок 2.10 – Сторінка перегляду та редагування пароля

Після успішної перевірки даних виконується взаємодія із сервером через метод `update_password`, який формує запит типу “update” та передає ідентифікатор запису, нові значення та токен авторизації. Фрагмент коду взаємодії із сервером наведений нижче.

```
request = self.__app._client.message(header="update",
    identifier=self.__identifier, password=password,
```

```

description=description,
token=self.__app.get_token())
result = self.__app._client.send(request)
answer = self.__app._client.get()

```

У разі успішної відповіді серверу дані оновлюються, після чого ініціюється повторне завантаження списку паролів через `self.__app.get_passwords`.

Аналогічно реалізовано функціонал видалення запису. При натисканні кнопки відкривається вікно підтвердження, після чого формується запит “delete” до сервера. У разі успішного видалення елемент інтерфейсу видаляється через `destroy`, а список паролів оновлюється.

На рисунку 2.11 зображена сторінка видалення пароля.

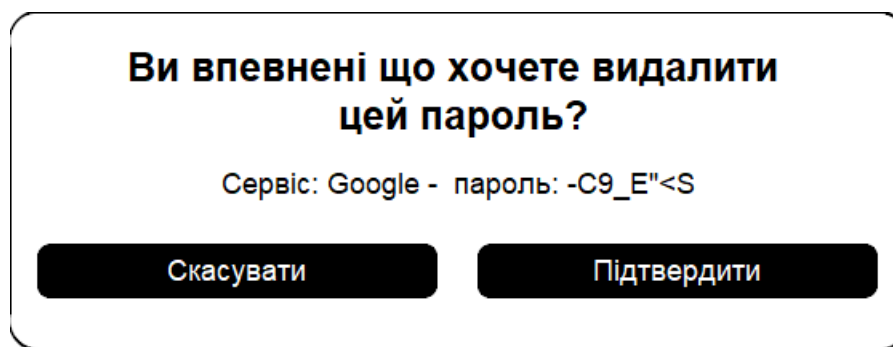


Рисунок 2.11 – Сторінка видалення пароля

В результаті клас `PasswordFrame` відображає додаткові дані щодо паролю, та надає можливість копіювання, перегляду, редагування та видалення запису із синхронізацією з сервером.

Для зміни налаштувань додатку застосовується клас `SettingsPage`, який об’єднує функціонал зміни теми оформлення, а також виходу з акаунта.

Інтерфейс сторінки налаштувань складається з заголовкової частини, блоку налаштувань оформлення та блоку керування акаунтом. Основна частина містить елементи керування, зокрема випадаючий список для вибору теми інтерфейсу, а також кнопку виходу з акаунта.

Код реалізації класу `SettingsPage` знаходиться в Додатку А пункті “Код реалізації класу `SettingsPage`”.

На рисунку 2.12 зображена сторінка налаштування додатку.

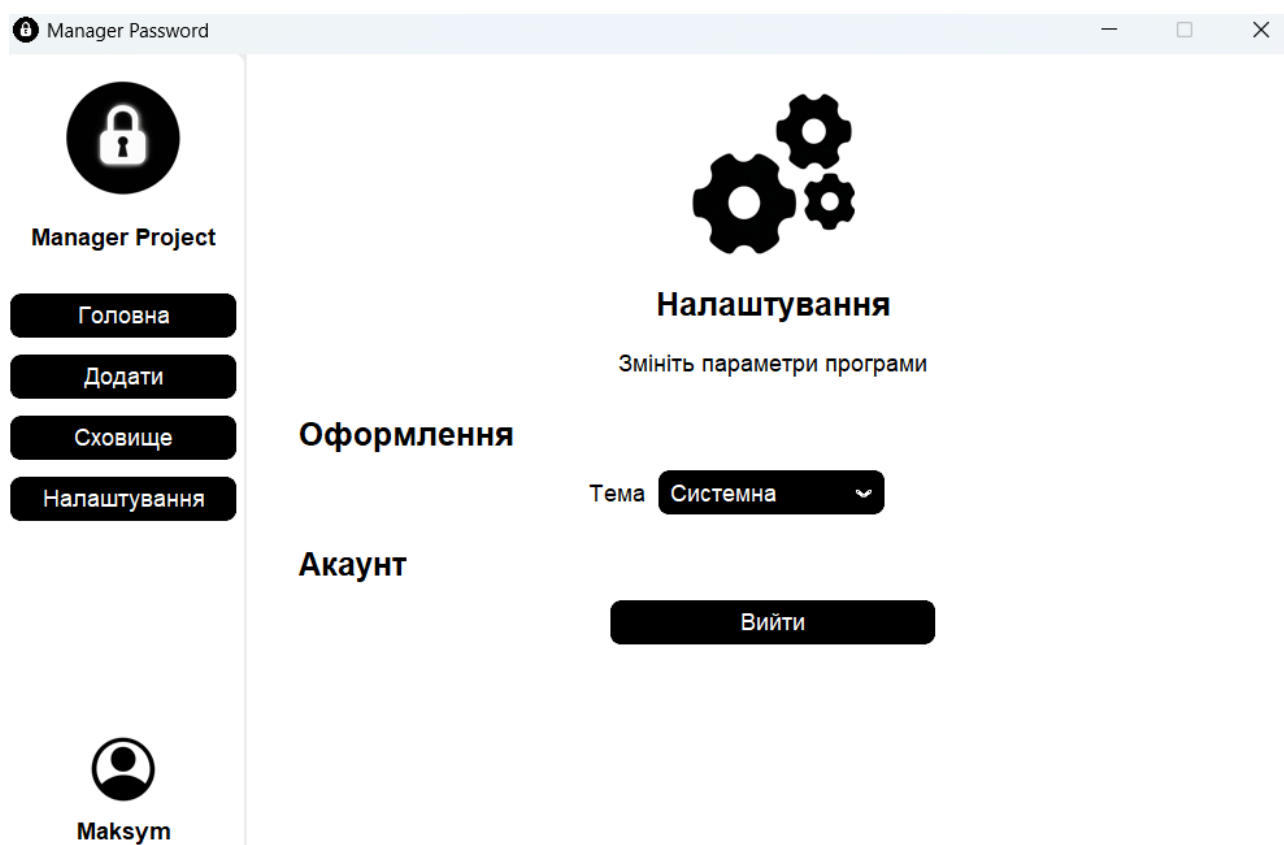


Рисунок 2.12 – Сторінка налаштування додатку

Одним із ключових елементів сторінки є вибір теми інтерфейсу. Для цього використовується віджет `CTkOptionMenu`, який дозволяє користувачу обрати один із варіантів: світлу, темну або системну тему. Вибір користувача обробляється методом `set_theme_option`, який змінює вигляд застосунку та відображає відповідне повідомлення.

Окрему важливу роль у класі відіграє функція виходу з акаунта. Вона реалізована у вигляді кнопки виходу, яка викликає метод `show_login` головного застосунку. Цей метод переводить користувача на екран автентифікації та завершує поточну сесію роботи.

Для збереження та отримання налаштувань використовується файл “`settings.json`”, який містить параметри конфігурації застосунку. Робота з ним реалізована через дві функції: `get_value_settings` та `set_value_settings`.

Функція отримання значення читає JSON-файл, завантажує дані та

повертає значення за вказаним ключем, її код наведений нижче.

```
def get_value_settings(setting_name):
    path = resource_path(os.path.join("settings",
    "settings.json"))
    with open(path, "r", encoding="utf-8") as file:
        data = json.load(file)
    if data.get(setting_name):
        return data[setting_name]
```

Функція запису, у свою чергу, оновлює відповідний параметр і перезаписує файл, її вміст наведений нижче.

```
def set_value_settings(setting_name, value):
    path = resource_path(os.path.join("settings",
    "settings.json"))
    with open(path, "r", encoding="utf-8") as file:
        data = json.load(file)
    data[setting_name] = value
    with open(path, "w", encoding="utf-8") as file:
        json.dump(data, file, indent=4, ensure_ascii=False)
```

Файл “settings.json” має просту структуру та зберігає поточну тему інтерфейсу, його вміст наведений нижче.

```
{
    "theme": "System"
}
```

Так було реалізовано інтерфейсні компоненти додатку: RefugePage, PasswordFrame та SettingsPage. RefugePage дозволяє відображення та пошук збережених паролів, PasswordFrame реалізує керування окремим записом (перегляд, копіювання, редагування та видалення), а SettingsPage відповідає за налаштування застосунку та вихід з акаунта. Усі класи разом формують інтерфейс для роботи з паролями та взаємодіють із сервером і локальними

					БР.КІ-41.00.00.000 ПЗ	Арк.
Змн.	Арк.	окум.	Підпис	Дат		38

налаштуваннями.

2.5 Розробка блоків зв'язку із сервером

Для прямої взаємодії із сервером, створений окремий клас Client. Він реалізує клієнтську частину застосунку та відповідає за безпечну взаємодію із сервером через TLS-з'єднання. Його основним призначенням є формування запитів, передача даних на сервер, отримання відповіді та її декодування у зручний для використання формат.

Код класу Client розміщений в додатку Б пункті “Код реалізації класу Client”.

Під час ініціалізації клас завантажує сертифікати ca.crt, client.crt та client.key (їх можна створити використовуючи OpenSSL) та створює TLS-контекст через клас TLSContext. На його основі формується з'єднання із сервером за допомогою SecureSocket.

Основна логіка формування запитів реалізована у методі message, який створює структуру даних залежно від типу операції:

```
def message(self, header, login="", email="", identifier="",
password="", service="", description="", token=""):
    if header == "authentication":
        return {"header": header, "login": login, "password":
password}
    elif header == "add":
        return {"header": header, "password": password,
"service": service, "description": description, "token":
token}
    elif header == "update":
        return {"header": header, "identifier": identifier,
"password": password, "description": description,
"token": token}
```

Перед відправкою всі дані кодуються у формат JSON у методі encode, після чого передаються через TLS-з'єднання методом send. Отримана відповідь

					БР.КІ-41.00.00.000 ПЗ	Арк.
Змн.	Арк.	окум.	Підпис	Дат		39

сервера зчитується методом `get` і декодується назад у Python-словник через `decode`.

Низькорівневий механізм взаємодії із сервером реалізує клас `SecureSocket`. Його основне призначення полягає у створенні стабільного каналу передачі даних, забезпеченні відправки та отримання повідомлень, а також обробці можливих мережових помилок.

Код класу `SecureSocket` розміщений у додатку Б пункті “Код реалізації класу `SecureSocket`”.

Під час ініціалізації клас зберігає параметри підключення (хост, порт) та TLS-контекст, після чого використовує їх для створення захищеного сокета.

З’єднання встановлюється методом `connect`, який створює звичайний TCP-сокет і обгортає його в SSL/TLS-обгортку. У разі помилок підключення (відсутність сервера, таймаут або мережові збої) метод повертає відповідне повідомлення про помилку.

Метод `send` відповідає за передачу даних на сервер. Перед відправкою перевіряється стан з’єднання, і за потреби воно автоматично відновлюється. У разі успішного підключення дані надсилаються через `sendall`, а всі можливі мережові виключення перехоплюються та обробляються із поверненням зрозумілих повідомлень про помилку.

Метод `get` реалізує отримання даних від сервера з використанням буферизації. Повідомлення читаються частинами до моменту появи символу нового рядка, який використовується як розділювач кінця повідомлення. Після цього отриманий рядок очищається та повертається у вигляді тексту.

Також реалізовано метод `close`, який завершує TLS-з’єднання та очищає ресурси. Усі критичні помилки (розрив з’єднання, таймаути, скидання сервером) перехоплюються та переводять стан клієнта у неактивний.

Для створення та налаштування захищеного з’єднання використовується клас `TLSContext`.

Його призначення полягає у конфігурації параметрів SSL/TLS, завантаженні сертифікатів та встановленні політики безпеки з’єднання. Під час

					БР.КІ-41.00.00.000 ПЗ	Арк.
Змн.	Арк.	окум.	Підпис	Дат		40

ініціалізації клас отримує шляхи до трьох основних файлів: сертифіката центру сертифікації (ca_cert), клієнтського сертифіката (client_cert) та приватного ключа (client_key). Ці дані зберігаються для подальшого використання при створенні TLS-контексту.

Код класу TLSContext розміщений у додатку Б пункті “Код реалізації класу TLSContext”.

Основна логіка реалізована у методі build_context, який створює об’єкт SSLContext з протоколом TLS_CLIENT і налаштовує його для безпечної роботи.

Нижче наведений код методу build_context.

```
def build_context(self):
    self.__context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)
    self.__context.load_verify_locations(cafile=self.__ca_cert)
    self.__context.load_cert_chain(certfile=self.__client_cert,
    keyfile=self.__client_key)
    self.__context.minimum_version = ssl.TLSVersion.TLSv1_2
    return self.__context
```

У процесі налаштування виконується завантаження довіреного сертифіката центру сертифікації, підключення клієнтського сертифіката та приватного ключа, це дозволяє взаємну автентифікацію між клієнтом і сервером. Також встановлюється мінімальна версія протоколу TLS – 1.2.

Загалом структура додатку готова, проте варто реалізувати код, що запускатиме весь додаток, його реалізація наведена нижче.

```
from gui.app import App
if __name__ == "__main__":
    app = App()
    app.mainloop()
```

Спочатку імпортується клас App, який містить усю логіку побудови інтерфейсу, навігації між сторінками та взаємодії з іншими компонентами

					БР.КІ-41.00.00.000 ПЗ	Арк.
Змн.	Арк.	окум.	Підпис	Дат		41

системи. Далі перевіряється умова, що гарантує виконання коду лише у випадку прямого запуску файлу, а не при імпорті як модуля.

Після цього створюється екземпляр класу App, який ініціалізує всі необхідні елементи інтерфейсу та підключення до внутрішніх модулів програми. Завершальним етапом є виклик методу mainloop, який запускає головний цикл обробки подій графічного інтерфейсу.

Перейдемо до реалізації серверної частини додатку.

2.6 Розробка модуля серверної взаємодії з клієнтом та базою даних

Сервер очікує клієнтського з'єднання. Коли з'єднання встановлене, він приймає повідомлення від клієнта і, в залежності від запитів відповідає.

Робота сервера починається із запуску та переходу в режим очікування підключення клієнта. Після встановлення з'єднання виконується оновлення токена користувача та визначення типу отриманого запиту.

Залежно від запиту сервер може виконувати автентифікацію, реєстрацію користувача, додавання, отримання, редагування або видалення паролів. Для операцій із паролями обов'язково виконується перевірка токена доступу. Якщо токен не знайдено або він недійсний, сервер повертає повідомлення про відмову.

Під час реєстрації та автентифікації здійснюється перевірка даних користувача в базі даних. У разі успішного виконання користувачу видається токен доступу. Для операцій над паролями сервер звертається до бази даних для пошуку, додавання, оновлення або видалення записів.

Після завершення будь-якої операції сервер виконує журналювання події та завершує поточний цикл обробки запиту.

Реалізуємо основу роботи сервера класом Handler.

Код класу Handler розміщений в додатку В пункті “Код реалізації класу Handler”.

					БР.КІ-41.00.00.000 ПЗ	Арк.
Змн.	Арк.	окум.	Підпис	Дат		42

Його головне призначення полягає у прийомі даних від клієнта, визначенні типу запиту, роботи із класом взаємодії з базою даних та формуванні відповіді у форматі JSON.

Після встановлення з'єднання метод `handle` запускає цикл прийому повідомлень, у якому дані зчитуються, декодуються та передаються до методу `route`, що виконує їх маршрутизацію. Нижче зображений фрагмент коду отримання повідомлення від клієнта.

```
while True:
    request = self.get()
    if not request:
        break
    request = self.decode(request)
    self.route(request)
```

Метод `route` відповідає за визначення типу отриманого повідомлення та вибір відповідної функції обробки.

Система підтримує такі типи запитів:

- автентифікація користувача;
- реєстрація нового користувача;
- додавання пароля;
- отримання списку паролів;
- редагування пароля;
- видалення пароля.

Для всіх операцій, окрім реєстрації та входу, виконується перевірка токена авторизації користувача. Сервер порівнює отриманий токен із токенами, що зберігаються у базі даних у вигляді Argon2-хешів.

Argon2 це сучасний алгоритм хешування паролів, який спеціально розроблений для безпечного зберігання паролів у базах даних, він є стійким до атак перебору, оскільки потребує значного обсягу пам'яті, що ускладнює використання GPU та спеціалізованих атакувальних пристроїв.

					БР.КІ-41.00.00.000 ПЗ	Арк.
Змн.	Арк.	окум.	Підпис	Дат		43

Функція authentication відповідає за перевірку облікових даних користувача. Після пошуку користувача у базі даних виконується перевірка пароля за допомогою алгоритму Argon2. У разі успішної автентифікації на основі пароля та криптографічної солі формується ключ КЕК, який використовується для розшифрування ключа DEK. Після цього генерується токен доступу, який хешується, зберігається в базі даних та повертається клієнту для подальшої авторизації.

Функція registration виконує створення нового користувача. Під час реєстрації генерується криптографічна сіль, створюються ключі КЕК і DEK, пароль хешується за допомогою Argon2, а персональні дані шифруються перед збереженням у базі даних. Після успішної реєстрації також створюється токен доступу.

Функції add_password, update_password та get_passwords забезпечують роботу зі сховищем паролів. Перед записом до бази даних пароль та опис шифруються за допомогою ключа DEK, а під час отримання інформації виконується їх розшифрування. Це дозволяє захистити конфіденційні дані навіть у разі компрометації бази даних.

Кожна операція перевіряє результат виконання SQL-запиту та формує відповідь у форматі JSON для передачі клієнту. Окремий метод generate_token відповідає за створення унікального токена доступу. Він генерує випадковий рядок та перевіряє його у базі даних на унікальність. Для генерації використовується модуль secrets.

Для передачі даних використовується JSON-формат із розділенням повідомлень символом нового рядка. Метод send кодує відповідь і передає її клієнту, а метод get накопичує вхідні дані у буфері до повного отримання повідомлення.

Для всіх криптографічних методів, реалізований окремий модуль “crypto.py”, що реалізує механізми шифрування файлів та конфіденційних даних, що використовуються в менеджері паролів.

					БР.КІ-41.00.00.000 ПЗ	Арк.
Змн.	Арк.	окум.	Підпис	Дат		44

Вміст файлу “crypto.py” знаходиться в додатку В пункті “Вміст файлу “crypto.py””.

Основою захисту є алгоритм AES-256-GCM, який забезпечує одночасно конфіденційність даних та контроль їх цілісності.

Для формування криптографічних ключів використовується алгоритм Scrypt, який створює ключ шифрування на основі пароля користувача та випадкової солі.

Ключ DEK шифрується за допомогою КЕК функціями wrap_dek та unwrap_dek, їх реалізації наведена нижче.

```
def wrap_dek(dek: bytes, kek: bytes) -> bytes:
    aes = AESGCM(kek)
    nonce = os.urandom(12)
    return nonce + aes.encrypt(nonce, dek, None)

def unwrap_dek(wrapped: bytes, kek: bytes) -> bytes:
    nonce = wrapped[:12]
    ciphertext = wrapped[12:]
    aes = AESGCM(kek)
    return aes.decrypt(nonce, ciphertext, None)
```

Функція wrap_dek генерує випадковий nonce довжиною 12 байт і шифрує DEK за допомогою AES-GCM. У результаті повертається об'єднання nonce і ciphertext, де ciphertext містить зашифровані дані разом із автентифікаційним тегом.

Функція unwrap_dek виконує зворотну операцію: відокремлює nonce з перших 12 байт і розшифровує решту даних за допомогою того самого КЕК. Якщо дані були змінені або ключ неправильний, AES-GCM автоматично викликає помилку автентифікації.

Для захисту файлів використовуються функції encrypt_file та decrypt_file. Вони виконують шифрування та розшифрування файлів блоками по 64 КБ, це дозволяє працювати навіть із файлами великого розміру без значного використання оперативної пам'яті.

					БР.КІ-41.00.00.000 ПЗ	Арк.
Змн.	Арк.	окум.	Підпис	Дат		45

Для реалізації багатопотокового TLS-сервера використовується клас SecureSocket. Його основне призначення полягає у прийомі захищених з'єднань від клієнтів, наданні доступу до спільних ресурсів сервера та веденні журналів подій.

Код класу SecureSocket розміщений в додатку В пункті “Код реалізації SecureSocket”.

Під час ініціалізації сервер створює TLS-контекст, підключає базу даних, ініціалізує механізм хешування паролів та налаштовує чергу логів. Окремо запускається фоновий потік, який відповідає за запис логів у файли, що дозволяє не блокувати основний потік обробки запитів.

Нижче наведений фрагмент коду задання потоку черги.

```
self.__queue_logs = Queue()
Thread(target=self.logs_worker, daemon=True).start()
```

Ключовою особливістю класу є перевизначення методу get_request, який обгортає звичайне TCP-з'єднання у TLS:

```
def get_request(self):
    newsocket, addr = super().get_request()
    conn = self.__ssl_context.wrap_socket(newsocket,
    server_side=True)
    return conn, addr
```

Клас надає доступ до спільних компонентів сервера через методи get_db, get_password_hasher та gat_queue_logs. Такий підхід дає можливість обробнику запитів (Handler) використовувати єдині екземпляри бази даних і хешера паролів.

Окремий потік logs_worker постійно обробляє чергу логів. Повідомлення розділяються на звичайні та помилки й записуються у відповідні файли.

Загалом клас SecureSocket реалізує багатопотоковий TLS-сервер із системою хешування та механізмом логування.

					БР.КІ-41.00.00.000 ПЗ	Арк.
Змн.	Арк.	окум.	Підпис	Дат		46

Для взаємодії із клієнтами на верхньому рівні, використовується клас `Server`. Він є верхньорівневим компонентом серверної частини застосунку та відповідає за ініціалізацію, конфігурацію та запуск TLS-захищеного багатопотокового сервера. Він об'єднує всі основні модулі системи, зокрема налаштування SSL-контексту, мережевий сервер та обробник запитів.

Код класу `Server` розміщений в додатку В пункті “Код реалізації класу `Server`”.

Під час ініціалізації клас завантажує сертифікати сервера і центру сертифікації, створює TLS-контекст через `TLSContext`, а також зчитує параметри конфігурації (порт і хост) із налаштувань програми. У випадку некоректних значень використовуються значення за замовчуванням.

Методи `set_port` та `set_host` відповідають за отримання параметрів запуску сервера. Дані читаються з конфігураційного файлу “`settings.json`”, після чого перевіряються на коректність. Порт перевіряється на допустимий діапазон, а IP-адреса — через модуль `ipaddress`.

Нижче зображений вміст файлу “`settings.json`”.

```
{
    "length_token": 8,
    "hosts": "0.0.0.0",
    "port": 9000
}
```

У випадку помилок сервер автоматично переходить на стандартні значення (9000 та 0.0.0.0).

Метод `create` ініціалізує екземпляр сервера `SecureSocket`, передаючи йому адресу, порт, TLS-контекст і клас обробника запитів `Handler`.

Запуск сервера здійснюється через метод `start`, який запускає безперервне прослуховування з'єднань за допомогою `server_forever`. Метод `stop` використовується для коректного завершення роботи сервера.

					БР.КІ-41.00.00.000 ПЗ	Арк.
Змн.	Арк.	окум.	Підпис	Дат		47

Як і в клієнті, для конфігурації захищеного з'єднання створений клас TLSContext.

Клас TLSContext відповідає за створення та налаштування TLS-захищеного середовища на сервері. Його основне призначення полягає у забезпеченні шифрування переданих даних та перевірці автентичності клієнтів під час встановлення з'єднання.

Код класу TLSContext розміщений в додатку В пункті “Код реалізації класу TLSContext”.

У процесі ініціалізації клас отримує шляхи до серверного сертифіката, приватного ключа та сертифіката центру сертифікації. На основі цих даних формується TLS-контекст, який використовується для всіх вхідних підключень.

Метод build_context створює SSL-контекст із режимом CLIENT_AUTH, що дозволяє вимагати автентифікацію клієнта за сертифікатом. Додатково вмикається обов'язкова перевірка сертифікатів та завантажується довірений СА.

Нижче наведений фрагмент реалізації методу build_context.

```
self.__context=ssl.create_default_context(ssl.Purpose.CLIENT_AUTH)
self.__context.verify_mode = ssl.CERT_REQUIRED
self.__context.load_verify_locations(cafile=self.__ca_cert)
self.__context.load_cert_chain( certfile=self.__server_cert,
keyfile=self.__server_key )
self.__context.minimum_version = ssl.TLSVersion.TLSv1_2
self.__context.options |= ssl.OP_SINGLE_ECDH_USE
```

Для зв'язку сервера із базою даних використовується клас DBMS. Він призначений для взаємодії серверної частини системи з базою даних SQLite та забезпечує створення структури бази даних, виконання операцій додавання, отримання, оновлення та видалення даних, а також обробку SQL-запитів через механізм черги та окремого потоку.

Код класу DBMS розміщений у додатку В пункті “Код реалізації класу DBMS”.

Під час створення об'єкта класу ініціалізується шлях до бази даних, створюється черга запитів, відкривається з'єднання з SQLite та запускається окремий потік-обробник, який виконує SQL-запити асинхронно. Додатково завантажується налаштування довжини токена з конфігураційних параметрів системи.

Особливу роль відіграє метод `create_data_base`, який виконує створення структури бази даних. У цьому методі активується підтримка зовнішніх ключів та створюються таблиці користувачів, паролів і токенів у разі їх відсутності.

Фрагмент створення таблиці користувачів наведений нижче:

```
CREATE TABLE IF NOT EXISTS users (  
    user_id INTEGER PRIMARY KEY AUTOINCREMENT,  
    login TEXT NOT NULL UNIQUE,  
    password TEXT NOT NULL,  
    email TEXT NOT NULL UNIQUE,  
    key BLOB NOT NULL,  
    salt BLOB NOT NULL)
```

Окрім базових даних користувача, у таблиці зберігаються криптографічні параметри: зашифрований ключ і сіль, що використовуються для захищеного зберігання або шифрування даних.

Ключовим елементом класу є метод `database_worker()`, який реалізує механізм черги запитів до бази даних. Метод працює у фоновому потоці та безперервно очікує надходження нових завдань. Кожен запит містить SQL-команду, параметри, чергу для повернення результату та тип операції. Після отримання запиту виконується SQL-команда. Залежно від типу запиту обробник повертає кількість змінених рядків, один запис або список записів. Після завершення результат передається через чергу методу виклику.

Для роботи з даними клас містить набір спеціалізованих методів: додавання та пошук користувачів, створення й оновлення токенів, додавання, отримання, редагування та видалення паролів. Кожен із цих методів формує SQL-запит і передає його до черги на виконання обробником бази даних.

					БР.КІ-41.00.00.000 ПЗ	Арк.
Змн.	Арк.	окум.	Підпис	Дат		49

Окремо реалізовано механізм керування токенами автентифікації, включаючи перевірку їх існування та оновлення, а також функцію визначення довжини токена через налаштування системи.

В результаті клас DBMS реалізує достатній набір функцій для роботи з базою даних у серверній частині системи.

Наступним кроком є написання основного сценарію запуску та завершення роботи серверної частини системи із використанням механізму захисту даних.

```
from connection.server import Server
from getpass import getpass
import time
from utils.crypto import *
from utils.find_path_file import resource_path
folder = resource_path("data")
server = None
password = None
decrypt_done = False
if __name__ == "__main__":
    try:
        password = getpass("Введіть пароль розшифрування: ")
        decrypt_folder(folder, password)
        decrypt_done = True
        server = Server()
        server.create()
        print(f"Server running on port: {server.get_port()},
hosts: {server.get_hosts()}")
        server.start()
    except KeyboardInterrupt:
        print("KeyboardInterrupt received")
    except Exception as e:
        print(f"[ERROR] {e}")
    finally:
        if server:
            try:
                server.stop()
                server.stop_db()
            except Exception as e:
                print(f"[STOP ERROR] {e}")
        if decrypt_done and password:
```

					БР.КІ-41.00.00.000 ПЗ	Арк.
Змн.	Арк.	окум.	Підпис	Дат		50

```

try:
    encrypt_folder(folder, password)
except Exception as e:
    print(f"[ENCRYPT ERROR] {e}")
time.sleep(0.5)
print("Server stopped")

```

На початку роботи програма визначає шлях до директорії з даними та запитує у користувача пароль для розшифрування. Після введення пароля виконується розшифрування робочої директорії.

Далі створюється об'єкт сервера та виконується його запуск. Після старту сервер працює на визначених портах і хостах. В кінці виконання здійснюється гарантоване завершення роботи сервера та бази даних. Після цього, якщо розшифрування було виконано, дані повторно шифруються для забезпечення їх захисту.

Висновок до розділу

В даному розділі було розроблено клієнтську та серверну частини додатка, які разом утворюють локальний додаток-менеджер паролів. Клієнтська частина забезпечує інтерфейс для користувача, виконання запитів та взаємодію із сервером, тоді як серверна частина відповідає за обробку запитів, роботу з базою даних, автентифікацію та збереження інформації.

					БР.КІ-41.00.00.000 ПЗ	Арк.
Змн.	Арк.	окум.	Підпис	Дат		51

3 ПЕРЕВІРКА ПРАЦЕЗДАТНОСТІ РОЗРОБЛЕНОГО ДОДАТКА

3.1 Перевірка працездатності клієнтської частини додатка

До запуску сервера вміст каталог data повністю зашифрований (рис.3.1).

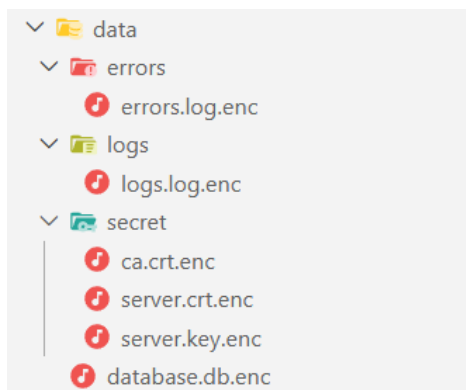


Рисунок 3.1 – Зашифрований вміст каталога data сервера

Переходимо в термінал, запускаємо сервер командою: `python manager_server.py`, та вводимо ключ розшифрування даних (рис.3.2).

```
Введіть пароль розшифрування:  
Server running on port: 9000, hosts: 0.0.0.0
```

Рисунок 3.2 – Запуск сервера

В результаті, вміст каталога data повністю розшифрувався (рис.3.3).

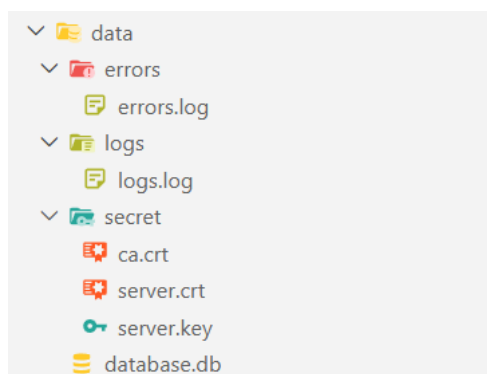


Рисунок 3.3 – Розшифрований вміст каталога data сервера

Тепер виконаємо запуск додатку, перейдемо на сторінку реєстрації, та введемо дані нового користувача (рис.3.4).

Рисунок 3.4 – Реєстрація користувача

Обираємо зареєструватись, та бачимо вікно головного меню (рис.3.5).

Рисунок 3.5 – Головна сторінка

					БР.КІ-41.00.00.000 ПЗ	Арк.
Змн.	Арк.	окум.	Підпис	Дат		53

Згенеруємо новий пароль (рис.3.6).

Manager Password

Manager Project

Головна

Додати

Сховище

Налаштування

User

Новий пароль

Ви можете згенерувати новий пароль

{Z-M62MiSy}MI%

✓ Нижній ✓ Верхній ✓ Цифри ✓ Спец-символи

Довжина пароля: 16

Згенерувати

Рисунок 3.6 – Генерація нового пароля

Перейдемо до сторінки додавання пароля, та введемо дані в поля введення (рис.3.7).

Manager Password

Manager Project

Головна

Додати

Сховище

Налаштування

User

Додати пароль

Збережіть новий пароль у віддаленому сховищі

{Z-M62MiSy}MI%

Google

Введіть додаткову інформацію щодо паролю

Account: user@gmail.com

Додати

Рисунок 3.7 – Додавання пароля

Тиснемо додати та переходимо до сторінки перегляду паролів (рис.3.8).

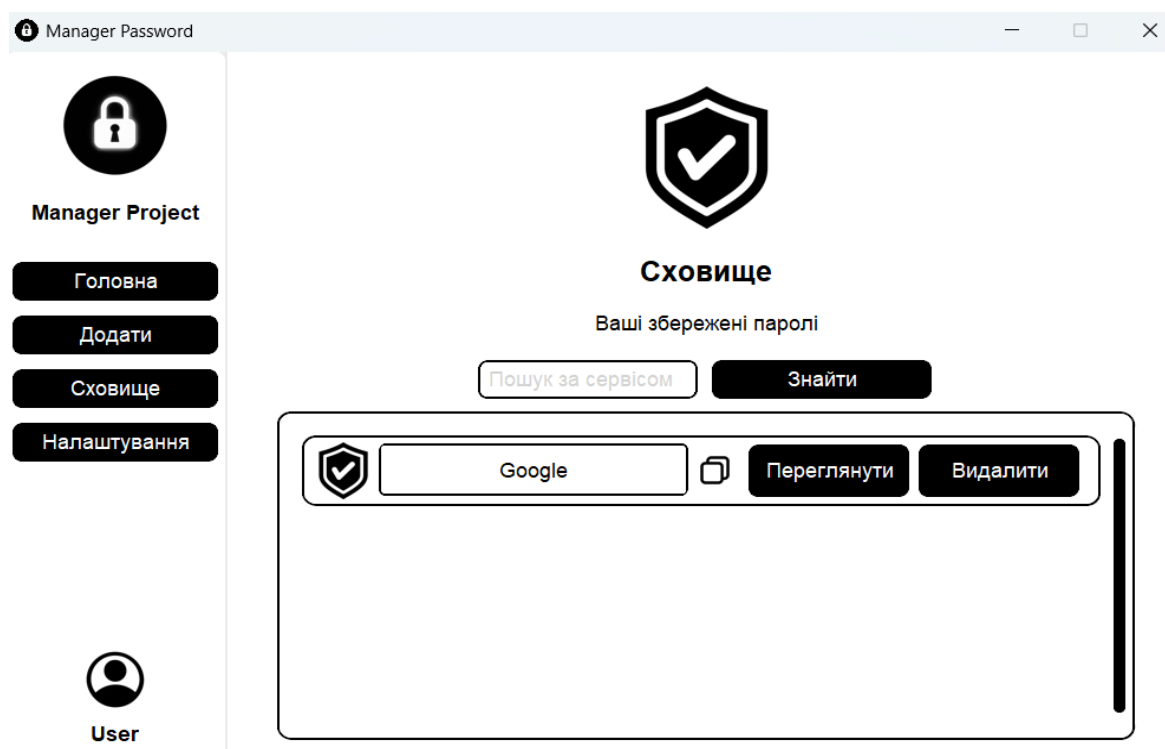


Рисунок 3.8 – Доданий пароль

Спробуємо додати ще один пароль, з здійснити пошук серед паролів (рис.3.9).

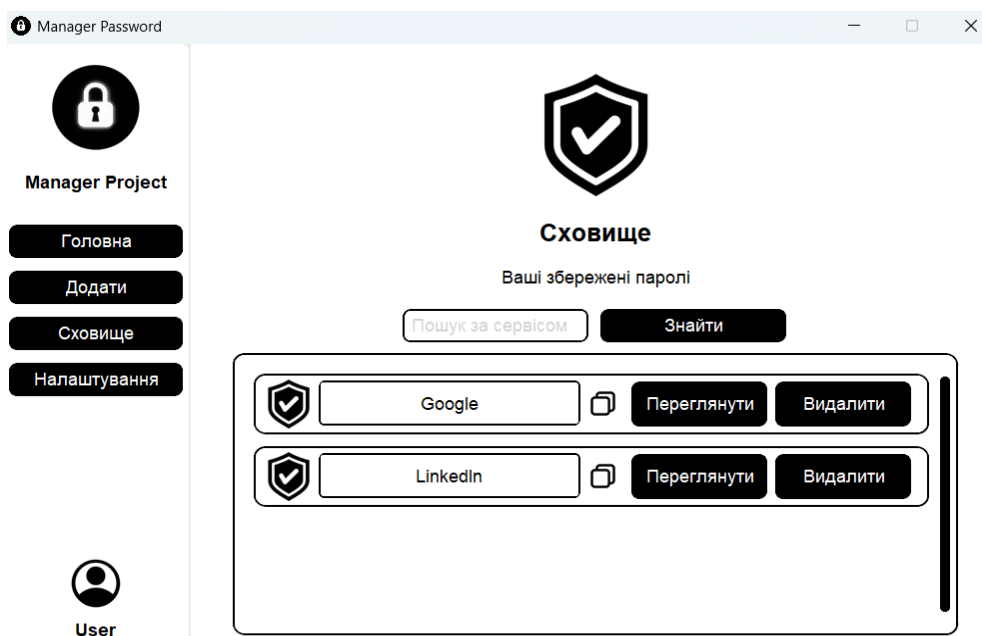


Рисунок 3.9 – Ще один доданий пароль у сховищі

Тепер виконаємо пошук за словом «In» (рис.3.10).

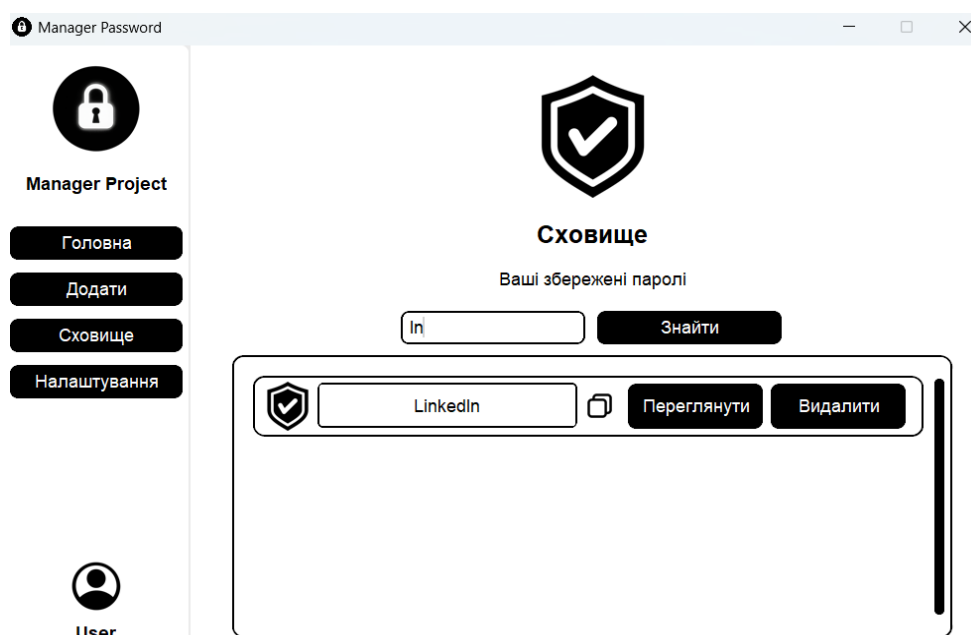


Рисунок 3.10 – Пошук за словом «In»

Як бачимо, додавання та пошук пароля працює коректно. Очистимо поле пошуку, та натиснемо “Знайти“, щоб вивести назад всі пароль користувача (рис.3.11).

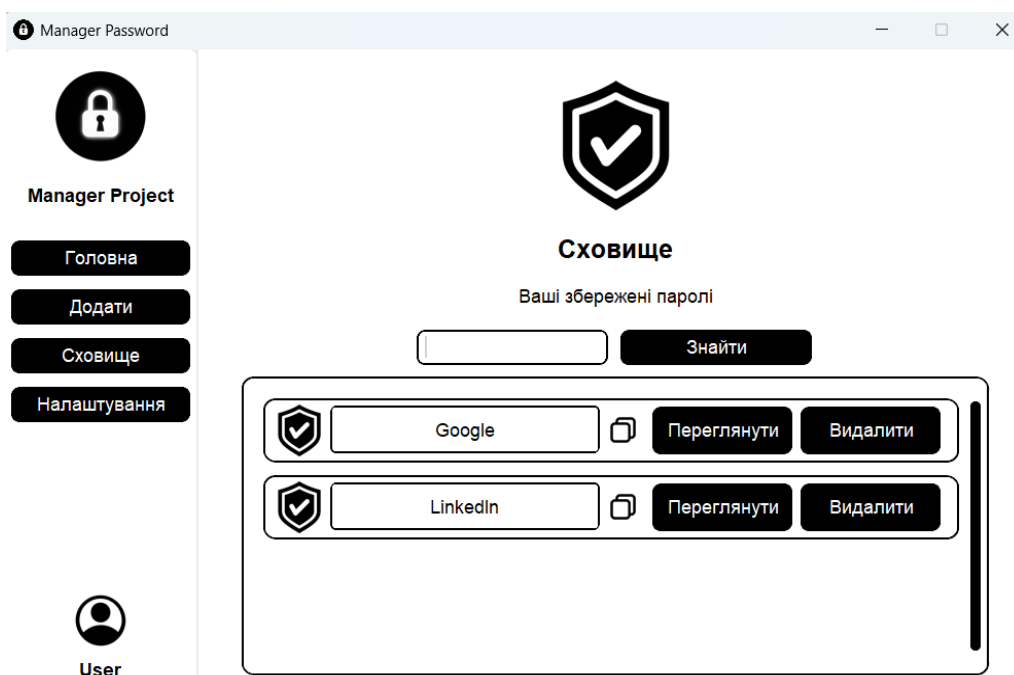


Рисунок 3.11 – Всі паролі користувача

Натиснемо кнопку змінити введемо додаткове значення в пароль сервісу LinkedIn (рис.3.12).

Пароль

Ви можете змінити параметри пароля

Сервіс: Пароль:

Додаток до пароля

Рисунок 3.12 – Додавання опису в пароль сервісу LinkedIn

Натиснемо зберегти, та перевіримо результат, знову переглянувши (рис.3.13).

Manager Password

Пароль

Ви можете змінити параметри пароля

Сервіс: Пароль:

Додаток до пароля

Пароль успішно змінений в сховищі

Рисунок 3.13 – Збережені зміни в паролі сервісу LinkedIn

При натисканні “Скасувати”, зміни для пароля не застосовуватимуться.
Тепер видалимо пароль від LinkedIn (рис.3.14)

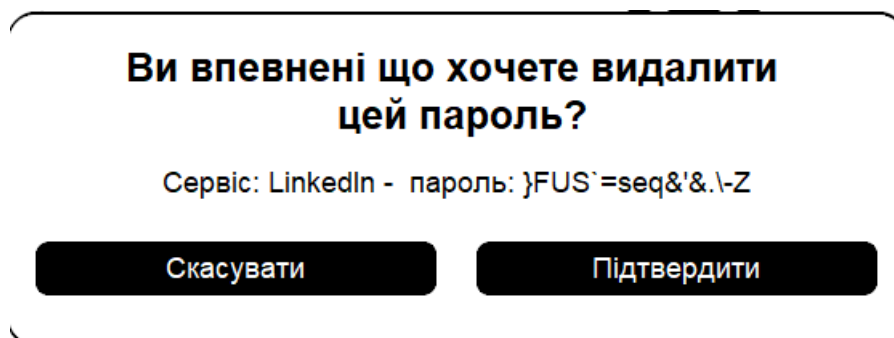


Рисунок 3.14 – Видалення паролю

Обираємо “Підтвердити” та бачимо, що пароля вже немає в сховищі, а повідомлення внизу вказує, що видалення пройшло успішно (рис.3.15).

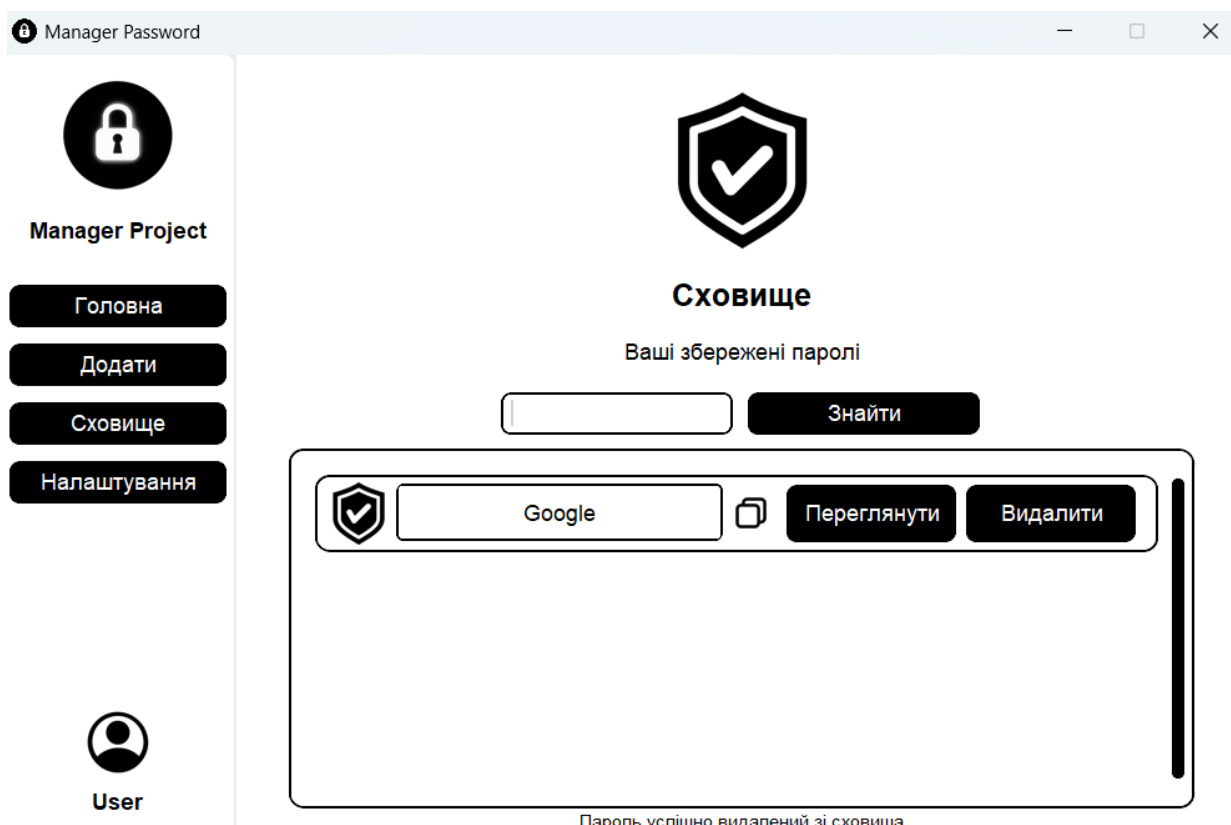


Рисунок 3.15 – Сховище після видалення пароля LinkedIn

Наступним кроком перейдемо до налаштувань, та змінимо тему на темну (рис.3.16).

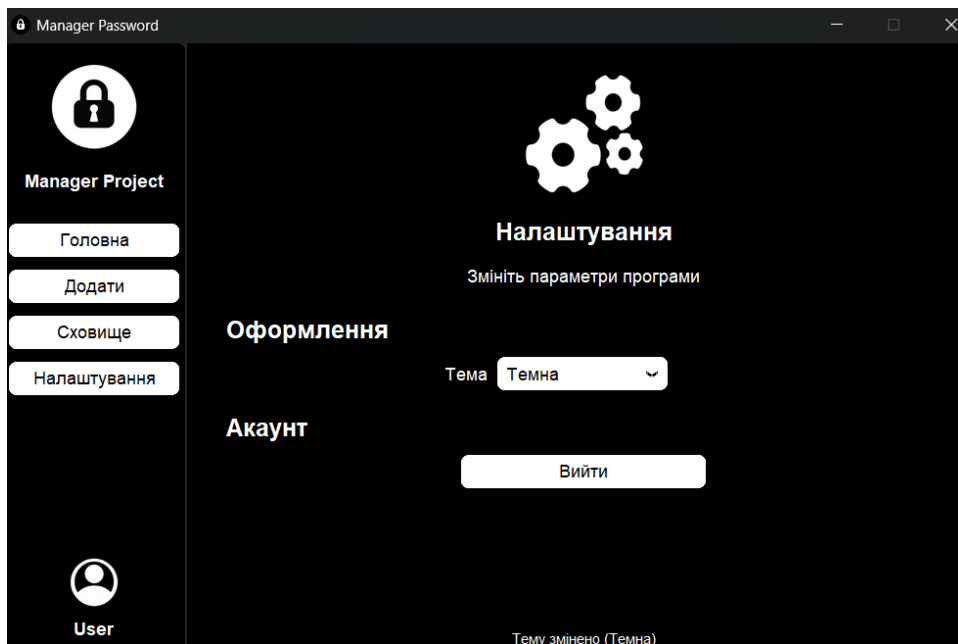


Рисунок 3.16 – Зміна теми на темну

Встановимо назад світлу тему та вийдемо із додатку. В результаті бачимо сторінку автентифікації (рис.3.17).

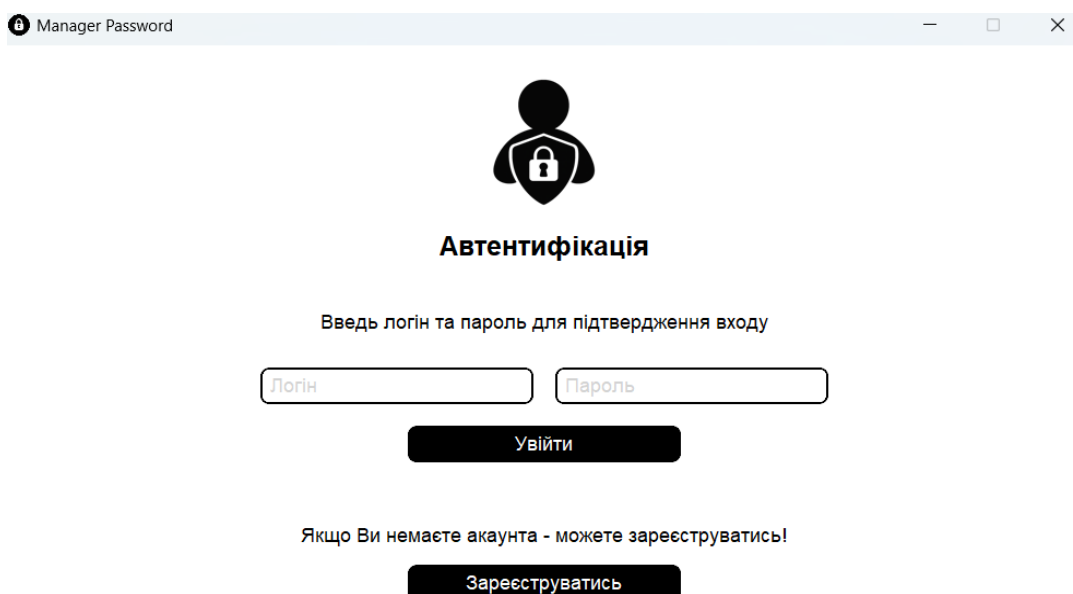
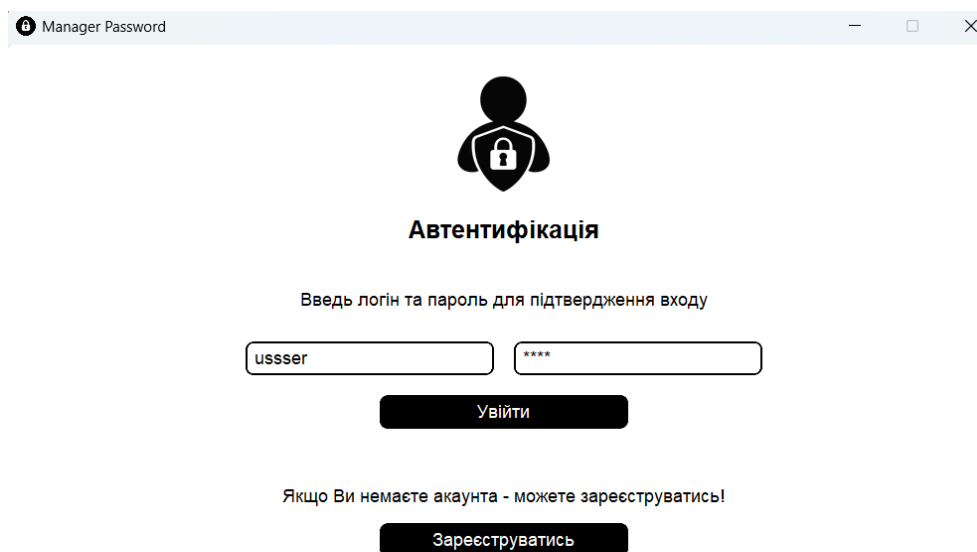


Рисунок 3.17 – Сторінка автентифікації після виходу із системи

					БР.КІ-41.00.00.000 ПЗ	Арк.
Змн.	Арк.	окум.	Підпис	Дат		59

Спробуємо ввести невірні дані щодо логіна та пароля для входу (рис.3.18).

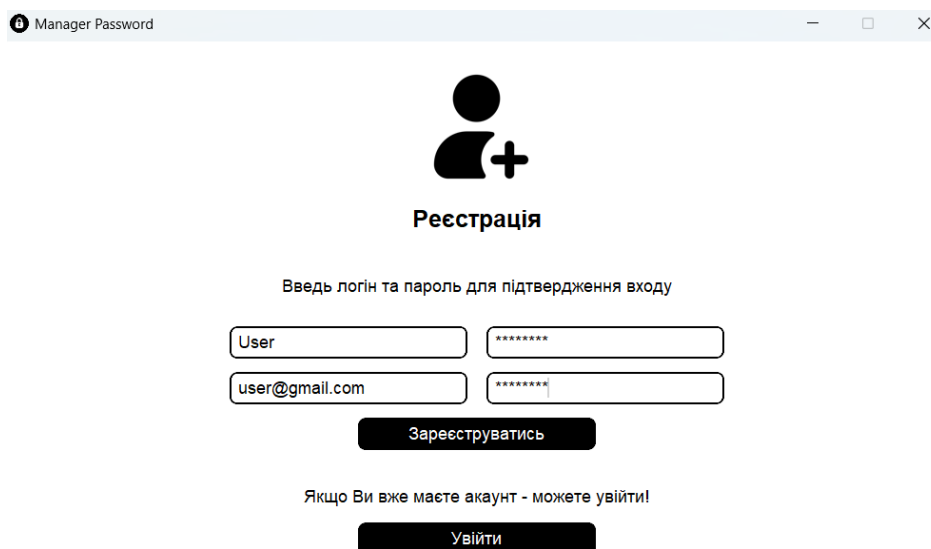


The screenshot shows a web browser window titled "Manager Password". The page has a header with a user icon and a lock icon. The main heading is "Автентифікація". Below it, the text "Введь логін та пароль для підтвердження входу" is displayed. There are two input fields: the first contains "ussser" and the second contains "*****". A black button labeled "Увійти" is below the fields. Below the button, the text "Якщо Ви немає акаунта - можете зареєструватись!" is shown, followed by a black button labeled "Зареєструватись". At the bottom, a message reads "Помилка: Логін чи пароль невірний".

Рисунок 3.18 – Помилка аутентифікації

Після вибору пункту “Увійти”, бачимо, що система не дозволяє вхід.

Також спробуємо зареєструвати вже зареєстрованого користувача (рис.3.19).

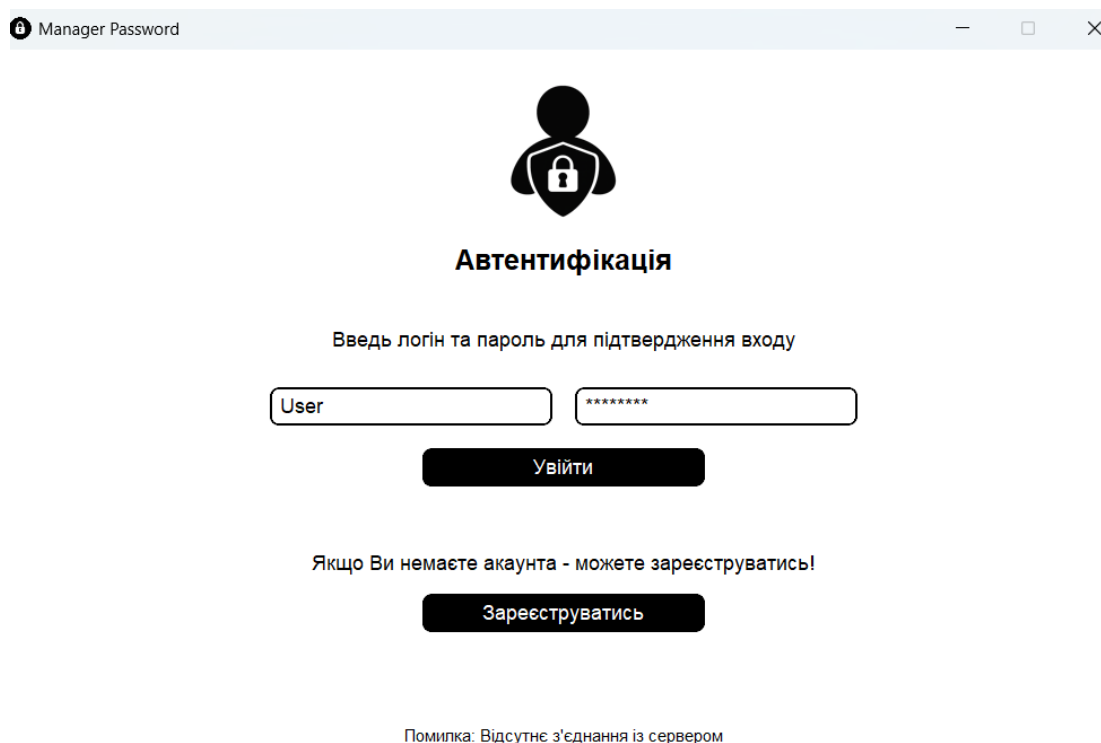


The screenshot shows a web browser window titled "Manager Password". The page has a header with a user icon and a plus icon. The main heading is "Реєстрація". Below it, the text "Введь логін та пароль для підтвердження входу" is displayed. There are four input fields: the first contains "User", the second contains "*****", the third contains "user@gmail.com", and the fourth contains "*****". A black button labeled "Зареєструватись" is below the fields. Below the button, the text "Якщо Ви вже маєте акаунт - можете увійти!" is shown, followed by a black button labeled "Увійти". At the bottom, a message reads "Помилка: Користувач вже зареєстрований".

Рисунок 3.19 – Помилка реєстрації

					БР.КІ-41.00.00.000 ПЗ	Арк.
						60
Змн.	Арк.	окум.	Підпис	Дат		

Як видно, користувач вже зареєстрований. Також вимкнемо сервер, та спробуємо увійти в систему (рис.3.20).



Manager Password



Автентифікація

Введь логін та пароль для підтвердження входу

User

Увійти

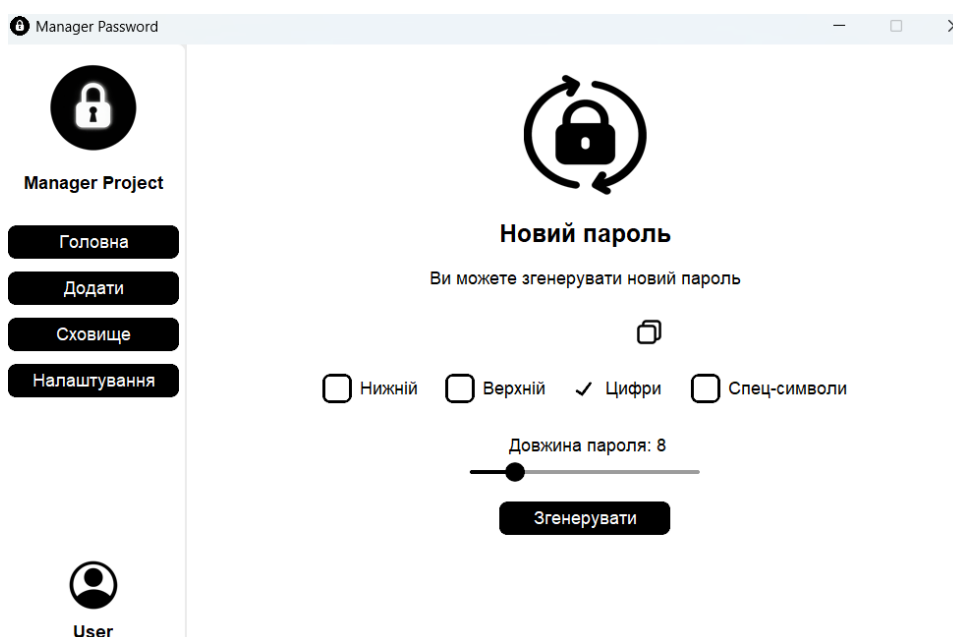
Якщо Ви немає акаунта - можете зареєструватись!

Зареєструватись

Помилка: Відсутнє з'єднання із сервером

Рисунок 3.20 – Спроба увійти, коли сервер вимкнений

Назад увімкнемо сервер, та увійдемо в систему (рис.3.21).



Manager Password


Manager Project

Головна

Додати

Сховище

Налаштування


User



Новий пароль

Ви можете згенерувати новий пароль



☐ Нижній ☐ Верхній ☒ Цифри ☐ Спец-символи

Довжина пароля: 8

Згенерувати

Рисунок 3.21 – Успішна автентифікація

					БР.КІ-41.00.00.000 ПЗ	Арк.
Змн.	Арк.	окум.	Підпис	Дат		61

У результаті проведеної перевірки встановлено, що клієнтська частина застосунку працює коректно та виконує всі передбачені функції.

3.2 Перевірка працездатності серверної частини додатка

Перевіримо логи сервера, що були збережені в процесі його роботи (рис.3.22).

```
data > logs > logs.log
1 New connection [2026-06-08 12:49:29.951451]
2 Registration: answer: True [2026-06-08 12:49:30.380541]
3 Get passwords: answer: False [2026-06-08 12:49:30.646069]
4 Add password: answer: True [2026-06-08 12:54:44.762104]
5 Get passwords: answer: True [2026-06-08 12:54:45.027388]
6 Add password: answer: True [2026-06-08 12:55:28.761834]
7 Get passwords: answer: True [2026-06-08 12:55:29.025097]
8 Update password: answer: True [2026-06-08 13:00:52.814496]
9 Get passwords: answer: True [2026-06-08 13:00:53.088006]
10 Delete password: answer: True [2026-06-08 13:03:23.882379]
11 Get passwords: answer: True [2026-06-08 13:03:24.173894]
12 Disconnected client [2026-06-08 13:04:49.146097]
13
14 New connection [2026-06-08 13:16:57.507610]
15 Authentication: answer: True [2026-06-08 13:16:58.022450]
16 Get passwords: answer: True [2026-06-08 13:16:58.350603]
17 Disconnected client [2026-06-08 13:17:10.792116]
18
```

Рисунок 3.22 – Записані логи сервера

З рисунка 3.22 видно, що сервер зберіг всі операції користувача. Також перевіримо базу даних (рис.3.23-3.25).

user_id ?	login	password	email	key	salt
Фільтр	Фільтр	Фільтр	Фільтр	Фільтр...	Фільтр...
1	2 User	\$argon2id\$v=19\$m=65536,t=3,p=4\$ngtFD...	FiQqervsHnAtTjw+zPMarT4J/...	BLOB	BLOB

Рисунок 3.23 – Вміст таблиці users

id	user_id ?	token
Фі...	Фільтр	Фільтр
1	2	\$argon2id\$v=19\$m=65536,t=3,p=4\$S4Qbz...

Рисунок 3.24 – Вміст таблиці tokens

	id	user_id ?	password	service	description
	Фі...	Фільтр	Фільтр	Фільтр	Фільтр
1	6	2	vFIHz/xYR39yynXi5Pf58BWGN/...	Google	ROP0xArnwgkqFvniVeP+/...

Рисунок 3.25 – Вміст таблиці passwords

З рисунків 3.23-3.25 видно, що в таблицях users, tokens, passwords є наявності збережена сервером інформація користувача.

Також перевірено процес передачі даних між клієнтом та сервером за допомогою Wireshark. Для цього було запущено Wireshark, передано дані між клієнтом та сервером та перевірено вміст перехоплених пакетів (рис.3.26).

14	0.029405500	127.0.0.1	127.0.0.1	TLSv1.3	1643 Application Data
15	0.029462800	127.0.0.1	127.0.0.1	TCP	44 64700 → 9000 [ACK] Seq=3967 Ack=6810 Win=58624 Len=0
16	0.546406200	127.0.0.1	127.0.0.1	TLSv1.3	140 Application Data
17	0.546489300	127.0.0.1	127.0.0.1	TCP	44 64700 → 9000 [ACK] Seq=3967 Ack=6906 Win=58624 Len=0
18	0.553141900	127.0.0.1	127.0.0.1	TLSv1.3	105 Application Data

> Frame 14: Packet, 1643 bytes on wire (13144 bits), 1643 bytes captured (13144 bits) on interface \Device	0000	02 00 00 00 45 00 0f
> Null/Loopback	0010	7f 00 00 01 7f 00 0f
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1	0020	16 44 fd c1 50 18 0f
> Transmission Control Protocol, Src Port: 9000, Dst Port: 64700, Seq: 5211, Ack: 3967, Len: 1599	0030	3a bd ef f9 a5 d7 3f
▼ Transport Layer Security	0040	f3 c1 fd 97 39 63 b1
[Stream index: 0]	0050	6b 11 d5 39 7e 67 c1
▼ TLSv1.3 Record Layer: Application Data Protocol: Application Data	0060	3c fa 56 f6 c6 7b 0f
Opaque Type: Application Data (23)	0070	ca 44 73 a1 d7 21 b1
Version: TLS 1.2 (0x0303)	0080	de 3e b9 da 28 60 b1
Length: 1594	0090	f5 da fa ff f5 5d 5c
Encrypted Application Data [...]: bdeff9a5d734c6f0766b8705e81bdaf3c1fd973963bfd19b9acda22ceed2ae6b11	00a0	eb 22 40 2b 38 81 1f
	00b0	f9 53 af 8f f2 b1 5f
	00c0	b2 ca 9d 1d 69 0e 6f
	00d0	64 de 18 97 d5 57 7c

Рисунок 3.26 – Вміст перехопленого пакету, який передавався між клієнтом та сервером

З рисунка 3.26 видно, що пакет містить зашифровану інформацію, а це підтверджує, що від клієнта до сервера передавалися зашифровані дані.

Висновок до розділу

У результаті проведеної перевірки працездатності розробленого локального додатка-менеджера паролів було встановлено, що він працює належним чином.

					БР.КІ-41.00.00.000 ПЗ	Арк.
Змн.	Арк.	окум.	Підпис	Дат		63

ВИСНОВКИ

У результаті виконання бакалаврської роботи створено локальний додаток-менеджер паролів, який надає можливість додавати, переглядати, змінювати, видаляти та генерувати паролі та взаємодіяти із базою даних.

У першому розділі було проведено аналіз існуючих менеджерів паролів KeePassXC (локальне рішення) та LastPass for Desktop (хмарний сервіс). На основі їх переваг і недоліків обрано клієнт-серверну архітектуру локального менеджера паролів, яка забезпечує централізоване зберігання даних та підтримку багатокористувацької роботи в локальній мережі. Для реалізації графічного інтерфейсу використано бібліотеку CustomTkinter, яка дозволяє створювати сучасні застосунки з мінімальними системними вимогами. Захист процесу взаємодії сервера з клієнтом забезпечується за допомогою бібліотеки ssl із використанням механізму mTLS, що гарантує взаємну автентифікацію клієнта і сервера. Для зберігання інформації обрано СУБД SQLite, яка характеризується простотою інтеграції, низькими вимогами до ресурсів та достатньою продуктивністю для локальних систем.

Під час виконання другого розділу було розроблено клієнтську та серверну частини локального додатка-менеджера паролів. Клієнтська частина забезпечує інтерфейс для користувача, виконання запитів та взаємодію із сервером, тоді як серверна частина відповідає за обробку запитів, роботу з базою даних, автентифікацію та збереження інформації.

У результаті проведеної перевірки працездатності розробленого локального додатка-менеджера паролів було встановлено, що він працює належним чином.

					БР.КІ-41.00.00.000 ПЗ	Арк.
Змн.	Арк.	окум.	Підпис	Дат		64

ПЕРЕЛІК ПОСИЛАНЬ НА ДЖЕРЕЛА

1. Python 3.16 documentation. *Python documentation*.
URL: <https://docs.python.org/3.16/> (дата звернення: 20.05.2026).
2. Documentation Introduction | CustomTkinter. Official Documentation And Tutorial | CustomTkinter. URL: <https://customtkinter.tomschimansky.com/documentation/> (дата створення: 10.04.2026).
3. Eliminating Obsolete Transport Layer Security (TLS) Protocol Configurations. *Publications*. URL: <https://publications.bsafes.com/docs/nsa/eliminating-obsolete-tls-u00197443-20/> (дата звернення: 03.05.2026).
4. TLS/SSL wrapper for socket objects. *Python documentation*.
URL: <https://docs.python.org/3/library/ssl.html> (дата звернення: 04.05.2026).
5. Generate secure random numbers for managing secrets. *Python documentation*. URL: <https://docs.python.org/3/library/secrets.html> (date of access: 06.04.2026).
6. Васильєв О.М. Програмування в Python. Теорія і практика : навч. посіб. Київ : Видавництво Ліра-К, 2023. 462 с.
7. Common string operations. *Python documentation*. URL: <https://docs.python.org/3/library/string.html> (дата звернення: 06.05.2026).
8. DB-API 2.0 interface for SQLite databases. *Python documentation*.
URL: <https://docs.python.org/3/library/sqlite3.html> (дата звернення: 14.05.2026).
9. Welcome to pyca/cryptography – Cryptography 49.0.0.dev1 documentation. *Welcome to pyca/cryptography – Cryptography 49.0.0.dev1 documentation*. URL: <https://cryptography.io/en/latest/> (дата звернення: 26.04.2026).
10. socketserver A framework for network servers. *Python documentation*.
URL: <https://docs.python.org/uk/3/library/socketserver.html> (дата звернення: 01.05.2026).
11. Комплексна безпека інформаційних мережевих систем: навчальний посібник для студентів спеціальності 174 «Автоматизація, комп'ютерно-

					БР.КІ-41.00.00.000 ПЗ	Арк.
Змн.	Арк.	окум.	Підпис	Дат		65

інтегровані технології та робототехніка» / Укладачі: А.Г. Микитишин, М.М. Митник, О.С. Голотенко, В.В. Карташов. – Тернопіль : ФОП Паляниця В.А., 2023. – 324 с.

12. Гапак О.М. Захист інформації в комп'ютерних системах. Підручник / О.М. Гапак, С.І. Балога .- Ужгород: видавництво ПП «АУТДОР-ШАРК», 2021. – 184 с.

13. Automate the boring stuff with python - automate the boring stuff with python. *Automate the Boring Stuff with Python - Automate the Boring Stuff with Python*. URL: <https://automatetheboringstuff.com/> (дата звернення: 20.04.2026).

14. Computer Networks: A Systems Approach – Computer Networks: A Systems Approach Version 6.2-dev documentation. *Computer Networks: A Systems Approach – Computer Networks: A Systems Approach Version 6.2-dev documentation*. URL: <https://book.systemsapproach.org/> (дата звернення: 27.04.2026).

15. CS50's introduction to programming with python. *CS50: Harvard's computer science courses and certificates | edX*. URL: <https://cs50.harvard.edu/python/> (дата звернення: 08.05.2026).

16. Python for beginners. *Python.org*. URL: <https://www.python.org/about/gettingstarted/> (дата звернення: 02.04.2026).

17. CS 144: Introduction to Computer Networking. *CS 144: Introduction to Computer Networking*. URL: <https://cs144.github.io/> (дата звернення: 18.05.2026).

18. OpenSSL Documentation. *Redirecting*. URL: <https://docs.openssl.org/master/> (дата звернення: 21.04.2026).

19. Wireshark Go Deep|Documentation. *Wireshark*. URL: <https://www.wireshark.org/docs/> (дата звернення: 02.04.2026).

20. Argon2-cffi 25.1.0 documentation. *argon2-cffi 25.1.0 documentation*. URL: <https://argon2-cffi.readthedocs.io/en/stable/> (дата звернення: 15.05.2026).

21. Welcome to pyca/cryptography – Cryptography 49.0.0.dev1 documentation. *Welcome to pyca/cryptography – Cryptography 49.0.0.dev1 documentation*. URL: <https://cryptography.io/en/latest/> (дата звернення: 10.04.2026).

					БР.КІ-41.00.00.000 ПЗ	Арк.
Змн.	Арк.	окум.	Підпис	Дат		66

22. System-specific parameters and functions. *Python documentation*. URL: <https://docs.python.org/3/library/sys.html> (дата звернення: 11.05.2026).

23. Miscellaneous operating system interfaces. *Python documentation*. URL: <https://docs.python.org/3/library/os.html> (дата звернення: 12.05.2026).

24. RFC 9293: Transmission Control Protocol (TCP). *IETF | Internet Engineering Task Force*. URL: <https://www.ietf.org/rfc/rfc9293.html> (дата звернення: 07.05.2026).

25. RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3 | RFC Editor. *RFC Editor*. URL: <https://www.rfc-editor.org/info/rfc8446/> (дата звернення: 28.04.2026).

					БР.КІ-41.00.00.000 ПЗ	Арк.
Змн.	Арк.	окум.	Підпис	Дат		67

ДОДАТОК А

Код реалізації класу App

```
import customtkinter as ctk
from PIL import Image
import os
from connection.client import Client
from gui.pages import GeneratePasswordPage, AddPasswordPage, RefugePage,
SettingsPage
from gui.login_pages import Authentication, Registration
from utils.find_path_file import resource_path
from utils.process_value_settings import get_value_settings,
set_value_settings
class App(ctk.CTk):
    def __init__(self):
        super().__init__()
        self.protocol("WM_DELETE_WINDOW", self.on_close)
        self.center_window(self, 800, 500)
        self.title("Manager Password")
        self.resizable(False, False)
        self.iconbitmap(resource_path(os.path.join("images",
"logo.ico")))
        ctk.set_default_color_theme(resource_path(
os.path.join("themes", "theme.json")))
        self.change_theme("System")
        self.__pages = {}
        self.__token = None
        self.__login = None
        self.__cut_login = None
        self.__current_theme = None
        self._client = Client()
        self._title_font = ctk.CTkFont(family="Nunito", size=20,
weight="bold")
        self._text_font = ctk.CTkFont(family="Nunito", size=14)
        self._message_font = ctk.CTkFont(family="Nunito", size=12)
        self.create_main_block()
        self.create_main_menu()
```

```

        self.create_main_pages()
        self.set_current_theme()
        self.create_login_pages()
        self.show_login()
        self.show_page("Authentication")
    def create_main_block(self):
        self.__menu = ctk.CTkFrame(self, width=200)
        self.__menu.pack(side="left", fill="y", padx=1)
        self.__page = ctk.CTkFrame(self)
        self.__page.pack(side="left", fill="both", expand=True)
    def create_main_menu(self):
        self.__menu_image = ctk.CTkImage(light_image=Image.open(
            resource_path(os.path.join("images",
            "main_logo_light_theme.png"))),
            dark_image=Image.open(resource_path(
            os.path.join("images", "main_logo_dark_theme.png"))),
            size=(70, 70))
        self.__menu_image_label = ctk.CTkLabel(
            self.__menu, image=self.__menu_image, text="")
        self.__menu_image_label.pack(pady=[20, 15])
        self.__menu_label = ctk.CTkLabel(
            self.__menu, text="Manager Project",
            font=ctk.CTkFont(family="Nunito", size=14, weight="bold"))
        self.__menu_label.pack(pady=[0, 20], padx=5)
        self.__menu_button_main = ctk.CTkButton(
            self.__menu, text="Головна", font=self._text_font,
            command=lambda: self.show_page("GeneratePasswordPage"))
        self.__menu_button_main.pack(pady=5, padx=5)
        self.__menu_button_add = ctk.CTkButton(
            self.__menu, text="Додати", font=self._text_font,
            command=lambda: self.show_page("AddPasswordPage"))
        self.__menu_button_add.pack(pady=5, padx=5)
        self.__menu_button_refuge = ctk.CTkButton(
            self.__menu, text="Сховище", font=self._text_font,
            command=lambda: self.show_page("RefugePage"))
        self.__menu_button_refuge.pack(pady=5, padx=5)
        self.__menu_button_settings = ctk.CTkButton(

```

```

        self.__menu, text="Налаштування", font=self._text_font,
command=lambda: self.show_page("SettingsPage"))
        self.__menu_button_settings.pack(pady=5, padx=5)
        self.__menu_bottom = ctk.CTkFrame(self.__menu,
fg_color="transparent")
        self.__menu_bottom.pack(side="bottom")
        self.__menu_bottom_image = ctk.CTkImage(light_image=Image.open(
resource_path(os.path.join("images",
"account_logo_light_theme.png"))),
dark_image=Image.open(resource_path(os.path.join(
"images", "account_logo_dark_theme.png"))),
size=(40, 40))
        self.__menu_bottom_image_label = ctk.CTkLabel(
self.__menu_bottom, image=self.__menu_bottom_image, text="")
        self.__menu_bottom_image_label.pack()
        self.__menu_bottom_image = ctk.CTkLabel(
self.__menu_bottom, text="", font=ctk.CTkFont(family="Nunito",
size=14, weight="bold"),)
        self.__menu_bottom_image.pack(
after=self.__menu_bottom_image_label, pady=[5, 10])
def create_login_pages(self):
    for PageClass in [Authentication, Registration]:
        page = PageClass(self, self)
        page.place(relx=0, rely=0, relwidth=1, relheight=1)
        self.__pages[PageClass.__name__] = page
def create_main_pages(self):
    for PageClass in [GeneratePasswordPage, AddPasswordPage,
RefugePage, SettingsPage,]:
        page = PageClass(self.__page, self)
        page.place(relx=0, rely=0, relwidth=1, relheight=1)
        self.__pages[PageClass.__name__] = page
def show_page(self, page_name):
    if page_name in ["GeneratePasswordPage", "RefugePage",
"AddPasswordPage", "SettingsPage"]:
        self.__pages[page_name].show_next_page()
        self.__pages[page_name].tkraise()
def show_main(self):

```

```

self.__pages["Authentication"].place_forget()
self.__pages["Registration"].place_forget()
self.__menu.pack(side="left", fill="y", padx=1)
self.__page.pack(side="left", fill="both", expand=True)
self.get_passwords()
self.__pages["GeneratePasswordPage"].clear_page()
self.__pages["RefugePage"].clear_page()
self.__pages["AddPasswordPage"].clear_page()
self.show_page("GeneratePasswordPage")
self.__menu_bottom_image.configure(text=self.__cut_login)
def show_login(self):
    self.__menu.pack_forget()
    self.__page.pack_forget()
    self._client.close()
    self.__pages["Authentication"].place(
        relx=0, rely=0, relwidth=1, relheight=1)
    self.__pages["Registration"].place(
        relx=0, rely=0, relwidth=1, relheight=1)
def get_passwords(self, is_show_message=True):
    request = self._client.message(header="get", token=self.__token)
    result = self._client.send(request)
    if not result:
        result = self._client.get()
    else:
        result = None
    self.__pages["RefugePage"].create_password_frames(
        result, is_show_message)
def get_token(self):
    return self.__token
def set_token(self, token):
    self.__token = token
def get_login(self):
    return self.__login
def set_login(self, login):
    self.__login = login
def get_cut_login(self):
    return self.__cut_login

```

```

def set_cut_login(self, login):
    if len(login) > 20:
        login = login[0:17]
        login += "..."
    self.__cut_login = login
def set_current_theme(self):
    self.__current_theme = get_value_settings("theme")
    if self.__current_theme == "Light":
        self.__pages["SettingsPage"].set_theme_option("Світла")
    elif self.__current_theme == "Dark":
        self.__pages["SettingsPage"].set_theme_option("Темна")
    elif self.__current_theme == "System":
        self.__pages["SettingsPage"].set_theme_option("Системна")
    else:
        self.__pages["SettingsPage"].set_theme_option("Системна")
        self.__pages["SettingsPage"].set_message_label(
            "Помилка: Некоректне значення в налаштуваннях, тема за
замовчуванням (Системна)")
def change_theme(self, theme):
    ctk.set_appearance_mode(theme)
    set_value_settings("theme", theme)
def center_window(self, window, width, height):
    screen_width = self.wininfo_screenwidth()
    screen_height = self.wininfo_screenheight()
    x = int((screen_width / 2) - (width / 2))
    y = int((screen_height / 2) - (height / 2))
    window.geometry(f"{width}x{height}+{x}+{y}")
def on_close(self):
    self._client.close()
    self.destroy()

```

Код реалізації класу Authentication

```

import customtkinter as ctk
from PIL import Image
import os

```

```

import re
from utils.find_path_file import resource_path
from utils.show_message import show_message
class Authentication(ctk.CTkFrame):
    def __init__(self, parent, app):
        super().__init__(parent)
        self.__app = app
        self.create_main_block()
        self.create_registration_block()
        self.create_message_block()
        error_connection = self.__app._client.connect()
        if error_connection:
            show_message(self.__message_label,
                        f"Помилка: {self.__app._client.connect()}")
    def create_main_block(self):
        self.__main_image =
        ctk.CTkImage(light_image=Image.open(resource_path(os.path.join("images",
"login_logo_light_theme.png"))),
        dark_image=Image.open(resource_path(os.path.join("images",
"login_logo_dark_theme.png"))),size=(75, 95))
        self.__main_image_label = ctk.CTkLabel(
            self, image=self.__main_image, text="")
        self.__main_image_label.pack(pady=[30, 0])
        self.__main_label = ctk.CTkLabel(
            self, text="Автентифікація", font=self.__app._title_font)
        self.__main_label.pack(pady=[20, 10])
        self.__main_text = ctk.CTkLabel(
            self, text="Введь логін та пароль для підтвердження входу",
font=self.__app._text_font)
        self.__main_text.pack(pady=20)
        self.__main_frame = ctk.CTkFrame(self, fg_color="transparent")
        self.__main_frame.pack(fill="x")
        for column, weight in enumerate((1, 0, 0, 1)):
            self.__main_frame.grid_columnconfigure(column, weight=weight)
        self.__main_frame.grid_rowconfigure(0, weight=1)
        self.__main_frame_login = ctk.CTkEntry(

```

```

        self.__main_frame, width=200, font=self.__app._text_font,
placeholder_text="Лорін")
    self.__main_frame_login.grid(row=0, column=1, padx=[0, 20])
    self.__main_frame_password = ctk.CTkEntry(
        self.__main_frame, width=200, font=self.__app._text_font,
placeholder_text="Пароль", show="*")
    self.__main_frame_password.grid(row=0, column=2)
    self.__main_button = ctk.CTkButton(
        self, width=200, text="Увійти", font=self.__app._text_font,
command=self.check_entry_main_block)
    self.__main_button.pack(pady=15)
    def create_registration_block(self):
        self.__registration_label = ctk.CTkLabel(
            self, text="Якщо Ви немає акаунта - можете зареєструватись!",
font=self.__app._text_font)
        self.__registration_label.pack(pady=[30, 0])
        self.__registration_button = ctk.CTkButton(
            self, width=200, text="Зареєструватись",
font=self.__app._text_font, command=self.show_registration_page)
        self.__registration_button.pack(pady=[10, 0])
    def create_message_block(self):
        self.__message_label = ctk.CTkLabel(
            self, text="", font=self.__app._message_font)
        self.__message_label.pack(side="bottom")
    def show_registration_page(self):
        self.__main_frame_login.delete(0, "end")
        self.__main_frame_password.delete(0, "end")
        self.__main_frame_login._activate_placeholder()
        self.__main_frame_password._activate_placeholder()
        self.__main_frame_password.configure(show="*")
        self.focus()
        show_message(self.__message_label, "")
        self.__app.show_page("Registration")
    def check_entry_main_block(self):
        login = self.__main_frame_login.get().strip()
        password = self.__main_frame_password.get().strip()
        if not login or not password:

```

```

        show_message(self.__message_label,
                      "Помилка: Введені не всі поля")
elif not (2 <= len(login) <= 50):
    show_message(self.__message_label,
                  "Помилка: Некоректний логін (від 2 до 50 символів)")
elif not (4 <= len(password) <= 25):
    show_message(self.__message_label,
                  "Помилка: Некоректний пароль (від 4 до 25
СИМВОЛІВ)")
else:
    show_message(self.__message_label, "")
    self.authentication(login, password)
def authentication(self, login, password):
    request = self.__app._client.message(
        header="authentication", login=login, password=password)
    result = self.__app._client.send(request)
    if result:
        show_message(self.__message_label, result)
        return
    answer = self.__app._client.get()
    if not answer:
        show_message(self.__message_label,
                      "Помилка: Час очікування відповіді вичерпаний")
        return
    if answer.get("header") == "authentication-respond" and
answer.get("answer") and answer.get("token"):
        self.__app.set_token(answer["token"])
        self.__app.set_login(login)
        self.__app.set_cut_login(login)
        self.__main_frame_login.delete(0, "end")
        self.__main_frame_password.delete(0, "end")
        self.__main_frame_login._activate_placeholder()
        self.__main_frame_password._activate_placeholder()
        self.__main_frame_password.configure(show="*")
        self.focus()
        self.__app.show_main()

```

```

else:
    show_message(self.__message_label,
                  "Помилка: Логін чи пароль невірний")

```

Код реалізації класу Registration

```

import customtkinter as ctk
from PIL import Image
import os
import re
from utils.find_path_file import resource_path
from utils.show_message import show_message
class Registration(ctk.CTkFrame):
    def __init__(self, parent, app):
        super().__init__(parent)
        self.__app = app
        self.__email_check_pattern = r"^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-
]+\. [a-zA-Z]{2,}$"
        self.create_main_block()
        self.create_authentication_block()
        self.create_message_block()
    def create_main_block(self):
        self.__main_image = ctk.CTkImage(
            light_image=Image.open(resource_path(os.path.join(
                "images", "registration_logo_light_theme.png"))),
            dark_image=Image.open(resource_path(os.path.join(
                "images", "registration_logo_dark_theme.png"))),
            size=(95, 95))
        self.__main_image_label = ctk.CTkLabel(
            self, image=self.__main_image, text="")
        self.__main_image_label.pack(pady=[30, 0])
        self.__main_label = ctk.CTkLabel(
            self, text="Рєєстрація", font=self.__app._title_font)
        self.__main_label.pack(pady=[20, 10])
        self.__main_label_text = ctk.CTkLabel(self, text="Введь логін та пароль
для підтвердження входу", font=self.__app._text_font)

```

```

self.__main_label_text.pack(pady=20)
self.__main_frame = ctk.CTkFrame(self, fg_color="transparent")
self.__main_frame.pack(fill="x")
for column, weight in enumerate((1, 0, 0, 1)):
    self.__main_frame.grid_columnconfigure(column, weight=weight)
self.__main_frame.grid_rowconfigure((0, 1), weight=1)
self.__main_frame_login = ctk.CTkEntry(
    self.__main_frame, width=200, font=self.__app._text_font,
placeholder_text="Логін")
self.__main_frame_login.grid(row=0, column=1, padx=[0, 20])
self.__main_frame_password = ctk.CTkEntry(
    self.__main_frame, width=200, font=self.__app._text_font,
placeholder_text="Пароль", show="*")
self.__main_frame_password.grid(row=0, column=2)
self.__main_frame_email = ctk.CTkEntry(
    self.__main_frame, width=200, font=self.__app._text_font,
placeholder_text="Електронна адреса")
self.__main_frame_email.grid(
    row=1, column=1, padx=[0, 20], pady=10)
self.__main_frame_password_correct = ctk.CTkEntry(
    self.__main_frame, width=200, font=self.__app._text_font,
placeholder_text="Повторіть пароль", show="*")
self.__main_frame_password_correct.grid(row=1, column=2, pady=10)
self.__main_button = ctk.CTkButton(
    self, width=200, text="Зареєструватись",
font=self.__app._text_font, command=self.check_entry_main_block)
self.__main_button.pack()
def create_authentication_block(self):
    self.__authentication_label = ctk.CTkLabel(
        self, text="Якщо Ви вже маєте акаунт - можете увійти!",
font=self.__app._text_font)
    self.__authentication_label.pack(pady=[30, 0])
    self.__authentication_button = ctk.CTkButton(
        self, width=200, text="Увійти", font=self.__app._text_font,
command=self.show_authentication_page)
    self.__authentication_button.pack(pady=[10, 0])
def create_message_block(self):

```

```

self.__message_label = ctk.CTkLabel(
    self, text="", font=self.__app._message_font)
self.__message_label.pack(side="bottom")
def show_authentication_page(self):
    self.__main_frame_login.delete(0, "end")
    self.__main_frame_password.delete(0, "end")
    self.__main_frame_email.delete(0, "end")
    self.__main_frame_password_correct.delete(0, "end")
    self.__main_frame_login._activate_placeholder()
    self.__main_frame_password._activate_placeholder()
    self.__main_frame_email._activate_placeholder()
    self.__main_frame_password_correct._activate_placeholder()
    self.__main_frame_password.configure(show="*")
    self.__main_frame_password_correct.configure(show="*")
    self.focus()
    show_message(self.__message_label, "")
    self.__app.show_page("Authentication")
def check_entry_main_block(self):
    login = self.__main_frame_login.get().strip()
    password = self.__main_frame_password.get().strip()
    email = self.__main_frame_email.get().strip()
    password_correct =
self.__main_frame_password_correct.get().strip()
    if not login or not password or not email or not password_correct:
        show_message(self.__message_label,
            "Помилка: Введені не всі поля")
    elif not (2 <= len(login) <= 50):
        show_message(self.__message_label,
            "Помилка: Некоректний логін (від 2 до 50 символів)")
    elif not (4 <= len(password) <= 25):
        show_message(self.__message_label,
            "Помилка: Некоректний пароль (від 4 до 25
СИМВОЛІВ)")
    elif password != password_correct:
        show_message(self.__message_label,

```

```

        "Помилка: Паролі не співпадають")
    else:
        if not (6 <= len(email) <= 254):
            show_message(self.__message_label,
                "Помилка: Некоректна електронна адреса (від
6 до 254 символів)")
        elif not re.match(self.__email_check_pattern, email):
            show_message(self.__message_label,
                "Помилка: Неправильна електронна адреса")
        else:
            show_message(self.__message_label, "")
            self.registration(login, password, email)
    def registration(self, login, password, email):
        request = self.__app._client.message(
            header="registration", login=login, password=password,
email=email)
        result = self.__app._client.send(request)
        if result:
            show_message(self.__message_label, result)
            return
        answer = self.__app._client.get()
        if not answer:
            show_message(self.__message_label,
                "Помилка: Час очікування відповіді вичерпаний")
            return
        if answer.get("header") == "registration-respond" and
answer.get("answer") and answer.get("token"):
            self.__app.set_token(answer["token"])
            self.__app.set_login(login)
            self.__app.set_cut_login(login)
            self.__main_frame_login.delete(0, "end")
            self.__main_frame_password.delete(0, "end")
            self.__main_frame_email.delete(0, "end")
            self.__main_frame_password_correct.delete(0, "end")
            self.__main_frame_login._activate_placeholder()
            self.__main_frame_password._activate_placeholder()
            self.__main_frame_email._activate_placeholder()

```

```

self.__main_frame_password_correct._activate_placeholder()
self.__main_frame_password.configure(show="*")
self.__main_frame_password_correct.configure(show="*")
self.focus()
self.__app.show_main()
else:
    show_message(self.__message_label,
                 "Помилка: Користувач вже зареєстрований")

```

Код реалізації класу GeneratePasswordPage

```

import customtkinter as ctk
from PIL import Image
import os
from utils.generate_password import set_options, generate_password
from utils.find_path_file import resource_path
from utils.show_message import show_message
class GeneratePasswordPage(ctk.CTkFrame):
    def __init__(self, parent, app):
        super().__init__(parent)
        self.__app = app
        self.__options = {}
        self.__length = 0
        self.__chars = ""
        self.__password = None
        self.create_main_block()
        self.create_generate_block()
        self.create_checkbox_block()
        self.create_message_block()
    def create_main_block(self):
        self.__main_image = ctk.CTkImage(light_image=Image.open(
            resource_path(os.path.join("images",
"generate_password_logo_light_theme.png"))),
        dark_image=Image.open(resource_path(os.path.join(
            "images", "generate_password_logo_dark_theme.png"))),
        size=(100, 100))

```

```

self.__main_image_label = ctk.CTkLabel(
    self, image=self.__main_image, text="")
self.__main_image_label.pack(pady=[30, 0])
self.__main_title_label = ctk.CTkLabel(
    self, text="Новий пароль", font=self.__app._title_font)
self.__main_title_label.pack(pady=[20, 0])
self.__main_text_label = ctk.CTkLabel(
    self, text="Ви можете згенерувати новий пароль",
font=self.__app._text_font)
self.__main_text_label.pack(pady=[10, 20])
def create_generate_block(self):
self.__generate_frame = ctk.CTkFrame(
    self, fg_color="transparent")
self.__generate_frame.pack(fill="x")
for column, weight in enumerate((1, 0, 1)):
    self.__generate_frame.grid_columnconfigure(
        column, weight=weight)
self.__generate_frame.grid_rowconfigure(0, weight=1)
self.__generate_frame_cell = ctk.CTkFrame(
    self.__generate_frame)
self.__generate_frame_cell.grid(row=0, column=1, sticky="ew")
self.__generate_frame_cell_label = ctk.CTkLabel(
    self.__generate_frame_cell, width=100, text="      ",
font=self.__app._text_font)
self.__generate_frame_cell_label.pack(side="left")
self.__generate_frame_cell_button_image =
ctk.CTkImage(light_image=Image.open(
    resource_path(os.path.join("images",
"copy_logo_light_theme.png"))),
    dark_image=Image.open(resource_path(
    os.path.join("images", "copy_logo_dark_theme.png"))),
    size=(20, 20))
self.__generate_frame_cell_button = ctk.CTkButton(
    self.__generate_frame_cell, width=25, text="",
fg_color="transparent", image=self.__generate_frame_cell_button_image,
hover_color=["#FFFFFF", "#000000"], command=self.copy_password)
self.__generate_frame_cell_button.pack(

```

```

        side="left", padx=[5, 0])
self.__generate_slider_label = ctk.CTkLabel(
    self, text="Довжина пароля: 8", font=self.__app._text_font)
self.__generate_slider_label.pack()
self.__generate_slider = ctk.CTkSlider(
    self, from_=4, to=25, number_of_steps=21,
command=self.change_slider_label)
self.__generate_slider.set(8)
self.__generate_slider.pack(pady=[0, 20])
self.__generate_button = ctk.CTkButton(
    self, text="Згенерувати", font=self.__app._text_font,
command=self.show_password)
self.__generate_button.pack()
def create_checkbox_block(self):
self.__checkbox_frame = ctk.CTkFrame(self, fg_color="transparent")
self.__checkbox_frame.pack(
    fill="x", pady=20, after=self.__generate_frame)
for column, weight in enumerate((1, 0, 0, 0, 0, 1)):
    self.__checkbox_frame.grid_columnconfigure(column,
weight=weight)
self.__checkbox_frame.grid_rowconfigure((0, 1), weight=1)
self.__checkbox_frame_lower = ctk.CTkCheckBox(
    self.__checkbox_frame, text="Нижній",
font=self.__app._text_font)
self.__checkbox_frame_lower.grid(row=0, column=1)
self.__checkbox_frame_upper = ctk.CTkCheckBox(
    self.__checkbox_frame, text="Верхній",
font=self.__app._text_font)
self.__checkbox_frame_upper.grid(row=0, column=2)
self.__checkbox_frame_digits = ctk.CTkCheckBox(
    self.__checkbox_frame, text="Цифри",
font=self.__app._text_font)
self.__checkbox_frame_digits.select()
self.__checkbox_frame_digits.grid(row=0, column=3)
self.__checkbox_frame_punctuation =
ctk.CTkCheckBox(
    self.__checkbox_frame, text="Спец-символи",
font=self.__app._text_font)

```

```

        self.__checkbox_frame_punctuation.grid(row=0, column=4)
def create_message_block(self):
    self.__message_label = ctk.CTkLabel(
        self, text="", font=self.__app._message_font)
    self.__message_label.pack(side="bottom")
def change_slider_label(self, value):
    self.__generate_slider_label.configure(
        text=f" Довжина пароля: {int(value)}")
def generate_password(self):
    self.__options["lower"] = self.__checkbox_frame_lower.get()
    self.__options["upper"] = self.__checkbox_frame_upper.get()
    self.__options["digits"] = self.__checkbox_frame_digits.get()
    self.__options["punctuation"] =
self.__checkbox_frame_punctuation.get()
    self.__chars = set_options(self.__options)
    self.__length = int(self.__generate_slider.get())
    self.__password = generate_password(self.__length, self.__chars)
def show_password(self):
    self.generate_password()
    if self.__password is not None:
        self.__generate_frame_cell_label.configure(
            text=self.__password)
        show_message(self.__message_label, "")
    else:
        self.__generate_frame_cell_label.configure(
            text=" ")
        show_message(self.__message_label,
            "Помилка: Не задані значення для пароля")
def copy_password(self):
    if self.__password is not None:
        self.clipboard_clear()
        self.clipboard_append(self.__password)
        self.__generate_frame_cell_label.configure(
            text=" ")
        self.__password = None
        show_message(self.__message_label,
            "Пароль скопійований до буфера обміну")

```

```

else:
    self.__generate_frame_cell_label.configure(
        text="                ")
    show_message(self.__message_label,
                  "Помилка: Пароль не згенерований")
def show_next_page(self):
    self.__message_label.configure(text="")
def clear_page(self):
    self.__generate_frame_cell_label.configure(text="                ")
    self.__checkbox_frame_lower.deselect()
    self.__checkbox_frame_upper.deselect()
    self.__checkbox_frame_digits.select()
    self.__checkbox_frame_punctuation.deselect()
    self.__generate_slider.set(8)
    self.change_slider_label(8)

```

Код реалізації класу AddPasswordPage

```

import customtkinter as ctk
from PIL import Image
import os
from utils.generate_password import set_options, generate_password
from utils.find_path_file import resource_path
from utils.show_message import show_message
class AddPasswordPage(ctk.CTkFrame):
    def __init__(self, parent, app):
        super().__init__(parent)
        self.__app = app
        self.create_main_block()
        self.create_add_block()
        self.create_message_block()
    def create_main_block(self):
        self.__main_image = ctk.CTkImage(light_image=Image.open(
            resource_path(os.path.join("images",
"add_password_logo_light_theme.png"))),
            dark_image=Image.open(resource_path(os.path.join(

```

```

        "images", "add_password_logo_dark_theme.png"))),
        size=(100, 100))
self.__main_image_label = ctk.CTkLabel(
    self, image=self.__main_image, text="")
self.__main_image_label.pack(pady=[30, 0])
self.__main_title_label = ctk.CTkLabel(
    self, text="Додати пароль", font=self.__app._title_font)
self.__main_title_label.pack(pady=[20, 0])
self.__main_text_label = ctk.CTkLabel(
    self, text="Збережіть новий пароль у віддаленому сховищі",
font=self.__app._text_font)
self.__main_text_label.pack(pady=[10, 20])
def create_add_block(self):
    self.__add_frame = ctk.CTkFrame(self, fg_color="transparent")
    self.__add_frame.pack(fill="x")
    for column, weight in enumerate((1, 0, 0, 1)):
        self.__add_frame.grid_columnconfigure(column, weight=weight)
    self.__add_frame.grid_rowconfigure(0, weight=1)
    self.__add_frame_password = ctk.CTkEntry(
        self.__add_frame, width=200, font=self.__app._text_font,
placeholder_text="Пароль")
    self.__add_frame_password.grid(row=0, column=1, padx=[0, 20])
    self.__add_frame_service = ctk.CTkEntry(
        self.__add_frame, width=200, font=self.__app._text_font,
placeholder_text="Назва сервісу")
    self.__add_frame_service.grid(row=0, column=2)
    self.__add_frame_description_text = ctk.CTkLabel(
        self, text="Введіть додаткову інформацію щодо паролю",
font=self.__app._text_font)
    self.__add_frame_description_text.pack(pady=5)
    self.__add_frame_description = ctk.CTkTextbox(
        self, width=420, height=150, wrap="word",
font=self.__app._text_font)
    self.__add_frame_description.pack()
    self.__add_frame_button = ctk.CTkButton(
        self, width=200, text="Додати", font=self.__app._text_font,
command=self.check_entry_add_block)

```

```

        self.__add_frame_button.pack(pady=[10, 0])
    def create_message_block(self):
        self.__message_label = ctk.CTkLabel(
            self, text="", font=self.__app._message_font)
        self.__message_label.pack(side="bottom")
    def check_entry_add_block(self):
        password = self.__add_frame_password.get().strip()
        service = self.__add_frame_service.get().strip()
        description = self.__add_frame_description.get("1.0",
"end").strip()
        if not password or not service:
            show_message(self.__message_label,
                "Помилка: Потрібно обов'язково ввести пароль та
сервіс")
        elif not (2 <= len(service) <= 25):
            show_message(self.__message_label,
                "Помилка: Некоректний сервіс (від 2 до 25
символів)")
        elif not (4 <= len(password) <= 25):
            show_message(self.__message_label,
                "Помилка: Некоректний пароль (від 4 до 25
символів)")
        elif len(description) > 250:
            show_message(self.__message_label,
                "Помилка: Некоректний опис (до 250 символів)")
        else:
            show_message(self.__message_label, "")
            self.add_password(password, service, description)
    def add_password(self, password, service, description):
        request = self.__app._client.message(
            header="add", password=password, service=service,
description=description, token=self.__app.get_token())
        result = self.__app._client.send(request)
        if result:
            show_message(self.__message_label, result)
            return
        answer = self.__app._client.get()

```

```

if not answer:
    show_message(self.__message_label,
                  "Помилка: Час очікування відповіді вичерпаний")
    return
if answer.get("header") == "add-respond" and answer.get("answer"):
    show_message(self.__message_label,
                  "Пароль успішно доданий до сховища")
    self.__add_frame_password.delete(0, "end")
    self.__add_frame_service.delete(0, "end")
    self.__add_frame_description.delete("0.0", "end")
    self.__add_frame_password._activate_placeholder()
    self.__add_frame_service._activate_placeholder()
    self.focus()
    self.__app.get_passwords()
else:
    show_message(self.__message_label,
                  "Помилка: Пароль вже наявний у базі даних, створіть
новий")
def show_next_page(self):
    self.__message_label.configure(text="")
    self.focus()
def clear_page(self):
    self.__add_frame_password.delete(0, "end")
    self.__add_frame_service.delete(0, "end")
    self.__add_frame_description.delete("0.0", "end")
    self.__add_frame_password._activate_placeholder()
    self.__add_frame_service._activate_placeholder()
    self.focus()

```

Код реалізації класу **RefugePage**

```

import customtkinter as ctk
from PIL import Image
import os
from utils.generate_password import set_options, generate_password

```

```

from utils.find_path_file import resource_path
from utils.show_message import show_message
class RefugePage(ctk.CTkFrame):
    def __init__(self, parent, app):
        super().__init__(parent)
        self.__app = app
        self.__frames = []
        self.__passwords = []
        self.create_main_block()
        self.create_refuge_block()
        self.create_message_block()
    def create_main_block(self):
        self.__main_image = ctk.CTkImage(light_image=Image.open(
            resource_path(os.path.join("images",
"refuge_logo_light_theme.png"))),
            dark_image=Image.open(resource_path(os.path.join(
                "images", "refuge_logo_dark_theme.png"))),
            size=(100, 100))
        self.__main_image_label = ctk.CTkLabel(
            self, image=self.__main_image, text="")
        self.__main_image_label.pack(pady=[30, 0])
        self.__main_title_label = ctk.CTkLabel(
            self, text="Сховище", font=self.__app._title_font)
        self.__main_title_label.pack(pady=[20, 0])
        self.__main_text_label = ctk.CTkLabel(
            self, text="Ваші збережені паролі", font=self.__app._text_font)
        self.__main_text_label.pack(pady=[10, 15])
    def create_refuge_block(self):
        self.__refuge_filter_search_frame = ctk.CTkFrame(self)
        self.__refuge_filter_search_frame.grid_columnconfigure(
            (0, 1, 2, 3), weight=0)
        self.__refuge_filter_search_frame.grid_rowconfigure(0, weight=1)
        self.__refuge_filter_search_frame.pack(pady=[0, 10])
        self.__refuge_filter_search_frame_entry = ctk.CTkEntry(
            self.__refuge_filter_search_frame, width=150,
font=self.__app._text_font, placeholder_text="Пошук за сервісом")

```

```

self.__refuge_filter_search_frame_entry.grid(row=0, column=1)
self.__refuge_filter_search_frame_button = ctk.CTkButton(
    self.__refuge_filter_search_frame, width=150, text="Знайти",
font=self.__app._text_font, command=lambda: self.fiter_password_frames())
self.__refuge_filter_search_frame_button.grid(
    row=0, column=2, padx=[10, 0])

self.__refuge_frame = ctk.CTkScrollableFrame(self, width=550,
height=170, corner_radius=10, border_width=2, border_color=["#000000",
"#FFFFFF"], fg_color=["#FFFFFF", "#000000"], label_fg_color=[
                                                                    "#FFFFFF", "#000000"],
label_text_color=["#000000",
                                                                    "#FFFFFF"],
scrollbar_fg_color="transparent", scrollbar_button_color=["#000000",
"#FFFFFF"],)

self.__refuge_frame.pack(padx=10)
self.__refuge_frame._scrollbar.grid_configure(padx=(0, 4))
def create_message_block(self):
    self._message_label = ctk.CTkLabel(
        self, text="", font=self.__app._message_font)
    self._message_label.pack(side="bottom")
def create_password_frames(self, answer, is_show_message):
    for frame in self.__frames:
        frame.destroy()
    self.__frames = []
    self.__passwords = []
    if not answer:
        show_message(self._message_label,
                    "Помилка: Виникла проблема під час передачі
повідомлення")
        return
    if answer.get("header") == "get-respond" and answer.get("answer")
and answer.get("passwords"):
        self.__passwords = answer["passwords"]
        self.__passwords.sort(key=lambda x: x[1])
        for element in self.__passwords:
            frame = PasswordFrame(parent=self,
master=self.__refuge_frame, identifer=element[0], password=element[1],
service=element[2], description=element[3], app=self.__app)

```

```

        self.__frames.append(frame)
    elif is_show_message:
        show_message(self._message_label,
                      "Даних користувача не знайдено")
def filter_password_frames(self):
    word = self.__refuge_filter_search_frame_entry.get()
    if not word:
        self.render_password_frames(self.__passwords)
        return
    results = self.find_passwords(word)
    self.render_password_frames(results)
def render_password_frames(self, passwords):
    for frame in self.__frames:
        frame.destroy()
    self.__frames = []
    for element in passwords:
        frame = PasswordFrame(
            parent=self,
            master=self.__refuge_frame,
            identifier=element[0],
            password=element[1],
            service=element[2],
            description=element[3],
            app=self.__app
        )
        self.__frames.append(frame)
        frame.pack(fill="x", padx=5, pady=5)
def find_passwords(self, word):
    word = word.lower()
    return [
        element for element in self.__passwords
        if word in element[2].lower()
    ]
def show_next_page(self):
    self._message_label.configure(text="")
def clear_page(self):
    self.__refuge_filter_search_frame_entry.delete(0, "end")

```

```
self.__refuge_filter_search_frame_entry._activate_placeholder()
self.focus()
```

Код реалізації класу PasswordFrame

```
import customtkinter as ctk
from PIL import Image
import os
from utils.generate_password import set_options, generate_password
from utils.find_path_file import resource_path
from utils.show_message import show_message
class PasswordFrame(ctk.CTkFrame):
    def __init__(self, parent, master, identifier, password, service,
description, app):
        super().__init__(master, height=55, fg_color=[
            "#FFFFFF", "#000000"], corner_radius=10, border_width=2,
border_color=["#000000", "#FFFFFF"])
        self.__parent = parent
        self.__identifier = identifier
        self.__password = password
        self.__cut_password = self.cut_word(self.__password)
        self.__service = service
        self.__cut_service = self.cut_word(self.__service)
        self.__description = description
        self.__app = app
        self.create_frame()
    def create_frame(self):
        self.pack(fill="x", padx=5, pady=5)
        self.pack_propagate(False)
        self.grid_columnconfigure((0, 1, 2, 3, 4, 5), weight=0)
        self.grid_rowconfigure(0, weight=1)
        self.frame_password_image = ctk.CTkImage(
            light_image=Image.open(resource_path(
                os.path.join("images", "refuge_logo_light_theme.png"))),
            dark_image=Image.open(resource_path(
                os.path.join("images", "refuge_logo_dark_theme.png"))),
```

```

        size=(40, 40)
    )
    self.image_label = ctk.CTkLabel(self,
        image=self.frame_password_image, text=""
    )
    self.image_label.grid(row=0, column=0, padx=[10, 5], pady=5)
    self.service_cell = ctk.CTkFrame(
        self, fg_color=["#FFFFFF", "#000000"],
        corner_radius=5, border_width=2,
        border_color=["#000000", "#FFFFFF"]
    )
    self.service_cell.grid(row=0, column=1, pady=5)
    self.service_label = ctk.CTkLabel(
        self.service_cell,
        width=200,
        text=self.__cut_service,
        font=self.__app._text_font
    )
    self.service_label.pack(pady=5, padx=5)
    self.copy_image = ctk.CTkImage(
        light_image=Image.open(resource_path(
            os.path.join("images", "copy_logo_light_theme.png"))),
        dark_image=Image.open(resource_path(
            os.path.join("images", "copy_logo_dark_theme.png"))),
        size=(20, 20)
    )
    self.copy_button = ctk.CTkButton(
        self,
        width=25,
        text="",
        fg_color="transparent",
        image=self.copy_image,
        hover_color=["#FFFFFF", "#000000"],
        command=self.copy_password
    )
    self.copy_button.grid(row=0, column=3)

```

```

self.view_button = ctk.CTkButton(
    self,
    width=110,
    height=37,
    text="Переглянути",
    font=self.__app._text_font,
    command=self.view_password_window
)
self.view_button.grid(row=0, column=4, padx=5)
self.delete_button = ctk.CTkButton(
    self,
    width=110,
    height=37,
    text="Видалити",
    font=self.__app._text_font,
    command=self.delete_password_window
)
self.delete_button.grid(row=0, column=5)
def copy_password(self):
    self.clipboard_clear()
    self.clipboard_append(self.__password)
    show_message(self.__parent._message_label,
                 "Пароль скопійований до буфера обміну")
def view_password_window(self):
    self.__view_password_window = ctk.CTkToplevel(self)
    self.center_window(self.__view_password_window, 450, 370)
    self.__view_password_window.overridereirect(True)
    self.__view_password_window.title(self.__service)
    self.__view_password_window.grab_set()
    self.__view_password_window.focus()
    self.__view_password_window_container = ctk.CTkFrame(
        self.__view_password_window,
        fg_color=["#FFFFFF", "#000000"],
        corner_radius=15,
        border_width=2,
        border_color=["#000000", "#FFFFFF"]
    )

```

```

        self.__view_password_window_container.pack(fill="both",
expand=True)

        self.__view_password_window_container_title_label = ctk.CTkLabel(
            self.__view_password_window_container, text="Пароль",
font=self.__app._title_font)
        self.__view_password_window_container_title_label.pack(pady=[20,
0])

        self.__view_password_window_container_text_label = ctk.CTkLabel(
            self.__view_password_window_container, text="Ви можете змінити
параметри пароля", font=self.__app._text_font)
        self.__view_password_window_container_text_label.pack(pady=[10,
20])

        self.__view_password_window_container_frame = ctk.CTkFrame(
            self.__view_password_window_container, fg_color="transparent")
        self.__view_password_window_container_frame.pack(fill="x",
padx=10)

        self.__view_password_window_container_frame.grid_columnconfigure(
            (0, 1, 2, 3), weight=1)
        self.__view_password_window_container_frame.grid_rowconfigure(
            0, weight=1)

        self.__view_password_window_container_frame_service_label =
ctk.CTkLabel(
            self.__view_password_window_container_frame, text="Сервіс:",
font=self.__app._text_font)
        self.__view_password_window_container_frame_service_label.grid(
            row=0, column=0)

        self.__view_password_window_container_frame_service =
ctk.CTkEntry(
            self.__view_password_window_container_frame, width=150,
font=self.__app._text_font)
        self.__view_password_window_container_frame_service.grid(
            row=0, column=1)
        self.__view_password_window_container_frame_service.insert(
            0, self.__service)
        self.__view_password_window_container_frame_service.configure(
            state="readonly")

```

```

        self.__view_password_window_container_frame_password_label =
ctk.CTkLabel(
            self.__view_password_window_container_frame, text="Пароль:",
font=self.__app._text_font)
        self.__view_password_window_container_frame_password_label.grid(
            row=0, column=2)
        self.__view_password_window_container_frame_password =
ctk.CTkEntry(
            self.__view_password_window_container_frame, width=150,
font=self.__app._text_font)
        self.__view_password_window_container_frame_password.grid(
            row=0, column=3)
        self.__view_password_window_container_frame_password.insert(
            0, self.__password)
        self.__view_password_window_container_description_label =
ctk.CTkLabel(
            self.__view_password_window_container, text="Додаток до
пароля",
font=self.__app._text_font)
        self.__view_password_window_container
_description_label.pack(pady=[
            10, 0])
        self.__view_password_window_container_description =
ctk.CTkTextbox(
            self.__view_password_window_container, width=420, height=150,
wrap="word", font=self.__app._text_font)
        self.__view_password_window_container_description.pack(
            padx=10, pady=10)
        self.__view_password_window_container_description.insert(
            "1.0", self.__description)
        self.__view_password_window_container_bootom_frame = ctk.CTkFrame(
            self.__view_password_window_container, fg_color="transparent")
        self.__view_password_window_container_bootom_frame.pack(
            fill="x",
padx=5)
        self.__view_password_window_container_bootom_frame.grid_c
olumnconfigure(
            (0, 1), weight=1)

```

```

self.__view_password_window_container_bootom_frame.grid_rowconfigure(
    0, weight=1)
    self.__view_password_window_container_bootom_frame_cancel =
ctk.CTkButton(
    self.__view_password_window_container_bootom_frame, width=200,
text="Скасувати",                                font=self.__app._text_font,
command=self.close_view_password_window)
    self.__view_password_window_container_bootom_frame_cancel.grid(
        row=0, column=0)
        self.__view_password_window_container_bootom_frame_save =
ctk.CTkButton(
    self.__view_password_window_container_bootom_frame, width=200,
text="Зберегти",                                font=self.__app._text_font,
command=self.check_entry_password_window)
    self.__view_password_window_container_bootom_frame_save.grid(
        row=0, column=1)
    self.__view_password_window.wait_window()
def close_view_password_window(self):
    self.__view_password_window.grab_release()
    self.__view_password_window.destroy()
    def check_entry_password_window(self):
        password =
self.__view_password_window_container_frame_password.get().strip()
        description =
self.__view_password_window_container_description.get(
    "1.0", "end").strip()
        if not password:
            self.__view_password_window.grab_release()
            self.__view_password_window.destroy()
            show_message(self.__parent._message_label,
                "Помилка: Потрібно обов'язково ввести пароль")
        elif not (4 <= len(password) <= 25):
            show_message(self.__message_label,
                "Помилка: Некоректний пароль (від 4 до 25
СИМВОЛІВ)")
        elif len(description) > 250:
            show_message(self.__message_label,
                "Помилка: Некоректний опис (до 250 символів)")

```

```

else:
    show_message(self.__parent._message_label, "")
    self.update_password(password=password,
description=description)
def update_password(self, password, description):
    request = self.__app._client.message(
        header="update",    identifier=self.__identifier,
password=password, description=description, token=self.__app.get_token())
    result = self.__app._client.send(request)
    if result:
        show_message(self.__parent._message_label, result)
        return
    answer = self.__app._client.get()
        if    answer.get("header")    ==    "update-respond"    and
answer.get("answer"):
        show_message(self.__parent._message_label,
            "Пароль успішно змінений у сховищі")
        self.__app.get_passwords()
    else:
        show_message(self.__parent._message_label,
            "Помилка: Не вдалось оновити пароль")
        self.__view_password_window.grab_release()
        self.__view_password_window.destroy()
def delete_password(self):
    request = self.__app._client.message(
        header="delete",    identifier=self.__identifier,
token=self.__app.get_token())
    result = self.__app._client.send(request)
    if result:
        show_message(self.__parent._message_label, result)
        return
    answer = self.__app._client.get()
        if    answer.get("header")    ==    "delete-respond"    and
answer.get("answer"):
        show_message(self.__parent._message_label,
            "Пароль успішно видалений зі сховища")
        self.pack_forget()

```

```

        self.destroy()
        self.__app.get_passwords(False)
    else:
        show_message(self.__parent._message_label,
                      "Помилка: Не вдалось видалити пароль")

def delete_password_window(self):
    self.__delete_password_window = ctk.CTkToplevel(self)
    self.center_window(self.__delete_password_window, 450, 170)
    self.__delete_password_window.overridereirect(True)
    self.__delete_password_window.title(self.__service)
    self.__delete_password_window.grab_set()
    self.__delete_password_window.focus()
    self.__delete_password_window_container = ctk.CTkFrame(
        self.__delete_password_window,
        fg_color=["#FFFFFF", "#000000"],
        corner_radius=15,
        border_width=2,
        border_color=["#000000", "#FFFFFF"]
    )
    self.__delete_password_window_container.pack(fill="both",
expand=True)

    self.__main_title_label = ctk.CTkLabel(
        self.__delete_password_window_container, text="Ви впевнені що
хочете видалити \n цей пароль?", font=self.__app._title_font)
    self.__main_title_label.pack(pady=[20, 0])
    self.__main_text_label = ctk.CTkLabel(
        self.__delete_password_window_container, text=f"Сервіс:
{self.__service} - пароль: {self.__password}",
font=self.__app._text_font)
    self.__main_text_label.pack(pady=[10, 20])
    self.__delete_password_window_bottom_frame = ctk.CTkFrame(
        self.__delete_password_window_container,
fg_color="transparent")
    self.__delete_password_window_bottom_frame.pack(fill="x", padx=5)
    self.__delete_password_window_bottom_frame.grid_columnconfigure(
        (0, 1), weight=1)
    self.__delete_password_window_bottom_frame.grid_rowconfigure(

```

```

        0, weight=1)
    self.__delete_password_window_bottom_frame_cancel = ctk.CTkButton(
        self.__delete_password_window_bottom_frame, width=200,
        text="Скасувати", font=self.__app._text_font,
        command=self.close_delete_password_window)
    self.__delete_password_window_bottom_frame_cancel.grid(row=0,
        column=0)

    self.__delete_password_window_bottom_frame_confirm =
    ctk.CTkButton(
        self.__delete_password_window_bottom_frame, width=200,
        text="Підтвердити", font=self.__app._text_font,
        command=self.delete_password)
    self.__delete_password_window_bottom_frame_confirm.grid(
        row=0, column=1)
    self.__delete_password_window.wait_window()
def close_delete_password_window(self):
    self.__delete_password_window.grab_release()
    self.__delete_password_window.destroy()
def center_window(self, window, width, height):
    self.update_idletasks()
    screen_width = self.winfo_screenwidth()
    screen_height = self.winfo_screenheight()
    x = (screen_width // 2) - (width // 2)
    y = (screen_height // 2) - (height // 2)
    window.geometry(f"{width}x{height}+{x}+{y}")
    @staticmethod
def cut_word(word):
    if len(word) > 20:
        word = word[0:17]
        word += "... "
    return word

```

Код реалізації класу SettingsPage

```
import customtkinter as ctk
```

```

from PIL import Image
import os
from utils.generate_password import set_options, generate_password
from utils.find_path_file import resource_path
from utils.show_message import show_message
class SettingsPage(ctk.CTkFrame):
    def __init__(self, parent, app):
        super().__init__(parent)
        self.__app = app
        self.__theme_options = ["Світла", "Темна", "Системна"]
        self.create_main_block()
        self.create_them_lang_block()
        self.create_account_block()
        self.create_message_block()
    def create_main_block(self):
        self.__main_image = ctk.CTkImage(light_image=Image.open(
            resource_path(os.path.join("images",
"settings_logo_light_theme.png"))),
            dark_image=Image.open(resource_path(os.path.join(
                "images", "settings_logo_dark_theme.png"))),
            size=(100, 100))
        self.__main_image_label = ctk.CTkLabel(
            self, image=self.__main_image, text="")
        self.__main_image_label.pack(pady=[30, 0])
        self.__main_title_label = ctk.CTkLabel(
            self, text="Налаштування", font=self.__app._title_font)
        self.__main_title_label.pack(pady=[20, 0])
        self.__main_text_label = ctk.CTkLabel(
            self, text="Змініть параметри програми",
font=self.__app._text_font)
        self.__main_text_label.pack(pady=[10, 20])
    def create_them_lang_block(self):
        self.__theme_language_title = ctk.CTkLabel(
            self, text="Оформлення", font=self.__app._title_font)
        self.__theme_language_title.pack(
            pady=[0, 10], padx=[40, 0], anchor="nw")
        self.__theme_language_frame = ctk.CTkFrame(

```

```

        self, fg_color="transparent")
self.__theme_language_frame.pack(fill="x")
for column, weight in enumerate((1, 0, 0, 1)):
    self.__theme_language_frame.grid_columnconfigure(
        column, weight=weight)
self.__theme_language_frame.grid_rowconfigure(0, weight=0)
self.__theme_language_frame_cell_theme = ctk.CTkFrame(
    self.__theme_language_frame, fg_color="transparent")
self.__theme_language_frame_cell_theme.grid(row=0, column=1)
self.__theme_language_frame_cell_theme_label = ctk.CTkLabel(
    self.__theme_language_frame_cell_theme, text="Тема",
font=self.__app._text_font)
self.__theme_language_frame_cell_theme_label.pack(
    side="left", anchor="nw", padx=[0, 10])
self.__theme_language_frame_cell_theme_option = ctk.CTkOptionMenu(
    self.__theme_language_frame_cell_theme,
values=self.__theme_options, font=self.__app._text_font,
command=self.set_theme_option
)
self.__theme_language_frame_cell_theme_option.pack(
    side="left", anchor="nw", padx=[0, 55])
def create_account_block(self):
    self.__account_title = ctk.CTkLabel(
        self, text="Акаунт", font=self.__app._title_font)
        self.__account_title.pack(pady=[20, 10], padx=[40, 0],
anchor="nw")
    self.__account_frame = ctk.CTkFrame(
        self, fg_color="transparent")
    self.__account_frame.pack(fill="x")
    for column, weight in enumerate((1, 0, 0, 1)):
        self.__account_frame.grid_columnconfigure(
            column, weight=weight)
    self.__account_frame.grid_rowconfigure(0, weight=1)
    self.__account_frame_cell_logout_button = ctk.CTkButton(
        self.__account_frame, width=200, text="Вийти",
font=self.__app._text_font, command=self.__app.show_login)

```

```

self.__account_frame_cell_logout_button.grid(row=0, column=1)
def create_message_block(self):
    self.__message_label = ctk.CTkLabel(
        self, text="", font=self.__app._message_font)
    self.__message_label.pack(side="bottom")
def set_message_label(self, text):
    self.__message_label.configure(text=text)
def set_theme_option(self, value):
    if self.__theme_options.index(value) == 0:
        self.__theme_language_frame_cell_theme_option.set(
            self.__theme_options[0])
        self.__app.change_theme("Light")
        self.__message_label.configure(text="Тема: світла")
    if self.__theme_options.index(value) == 1:
        self.__theme_language_frame_cell_theme_option.set(
            self.__theme_options[1])
        self.__app.change_theme("Dark")
        self.__message_label.configure(text="Тема: темна")
    if self.__theme_options.index(value) == 2:
        self.__theme_language_frame_cell_theme_option.set(
            self.__theme_options[2])
        self.__app.change_theme("System")
        self.__message_label.configure(text="Тема: системна")
def show_next_page(self):
    label = self.__message_label.cget("text")
    if label != "Помилка: Некоректне значення в налаштуваннях, тема за
замовчуванням (Системна)":
        self.__message_label.configure(text="")

```

ДОДАТОК Б

Код реалізації класу Client

```
import json
import os
from connection.tls_context import TLSContext
from connection.secure_socket import SecureSocket
from utils.find_path_file import resource_path
class Client:
    def __init__(self):
        self.__ca_cert = resource_path(os.path.join("secret", "ca.crt"))
        self.__client_cert = resource_path(
            os.path.join("secret", "client.crt"))
        self.__client_key = resource_path(os.path.join("secret",
"client.key"))
        self.__hostname = "127.0.0.1"
        self.__port = 9000
        self.__tls = TLSContext(self.__ca_cert,
                                self.__client_cert, self.__client_key)
        self.__context = self.__tls.build_context()
        self.__connetion = SecureSocket(
            self.__hostname, self.__port, self.__context)
    def connect(self):
        return self.__connetion.connect()
    def close(self):
        self.__connetion.close()
    def message(self, header, login="", email="", identifier="",
password="", service="", description="", token=""):
        if header == "authentication":
            return {"header": header, "login": login, "password": password}
        elif header == "registration":
            return {"header": header, "login": login, "password": password,
"email": email}
        elif header == "add":
```

```

        return {"header": header, "password": password, "service":
service, "description": description, "token": token}
    elif header == "get":
        return {"header": header, "token": token}
    elif header == "update":
        return {"header": header, "identifier": identifier, "password":
password, "description": description, "token": token}
    elif header == "delete":
        return {"header": header, "identifier": identifier, "token":
token}

    def send(self, data):
        message = self.encode(data)
        return self.__connetion.send(message)

    def get(self):
        message = self.__connetion.get()
        if message:
            message = self.decode(message)
            return message

    def encode(self, data):
        message = json.dumps(data)
        message += "\n"
        message = message.encode()
        return message

    def decode(self, data):
        message = json.loads(data)
        return message

```

Код реалізації класу SecureSocket

```

import socket

class SecureSocket:
    def __init__(self, host, port, ssl_context):
        self.__host = host
        self.__port = port
        self.__ssl_context = ssl_context
        self.__socket = None

```

```

self.__secure_socket = None
self.__connected = False
self.__buffer = ""
def connect(self):
    try:
        self.__socket = socket.create_connection(
            (self.__host, self.__port))
        self.__secure_socket = self.__ssl_context.wrap_socket(
            self.__socket,
            server_hostname=self.__host
        )
        self.__secure_socket.settimeout(60)
        self.__connected = True
    except socket.gaierror:
        self.__connected = False
        return "Не вдалося знайти IP-адресу для вказаного хоста"
    except ConnectionRefusedError:
        self.__connected = False
        return "Відсутнє з'єднання із сервером"
    except socket.timeout:
        self.__connected = False
        return "Вичерпаний час з'єднання із сервером"
    except OSError:
        self.__connected = False
        return "Мережева помилка"
    except Exception:
        self.__connected = False
        return "Помилка з'єднання"
def close(self):
    try:
        if self.__secure_socket:
            self.__secure_socket.close()
    finally:
        self.__connected = False
        self.__secure_socket = None
def send(self, message):
    try:

```

```

    if not self.__connected:
        error_connection = self.connect()
        if error_connection:
            return f"Помилка: {error_connection}"
        self.__secure_socket.sendall(message)
except BrokenPipeError:
    self.__connected = False
    return "Помилка: Сервер закрив з'єднання"
except ConnectionResetError:
    self.__connected = False
    return "Помилка: З'єднання було скинуто сервером"
except ConnectionAbortedError:
    self.__connected = False
    return "Помилка: З'єднання було перервано"
except socket.timeout:
    self.__connected = False
    return "Помилка: Перевищено час очікування"
except OSError:
    self.__connected = False
    return f"Помилка: Мережева помилка"
except Exception:
    self.__connected = False
    return f"Помилка: Не вдалось надіслати дані"
def get(self):
    try:
        while True:
            if "\n" in self.__buffer:
                break
            chunk = self.__secure_socket.recv(4096).decode()
            if not chunk:
                return None
            self.__buffer += chunk
        message, self.__buffer = self.__buffer.split("\n", 1)
        return message.strip()
    except socket.timeout:
        self.__connected = False

```

Код реалізації класу TLSContext

```

import ssl
class TLSContext:
    def __init__(self, ca_cert, client_cert, client_key):
        self.__ca_cert = ca_cert
        self.__client_cert = client_cert
        self.__client_key = client_key
    def build_context(self):
        self.__context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)
        self.__context.load_verify_locations(cafile=self.__ca_cert)
        self.__context.load_cert_chain(
            certfile=self.__client_cert,
            keyfile=self.__client_key
        )
        self.__context.minimum_version = ssl.TLSVersion.TLSv1_2
        return self.__context

```

Вміст файлу «theme.json»

```

{
  "CTk": {
    "fg_color": ["#FFFFFF", "#000000"]
  },
  "CTkToplevel": {
    "fg_color": ["#FFFFFF", "#000000"]
  },
  "CTkFrame": {
    "corner_radius": 6,
    "border_width": 0,
    "fg_color": ["#FFFFFF", "#000000"],
    "top_fg_color": ["#FFFFFF", "#000000"],
    "border_color": ["#000000", "#FFFFFF"]
  },
}

```

```

"CTkButton": {
    "corner_radius": 6,
    "border_width": 0,
    "fg_color": ["#000000", "#FFFFFF"],
    "hover_color": ["#9C9C9C", "#9C9C9C"],
    "border_color": ["#000000", "#FFFFFF"],
    "text_color": ["#FFFFFF", "#000000"],
    "text_color_disabled": ["#B3B3B3", "#B3B3B3"]
},
"CTkLabel": {
    "corner_radius": 0,
    "fg_color": "transparent",
    "text_color": ["#000000", "#FFFFFF"]
},
"CTkEntry": {
    "corner_radius": 6,
    "border_width": 2,
    "fg_color": ["#FFFFFF", "#000000"],
    "border_color": ["#000000", "#FFFFFF"],
    "text_color": ["#000000", "#FFFFFF"],
    "placeholder_text_color": ["#D3D3D3", "#555555"]
},
"CTkCheckBox": {
    "corner_radius": 6,
    "border_width": 2.5,
    "fg_color": ["#FFFFFF", "#000000"],
    "border_color": ["#000000", "#FFFFFF"],
    "hover_color": ["#FFFFFF", "#000000"],
    "checkmark_color": ["#000000", "#FFFFFF"],
    "text_color": ["#000000", "#FFFFFF"],
    "text_color_disabled": ["#9C9C9C", "#9C9C9C"]
},
"CTkSwitch": {
    "corner_radius": 1000,
    "border_width": 3,
    "button_length": 0,
    "fg_color": ["#FFFFFF", "#000000"],

```

```

    "progress_color": ["#D3D3D3", "#555555"],
    "button_color": ["#FFFFFF", "#000000"],
    "button_hover_color": ["#A9A9A9", "#A9A9A9"],
    "text_color": ["#000000", "#FFFFFF"],
    "text_color_disabled": ["#D3D3D3", "#555555"]
},
"CTkRadioButton": {
    "corner_radius": 1000,
    "border_width_checked": 6,
    "border_width_unchecked": 3,
    "fg_color": ["#FFFFFF", "#000000"],
    "border_color": ["#000000", "#FFFFFF"],
    "hover_color": ["#A9A9A9", "#A9A9A9"],
    "text_color": ["#000000", "#FFFFFF"],
    "text_color_disabled": ["#D3D3D3", "#555555"]
},
"CTkProgressBar": {
    "corner_radius": 1000,
    "border_width": 0,
    "fg_color": ["#FFFFFF", "#000000"],
    "progress_color": ["#D3D3D3", "#555555"],
    "border_color": ["#000000", "#FFFFFF"]
},
"CTkSlider": {
    "corner_radius": 1000,
    "button_corner_radius": 1000,
    "border_width": 6,
    "button_length": 0,
    "fg_color": ["#9C9C9C", "#9C9C9C"],
    "progress_color": ["#000000", "#FFFFFF"],
    "button_color": ["#000000", "#FFFFFF"],
    "button_hover_color": ["#A9A9A9", "#A9A9A9"]
},
"CTkOptionMenu": {
    "corner_radius": 6,
    "fg_color": ["#000000", "#FFFFFF"],
    "button_color": ["#000000", "#FFFFFF"],

```

```

    "button_hover_color": ["#9C9C9C", "#9C9C9C"],
    "text_color": ["#FFFFFF", "#000000"],
    "text_color_disabled": ["#B3B3B3", "#B3B3B3"]
},
"CTkComboBox": {
    "corner_radius": 6,
    "border_width": 2,
    "fg_color": ["#FFFFFF", "#000000"],
    "border_color": ["#000000", "#FFFFFF"],
    "button_color": ["#FFFFFF", "#000000"],
    "button_hover_color": ["#A9A9A9", "#A9A9A9"],
    "text_color": ["#000000", "#FFFFFF"],
    "text_color_disabled": ["#D3D3D3", "#555555"]
},
"CTkScrollbar": {
    "corner_radius": 1000,
    "border_spacing": 4,
    "fg_color": "transparent",
    "button_color": ["#FFFFFF", "#000000"],
    "button_hover_color": ["#A9A9A9", "#A9A9A9"]
},
"CTkSegmentedButton": {
    "corner_radius": 6,
    "border_width": 2,
    "fg_color": ["#FFFFFF", "#000000"],
    "selected_color": ["#A9A9A9", "#A9A9A9"],
    "selected_hover_color": ["#9C9C9C", "#9C9C9C"],
    "unselected_color": ["#D3D3D3", "#555555"],
    "unselected_hover_color": ["#D3D3D3", "#555555"],
    "text_color": ["#000000", "#FFFFFF"],
    "text_color_disabled": ["#D3D3D3", "#555555"]
},
"CTkTextbox": {
    "corner_radius": 6,
    "border_width": 2,
    "fg_color": ["#FFFFFF", "#000000"],
    "border_color": ["#000000", "#FFFFFF"],

```

```

    "text_color": ["#000000", "#FFFFFF"],
    "scrollbar_button_color": ["#000000", "#FFFFFF"],
    "scrollbar_button_hover_color": ["#9C9C9C", "#9C9C9C"]
},
"CTkScrollableFrame": {
    "corner_radius": 0,
    "border_width": 2.5,
    "fg_color": ["#000000", "#000000"],
    "border_color": ["#000000", "#FFFFFF"],
    "label_fg_color": ["#FFFFFF", "#000000"],
    "label_text_color": ["#000000", "#FFFFFF"],
    "scrollbar_fg_color": ["#E0E0E0", "#1A1A1A"],
    "scrollbar_button_color": ["#808080", "#666666"],
    "scrollbar_button_hover_color": ["#606060", "#999999"]
},
"DropdownMenu": {
    "fg_color": ["#FFFFFF", "#000000"],
    "hover_color": ["#9C9C9C", "#9C9C9C"],
    "text_color": ["#000000", "#FFFFFF"]
},
"CTkFont": {
    "macOS": {"family": "Nunito", "size": 18, "weight": "normal"},
    "Windows": {"family": "Nunito", "size": 13, "weight": "normal"},
    "Linux": {"family": "Nunito", "size": 13, "weight": "normal"}
}
}

```

ДОДАТОК В

Код реалізації класу Handler

```
from argon2.exceptions import VerifyMismatchError
from datetime import datetime
import socketserver
import json
from utils.generate_token import generate_token
from utils.crypto import derive_master_key, generate_salt,
generate_file_key, wrap_dek, unwrap_dek, encrypt_data, decrypt_data
class Handler(socketserver.BaseRequestHandler):
    def handle(self):
        self.__buffer = ""
        self.__dbms = self.server.get_db()
        self.__password_hasher = self.server.get_password_hasher()
        self.__queue_logs = self.server.get_queue_logs()
        self.__dek = None
        log = f"\nNew connection [{datetime.now()}]"
        self.__queue_logs.put((log, False))
        try:
            while True:
                request = self.get()
                if not request:
                    log = f"Disconnected client [{datetime.now()}]"
                    self.__queue_logs.put((log, False))
                    break
                request = self.decode(request)
                self.route(request)
        except Exception as e:
            log = f"[ERROR] {type(e).__name__} ({e}) [{datetime.now()}]"
            self.__queue_logs.put((log, True))
            raise Exception(log)
        finally:
            self.__dek = None
            self.request.close()
```

```

def route(self, request):
    if request.get("header") == "authentication" and
request.get("login") and request.get("password"):
        self.authentication(request)
    elif request.get("header") == "registration" and
request.get("login") and request.get("password") and
request.get("email"):
        self.registration(request)
    else:
        token = request.get("token")
        if token:
            result = self.__dbms.select_tokens_tokens()
            is_token = False
            for element in result:
                try:
                    self.__password_hasher.verify(element[1], token)
                    user_id = element[0]
                    is_token = True
                except:
                    pass
            if is_token:
                is_token = False
                if request.get("header") == "add" and
request.get("password") and request.get("service"):
                    self.add_password(user_id, request)
                elif request.get("header") == "get":
                    self.get_passwords(user_id)
                elif request.get("header") == "update" and
request.get("identifier") and request.get("password"):
                    self.update_password(user_id, request)
                elif request.get("header") == "delete" and
request.get("identifier"):
                    self.delete_password(user_id, request)
            else:
                raise Exception(
                    f"[WARNING] Uncorrect message from client
{self.client_address} - {request}")

```

```

        else:
            raise Exception(
                f"[WARNING] No authorized client
{self.client_address}")
        else:
            raise Exception(
                f"[WARNING] Uncorrect message from client
{self.client_address} - {request}")
    def authentication(self, request):
        result = self.__dbms.select_user_users(
            login=request["login"])
        if result:
            user_id = result[0]
            hash_password_db = result[1]
            password_user = request["password"]
            wraped_dek = result[2]
            salt = result[3]
            try:
                self.__password_hasher.verify(hash_password_db,
password_user)
                kek = derive_master_key(password_user, salt)
                self.__dek = unwrap_dek(wraped_dek, kek)
                token = self.generate_token()
                if token:
                    hashed_token = self.__password_hasher.hash(token)
                    db_rows = self.__dbms.update_token_tokens(
                        user_id=user_id, token=hashed_token)
                    if db_rows:
                        answer = self.message(
                            header="authentication-respond", answer=True,
token=token)
                        log = f"Authentication: answer: {answer["answer"]}
[{{datetime.now()}}]"
                    else:
                        answer = self.message(
                            header="authentication-respond", answer=False)

```

```

        log = f"Authentication: answer: {answer['answer']}"
    [{datetime.now()}] "
    else:
        answer = self.message(
            header="authentication-respond", answer=False)
        log = f"Authentication: answer: {answer['answer']}"
    [{datetime.now()}] "
    except VerifyMismatchError:
        answer = self.message(
            header="authentication-respond", answer=False)
        log = f"Authentication: answer: {answer['answer']}"
    [{datetime.now()}] "
    else:
        answer = self.message(
            header="authentication-respond", answer=False)
        log = f"Authentication: answer: {answer['answer']}"
    [{datetime.now()}] "
    self.__queue_logs.put((log, False))
    self.send(answer)
def registration(self, request):
    salt = generate_salt()
    password_user = request["password"]
    email_user = request["email"]
    if isinstance(password_user, bytes):
        password_user = password_user.decode()
    password_user_key = derive_master_key(password_user, salt)
    self.__dek = generate_file_key()
    encrypted_email = encrypt_data(self.__dek, email_user)
    wrapped_dek = wrap_dek(self.__dek, password_user_key)
    result = self.__dbms.search_user_users(
        login=request["login"], email=encrypted_email)
    result = bool(result[0])
    if not result:
        hash_password_user = self.__password_hasher.hash(
            request["password"])
        user_id = self.__dbms.insert_user_users(

```

```

        login=request["login"], password=hash_password_user,
email=encrypted_email, key=wrapped_dek, salt=salt)
        if user_id:
            user_id = user_id[0]
            token = self.generate_token()
            if token:
                hashed_token = self.__password_hasher.hash(token)
                db_rows = self.__dbms.insert_token_tokens(
                    user_id=user_id, token=hashed_token)
                if db_rows:
                    answer = self.message(
                        header="registration-respond", answer=True,
token=token)

                    log = f"Registration: answer: {answer["answer"]}
[{{datetime.now()}}]"
                else:
                    answer = self.message(
                        header="registration-respond", answer=False)
                    log = f"Registration: answer: {answer["answer"]}
[{{datetime.now()}}]"
                else:
                    answer = self.message(
                        header="registration-respond", answer=False)
                    log = f"Registration: answer: {answer["answer"]}
[{{datetime.now()}}]"
                else:
                    answer = self.message(
                        header="registration-respond", answer=False)
                    log = f"Registration: answer: {answer["answer"]}
[{{datetime.now()}}]"
                else:
                    answer = self.message(header="registration-respond",
answer=False)

                    log = f"Registration: answer: {answer["answer"]}
[{{datetime.now()}}]"

                self.__queue_logs.put((log, False))
                self.send(answer)

```

```

def add_password(self, user_id, request):
    password_user = request["password"]
    description_user = request["description"]
    encrypt_password_user = encrypt_data(self.__dek, password_user)
    encrypt_description_user = encrypt_data(self.__dek,
description_user)
    db_rows = self.__dbms.insert_password_passwords(
        user_id=user_id, password=encrypt_password_user,
service=request["service"], description=encrypt_description_user)
    if db_rows:
        answer = self.message(header="add-respond", answer=True)
    else:
        answer = self.message(header="add-respond", answer=False)
        log = f"Add password: answer: {answer["answer"]}
[{{datetime.now()}}]"
        self.__queue_logs.put((log, False))
        self.send(answer)
def get_passwords(self, user_id):
    result = self.__dbms.select_password_passwords(user_id)
    if result:
        passwords_user = []
        for element in result:
            decrypted_password_user = decrypt_data(self.__dek,
element[1])
            decrypted_description_user = decrypt_data(
                self.__dek, element[3])
            passwords_user.append(
                (element[0], decrypted_password_user, element[2],
decrypted_description_user))
            answer = self.message(header="get-respond",
                answer=True, passwords=passwords_user)
        else:
            answer = self.message(header="get-respond", answer=False)
            log = f"Get passwords: answer: {answer["answer"]}
[{{datetime.now()}}]"
            self.__queue_logs.put((log, False))
            self.send(answer)

```

```

def update_password(self, user_id, request):
    password_user = request["password"]
    description_user = request["description"]
    encpted_password_user = encrypt_data(self.__dek, password_user)
    encpted_description_user = encrypt_data(self.__dek,
description_user)
    db_rows = self.__dbms.update_password_passwords(
        user_id=user_id, identifier=request["identifier"],
password=encpted_password_user, description=encpted_description_user)
    if db_rows:
        answer = self.message(header="update-respond", answer=True)
    else:
        answer = self.message(header="update-respond", answer=False)
        log = f"Update password: answer: {answer["answer"]}
[{{datetime.now()}}]"
        self.__queue_logs.put((log, False))
        self.send(answer)
def delete_password(self, user_id, request):
    db_rows = self.__dbms.delete_password_passwords(
        user_id=user_id, identifier=request["identifier"])
    if db_rows:
        answer = self.message(header="delete-respond", answer=True)
    else:
        answer = self.message(header="delete-respond", answer=False)
        log = f"Delete password: answer: {answer["answer"]}
[{{datetime.now()}}]"
        self.__queue_logs.put((log, False))
        self.send(answer)
def generate_token(self):
    for i in range(8, 15):
        token = generate_token(self.__dbms.get_length_token())
        result = self.__dbms.search_token_tokens(token)
        result = bool(result[0])
        if not result:
            return token
        self.__dbms.set_length_token(i)
def message(self, header, answer, token="", passwords=""):

```

```

if header == "authentication-respond" and answer:
    return {"header": header, "answer": answer, "token": token}
elif header == "authentication-respond" and not answer:
    return {"header": header, "answer": answer}
elif header == "registration-respond" and answer:
    return {"header": header, "answer": answer, "token": token}
elif header == "registration-respond" and not answer:
    return {"header": header, "answer": answer}
elif header == "add-respond":
    return {"header": header, "answer": answer}
elif header == "get-respond" and answer:
    return {"header": header, "answer": answer, "passwords":
passwords}
elif header == "get-respond" and not answer:
    return {"header": header, "answer": answer}
elif header == "update-respond":
    return {"header": header, "answer": answer}
elif header == "delete-respond":
    return {"header": header, "answer": answer}
def send(self, data):
    data = self.encode(data)
    self.request.sendall(data)
def get(self):
    try:
        while True:
            if "\n" in self.__buffer:
                break
            chunk = self.request.recv(4096).decode()
            if not chunk:
                return None
            self.__buffer += chunk
        message, self.__buffer = self.__buffer.split("\n", 1)
        return message.strip()
    except ConnectionResetError:
        return None
def encode(self, data):
    message = json.dumps(data)

```

```

        message += "\n"
        message = message.encode()
        return message
    def decode(self, data):
        message = json.loads(data)
        return message

```

Код реалізації класу SecureSocket

```

from queue import Queue
from threading import Thread
from argon2 import PasswordHasher
import socketserver
import os
from utils.dbms import DBMS
from utils.find_path_file import resource_path
class SecureSocket(socketserver.ThreadingTCPServer):
    def __init__(self, addr, handler, ssl_context):
        super().__init__(addr, handler)
        self.__logs_running = True
        self.__ssl_context = ssl_context
        self.allow_reuse_address = True
        self.__db = DBMS(resource_path(os.path.join("data",
"database.db")))
        self.__password_hasher = PasswordHasher()
        self.__log_file_path = resource_path(
            os.path.join("data", "logs", "logs.log"))
        self.__error_file_path = resource_path(
            os.path.join("data", "errors", "errors.log"))
        self.__queue_logs = Queue()
        Thread(target=self.logs_worker, daemon=True).start()
    def get_request(self):
        newsocket, addr = super().get_request()
        conn = self.__ssl_context.wrap_socket(
            newsocket,
            server_side=True
        )

```

```

        return conn, addr
def get_db(self):
    return self.__db
def get_password_hasher(self):
    return self.__password_hasher
def get_queue_logs(self):
    return self.__queue_logs
def logs_worker(self):
    while self.__logs_running or not self.__queue_logs.empty():
        item = self.__queue_logs.get()
        if item is None:
            self.__queue_logs.task_done()
            break
        log, is_error = item
        try:
            if is_error:
                with open(self.__error_file_path, "a") as error_file:
                    print(log, file=error_file)
            else:
                with open(self.__log_file_path, "a") as log_file:
                    print(log, file=log_file)
        finally:
            self.__queue_logs.task_done()
def stop(self):
    self.shutdown()
    self.server_close()
    self.__db.stop()
    self.stop_logs()
    self.__queue_logs.join()
def stop_logs(self):
    self.__logs_running = False
    self.__queue_logs.put(None)

```

Код реалізації класу Server

```

from connection.tls_context import TLSContext

```

```

from connection.secure_socket import SecureSocket
from connection.handler import Handler
import ipaddress
import os
from utils.find_path_file import resource_path
from utils.get_value_settings import get_value_settings
class Server:
    def __init__(self):
        server_cert = resource_path(
            os.path.join("data", "secret", "server.crt"))
        server_key = resource_path(
            os.path.join("data", "secret", "server.key"))
        ca_cert = resource_path(os.path.join("data", "secret", "ca.crt"))
        self.__tls = TLSContext(server_cert, server_key, ca_cert)
        self.__ssl_context = self.__tls.build_context()
        self.__server = None
        self.__port = None
        self.__hosts = None
        self.set_port()
        self.set_hosts()
    def create(self):
        self.__server = SecureSocket(
            (self.__hosts, self.__port),
            Handler,
            self.__ssl_context
        )
    def start(self):
        self.__server.serve_forever()
    def stop(self):
        self.__server.stop()
    def get_port(self):
        return self.__port
    def set_port(self):
        try:
            port = get_value_settings("port")
            if port and (0 < port < 65536):
                self.__port = port

```

```

        else:
            print(
                f"[TERMINAL ERROR] Incorrect port setting {port}, port
is set to the default value of 9000")
            self.__port = 9000
    except Exception as e:
        print(f"[TERMINAL ERROR] Set port {type(e).__name__} {e}")
def get_hosts(self):
    return self.__hosts
def set_hosts(self):
    try:
        hosts = get_value_settings("hosts")
        try:
            ipaddress.ip_address(hosts)
            self.__hosts = hosts
        except ValueError:
            print(
                f"[TERMINAL ERROR] Incorrect hosts setting {hosts},
hosts is set to the default value of (0.0.0.0)")
            self.__hosts = "0.0.0.0"
    except Exception as e:
        print(f"[TERMINAL ERROR] Set hosts {type(e).__name__} {e}")
def stop_db(self):
    self.__server.get_db().stop()

```

Код реалізації класу TLSContext

```

import ssl
class TLSContext:
    def __init__(self, server_cert, server_key, ca_cert):
        self.__server_cert = server_cert
        self.__server_key = server_key
        self.__ca_cert = ca_cert
        self.__context = None
    def build_context(self):

```

```

self.__context = ssl.create_default_context(ssl.Purpose.CLIENT_AUTH)
    self.__context.verify_mode = ssl.CERT_REQUIRED
    self.__context.load_verify_locations(cafile=self.__ca_cert)
    self.__context.load_cert_chain(
        certfile=self.__server_cert,
        keyfile=self.__server_key
    )
    self.__context.minimum_version = ssl.TLSVersion.TLSv1_2
    self.__context.options |= ssl.OP_SINGLE_ECDH_USE
    return self.__context

```

Код реалізації класу DBMS

```

from queue import Queue
from threading import Thread
import sqlite3 as sql
from utils.get_value_settings import get_value_settings

class DBMS:
    def __init__(self, data_base):
        self.__data_base = data_base
        self.__length_token = None
        self.__queue = Queue()
        self.__thread = Thread(target=self.database_worker, daemon=False)
        self.__thread.start()

        self.__con = sql.connect(self.__data_base,
check_same_thread=False)
        self.__cur = self.__con.cursor()
        self.set_length_token()
    def create_data_base():
        try:
            with sql.connect("database.db") as con:
                cur = con.cursor()
                cur.execute("PRAGMA foreign_keys = ON")
                cur.execute("""CREATE TABLE IF NOT EXISTS users (
                    user_id INTEGER PRIMARY KEY AUTOINCREMENT,

```

```

        login TEXT NOT NULL UNIQUE,
        password TEXT NOT NULL,
        email TEXT NOT NULL UNIQUE,
        key BLOB NOT NULL,
        salt BLOB NOT NULL
    ) """)

    cur.execute("""CREATE TABLE IF NOT EXISTS passwords (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        user_id INTEGER NOT NULL,
        password TEXT NOT NULL UNIQUE,
        service TEXT NOT NULL,
        description TEXT NOT NULL,
        FOREIGN KEY (user_id)
            REFERENCES users(user_id)
            ON DELETE CASCADE
    ) """)

    cur.execute("""CREATE TABLE IF NOT EXISTS tokens (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        user_id INTEGER NOT NULL,
        token TEXT NOT NULL,
        FOREIGN KEY (user_id)
            REFERENCES users(user_id)
            ON DELETE CASCADE
    ) """)

    return True

except Exception as e:
    print(f"[ERROR] {e}")

def insert_user_users(self, login, password, email, key, salt):
    queue_answer = Queue()
    self.__queue.put((
        """INSERT INTO users (login, password, email, key, salt) VALUES
        (?, ?, ?, ?, ?) RETURNING user_id""", (login, password, email, key, salt),
        queue_answer, 1))
    user_id = queue_answer.get(timeout=5)
    return user_id

def select_user_users(self, login):
    queue_answer = Queue()

```

```

        self.__queue.put((
            """SELECT user_id, password, key, salt FROM users WHERE login
= ?""", (login,), queue_answer, 1))
        result = queue_answer.get(timeout=5)
        return result

    def select_user_tokens(self, token):
        queue_answer = Queue()
        self.__queue.put((
            """SELECT user_id FROM tokens WHERE token = ?""", (token,),
queue_answer, 1))
        user_id = queue_answer.get(timeout=5)
        return user_id

    def select_tokens_tokens(self):
        queue_answer = Queue()
        self.__queue.put((
            """SELECT user_id, token FROM tokens""", (), queue_answer, 2))
        tokens = queue_answer.get(timeout=5)
        return tokens

    def search_token_tokens(self, token):
        queue_answer = Queue()
        self.__queue.put((
            """SELECT EXISTS (SELECT 1 FROM tokens WHERE token = ? )""",
(token,), queue_answer, 1))
        result = queue_answer.get(timeout=5)
        return result

    def search_user_users(self, login, email):
        queue_answer = Queue()
        self.__queue.put((
            """SELECT EXISTS (SELECT 1 FROM users WHERE login = ? OR email
= ?)""", (login, email,), queue_answer, 1))
        result = queue_answer.get(timeout=5)
        return result

    def insert_token_tokens(self, user_id, token):
        queue_answer = Queue()
        self.__queue.put((

```

```

        """INSERT INTO tokens (user_id, token) VALUES (?, ?)""",
        (user_id, token,), queue_answer, 0))
        db_rows = queue_answer.get(timeout=5)
        return db_rows

    def update_token_tokens(self, user_id, token):
        queue_answer = Queue()
        self.__queue.put((
            """UPDATE tokens SET token = ? WHERE user_id = ?""", (token,
            user_id,), queue_answer, 0))
        db_rows = queue_answer.get(timeout=5)
        return db_rows

    def insert_password_passwords(self, user_id, password, service,
        description):
        queue_answer = Queue()
        self.__queue.put((
            """INSERT INTO passwords (user_id, password, service,
            description) VALUES (?, ?, ?, ?)""", (user_id, password, service,
            description,), queue_answer, 0))
        db_rows = queue_answer.get(timeout=5)
        return db_rows

    def select_password_passwords(self, user_id):
        queue_answer = Queue()
        self.__queue.put((
            """SELECT id, password, service, description FROM passwords
            WHERE user_id = ?""", (user_id,), queue_answer, 2))
        user_id = queue_answer.get(timeout=5)
        return user_id

    def update_password_passwords(self, user_id, identifier, password,
        description):
        queue_answer = Queue()
        self.__queue.put((
            """UPDATE passwords SET password = ?, description = ? WHERE
            user_id = ? AND id = ?""", (password, description, user_id, identifier),
            queue_answer, 0))
        db_rows = queue_answer.get(timeout=5)
        return db_rows

    def delete_password_passwords(self, user_id, identifier):

```

```

queue_answer = Queue()
self.__queue.put((
    """DELETE FROM passwords WHERE user_id = ? AND id = ?""",
    (user_id, identifier,), queue_answer, 0))
db_rows = queue_answer.get(timeout=5)
return db_rows
def get_length_token(self):
    return self.__length_token
def set_length_token(self):
    try:
        length_token = get_value_settings("length_token")
        if 4 < length_token < 15:
            self.__length_token = length_token
        else:
            print(
                f"[TERMINAL ERROR] Incorrect length token setting
{length_token}, length token is set to the default value of 8")
            self.__length_token = 8
    except Exception as e:
        print(f"[TERMINAL ERROR] DBMS {type(e).__name__} {e}")
def database_worker(self):
    while True:
        item = self.__queue.get()
        try:
            if item is None:
                break
            sql_request, parameters, queue_answer, type_request = item
            self.__cur.execute(sql_request, parameters)
            if type_request == 0:
                result = self.__cur.rowcount
            elif type_request == 1:
                result = self.__cur.fetchone()
            else:
                result = self.__cur.fetchall()
            self.__con.commit()
            queue_answer.put(result)

```

```

except Exception as e:
    print(f"[ERROR] {e}")
    queue_answer.put(None)
finally:
    self.__queue.task_done()
def stop(self):
    self.__queue.put(None)
    self.__thread.join()
    self.__con.close()

```

Вміст файлу “crypto.py”

```

import os
from cryptography.hazmat.primitives.ciphers.aead import AESGCM
from cryptography.hazmat.primitives.kdf.scrypt import Scrypt
import base64
MAGIC = b"QENC"
VERSION = b"\x01"
CHUNK_SIZE = 64 * 1024
def generate_salt() -> bytes:
    return os.urandom(16)
def wrap_dek(dek: bytes, kek: bytes) -> bytes:
    aes = AESGCM(kek)
    nonce = os.urandom(12)
    return nonce + aes.encrypt(nonce, dek, None)
def unwrap_dek(wrapped: bytes, kek: bytes) -> bytes:
    nonce = wrapped[:12]
    ciphertext = wrapped[12:]
    aes = AESGCM(kek)
    return aes.decrypt(nonce, ciphertext, None)
def derive_master_key(password: str | bytes, salt: bytes) -> bytes:
    if isinstance(password, str):
        password = password.encode("utf-8")
    kdf = Scrypt(
        salt=salt,
        length=32,

```

```

        n=2**14,
        r=8,
        p=1,
    )
    return kdf.derive(password)
def generate_file_key() -> bytes:
    return os.urandom(32)
def encrypt_file(path: str, password: str):
    try:
        salt = os.urandom(16)
        master_key = derive_master_key(password, salt)
        file_key = generate_file_key()
        aes_master = AESGCM(master_key)
        wrapped_key = aes_master.encrypt(b"0" * 12, file_key, None)
        tmp_path = path + ".enc.tmp"
        with open(path, "rb") as fin, open(tmp_path, "wb") as fout:
            fout.write(MAGIC)
            fout.write(VERSION)
            fout.write(salt)
            fout.write(wrapped_key)
            aes_file = AESGCM(file_key)
            while True:
                chunk = fin.read(CHUNK_SIZE)
                if not chunk:
                    break
                nonce = os.urandom(12)
                encrypted = aes_file.encrypt(nonce, chunk, None)
                fout.write(nonce)
                fout.write(encrypted)
            os.replace(tmp_path, path + ".enc")
            os.remove(path)
        return True
    except Exception as e:
        return False
def decrypt_file(path: str, password: str):
    try:
        tmp_path = path + ".tmp"

```

```

with open(path, "rb") as f:
    magic = f.read(4)
    if magic != MAGIC:
        raise ValueError("Invalid file")
    version = f.read(1)
    if version != VERSION:
        raise ValueError("Unsupported version")
    salt = f.read(16)
    wrapped_key = f.read(32 + 16)
    master_key = derive_master_key(password, salt)
    aes_master = AESGCM(master_key)
    file_key = aes_master.decrypt(b"0" * 12, wrapped_key, None)
    aes_file = AESGCM(file_key)
    with open(tmp_path, "wb") as fout:
        while True:
            nonce = f.read(12)
            if not nonce:
                break
            chunk = f.read(CHUNK_SIZE + 16)
            if not chunk:
                break
            data = aes_file.decrypt(nonce, chunk, None)
            fout.write(data)
    os.replace(tmp_path, path.replace(".enc", ""))
    os.remove(path)
    return True
except Exception as e:
    return False

def encrypt_folder(folder: str, password: str):
    for root, _, files in os.walk(folder):
        for name in files:
            path = os.path.join(root, name)
            if not path.endswith(".enc"):
                if not encrypt_file(path, password):
                    raise Exception("Не вдалось зашифрувати дані сервера")

def decrypt_folder(folder: str, password: str):

```

```
for root, _, files in os.walk(folder):
    for name in files:
        path = os.path.join(root, name)
        if path.endswith(".enc"):
            if not decrypt_file(path, password):
                raise Exception("Неправильний пароль розшифрування")
def encrypt_data(dek: bytes, data: str) -> str:
    aes = AESGCM(dek)
    nonce = os.urandom(12)
    ciphertext = aes.encrypt(nonce, data.encode(), None)
    return base64.b64encode(nonce + ciphertext).decode()
def decrypt_data(dek: bytes, data: str) -> str:
    aes = AESGCM(dek)
    raw = base64.b64decode(data.encode())
    nonce = raw[:12]
    ciphertext = raw[12:]
    return aes.decrypt(nonce, ciphertext, None).decode()
```

БІБЛІОГРАФІЧНА ДОВІДКА

Тема бакалаврської роботи: *Розробка локального додатку-менеджера паролів засобами Python*

Обсяг пояснювальної записки 67 аркушів:

6 таблиць;

42 рисунки;

3 додатки.

Дата завершення роботи: *11 червня 2026р.*

Підпис студента- _____*Нейлюк М.Р.*