МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 123 «Комп'ютерна інженерія»

Освітньо-професійна програма: «Комп'ютерна інженерія»

Група: 2БКС-29

КВАЛІФІКАЦІЙНА РОБОТА

здобувача освіти денної форми навчання

БКС.29.13.000.КРБ

КОДРЯНУ
ПАВЛА МИХАЙЛОВИЧА

м. Одеса
2025 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: **123 «Комп'ютерна інженерія»**

Освітньо-професійна програма: **«Комп'ютерна інженерія»**

Група: **2БКС-29**

ПОЯСНЮВАЛЬНА ЗАПИСКА

До кваліфікаційної роботи бакалавра на тему: **Аналіз методів генерації та керування паролями з використанням алгоритму AES**

Проектний матеріал складається з пояснювальної записки на 85 сторінках та графічного (презентаційного) матеріалу на 12 аркушах (слайдах)

Виконавець _____ (Кодряну П.М.)

Керівник проекту _____ (Іванова Л.В.)

Консультанти:

з розділу охорони праці та техніки безпеки _____ (Чорновол Н.І.)

з нормоконтролю _____ (Петрашова В.І.)

старший консультант _____ (Кривченко Ю.В.)

До захисту допущений

Завідувач кафедри _____ (Іванова Л.В.)

Завідувач відділення _____ (Краснокутська К.Г.)

Захист «25» 06 2025 р. Протокол ЕК № 1

Оцінка ЕК 5 (відмінно) / 98

Секретар ЕК _____

АНОТАЦІЯ

У кваліфікаційній роботі розглядається питання розробки локального програмного застосунку для безпечного керування паролями та застосовується підхід, що забезпечує повний контроль користувача над власними даними. Основна мета роботи – створити надійний менеджер паролів, що використовує симетричне шифрування за стандартом AES-256 для захисту чутливої інформації від несанкціонованого доступу.

У рамках проекту проведено ґрунтовний аналіз існуючих комерційних рішень для керування паролями та обґрунтовано вибір технологічного стеку. Особливу увагу приділено реалізації криптографічного ядра на базі бібліотеки cryptography, що забезпечує шифрування всієї бази даних. Для автентифікації користувача застосовано механізм виведення ключа шифрування з майстер-пароля за допомогою хеш-функції SHA-256, що унеможливлює зберігання ключа у відкритому вигляді.

Робота передбачає розробку програмного застосунку на мові Python із використанням фреймворку PyQt6 для створення інтуїтивного графічного інтерфейсу. Застосунок забезпечує повний цикл керування записами, включає генератор криптографічно стійких паролів на основі модуля secrets, а також реалізує інтелектуальний пошук з допущеннями. Динамічні налаштування мови та теми інтерфейсу дозволяють адаптувати програму до потреб користувача.

Запропонований підхід, що ґрунтується на поєднанні надійних криптографічних алгоритмів, локального зберігання даних та прозорості відкритого коду, має сприяти підвищенню рівня особистої кібербезпеки користувачів. Створений інструмент надає ефективне рішення для захисту цифрової ідентичності та запобігання несанкціонованому доступу до приватних облікових записів. Розроблений інструмент вирішує поширену проблему повторного використання слабких паролів, автоматизуючи створення та зберігання унікальних комбінацій для кожного сервісу. Все це робить програмний продукт не просто зручним, а й важливим елементом для забезпечення комплексного захисту в сучасному цифровому середовищі.

ANNOTATION

This qualification thesis examines the development of a local software application for secure password management and applies an approach that ensures the user's complete control over their own data. The main goal of the work is to create a reliable password manager that utilizes symmetric encryption based on the AES-256 standard to protect sensitive information from unauthorized access.

Within the project, a thorough analysis of existing commercial password management solutions was conducted, and the choice of the technology stack was justified. Special attention was paid to the implementation of the cryptographic core using the cryptography library, which ensures the encryption of the entire database. For user authentication, a mechanism for deriving the encryption key from a master password using the SHA-256 hash function is employed, which prevents storing the key in plaintext.

The work involves the development of a software application in Python using the PyQt6 framework to create an intuitive graphical user interface. The application provides a full record management cycle, includes a generator for cryptographically strong passwords based on the secrets module, and implements fuzzy search functionality. Dynamic language and theme settings allow for adapting the program to the user's needs.

The proposed approach, based on a combination of robust cryptographic algorithms, local data storage, and the transparency of open source, is intended to enhance users' personal cybersecurity. The developed tool provides an effective solution for protecting digital identity and preventing unauthorized access to private accounts. The developed tool addresses the common problem of reusing weak passwords by automating the creation and storage of unique combinations for each service. All of this makes the software product not just a convenience, but also an important element for ensuring comprehensive protection in the modern digital environment.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Відділення *Комп'ютерних систем* Кафедра *Комп'ютерної інженерії*

Спеціальність *123 «Комп'ютерна інженерія»*

Освітньо-професійна програма *«Комп'ютерна інженерія»*

ЗАТВЕРДЖУЮ:

Заст. дир.з НВР *Беркань І.В.*

"28" 05 2025 р.

ЗАВДАННЯ

на кваліфікаційну роботу бакалавра

здобувачеві освіти *Кодряну Павлу Михайловичу*

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *Аналіз методів генерації та керування паролями з використанням алгоритму AES*

затверджена наказом по коледжу від "14" 11 20 р. № 246

2. Термін здачі студентом кваліфікаційної роботи *15.06.2025*

3. Вихідні дані до роботи *Вимоги до інформаційних систем; Функціональні вимоги до програмного забезпечення; Нефункціональні вимоги до програмного забезпечення; Вимоги до дизайну та інтерфейсу користувача;*

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)

Аналітичний огляд існуючих рішень; Аналіз алгоритму AES; Технічне завдання до проєкту; Проєктування та розробка програми; Тестування функціоналу додатку; Охорона праці та техніка безпеки; Висновки; Перелік використаних інформаційних джерел; Додатки;

5. Перелік графічного матеріалу (слайдів мультимедійної презентації) *Типи менеджерів паролів; Алгоритм шифрування AES-256; Архітектура додатку; Інструменти для розробки; Алгоритм взаємодії модулів програми; Функція генерації паролів; Схема класу VaultManager; Виведення ключа з майстер-паролю; Алгоритм взаємодії користувача з додатком; Аналіз ефективності шифрування; Демонстрація застосунку;*

6. Консультанти по кваліфікаційній роботі, із зазначенням розділів, що їх стосуються

Розділ	Консультант	ПІДПИС	
		Завдання видав	Завдання прийняв
Основний розділ	*Іванова Л.В.*		
Розділ охорони праці	*Чорновол Н.І.*		
Нормоконтроль	*Петрашова В.І.*		
Старший консультант	*Кривченко Ю.В.*		

7. Дата видачі завдання _____

Керівник роботи *Іванова Л.В.* _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

Пор. №	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1.	*Аналіз технічного завдання*	*4.05.2025*	виконав
2.	*Огляд існуючих рішень*	*8.05.2025*	виконав
3.	*Аналіз алгоритму AES*	*10.05.2025*	виконав
4.	*Формування технічного завдання до проєкту*	*12.05.2025*	виконав
5.	*Вибір бібліотек для програми*	*15.05.2025*	виконав
6.	*Розробка архітектури програми*	*18.05.2025*	виконав
7.	*Розробка класу менеджера сховища*	*22.05.2025*	виконав
8.	*Розробка логіки шифрування даних*	*24.05.2025*	виконав
9.	*Розробка логіки хешування даних*	*26.05.2025*	виконав
10.	*Розробка елементів інтерфейсу*	*28.05.2025*	виконав
11.	*Перевірка на вимоги користувача*	*31.05.2025*	виконав
12.	*Виконання розділу охорони праці*	*2.06.2025*	виконав
13.	*Перевірка роботи на плагіат*	*6.06.2025*	виконав
14.	*Підготовка до малого захисту*	*10.06.2025*	виконав
15.	*Розробка мультимедійної презентації*	*12.062025*	виконав
16.	*Підготовка до основного захисту*	*15.06.2025*	виконав
17.	*Оформлення мультимедійної презентації*	*18.06.2025*	виконав

Здобувач освіти _____
(підпис)

Керівник роботи _____
(підпис)

Формат	Зона	Поз.	Позначення	Назва	Кіл.	Примітка
				Документація		
			БКС 29. 13 000. 00 КРБ	Випускна робота		
А4			БКС 29. 13 000. 00 КРБ ПЗ	Пояснювальна записка	1	

Зм.	Арк.	№ докум.	Підпис	Дата	БКС 29. 13 000. 00 КРБ		
Розробив		Кодряну П.М.		20.06	Аналіз методів генерації та керування паролями з використанням AES	Літ.	Аркуш 4 / Аркушів 85
Перевірив		Іванова Л.В.		20.06		Н Д П	
Н. Контр.		Петрашова В.І.		20.06.25		ВСП "ОТФК ОНТУ" гр.2БКС-29	
Затверд.		Кривченко Ю.В.		23.06			

ЗМІСТ

Вступ.....9
1 Основний розділ.....11
1.1 Аналітичний огляд існуючих рішень.....11
1.2 Аналіз алгоритму AES.....16
1.3 Технічне завдання до проєкту.....20
1.3.1 Структура проєкту.....20
1.3.2 Вибір інструментів для розробки.....21
1.4 Проєктування та розробка програми.....30
1.5 Тестування функціоналу додатку.....59
2 Розділ охорони праці та техніки безпеки.....68
2.1 Аналіз небезпечних і шкідливих факторів, що впливають на програміста під час розробки системи….....68
2.2 Виробниче приміщення…….....68
2.3 Вимоги до робочого середовища під час роботи за ПК.....69
2.4 Організація робочого місця з ПК.....69
2.4.1 Освітлення робочого місця.....70
2.4.2 Мікроклімат.....70
2.4.3 Шум.....70
2.4.4 Електробезпека.....71
2.5 Пожежна безпека при роботі з ВДТ.....72
Висновки.....73
Перелік використаних інформаційних джерел.....74
Додаток А. Фрагмент коду модуля vault.py на мові Python додатку для керування сховищем паролів.....75
Додаток Б. Слайди мультимедійної презентації.....80

ВСТУП

Швидкий розвиток інформаційних технологій та стихійне зростання обсягів оброблюваних даних супроводжуються зростаючими загрозами несанкціонованого доступу та кіберзлочинності. Одним із ключових бар'єрів захисту інформаційних ресурсів є надійна автентифікація користувачів, основу якої традиційно становлять паролі.

Водночас з підвищенням вимог до складності та унікальності паролів користувачі все частіше стикаються з проблемою їх генерації, зберігання й керування. У результаті багато хто використовує прості та легко вгадуванні поєднання символів, що значно знижує загальний рівень безпеки систем.

Алгоритм AES (Advanced Encryption Standard) сьогодні є одним із найпоширеніших симетричних шифрів, що гарантує високу стійкість до криптоаналізу та ефективну апаратну реалізацію. Використання AES-шифрування у менеджерах паролів забезпечує надійну конвертацію секретної інформації в безпечні блоки даних із подальшим розшифруванням лише за наявності коректного ключа. Завдяки низькому накладному навантаженню та широкій підтримці на рівні операційних систем і програмних бібліотек, AES-базовані рішення стали стандартом де-факто для захисту конфіденційних даних.

В Україні, незважаючи на активне впровадження цифрових технологій, питання ефективного генерування та зберігання паролів досі потребує додаткових досліджень та адаптації готових рішень до вітчизняних реалій. Більшість існуючих менеджерів паролів орієнтовані на міжнародний ринок і не враховують специфіку локальних IT-інфраструктур та нормативно-правового поля. Це створює необхідність розробки оптимізованого інструменту, який поєднає перевірені криптографічні методи (зокрема AES) із зручним інтерфейсом і локалізованими налаштуваннями.

Метою даної кваліфікаційної роботи є аналіз існуючих методів

генерації й керування паролями з застосуванням алгоритму AES та створення програмного застосунку — менеджера паролів.

В основному розділі проведено огляд та порівняння сучасних алгоритмів генерації випадкових та легко запам’ятовуваних паролів. Дослідження принципів роботи та безпеки AES у контексті шифрування сховищ паролів. Вибір і обґрунтування архітектури програми менеджера паролів із криптографічним ядром на основі AES. Розробка та реалізація функціональності генерації, зберігання й захищеного доступу до паролів.

Структурно робота складається з основного розділу, присвяченого огляду алгоритмів генерації паролів і принципів AES-шифрування, проєктуванню та реалізацію програми; Розділу охорони праці. Таким чином, результатом дослідження стане повнофункціональний менеджер паролів із надійним криптографічним ядром на основі AES, який може бути використаний як в особистих, так і в корпоративних інформаційних системах для підвищення рівня захищеності облікових даних.

1 ОСНОВНИЙ РОЗДІЛ

1.1 Огляд існуючих рішень

Управління паролями є критично важливим аспектом інформаційної безпеки як для окремих користувачів, так і для організацій. Як зазначалося раніше, проблема генерації, зберігання та використання надійних паролів стоїть досить гостро. Для вирішення цього завдання існує цілий клас програмного забезпечення — менеджери паролів.

Менеджери паролів допомагають генерувати, зберігати та вводити складні паролі із зашифрованої бази даних. Вони дозволяють користувачам запам'ятовувати лише один майстер-пароль для доступу до всього сховища.

Менеджери паролів можна класифікувати за кількома ознаками. Однією з ключових є спосіб зберігання даних. За цією ознакою їх поділяють на:

- Хмарні (онлайн) менеджери паролів: Дані (зашифрована база паролів) зберігаються на серверах постачальника послуг. Доступ до них можливий з будь-якого пристрою через інтернет, зазвичай через веб-інтерфейс або спеціальні застосунки.

- Локальні (офлайн) менеджери паролів: База даних паролів зберігається безпосередньо на пристрої користувача (комп'ютері, смартфоні, USB-накопичувачі). Доступ до даних зазвичай не потребує підключення до інтернету.

- Гібридні менеджери паролів: Поєднують риси локальних та хмарних. Наприклад, база даних може зберігатися локально, але синхронізуватися через хмарний сервіс (який може належати як розробнику менеджера, так і сторонньому провайдеру).

- Апаратні менеджери паролів: Дані зберігаються на спеціалізованому фізичному пристрої.

Іншими важливими критеріями класифікації є тип доступу (браузерні розширення, десктопні програми, мобільні застосунки) та модель ліцензування (комерційні продукти з платною підпискою або одноразовою покупкою, безкоштовні програми та продукти з відкритим вихідним кодом). Більшість сучасних менеджерів паролів, незалежно від типу, використовують надійні

алгоритми шифрування для захисту збережених даних. Стандартом де-факто для шифрування в таких системах є AES (Advanced Encryption Standard), переважно з довжиною ключа 256 біт (AES-256).

Було розглянуто реальні рішення, до популярних хмарних менеджерів паролів можна віднести: 1Password, LastPass, BitWarden.

1Password – преміальний хмарний менеджер паролів, можна побачити на рис. 1.1. Відомий своїм зручним інтерфейсом та набором функцій безпеки. Орієнтований як на індивідуальних користувачів, так і на сімейне та бізнес-використання. Використовує AES-256 для шифрування.

Рисунок 1.1. Зображення логотипу 1Password

До переваг можна віднести: зручний та інтуїтивно зрозумілий інтерфейс, високий рівень безпеки та надійності шифрування, широка функціональність, підтримка різних платформ, можливість створення кількох сховищ.

Недоліки: відсутня повністю безкоштовна версія (тільки пробний період), закритий вихідний код, може бути дорожчим порівняно з деякими конкурентами.

1Password є преміальним хмарним рішенням, що вирізняється високою зручністю, відмінним дизайном та потужними функціями безпеки. Хоча це платний продукт із закритим кодом, він пропонує один з найкращих користувацьких досвідів та надійний захист для тих, хто готовий інвестувати у свою цифрову безпеку та зручність.

LastPass – один з найвідоміших хмарних менеджерів паролів, який пропонує як безкоштовний, так і платний плани з розширеними функціями, логотип

застосунку можна побачити на рис. 1.2. Використовує AES-256 шифрування.

Рисунок 1.2. Логотип LastPass

Переваги**:** наявність безкоштовної версії, зручність у використанні та мультиплатформність, автоматичне заповнення форм і генерація надійних паролів, можливість синхронізації між необмеженою кількістю пристроїв одного типу у безкоштовній версії.

Недоліки**:** наявність інцидентів з безпекою в минулому, що підірвало довіру деяких користувачів, деякі розширені функції доступні лише в платних версіях, дещо застарілий інтерфейс.

LastPass залишається популярним хмарним менеджером завдяки своїй доступності, зокрема наявності безкоштовного плану, та широкій функціональності. Незважаючи на минулі інциденти з безпекою, він залишається зручним рішенням для багатьох користувачів, хоча питання довіри може бути вирішальним для певної їх частини.

BitWarden – менеджер паролів з відкритим вихідним кодом, логотип на рис. 1.3, який пропонує як хмарний сервіс, так і можливість самостійного розміщення. Використовує AES-256.

Рисунок 1.3. Логотип BitWarden

Переваги**:** відкритий вихідний код, безкоштовна версія з основними функціями, високий рівень безпеки, підтримка широкого спектра платформ, можливість самостійного розміщення.

Недоліки**:** деякі розширені функції доступні лише в платній версії, інтерфейс користувача може здатися менш продуманим.

Bitwarden є чудовим вибором для тих, хто шукає баланс між функціональністю, безпекою та вартістю, особливо цінуючи прозорість відкритого коду. Його безкоштовний план та можливість самостійного розміщення роблять його універсальним рішенням як для індивідуальних користувачів, так і для технічно підкованих ентузіастів.

З локальних менеджерів паролів можна виділити: KeePass, Enpass.

KeePass – безкоштовний локальний менеджер паролів з відкритим вихідним кодом, логотип на рис. 1.4. Дані зберігаються в зашифрованому файлі локально на пристрої користувача.

Рисунок 1.4. Логотип KeePass

Переваги**:** повністю безкоштовний та з відкритим вихідним кодом, високий рівень безпеки завдяки локальному зберіганню, повний контроль користувача над своїми даними, велика кількість плагінів для розширення функціональності, не потребує підключення до Інтернету.

Недоліки**:** інтерфейс може виглядати застарілим, та менш інтуїтивним для новачків, може бути складнішим у налаштуванні та використанні для менш технічно обізнаних користувачів, офіційно доступний переважно для Windows.

KeePass є класичним прикладом надійного локального менеджера паролів з відкритим кодом, що надає користувачеві повний контроль над своїми даними та максимальну безпеку за умови правильного використання. Хоча його інтерфейс

може здатися застарілим, а синхронізація потребує ручного налаштування, він залишається безкоштовним та потужним інструментом для тих, хто пріоритезує контроль та автономність.

Enpass - менеджер паролів, логотип можна побачити на рис. 1.5, який пропонує локальне зберігання даних за замовчуванням, але з можливістю синхронізації через власні хмарні облікові записи користувача (наприклад, iCloud, Dropbox, Google Drive, OneDrive), а не через сервери розробника.

Рисунок 1.5. Логотип Enpass

Переваги**:** користувач контролює процес синхронізації, підтримка різних платформ, зберігає не тільки паролі а й іншу конфіденційну інформацію.

Недоліки**:** безкоштовна версія для мобільних пристроїв може мати обмеження, інтерфейс може бути дещо складним, хоча дані шифруються локально, безпека синхронізації залежить від надійності обраного користувачем хмарного сервісу.

Enpass пропонує гнучкий підхід, поєднуючи переваги локального зберігання з можливістю синхронізації через обрані користувачем хмарні сервіси. Це робить його привабливим для тих, хто хоче контролювати, де зберігаються їхні зашифровані дані, уникаючи при цьому серверів самого розробника, і готовий розглянути як підписку, так і одноразову покупку.

На основі огляду існуючих рішень, та підходів до реалізації менеджеру паролів, можна виділити наступні варіанти: хмарні, локальні. Для розробки додатку було обрано підхід, що передбачає локальне зберігання даних та відкритий вихідний код. Рішення з такою архітектурою мають більшу довіру, оскільки відкритий вихідний код дозволяє будь-кому перевірити, що саме програма робить з даними користувача. Крім того, локальне зберігання надає

користувачеві повний контроль над власною інформацією, це дозволяє усунути ризики, пов’язані з провайдером послуг.

Проте, незважаючи на очевидні переваги, такий підхід має і свій зворотний бік. Коли користувач отримує повний контроль, на нього автоматично перекладається і вся повнота відповідальності за збереження та безпеку своїх даних. Це, безперечно, створює вищий поріг входу для новачків або менш технічно обізнаних людей, яким може бути складно розібратися в усіх нюансах. Однак важливо розуміти, що саме такий тотальний контроль і є запорукою максимальної безпеки, адже він дозволяє повністю виключити вразливості, пов'язані з третіми сторонами, такі як злами серверів чи витоки даних у посередників.

1.2 Аналіз алгоритму AES

У сучасному цифровому світі, де захист інформації має першочергове значення, надійні алгоритми шифрування відіграють ключову роль. Одним з найпоширеніших та найдовіреніших симетричних алгоритмів шифрування є Advanced Encryption Standard (AES). Його використання у менеджерах паролів забезпечує фундаментальний рівень безпеки для збережених облікових даних. Даний розділ присвячений огляду принципів роботи AES, аналізу його безпеки залежно від довжини ключа та обґрунтуванню його застосування для захисту сховищ паролів.

Advanced Encryption Standard (AES) – це симетричний блочний шифр, офіційно прийнятий урядом США для захисту електронних даних. Стандарт був опублікований Національним інститутом стандартів і технологій (NIST) у 2001 році під назвою FIPS PUB 197 після відкритого конкурсу, в якому переміг алгоритм Rijndael, розроблений бельгійськими криптографами Вінсентом Рейменом та Йоаном Дайменом. AES прийшов на зміну попередньому стандарту Data Encryption Standard (DES), який став вразливим до атак через недостатню довжину ключа.

Як симетричний шифр, AES використовує однаковий ключ як для

шифрування, так і для дешифрування даних. Це означає, що безпека всієї системи залежить від збереження цього ключа в таємниці. AES є блочним шифром, тобто він оперує даними у вигляді блоків фіксованого розміру.

Ключовими характеристиками алгоритму AES, що визначають його роботу та рівень безпеки, є:

Розмір блоку; AES завжди працює з блоками даних розміром 128 біт, незалежно від довжини ключа.

Довжина ключа; Стандарт AES підтримує три можливі довжини ключів:

- AES-128. 128-бітний ключ.
- AES-192. 192-бітний ключ.
- AES-256. 256-бітний ключ.

Кількість раундів; Процес шифрування в AES складається з певної кількості раундів (циклів) перетворень. Кількість раундів залежить від довжини ключа:

- 10 раундів для 128-бітних ключів.
- 12 раундів для 192-бітних ключів.
- 14 раундів для 256-бітних ключів.

В основі алгоритму AES лежить багатораундова структура, де для ключа довжиною 256 біт виконується 14 таких раундів. Мета кожного раунду — послідовно і незворотно ускладнити зв'язок між відкритим текстом та шифротекстом, реалізуючи ключові принципи стійкої криптографії, сформульовані Клодом Шенноном: конфузію (заплутування) та дифузію (розсіювання). Це досягається за допомогою чотирьох різних математичних перетворень: SubBytes (нелінійна заміна байтів для забезпечення конфузії), ShiftRows (асиметричний зсув рядків), MixColumns (перемішування даних у стовпцях для досягнення дифузії) та AddRoundKey (побітове додавання раундового ключа). Раундові ключі, які використовуються на етапі AddRoundKey, генеруються з нього за допомогою окремого алгоритму розширення, що забезпечує унікальність ключового матеріалу.

Візуально цей ітеративний процес, що складає основу стійкості алгоритму, детально зображений на рисунку 1.6

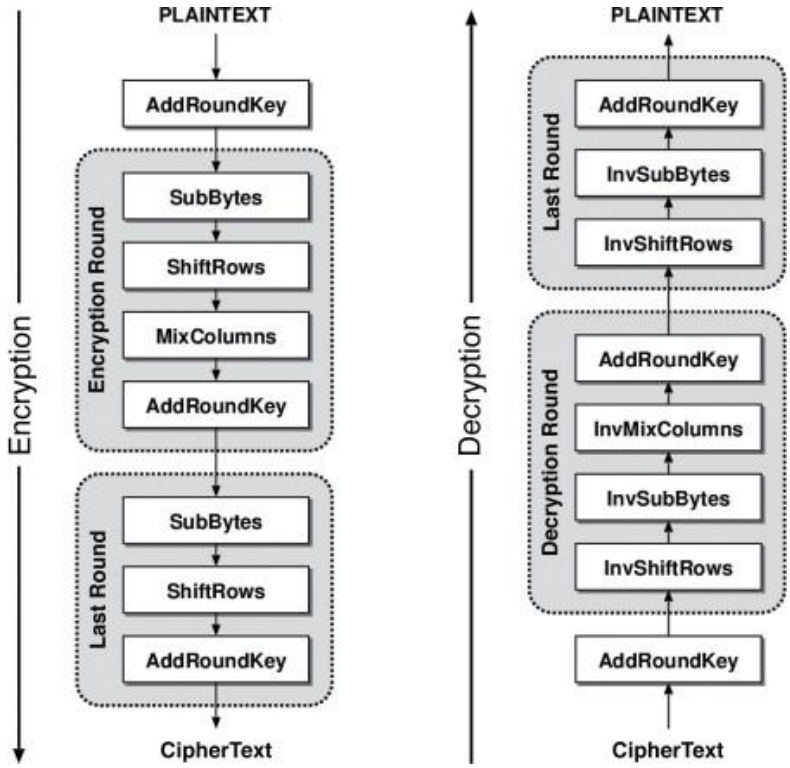


Рисунок 1.6. Кроки перетворень для 14-раундового ключа

Безпека AES проти атаки повним перебором (brute-force attack), тобто спроби підібрати ключ шляхом перебору всіх можливих комбінацій, експоненційно зростає з довжиною ключа, це можна побачити у таблиці табл. 1.1.

Таблиця 1.1. Таблиця порівняння ефективності AES.

Характеристика	*AES-128*	*AES-192*	*AES-256*
Довжина ключа (біт)	128	192	256
Кількість раундів	10	12	14
Кількість можливих ключів	$2^{128} = 3.4 * 10^{38}$	$2^{192} = 6.2 * 10^{57}$	$2^{256} = 1.1 * 10^{77}$
Час для повного перебору (для суперкомп'ютера)	Декілька мільярдів років	Більше, віку Всесвіту в квадрильйонах разів	За межами уявного часу

Більш наглядно, число можливих комбінацій для AES-256, виглядає так.

115,792,089,237,316,195,423,570,985,008,687,907,853,269,984,665,640,564,039,457,584,007,913,129,639,936.

Це 115 квінтильйонів трильйонів трильйонів трильйонів трильйонів варіантів – кількість варіантів настільки астрономічно велика, що немає жодного реалістичного способу її подолати повним перебором.

Важливо зазначити, що окрім атаки повного перебору, існують і інші криптоаналітичні методи. Однак, за понад 20 років використання, проти повнораундового AES не було знайдено практично реалізованих атак, які б суттєво скорочували час зламу порівняно з повним перебором, за умови коректної реалізації алгоритму. Атаки по бічних каналах (наприклад, аналіз часу виконання або енергоспоживання) та атаки на пов'язаних ключах є радше проблемами конкретних програмних чи апаратних реалізацій, аніж вразливостями самого математичного апарату AES.

Хоча довші ключі забезпечують вищий теоретичний рівень безпеки, вони також впливають на продуктивність через більшу кількість раундів обробки.

- AES-128: ~100% швидкість.
- AES-192: ~85-90% від швидкості AES-128.
- AES-256: ~70-75% від швидкості AES-128.

Для такого специфічного програмного рішення, як менеджер паролів, де операції шифрування та дешифрування стосуються відносно невеликих обсягів даних (зазвичай, це текстові рядки), будь-яке, навіть теоретично можливе, зниження швидкості при використанні найнадійнішого алгоритму AES-256 є цілком непомітним для сучасних систем. Навіть на мобільних пристроях затримки вимірюються мілісекундами, залишаючись далеко за межами людського сприйняття.

Безумовно, при виборі довжини ключа AES для будь-якого застосунку необхідно знаходити раціональний баланс між бажаним рівнем безпеки та можливою втратою продуктивності. Однак у випадку з менеджером паролів, де на кону стоїть доступ до всіх цифрових активів користувача, пріоритет завжди однозначно надається безпеці.

В результаті, остаточний вибір був зроблений на користь AES-256, оскільки цей стандарт пропонує найвищий рівень криптографічного захисту, доступний на

сьогодні. У контексті зберігання настільки чутливої інформації, використання AES-256 є не просто виправданим, а фактично єдиним рекомендованим підходом. Це забезпечує не лише максимальний запас міцності на десятиліття вперед, але й дарує користувачеві надзвичайно важливий психологічний комфорт, що ґрунтується на усвідомленні практичної незламності обраного методу шифрування.

1.3 Технічне завдання до проєкту

За результатами аналітичного дослідження було визначено, що розроблювальний менеджер паролів буде локальним, з відкритим вихідним кодом. В якості алгоритму шифрування було обрано AES-256, що використовує ключ довжиною 256 біт. На основі цих вимог можна детальніше визначити структуру програми.

1.3.1 Структура проєкту

Для початку було визначено основні функції які має виконувати програма:

- Створення захищеного сховища даних, для зберігання даних користувача (паролі, логіни, назви ресурсів тощо).
- Додавання, редагування, видалення записів у сховищі.
- Можливість імпорту існуючого файлу сховища даних.
- Шифрування даних алгоритмом AES-256.
- Використання файлу-ключа або майстер-паролю для генерації ключа шифрування.
- Вбудований генератор унікальних та надійних паролів, з можливістю налаштування довжини та набору символів.
- Реалізація нечіткого пошуку (fuzzy search), що дозволяє знаходити записи без необхідності вводити точний запит.
- Сучасний та інтуїтивний графічний інтерфейс користувача, для взаємодії з усіма функціями застосунку.
- Гнучкі та динамічні налаштування мови та теми, через графічний інтерфейс користувача.

На основі визначених функцій можна перейти до опису архітектури. Визначимо всю ключові системи:

- Інтерфейс користувача.
- Система керування сховищем.
- Система шифрування / дешифрування даних.
- Система генерації випадкових паролів.
- Система пошуку з допущеннями.

Отриману структуру проєкту та взаємодію між різними системами можна представити у вигляді діаграми на рис. 1.7.

Рисунок 1.7. UML-діаграма архітектури додатку

1.3.2 Вибір інструментів для розробки

Застосунок буде розроблено для операційної системи Windows 11 з використанням мови програмування Python.

Windows 11 — найновіша пропрієтарна версія лінійки операційних систем Windows NT компанії Microsoft, яка випущена 24 червня 2021 року.

Python - це високорівнева мова програмування загального призначення з динамічною типізацією та автоматичним керуванням пам'яттю, орієнтована на підвищення продуктивності розробника, читабельності коду та його якості, а також на забезпечення переносимості написаних нею програм.

Для того щоб обрати необхідні інструменти розробки, а саме бібліотеки Python, було розглянуто наступні варіанти:

Графічний інтерфейс користувача. Система інтерфейсу користувача є критично важливою для забезпечення зручної взаємодії користувача з усіма функціями додатку. Інтерфейс має бути інтуїтивно зрозумілим, сучасним та

підтримувати всі заплановані функції.

Серед найпопулярніших бібліотек для розробки GUI-застосунків на Python, можна виділити: PyQt6, Kivy та Tkinter.

PyQt6. PyQt6 – це набір Python-прив’язок для кросплатформенного фреймворку Qt 6, розроблений компанією Riverbank Computing. Qt є потужним та зрілим інструментарієм для створення графічних інтерфейсів користувача (GUI) з багатим набором віджетів, можливостями для роботи з графікою, мультимедіа, мережею та базами даних. PyQt6 дозволяє розробникам на Python використовувати всю міць Qt, створюючи нативні на вигляд додатки для різних операційних систем, включаючи Windows. Він підтримує систему сигналів та слотів для обробки подій, дизайнер Qt Designer для візуального проєктування інтерфейсів та має широку документацію.

PyQt6 є потужним інструментом для створення застосунків з графічним інтерфейсом. Він надає велику кількість готових компонентів та інструментів, що значно прискорює розробку складних GUI, хоча може мати дещо вищий поріг входу порівняно з більш простими бібліотеками.

Tkinter; Tkinter – є стандартною бібліотекою GUI для Python, яка постачається разом з інтерпретатором. Вона є обгорткою над інструментарієм Tcl/Tk, що забезпечує її кросплатформеність. Tkinter відносно простий у вивченні та використанні для створення базових графічних інтерфейсів. Бібліотека включає набір основних віджетів, таких як кнопки, мітки, текстові поля, меню тощо. Через свою простоту та доступність "з коробки", Tkinter часто використовується для невеликих утиліт або навчальних проєктів.

Tkinter це легкий та доступний варіант для створення простих і доволі примітивних GUI без необхідності встановлення інших залежностей або бібліотек. Однак, для створення сучасних та складних інтерфейсів його можливостей може бути недостатньо, а зовнішній вигляд додатків може виглядати дещо застарілим без додаткових зусиль по кастомізації.

Kivy. Kivy – це бібліотека Python з відкритим вихідним кодом для швидкої розробки додатків, які використовують інноваційні користувацькі інтерфейси,

такі як мультитач-додатки. Kivy працює на Linux, Windows, OS X, Android, iOS та Raspberry Pi. Він використовує власну мову розмітки Kv-lang для опису інтерфейсу, що дозволяє відокремити логіку від представлення. Kivy відомий своєю гнучкістю у створенні нестандартних, візуально привабливих інтерфейсів та підтримкою апаратного прискорення графіки через OpenGL.

Kivy це доволі потужний інструмент для створення сучасних, анімованих інтерфейсів, особливо якщо планується кросплатформеність, включаючи мобільні пристрої. Однак, його підхід до побудови GUI відрізняється від традиційних фреймворків, що може вимагати додаткового часу на вивчення.

Отже, для реалізації інтерфейсу користувача було обрано бібліотеку PyQt6. Незважаючи на потенційно вищий поріг входження, вона надає найширший набір інструментів для створення сучасного, та візуально привабливого десктопного додатку, що відповідає вимогам до розроблюваного менеджера паролів. Велика спільнота та документація забезпечують належну підтримку та безліч прикладів у мережі, у випадку виникнення питань чи проблем.

Система керування сховищем. Ця система відповідає за створення, читання, запис та модифікацію файлу сховища, у якому будуть зберігатися зашифровані дані користувача. Для зберігання структурованих даних можна використовувати різні формати файлів або бази даних. Серед найпопулярніших інструментів та бібліотек для роботи з даними в середовищі Python можна виділити таки: orjson, pickle, sqlite3.

orjson. orjson – це швидка, коректна бібліотека для серіалізації Python датакласів, datetime, numpy та інших типів у JSON і десеріалізації JSON у Python об'єкти. Вона написана на Rust і значно перевершує стандартну бібліотеку json та інші бібліотеки Python JSON за швидкістю. orjson строго дотримується специфікації JSON, забезпечує кращу обробку помилок та підтримує серіалізацію типів даних, які стандартний json не обробляє без додаткових зусиль (наприклад, datetime, uuid).

orjson є чудовим вибором для роботи з JSON-даними, коли швидкість та коректність є пріоритетом. Його здатність ефективно серіалізувати різноманітні

типи даних Python робить його зручним для зберігання структурованої інформації, такої як записи менеджера паролів, у форматі, який також легко читається людиною (перед шифруванням).

pickle. Модуль pickle реалізує бінарні протоколи для серіалізації та десеріалізації об'єктної структури Python. "Піклінг" - це процес перетворення об'єкта Python в потік байтів, а "анпіклінг" - це зворотна операція. pickle може серіалізувати практично будь-який об'єкт Python, включаючи користувацькі класи та функції, що робить його дуже гнучким. Однак, важливою особливістю pickle є те, що він не є безпечним проти помилкових або зловмисно створених даних. Десеріалізація даних з ненадійного джерела може призвести до виконання довільного коду.

pickle пропонує дуже гнучкий та потужний механізм серіалізації об’єктів Python, зберігаючи їхню структуру та тип. Проте, його використання вимагає особливої обережності через ризики безпеки при десеріалізації даних з неперевірених джерел. Для сховища паролів, яке повністю контролюється додатком та шифрується, цей ризик може бути мінімізований, але перевага читабельності JSON перед шифрування – втрачається.

sqlite3. Модуль sqlite3 надає інтерфейс для роботи з легкою дисковою базою даних SQLite. SQLite – це C-бібліотека, що реалізує невелику, швидку, самодостатню, високонадійну, повнофункціональну SQL-систему управління базами даних. Замість зберігання даних у простому файлі (JSON або pickle), можна використовувати базу даних SQLite, що може надати переваги у структуруванні даних, індексації та виконанні складних запитів, особливо при великій кількості записів.

sqlite3 пропонує більш потужний підхід до зберігання даних, ніж прості файли, особливо якщо очікується велика кількість записів та потрібні можливості реляційних баз даних. Однак, для простого сховища паролів, де основна операція – це завантаження та збереження всього файлу, це може бути надлишковим рішенням, а шифрування всієї бази даних потребуватиме окремих механізмів, що зробить систему складніше ніж вона має бути.

Для реалізації системи керування сховищем було обрано бібліотеку orjson. Формат JSON є достатньо гнучким для зберігання даних менеджера паролів, а також, на відміну від формату pickle, є стандартом обміну даними, що легко читається людиною. Це може бути корисним для відлагодження або аналізу структури даних.

Бібліотека orjson була обрана завдяки її високій продуктивності: вона забезпечує значно вищу швидкість серіалізації та десеріалізації порівняно зі стандартним модулем json, гарантуючи при цьому коректність обробки даних.

Таким чином, сховище буде реалізовано у вигляді єдиного JSON-файлу. Частина цього файлу, що містить дані користувача, буде зашифрована і зможе бути розшифрована лише за наявності відповідного ключа.

Система шифрування та дешифрування даних. Це ядро безпеки менеджера паролів. Система має надійно шифрувати дані сховища за допомогою алгоритму AES-256, використовуючи ключ у вигляді файлу, або похідний від майстер-пароля користувача. З безпечних та популярних варіантів можна виділити: cryptography, PyCryptodome, pynacl.

cryptography. Бібліотека cryptography має на меті надати високорівневі рецепти та низькорівневі інтерфейси до поширених криптографічних алгоритмів, таких як симетричні шифри, хеш-функції та алгоритми з відкритим ключем. Вона розробляється з акцентом на безпеку, простоту використання та уникнення поширених помилок при роботі з криптографією. Для симетричного шифрування вона підтримує AES з різними режимами роботи (наприклад, GCM, CBC). cryptography активно підтримується та рекомендується багатьма експертами з безпеки Python.

cryptography є сучасною та безпечною бібліотекою для криптографічних задач на Python. Вона надає всі необхідні інструменти для реалізації надійного шифрування AES-256. Її активна розробка та фокус на безпеці роблять її кращим вибором для критично важливих завдань.

PyCryptodome. PyCryptodome є форком бібліотеки PyCrypto, яка тривалий час була стандартом де-факто для криптографії в Python, але її розробка

припинилася. PyCryptodome містить як низькорівневі, так і високорівневі криптографічні примітиви, включаючи симетричні шифри (AES, DES, Blowfish), асиметричні шифри (RSA, ECC), хеш-функції (SHA2, SHA3, BLAKE2) та генератори випадкових чисел. Вона прагне бути прямою заміною PyCrypto з додаванням нових алгоритмів та виправленням відомих проблем.

PyCryptoDome є доволі потужною та багатофункціональною бібліотекою. Вона підтримує широкий спектр алгоритмів, включаючи AES, і може бути хорошим вибором. Однак, cryptography часто вважається більш сучасною та безпечною альтернативою завдяки своєму дизайну.

pynacl. PyNaCl є Python-прив'язками до бібліотеки Networking and Cryptography library (NaCl). NaCl – це сучасна, проста у використанні криптографічна бібліотека, розроблена з акцентом на високий рівень безпеки та запобігання поширеним помилкам криптографічного дизайну. PyNaCl надає доступ до таких примітивів, як шифрування з відкритим ключем, симетричне шифрування, хешування та цифровий підпис. Вона орієнтована на надання невеликого набору потужних та безпечних інструментів.

PyNaCl пропонує дуже високий рівень безпеки та простоту використання для певного набору криптографічних завдань, часто використовуючи більш сучасні алгоритми, ніж традиційні. Хоча вона підтримує симетричне шифрування (на якому працює AES), її основний набір алгоритмів можуть дещо відрізнятися від прямої реалізації AES з усіма його стандартними режимами, порівняно з cryptography.

Отже, для системи шифрування та дешифрування даних було обрано бібліотеку cryptography. Вона є галузевим стандартом для криптографічних операцій на Python, активно підтримується, має відмінну документацію та надає безпечні реалізації алгоритму AES-256.

Система генерації випадкових паролів. Ця система має генерувати криптографічно стійкі унікальні паролі. Система повинна дозволяти користувачеві задавати довжину пароля та обирати набір символів для генерації (наприклад, великі та малі літери, цифри, спеціальні символи). Для реалізації

цього функціоналу необхідно обрати модуль що надає інструменти для генерації випадкових чисел. В стандартній бібліотеці Python для цього є наступні модулі: random, secrets, os.urandom.

random. Модуль random у Python реалізує генератори псевдовипадкових чисел для різних розподілів. Він широко використовується для моделювання, ігор та інших завдань, де потрібна випадковість. Однак, розробники Python явно застерігають, що функції з модуля random не слід використовувати для цілей безпеки або криптографії, оскільки генератор псевдовипадкових чисел, який він використовує (Mersenne Twister), не є криптографічно стійким.

Модуль random підходить для багатьох завдань, але категорично не рекомендований для генерації паролів, оскільки він генерує саме псевдовипадкові числа на основні часу систему, що створює ризики, і відкриває можливості для певних атак.

secrets. Модуль secrets був введений у Python 3.6 і призначений для генерації криптографічно стійких випадкових чисел, придатних для управління секретами, такими як паролі, токени безпеки тощо. Він використовує найбільш безпечне джерело випадковості, доступне в операційній системі. Функції з цього модуля, такі як secrets.choice(), secrets.token_bytes(), secrets.token_hex(), secrets.token_urlsafe(), спеціально розроблені для генерації безпечних випадкових послідовностей.

Модуль secrets є ідеальним вибором для генерації паролів та інших криптографічних секретів на Python. Його пряме призначення – це забезпечувати криптографічну стійкість генерованих даних, що є ключовою вимогою для системи генерації паролів.

os.urandom(). Функція urandom(n), з модуля os, повертає рядок з n випадкових байтів, придатних для криптографічного використання. Вона отримує дані з джерела випадковості, наданого операційною системою (наприклад, /dev/urandom у Unix-подібних системах або CryptGenRandom у Windows). Фактично, модуль secrets використовує os.urandom() "під капотом" для отримання сирих випадкових даних.

urandom надає доступ до криптографічно стійких випадкових байтів, але для генерації паролів з певних наборів символів або певної довжини може знадобитися додаткова логіка обробки цих випадкових байтів. Модуль secrets надає зручніші високорівневі функції для цих завдань, які вже використовують urandom.

Для системи генерації паролів було обрано стандартний модуль secrets. Він спеціально розроблений для криптографічних потреб, гарантує використання надійного джерела випадковості та надає зручні функції для генерації безпечних токенів або вибору випадкових елементів із послідовностей. Це робить його його оптимальним вибором для створення складних, унікальних, та криптографічно стійких паролів.

Система нечіткого пошуку; Ця система дозволить користувачеві знаходити записи у сховищі, навіть якщо пошуковий запит містить невеликі помилки або є неточним, система буде аналізувати схожість запиту з наявними даними та відображатиме найбільш релевантні результати. Можна виділити кілька популярних та перевірених бібліотек: thefuzz, RapidFuzz, difflib.

thefuzz. thefuzz – це бібліотека Python, яка використовує відстань Левенштейна для обчислення подібності між рядками. Вона надає прості у використанні функції для визначення відсотка схожості між двома рядками, а також для пошуку найкращих збігів у списку рядків. thefuzz може обробляти різні типи нечітких збігів, такі як проста схожість, схожість часткових рядків, схожість з урахуванням порядку слів тощо.

Для реалізації функції пошуку записів у розробленому менеджері паролів було обрано бібліотеку thefuzz. Її функціональність базується на обчисленні відстані Левенштейна, що дозволяє кількісно оцінити різницю між двома рядками у вигляді "вартості редакції" — мінімальної кількості операцій (вставка, видалення, заміна), необхідних для перетворення одного рядка на інший. Завдяки цьому, додаток може коректно ідентифікувати запис, навіть якщо користувач припустився незначної помилки при введенні, наприклад, "Gmial" замість "Gmail". Таким чином, інтеграція thefuzz дозволить значно покращити

користувацький досвід, не вимагаючи при цьому впровадження складних повнотекстових пошукових систем, що є оптимальним рішенням для даного проєкту.

RapidFuzz. RapidFuzz – це бібліотека для нечіткого зіставлення рядків, написана на C++ з обгортками для Python. RapidFuzz реалізує різні алгоритми обчислення відстані між рядками, включаючи відстань Левенштейна, Геммінга, Джаро-Вінклера та інші.

RapidFuzz є чудовою альтернативою, якщо продуктивність пошуку є критичним фактором, наприклад, при роботі з дуже великими списками даних.

difflib. Модуль difflib надає класи та функції для порівняння послідовностей, включаючи файли, рядки та списки. Він може використовуватися для генерації "різниці" (diff) у різних форматах, а також для обчислення коефіцієнтів схожості між послідовностями.

difflib є вбудованим інструментом, який може використовуватися для базових завдань порівняння рядків та визначення схожості. Однак, thefuzz або RapidFuzz, надають більш гнучкі та прості у використанні API.

В результаті – для системи пошуку з допущеннями було обрано бібліотеку thefuzz. Простота інтеграції та зрозумілість її API роблять її гарним вибором для розробки менеджера паролів. Інтеграція thefuzz дозволить значно покращити користувацький досвід

Після огляду популярних та рентабельних інструментів для реалізаціїї описаних раніше систем для застосунку – менеджера паролів, було обрано наступні бібліотеки:

- Інтерфейс користувача – PyQt6.
- Система керування сховищем – orjson.
- Система шифрування / дешифрування даних – cryptography.
- Система генерації випадкових паролів – secrets.
- Система пошуку з допущеннями – thefuzz.

Програма буде реалізована під операційну систему Windows 11, з використанням мови програмування Python 3.13.3.

1.4 Проєктування та розробка програми

Перед початком розробки програмного коду, необхідно встановити Python, та налаштувати віртуальне оточення. Для встановлення Python версії 3.13 було використано вбудований менеджер пакетів – winget. Процес інсталяції ініціюється виконанням у командному рядку наступної команди:

winget install Python.Python.3.13

Для керування залежностями проєкту та уникнення конфліктів між бібліотеками було створено віртуальне оточення з допомогою стандартного модуля Python venv. У кореневій директорії проєкту було виконано команду:

python -m venv venv

Ця команда створює піддиректорію venv, що містить ізольовану версію інтерпретатора Python та інструменти керування пакетами. Для подальшої роботи, зокрема для встановлення необхідних бібліотек, віртуальне оточення необхідно активувати. Це виконується запуском відповідного скрипту - activate:

venv\Scripts\activate

Після активації, на початку запрошення командного рядка з'явиться позначка з назвою оточення (venv).

Було встановлено всі необхідні залежності проєкту за допомогою стандартного менеджера пакетів python – pip, команда на рис. 1.8.

```
(venv) D:\...\PasswordManager > pip install PyQt6 cryptography orjson thefuzz
Requirement already satisfied: PyQt6 in d:\programming\pyth\passwordmanager\venv
\lib\site-packages (6.8.1)
Requirement already satisfied: cryptography in d:\programming\pyth\passwordmanag
er\venv\lib\site-packages (44.0.2)
```

Рисунок 1.8. Встановлення залежностей

Варто згадати що secrets це стандартний модуль, тому його не потрібно встановлювати окремо.

Було створено більш привабливу структуру проєкту. Для проєкту було використано об'єктно-орієнтовану парадигму. Класи розміщені в окремих модулях для читабельності, та зручної масштабованості. Файлову структуру проєкту можна побачити на рис. 1.9.

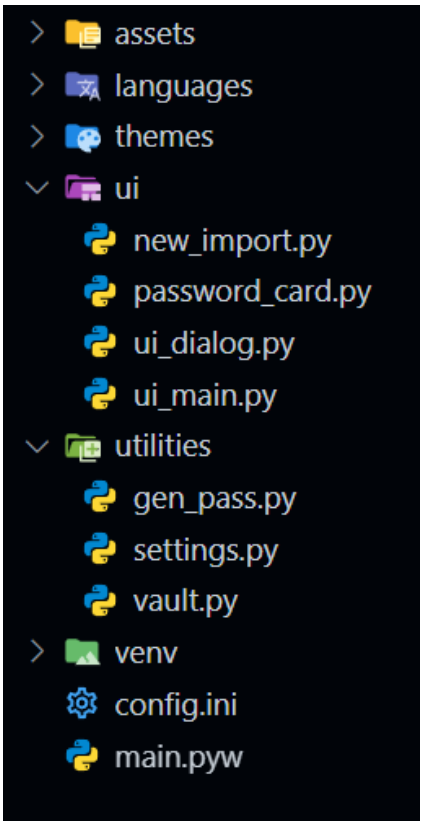


Рисунок 1.9. Файлова структура проєкту

Структура проєкту передбачає логічний поділ коду на модулі, що розміщені у відповідних директоріях. Такий підхід забезпечує високу читабельність коду та спрощує його подальшу підтримку й масштабування. Опис ключових директорій проєкту:

assets – директорія що містить статичні ресурси, для інтерфейсу.

languages – містить файли локалізації (.qm/.ts), для динамічної зміни мови.

themes – містить піддиректорії та файли .qss для динамічної зміни теми.

ui – містить модулі Python для реалізації різних частин інтерфейсу.

utilities – містить допоміжні модулі для роботи з сховищем.

venv – директорія віртуального оточення python, що містить ізольований інтерпретатор, та всі встановлені залежності проєкту.

Опис основних файлів та їх призначення:

main.pyw – головний файл програми – точка входу до застосунку, об'єднує всі інші модулі, та містить основну логіку застосунку.

config.ini – файл конфігурації.

ui/*ui_main.py* – модуль що містить клас головного вікна програми.

ui/ui_dialog.py – модуль що містить класи для різних діалогових вікон програми.

ui*/new_import.py* – містить класи для вікон створення / імпорту сховища.

ui*/password_card.py* – містить класи для віджету списку паролів.

utilities/*gen_pass.py* – містить функцію для генерації унікальних паролів.

utilities/*vault.py* – містить клас менеджеру сховища.

utilities/*settings.py* – містить функції для зміни або читання налаштувань.

Головний алгоритм програми та взаємодію між різними модулями, можна побачити на UML-діаграмі на рис. 1.10.

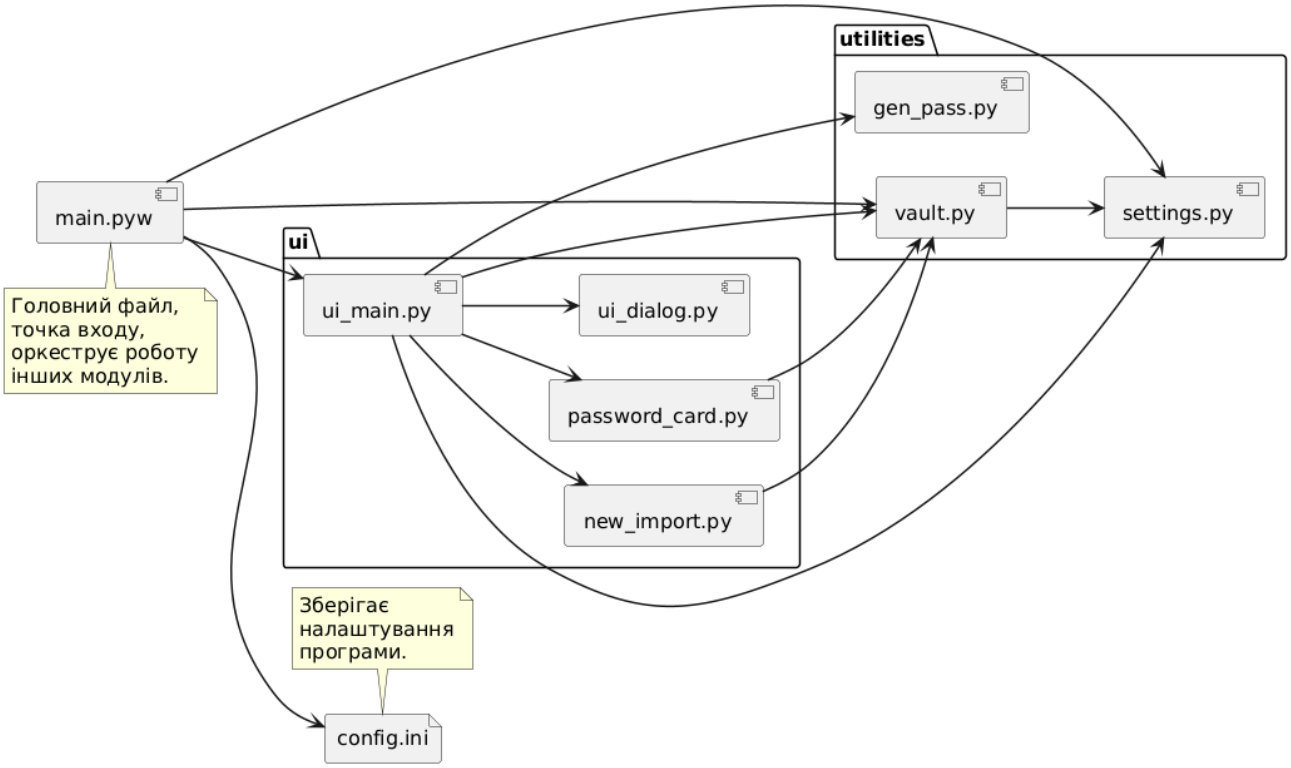


Рисунок 1.10. Головний алгоритм взаємодії модулів між собою

Після налаштування робочого середовища та визначення архітектури було розпочато етап реалізації програмного коду. Розробку було розпочато зі створення класу для керування сховищем. Цей клас поєднує всі інструменти необхідні для роботи з сховищем.

Клас VaultManager, реалізований у модулі vault.py, є центральним елементом системи керування даними.

Метод __init__ це конструктор класу, можна побачити на рис. 1.11, він

завжди викликається коли створюється екземпляр класу. В цьому методі ініціалізуються основні атрибути та йде спроба завантажити сховище, якщо воно вже існує.

```
class VaultManager():
    def __init__(self) -> None:
        self.base_path: Optional[str] = get_vault_path()
        self.key_path: Optional[str] = get_key_path()
        self.key = None
        self.entries: Optional[Vault] = None
        self.key_type: Optional[Literal['password', 'key']] = None
        self.salt: Optional[bytes] = None
        self.nonce: Optional[bytes] = None
        self.invalid_check = False

        if vault_exists(self.base_path):
            self.read_essentials()

        if self.key_type == 'key' and key_exists(self.key_path):
            self.read_key()
            self.read_vault()
```

Рисунок 1.11. Код конструктора __init__

Логіка роботи класу менеджера сховища має в основі взаємодію зі збереженим станом, тобто, якщо сховище вже існує, то йде спроба його завантажити, і всі операції над сховищем можна провести через цей клас на завантаженому сховищі.

Було створено метод для генерації ключа від сховища даних користувача, оголошення методу на рис. 1.12.

```
def create_key(self, key_path: str):

    with open(key_path, 'wb') as file:
        file.write(os.urandom(32))

    self.read_key()
```

Рисунок 1.12. Оголошення методу create_key

Для читання ключа створено метод read_key, оголошення можна побачити на рис. 1.13.

```python
def read_key(self) -> Optional[bytes]:

    if not self.key_path: return

    with open(self.key_path, 'rb') as file:
        out = file.read().strip()

    self.key = out
```

Рисунок 1.13. Оголошення методу read_key

Ключ представляє собою .txt файл що зберігає 256 випадкових бітів, згенерованих функцією os.urandom().

Створено метод для шифрування даних, код можна побачити на рис. 1.14.

```python
def _encode_data(self) -> Optional[bytes]:

    if not self.key_type or not self.salt or not self.nonce:
        return None

    if self.entries is None:
        return None

    if self.key is None:
        return None

    cipher = Cipher(
        algorithm=algorithms.AES(self.key),
        mode=modes.CBC(self.nonce),
        backend=default_backend()
    )

    prepared_data = orjson.dumps(self.entries)

    padder = padding.PKCS7(128).padder()
    padded_data = padder.update(prepared_data) + padder.finalize()

    encryptor = cipher.encryptor()
    try:
        encrypted_data = encryptor.update(padded_data) + encryptor.finalize()
    except ValueError:
        return
```

Рисунок 1.14. Реалізація методу_encode_data

Перед шифрування спочатку робиться ряд перевірок щоб впевнитися що сховище успішно завантажено, є доступ до ключа, та всіх необхідних даних.

Після чого ініціалізується клас Cipher з бібліотеки cryptography, який використовує раніше прочитаний ключ – self.key, в режимі CBC з вектором

ініціалізації self.nonce (16 випадкових байтів).

Далі дані користувача перетворюються на послідовність байтів, після чого ініціалізується об’єкт padder для додавання заповнювача, тому що AES працює з блоками даних фіксованої довжини (128 біт), після чого заповнювач додається до послідовності байтів (даних користувача).

Створюється об’єкт encryptor() який і буде виконувати шифрування, далі вже підготовлені дані з заповнювачем передаються об’єкту encryptor який послідовно обробляє їх та фіналізує процес створюючи зашифровані дані, яким надається назва encrypted_data, якщо цей процес був успішним, шифровані дані повертаються для подальшої обробки.

Метод дешифрації реалізовано аналогічним принципом, однак замість об’єкту encryptor, використовується об’єкт decryptor().

Функціонал для шифрування/дешифрування даних розроблено. Реалізовано метод для створення нового файлу сховища, можна побачити на рис. 1.15.

```
def create_vault(self, key_type: Literal['password', 'key'],
vault_path: str) → bool:

    if not vault_path:
        return False

    self.base_path = vault_path
    self.key_type = key_type
    self.nonce = os.urandom(16)
    self.entries = {}

    return self.write_to_vault()
```

Рисунок 1.15. Метод для створення файлу сховища

Цей метод ініціалізує атрибути екземпляра класу, такі як шлях до сховища, тип ключа, генерує та зберігає значення nonce яке необхідно для процесу шифрування. Також створюється пустий словник Python якому дається ім’я self.entries, це і є дані користувача такі як назви сайтів, логіни, паролі і так далі.

Цей метод готує дані в пам’яті об’єкта. Для їх збереження на диск було реалізовано окремий метод, що відповідає за запис даних у файл сховища. Фрагмент коду цього методу наведено на рис. 1.16.

```python
def write_to_vault(self) -> bool:

    if self.base_path is None:
        return False

    if self.entries is None:
        return False

    if not self.key_type or not self.salt or not self.nonce:
        return False

    encoded_data = self._encode_data()

    if encoded_data is None:
        return False
```

```python
encoded_vault = {
    'key_type': self.key_type,
    'salt': base64.b64encode(self.salt).decode('utf-8'),
    'nonce': base64.b64encode(self.nonce).decode('utf-8'),
    'data': base64.b64encode(encoded_data).decode('utf-8')
}

try:
    with open(self.base_path, 'wb') as file:
        file.write(orjson.dumps(encoded_vault))
    return True
except (OSError, IOError):
    return False
```

Рисунок 1.16. Фрагмент коду метода write_to_vault

Після виконання необхідних перевірок створюється словник що містить всю інформацію необхідну для подальшого завантаження та дешифрування сховища, цей словник містить тип ключа (ключ/пароль), сіль для майбутнього хешування паролю, значення nonce для шифрування, та зашифровані дані користувача. Сіль, nonce та дані користувача представлені у байтовому вигляді, оскільки тип ключа це рядок python, то перед записом до сховища необхідно ці дані перевести у єдиний формат, для цього всі байти переводяться у base64 що перетворюється на рядки Python. Вже після цього використовується .dumps для того щоб перетворити цей словник на байти для запису до файлу.

Для того щоб працювати з сховищем, його потрібно зчитувати з файлу, для

цього створено відповідний метод, який можна побачити на рис. 1.17.

```
def read_vault(self) → bool:

    if not self.base_path or not vault_exists(self.base_path):
         return False

    with open(self.base_path, 'rb') as file:
        raw_data = file.read()
        encoded_vault: EncodedVault = orjson.loads(raw_data)

    encrypted_data = base64.b64decode(encoded_vault['data'])

    if not self.key_type or not self.salt or not self.nonce:
        self.entries = None
        return False

    if encrypted_data is None:
        self.entries = {}
    else:
        self.entries = self._decode_data(encrypted_data)

    return True
```

Рисунок 1.17. Код для методу read_vault

Цей метод відповідає за зчитування файлу сховища, і завантаження його вмісту в атрибути об’єкта VaultManager. Після перевірки на наявність сховища, зчитується вміст файлу у змінну encoded_vault. Після чого виконується валідація завантажених даних перед процесом дешифрації. Якщо перевірки пройдено, буде виконано процес дешифрації, та у self.entries буде записано словник з дешифрованими даними користувача для подальшої обробки інтерфейсом.

Реалізовано можливість створювати та читати сховище, шифрувати та дешифрувати дані користувача з використанням ключа у вигляді файлу .txt з 256 випадковими бітами. Але за умовою, застосунок також має підтримувати майстер-пароль у вигляді ключа. Для цього реалізовано метод (рис. 1.18), який виводить ключ з паролю, для цього використовується алгоритм хешування SHA-256. Хешування це процес перетворення вхідних даних (паролю) у вихідний бітовий рядок фіксованої довжини, тобто пароль користувача буде перетворюватися на 256 бітів хешу, саме підходящий формат для ключа AES.

```python
def derive_key_from_password(self, password: str):

    if self.salt is None:
        return

    kdf = PBKDF2HMAC(
        algorithm=hashes.SHA256(),
        length=32,
        salt=self.salt,
        iterations=1_500_000,
        backend=default_backend()
    )

    encoding_key = kdf.derive(password.encode("utf-8"))
    self.key = encoding_key
    return self.key
```

Рисунок 1.18. Метод derive_key_from_password

Ключовим елементом безпеки при виведені ключа з пароля є використання солі (salt). Хеш-функції є детермінованими: для однакових вхідних даних вони завжди генерують однаковий результат. Ця властивість робить можливими так звані атаки за допомогою райдужних таблиць (rainbow table attack), де зловмисники використовують заздалегідь обчислені хеші для популярних паролів. Сіль – це унікальний, випадково згенерований для кожного сховища набір даних, у даній реалізації це 16 байт доданих до паролю. Оскільки сіль зберігається разом зі сховищем і є унікальною, навіть однакові паролі для різних сховищ будуть генерувати абсолютно різні ключі. Такий підхід робить rainbow-table attack неефективними, оскільки зловмиснику доведеться генерувати нову таблицю для кожної унікальної солі, що вимагає дуже багато ресурсів, оскільки алгоритми хешування спеціально створені таким чином, що вони працюють повільно. Саме для того щоб сповільнити всі потенційні атаки повним перебором.

У створеному методі використовується функціонал модуля cryptography. А саме створюється клас PBKDF2HMAC, що надає функцію виведення ключа з паролю, вказується довжина хешу у 32 байти – підходящий формат ключа для AES256, передається сіль, кількість ітерацій, та вказується алгоритм хешування

SHA256 (рис. 1.19). Я буду використовувати 1.5m ітерацій для отримання ключа з паролю, це призводить до затримки приблизно в 1 секунду. Після чого для виведення ключа з паролю користувача використовується метод derive().

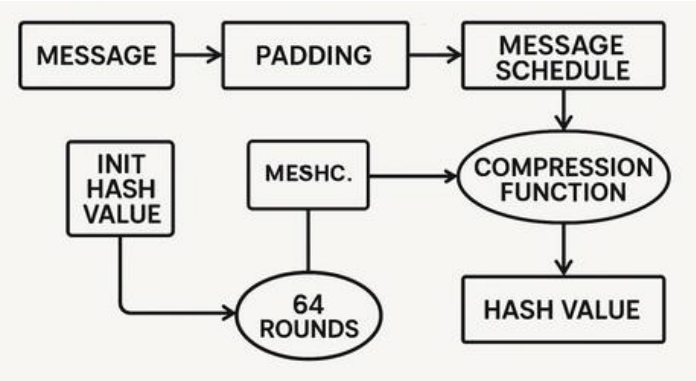


Рисунок 1.19. Алгоритм хешування SHA-256

Після реалізації основного функціоналу роботи зі сховищем було розроблено методи для маніпулювання записами. Першим з них є метод для додавання нових записів, фрагмент коду на рис. 1.20.

Цей метод приймає як аргумент словник, що містить дані нового запису. Перед додаванням виконується низка перевірок. Якщо всі перевірки пройдено, новий запис додається до словника self.entries. Для забезпечення унікальної ідентифікації кожного запису як ключ у словнику використовується унікальний ідентифікатор (UUID), згенерований за допомогою стандартного модуля Python. Приклад UUID: 748a310a-51d2-4588-ae5e-a661aa431304.

```
def add_entry(self, new_entry: VaultEntry) → Optional[str]:

    if self.entries is None:
        print(self.entries)
        return None

    if new_entry.get('site') is None: return
    if new_entry.get('login') is None: return
    if new_entry.get('password') is None: return
    if new_entry.get('notes') is None: return

    new_id = str(uuid.uuid4())

    self.entries[new_id] = new_entry

    return new_id
```

Рисунок 1.20. Фрагмент коду для методу add_entry

Створено метод для редагування раніше створених записів у сховищі, його можна побачити на рис. 1.21.

```
def update_entry(self, id: str, new_entry: VaultEntry) → bool:

    if self.entries is None: return False

    if new_entry.get('site') is None: return False
    if new_entry.get('login') is None: return False
    if new_entry.get('password') is None: return False
    if new_entry.get('notes') is None: return False

    if self.entries.get(id) is not None:
        self.entries[id] = new_entry
        return True

    return False
```

Рисунок 1.21. Метод update_entry

Цей метод отримує на вхід id запису котрий потребує редагування, і запис що містить оновлену інформацію, після чого змінюється запис у словнику self.entries. Ці методи не змінюють сам файл сховища, тому після їх використання необхідно викликати метод для запису сховища у файл – write_to_vault().

Далі було створено метод для видалення вже існуючого запису сховища, який можна побачити на рис. 1.22.

```
def delete_entry(self, entry_id: str) → bool:

    if self.entries is None:
        return False

    del self.entries[entry_id]

    return True
```

Рисунок 1.22. Метод delete_entry

Цей метод приймає тільки id запису, і якщо сховище завантажено то такий запис буде видалено. Після чого обов'язково треба викликати метод write_to_vault щоб зберегти всі зміни у файлі сховища.

Таким чином, було завершено реалізацію класу VaultManager, який

інкапсулює всю логіку роботи зі сховищем. Було реалізовано функціонал створення та читання файлу сховища, шифрування та дешифрування даних користувача за допомогою AES-256, додано методи для генерації випадкового ключа та виведення ключа з майстер-паролю. Методи для додавання, зміни, видалення записів користувача.

Наступним етапом розробки є створення графічного інтерфейсу користувача за допомогою бібліотеки PyQt6. Центральним елементом інтерфейсу є головне вікно програми, реалізоване у вигляді окремого класу. Цей клас наслідується від базового класу QWidget бібліотеки PyQt6. Основна логіка ініціалізації та компонування вікна зосереджена в методі init_ui(). На початковому етапі в цьому методі створюються та налаштовуються всі ключові елементи інтерфейсу: кнопки, текстові мітки, поля для введення тощо, це можна побачити на рис. 1.23.

```
class Ui_MainWindow(QWidget):
    def __init__(self):
        super().__init__()
        self.init_ui()

    def init_ui(self):
        self.main_layout = QVBoxLayout()
        self.button_layout = QHBoxLayout()

        self.create_button = QPushButton()
        self.import_button = QPushButton()
        self.add_button = QPushButton()
        self.info_button = QPushButton()
        self.settings_button = QPushButton()

        self.create_button.setObjectName('CreateButton')
        self.import_button.setObjectName('ImportButton')
        self.add_button.setObjectName('AddButton')
        self.info_button.setObjectName('InfoButton')
        self.settings_button.setObjectName('SettingsButton')
```

Рисунок 1.23. Створення основних елементів

Далі об'єктам надаються необхідні параметри, такі як розміри, назви, необхідні політики для керування розташуванням і так далі, це можна побачити на рис. 1.24.

```
        self.info_button.setObjectName('InfoButton')
        self.settings_button.setObjectName('SettingsButton')

        self.create_button.setFixedSize(50, 30)
        self.add_button.setFixedSize(30, 30)
        self.import_button.setFixedSize(60, 30)
        self.info_button.setFixedSize(20, 20)
        self.settings_button.setFixedSize(20, 20)

        self.info_text = QLabel()
        self.info_text.setFixedSize(350, 30)

        self.add_button.setIconSize(QSize(20, 20))
        self.info_button.setIconSize(self.info_button.size())
        self.settings_button.setIconSize(self.settings_button.size())

        self.search_field = QLineEdit()
        self.search_field.setObjectName('SearchField')

        self.password_list = SiteList()
        self.password_list.setObjectName('PasswordList')

        self.init_language()
```

Рисунок 1.24. Ініціалізація параметрів для основних елементів

Додаються віджети до головного макету (рис. 1.25), у PyQt6 макет регулює правила розміщення віджетів.

```
self.button_layout.addWidget(self.create_button)
self.button_layout.addWidget(self.import_button)
self.button_layout.addWidget(self.add_button)
self.button_layout.addWidget(self.info_text)
self.button_layout.addWidget(self.info_button)
self.button_layout.addWidget(self.settings_button)

self.main_layout.addLayout(self.button_layout)
self.main_layout.addWidget(self.search_field)
self.main_layout.addWidget(self.password_list)

self.setLayout(self.main_layout)
```

Рисунок 1.25. Додано віджети до головного макету

Для реалізації функціоналу динамічної зміни мови, було застосовано підхід що базується на механізмі подій PyQt6. Текстові значення для всіх віджетів задаються в окремому методі. Далі було перевизначено обробник подій changeEvent() на виконання методу для встановлення тексту (рис. 1.26). Цей процес пов’язаний з роботою класу Qtranslator.

```
def changeEvent(self, event):
    if event.type() == QEvent.Type.LanguageChange:
        self.init_language()
    super().changeEvent(event)

def init_language(self):
    self.create_button.setText(self.tr('New'))
    self.import_button.setText(self.tr('Import'))
    self.info_text.setText(self.tr('Welcome to Password Manager!'))
    self.search_field.setPlaceholderText(self.tr('Search...'))
```

Рисунок 1.26. Методи для динамічної зміни мови

Основний виконуваний модуль main.pyw містить функцію main(), яка виступає точкою входу в програму. Вона виконує ініціалізацію всіх ключових компонентів, об'єднання модулів та запуск головного циклу обробки подій. Створюється екземпляр QApplication, це ядро будь-якого застосунку на PyQt6. Для реалізації динамічної зміни мови використовується вбудований у PyQt6 механізм QTranslator. Далі, за допомогою методу load(), завантажується відповідний файл локалізації, та перекладач встановлюється у ядро застосунку, фрагмент коду можна побачити на рис. 1.27.

```
def main():
    app = QApplication(sys.argv)

    # Install translator, with current language settings
    translator = QTranslator(app)
    translator.setObjectName('MainTranslator')
    translator.load(f'languages\\{load_settings(type='GENERAL',
                    parameter='language')}.qm')
```

Рисунок 1.27. Встановлення мови графічного інтерфейсу додатку

На наступному етапі було реалізовано механізм для забезпечення глобального доступу до езкемпляра менеджера сховища (рис. 1.28). Для цього об'єкт класу VaultManager було ініціалізовано та прикріплено як динамічний атрибут до ядра програми. На завершення етапу ініціалізації зчитуються збережені налаштування користувача, і відповідна тема встановлюється для всього застосунку через об'єкт ядра застосунку - app. Для цього завантажується відповідний файл стилів (QSS), вміст якого застосовується до всіх елементів інтерфейсу.

```
# Add VaultManager instance to the global eventloop
setattr(app, '_vault_manager', VaultManager())

app.setStyleSheet(read_theme())

window = MainWindow()
window.show()

sys.exit(app.exec())
```

Рисунок 1.28. Екземпляр класу VaultManager додається до ядра програми

Для запуску застосунку, створено наступний код на рис. 1.29, який забезпечує безпечний запуск, тобто якщо main.pyw буде імпортовано в якості модуля, то функція main не буде виконана.

```
if __name__ == '__main__':
    main()
```

Рисунок 1.29. – Код для запуску застосунку

Логіка для реалізації головного вікна, та запуску застосунку готова, переходимо до реалізації всіх інших елементів інтерфейсу.

Для реалізації списку паролів створено клас для керування елементами списку, та клас що представляє сам елемент. Створено клас керування елементами SiteList (рис. 1.30).

```
class SiteList(QWidget):
    fuzzy_search_signal = pyqtSignal(str)
    update_info = pyqtSignal(str)
```

Рисунок 1.30. Оголошення класу SiteList

У методі-конструкторі даного компонента реалізовано ключову логіку, яка відповідає за початкову побудову макету та конфігурацію його елементів. Зокрема, тут визначаються фундаментальні правила для керування розміром, такі як фіксовані габарити, а також політики вирівнювання елементів. Візуальний результат застосування цих правил для компонування елементів продемонстровано на рис. 1.31.

```
def __init__(self):
    super().__init__()
    self.setObjectName('SiteList')
    self.items_map = {}
    self.current_expanded = None
    self.divider = QSpacerItem(0, 20, QSizePolicy.Policy.Minimum,
                               QSizePolicy.Policy.Fixed)

    self.container_widget = QWidget()
    self.container_widget.setObjectName('SiteListContainer')
    self.container_layout = QVBoxLayout(self.container_widget)
    self.container_layout.setAlignment(Qt.AlignmentFlag.AlignTop)
    self.container_layout.setContentsMargins(0, 0, 0, 0)
    self.container_layout.setSpacing(0)
```

Рисунок 1.31. Створення макетів та правил

Наступним кроком створено логіку для обробки двох типів ключа. На початку отримується екземпляр класу менеджера сховища з ядра програми. Далі перевіряється тип ключа для сховища, якщо це сховище з майстер-паролем, то виконується метод handle_password_type(). Інакше йде спроба отримати дані сховища, оскільки тип у вигляді файлу-ключа, передбачає автоматичне завантаження ключа, а значить всі дані користувача вже мають бути завантажені в пам’ять і готові для використання, якщо саме сховище було успішно завантажено. Далі виконується метод fill_passwords, який заповнює список записів даними користувача, фрагмент коду можна побачити на рис. 1.32.

```
# Instance of VaultManager from app's eventloop·
self._vault_manager: VaultManager = getattr(QApplication.instance(),
                                            '_vault_manager')
self.entries = None

if self._vault_manager.key_type == 'password':
    self.handle_password_type()
else:
    self.entries = self._vault_manager.get_all_entries()

if self.entries is not None:
    self.fill_passwords(entries=self.entries)

self.welcome_window = WelcomeWidget()

if self.entries is None:
    self.show_welcome_widget()
```

Рисунок 1.32. Обробка двох типів ключа

Створено віджет та зону з прокручуванням, для елементів що розкриваються. Далі всі елементи додаються до головного макету (рис. 1.33).

```
# Details widget
self.details_widget = DetailsWidget()
self.details_widget.setVisible(False)
self.details_widget.delete_requested.connect(self.delete_entry)
self.details_widget.edit_requested.connect(self.edit_entry)

self.scroll_area = QScrollArea()
self.scroll_area.setWidgetResizable(True)
self.scroll_area.setWidget(self.container_widget)

layout = QVBoxLayout(self)
layout.setContentsMargins(0, 0, 0, 0)
layout.setSpacing(0)

layout.addWidget(self.scroll_area)
```

Рисунок 1.33. Логіка зони для елементів що розкриваються

Було створено всі необхідні методи:

show_welcome_widget() – логіка для створення підказки на першому запуску, коли сховище не завантажено.

hide_welcome_widget() – видалення підказки на першому запуску

handle_password_type() – відповідає за створення промпту для користувача, для введення майстер-паролю.

update_list_data() – метод для оновлення списку даних користувача, використовується після імпорту нового сховища, для очищення списку та додавання нових даних.

fill_password() – відповідає за заповнення списку після запуску застосунку.

add_entry() – відповідає за додавання нового запису через графічний інтерфейс.

delete_entry() – метод для видалення запису зі списку, також модифікує файл сховища.

edit_entry() – відповідає за редагування записів у списку, модифікує файл сховища.

fuzzy_search() – відповідає за нечіткий пошук, для реалізації було використано модуль thefuzz (рис. 1.34).

expand_rules() – відповідає за обробку логіки елементів що розкриваються.

```
if self.entries is not None:
    list_of_site_names = [(id, entry['site'], fuzz.partial_ratio(filter,
    entry['site'])) for id, entry in self.entries.items()]

sorted_entries = sorted(
    list_of_site_names,
    key=lambda item: item[2],
)
```

Рисунок 1.34. Логіка для пошуку з допущеннями

Далі було створено клас який представляє віджет з деталями, для елементів що розкриваються (рис. 1.35).

```
class DetailsWidget(QWidget):
    delete_requested = pyqtSignal(str, name='DeleteRequest')
    edit_requested = pyqtSignal(str, name='EditRequest')
    update_info = pyqtSignal(str)
```

Рисунок 1.35. Оголошення класу DetailsWidget

Цей клас являє собою віджет з деталями для кожного запису який включає таку інформацію як логін, пароль та замітки. Також було додано функцію для копіювання логіну/паролю кліком по відповідному полю, та перемикач видимості для пароля, за замовчуванням пароль схований.

Далі створено клас вікна для редагування існуючих записів, який буде викликатися після натиску на кнопку для редагування у розкритому віджеті для деталей (рис. 1.36).

```
class EditDialog(QDialog):
    def __init__(self, site: str, login: str, password: str, notes: str):
        super().__init__()
        self.setObjectName('EditDialog')
```

Рисунок 1.36. Оголошення класу EditDialog

Було створено клас вікна для відображення діалогового вікна для отримання майстер-паролю (рис. 1.37).

```
class PasswordPrompt(QDialog):
    def __init__(self):
        super().__init__()
        self.setObjectName('PasswordPrompt')
```

Рисунок 1.37. Оголошення класу PasswordPrompt

На останок було створено віджет для відображення підказки для

користувача, коли сховище ще не існує (рис. 1.38).

```
class WelcomeWidget(QWidget):
    create_signal = pyqtSignal()
    import_signal = pyqtSignal()
```

Рисунок 1.38. Оголошення класу WelcomeWidget

Після створення всіх необхідних класів, модуль password_card завершено. Було створено кастомний список елементів, де кожен елемент розкривається та відображає дані користувача, з можливістю зміни, видалення та додавання нових елементів але тепер з використанням графічного інтерфейсу.

Створення класів інтерфейсу для інших вікон програми. Для початку реалізовано клас для вікна створення нового сховища (рис. 1.39).

```
class NewDialog(QDialog):
    def __init__(self):
```

Рисунок 1.39. Оголошення класу NewDialog

Цей клас відповідає за відображення вікна для створення нового сховища. Далі було створено клас для імпорту вже існуючого сховища (рис. 1.40).

```
class ImportDialog(QDialog):
    def __init__(self):
        super().__init__()
```

Рисунок 1.40. Оголошення класу ImportDialog

Цей клас відповідає за імпорт вже існуючого сховища, користувачу необхідно вказати шлях до сховища та ключа, або передати майстер-пароль.

Класи для вікон створення/імпорту сховища готові, отже модуль new_import завершено. Створено клас для діалогового вікна налаштувань програми, оголошення можна побачити на рис. 1.41.

```
class SettingsWindow(QDialog):
    def __init__(self):
        super().__init__()
        self.setObjectName('SettingsWidget')
```

Рисунок 1.41. Оголошення класу SettingsWindow

Цей клас відповідає за відображення діалогового вікна для зміни

налаштувань теми та мови, підтримується дві теми – темна/світла, та дві мови – англійська/українська.

Створено клас для діалогового вікна додавання нового запису до сховища (рис. 1.42)

```
class AddDialog(QDialog):
    def __init__(self):
        super().__init__()
```

Рисунок 1.42. Оголошення класу AddDialog

Цей клас відповідає за створення діалогового вікна для додавання нового запису до сховища, користувач має ввести назву сайту або ресурсу, логін/пароль та за бажанням можна додати замітки.

На цьому модуль ui_dialog реалізовано. Далі було створено основний клас програми який буде об’єднувати всі модулі та містити основну логіку програми (рис. 1.43).

```
class MainWindow(QMainWindow):
    def __init__(self) → None:
        super().__init__()
        self.ui = ui.Ui_MainWindow()
        self.setCentralWidget(self.ui)
        self.setWindowTitle("Password Manager")
        self.setWindowIcon(QIcon('assets\\main_window.ico'))
        self.setFixedSize(450, 295)
```

Рисунок 1.43. Оголошення класу MainWindow

У конструкторі головного вікна програми відбувається ініціалізація та компонування основного інтерфейсу. Створюється екземпляр віджета основного вікна і встановлюється як центральний компонент вікна.

Після цього здійснюється доступ до глобального екземпляра VaultManager, який було ініціалізовано на етапі запуску застосунку (рис. 1.44). Якщо дані були успішно завантажені, заголовок головного вікна динамічно оновлюється — до нього додається назва активного файлу сховища. Це допомагає уникнути плутанини при роботі з кількома різними сховищами.

```
self.vault_instance: VaultManager = getattr(QApplication.instance(), '_vault_man
ager')
self.update_title(path=load_settings(type='ESSENTIALS', parameter='base_path'))
```

Рисунок 1.44. Доступ до екземпляра класу VaultManager

Далі необхідні сигнали було під'єднано з об'єкта списку записів до методів класу MainWindow які будуть описані пізніше (рис. 1.45).

```
self.ui.password_list.welcome_window.create_signal.connect(self.create_button)
self.ui.password_list.welcome_window.import_signal.connect(self.import_button)
self.ui.password_list.details_widget.update_info.connect(self.set_info)
self.ui.password_list.update_info.connect(self.set_info)
```

Рисунок 1.45. Під'єднання сигналів об'єкта списку паролів

Тепер необхідно під'єднати сигнали всіх інших елементів графічного інтерфейсу користувача які були створені раніше до відповідних методів головного класу які будуть створені пізніше (рис. 1.46).

```
self.ui.create_button.clicked.connect(self.create_button)
self.ui.import_button.clicked.connect(self.import_button)
self.ui.add_button.clicked.connect(self.add_button)
self.ui.settings_button.clicked.connect(self.init_settings)
self.ui.info_button.clicked.connect(self.show_help)
self.ui.search_field.textChanged.connect(self.search_passwords)
```

Рисунок 1.46. Під'єднані сигнали елементів графічного інтерфейсу

Логіка конструктору реалізована, тепер створимо самі методи які будуть виконуватися при отриманні необхідних сигналів. Було створено метод для обробки кліку на кнопку для створення сховища (рис. 1.47).

```
def create_button(self) -> None:
    new_vault_window = NewDialog()

    if new_vault_window.exec() != NewDialog.DialogCode.Accepted:
        return

    vault_path = new_vault_window.get_vault_path()

    if vault_path is None:
        return

    vault_name = Path(vault_path).name
```

Рисунок 1.47. Оголошення методу create_button

Створюється екземпляр раніше створеного класу для вікна створення нового сховища, і якщо користувач передав валідні шляхи то отримується інформація з вікна, спочатку шлях до сховища, потім ім'я сховища.

Далі йде обробка двох можливих типів ключа, для початку обробляється варіант коли користувач створює сховище з майстер-паролем, це можна побачити на рис. 1.48.

```
if new_vault_window.password_switch.isChecked():
    vault_pass = new_vault_window.get_password()
    save_settings(type='ESSENTIALS', settings={'base_path': vault_path})
    self.vault_instance.base_path = vault_path

    self.vault_instance.generate_salt()
    self.vault_instance.derive_key_from_password(password=vault_pass)

    self.vault_instance.create_vault(key_type='password', vault_path=vault_path)
    self.vault_instance.read_vault()
    return
```

Рисунок 1.48. Логіка для обробки ключа у вигляді паролю

Логіка створення нового сховища починається з аналізу обраного користувачем типу ключа. У випадку, коли обрано варіант з майстер-паролем, зчитується пароль, оновляється шлях до сховища у файлі налаштувань, та у екземплярі менеджера сховища. Далі викликаються методи менеджера сховища, виконується генерація солі для хешування паролю, виводиться ключ з майстер-паролю, і створюється сховище. Після чого сховище завантажується щоб поточний екземпляр менеджера сховища працював з актуальними даними.

Наступним кроком створено логіку для обробки варіанта ключа у вигляді файлу (рис. 1.49).

```
else:
    key_path = new_vault_window.get_key_path()

    if key_path is None:
        return None

    save_settings(type='ESSENTIALS', settings={'base_path': vault_path})
    save_settings(type='ESSENTIALS', settings={'key_path': key_path})

    self.vault_instance.base_path = vault_path
    self.vault_instance.key_path = key_path

    self.vault_instance.create_key(key_path=key_path)
    self.vault_instance.create_vault(key_type='key', vault_path=vault_path)
    self.vault_instance.read_vault()
```

Рисунок 1.49. Логіка для обробки ключа у вигляді файлу

Оскільки логіка вибору типу ключа реалізована через умовний оператор,

блок коду для створення сховища з ключовим файлом виконується, якщо не було обрано варіант з майстер-паролем. Немає необхідності в додатковій перевірці.

Спочатку зчитується шлях до файлу, який буде використовуватися як ключ. Потім шляхи до файлу сховища та до файлу-ключа зберігаються в конфігураційному файлі програми. Відповідні атрибути оновлюються і в поточному екземплярі VaultManager, після чого генерується файл-ключ.

В кінці оновлюється заголовок головного вікна програми на назву нового сховища, оновлюється список паролів у графічному інтерфейсі, та відображається інформаційний текст щодо успішно створеного нового сховища (рис. 1.50).

```
self.update_title(path=vault_path)
self.ui.password_list.update_list_data()
self.set_info(info=f'[{vault_name}]: ' + self.tr('created!'))
```

Рисунок 1.50. Оновлення інтерфейсу після створення сховища

На цьому метод для обробки події створення нового сховища завершено. Далі було створено метод для імпорту вже існуючого сховища. Цей метод за своєю логікою дуже схожий на попередній, оскільки також передбачає взаємодію з файловою системою та завантаження даних. Однак, замість генерації, він використовує спеціальне вікно для імпорту сховища, і нове сховище, відповідно, не буде створено у процесі імпорту, а лише завантажено з обраного файлу (рис. 1.51).

```
def import_button(self):
    import_vault_window = ImportDialog()

    if import_vault_window.exec() != ImportDialog.DialogCode.Accepted:
        return

    vault_path = import_vault_window.get_vault_path()

    if vault_path is None:
        return

    vault_name = Path(vault_path).name
```

Рисунок 1.51. Оголошення методу import_button

Було створено логіку для кнопки додавання нових записів до сховища записів користувача (рис. 1.52).

```
def add_button(self) -> None:
    add_entry = ui_dialog.AddDialog()

    if add_entry.exec() == add_entry.DialogCode.Accepted:

        new_entry: VaultEntry = add_entry.get_data()
        id = self.vault_instance.add_entry(new_entry=new_entry)
        self.vault_instance.write_to_vault()
        self.vault_instance.read_vault()

        self.ui.password_list.add_entry(id=id, entry=new_entry)
        self.set_info(info=f'[{new_entry['site']}]: ' +
                      self.tr('was added!'))
```

Рисунок 1.52. Метод add_button

На початку створюється екземпляр класу діалогового вікна для додавання нових записів. Далі зчитуються всі дані користувача, по-перше додається запис до сховища що завантажено у екземплярі менеджера сховища, записується у файл, та зчитується знову щоб переконатися що дані актуальні. Вже потім оновлюється список паролів у графічному інтерфейсі, та відображається інформаційний текст щодо створення нового запису.

Було створено методу для обробки динамічних налаштувань мови та теми (рис. 1.53).

```
def init_settings(self) -> None:

    settings = ui_dialog.SettingsWindow()

    if settings.exec() == ui_dialog.QDialog.DialogCode.Accepted:
        selected_settings = settings.get_selected_options()
    else:
        return

    current_lang_settings = load_settings(
        type='GENERAL', parameter='language')
    current_theme_settings = load_settings(
        type='GENERAL', parameter='theme')

    save_settings(type='GENERAL', settings=selected_settings)
```

Рисунок 1.53. Оголошення методу init_settings

На початку створюється екземпляр класу діалогового вікна для налаштувань, та зчитуються всі параметри які обрав користувач. Після чого

завантажуються поточні налаштування теми та мови які зберігаються у файлі налаштувань (config.ini), нові налаштування записуються у файл.

Після чого обробляються змінені налаштування, починаючи з мови (рис. 1.54). Процес динамічної зміни мови реалізовано шляхом повної заміни активного об'єкта-перекладача. З ядра застосунку видаляється раніше завантажений об'єкт QTranslator. Це необхідно для того, щоб звільнити місце для нового перекладу та уникнути конфліктів. Створюється новий, порожній екземпляр класу Qtranslator, у який завантажується скомпільований .qm-файл. Новий екземпляр QTranslator з завантаженим перекладом встановлюється як активний для всього застосунку через виклик методу installTranslator().

```
if current_lang_settings != selected_settings['language']:

    capp = cast(QCoreApplication, QCoreApplication.instance())

    translator = capp.findChild(QTranslator, 'MainTranslator')

    if translator:
        capp.removeTranslator(translator)

    translator = QTranslator(capp)
    translator.setObjectName('MainTranslator')

    app = cast(QApplication, QApplication.instance())

    if translator.load(f"languages\\{selected_settings['language']}.qm"):
        app.installTranslator(translator)
```

Рисунок 1.54. Обробка зміни мови інтерфейсу

Варто зазначити, що ця операція змінить мову лише для нових елементів програми, які будуть створені вже після її виклику. Для того щоб мова динамічно змінилась і для всіх вже створених елементів інтерфейсу, необхідно було заздалегідь реалізувати спеціальний механізм. Для цього для кожного важливого елементу було переписано базовий метод changeEvent. У фреймворку PyQt6 це спеціальний метод-обробник, який система автоматично викликає, коли відбувається певна системна подія, зокрема QEvent::LanguageChange. Таким чином, при зміні мови кожен елемент отримує сигнал і може оновити свій текст (рис. 1.55).

```python
def changeEvent(self, event):
    if event.type() == QEvent.Type.LanguageChange:
        self.init_language()
    super().changeEvent(event)

def init_language(self):
    self.create_button.setText(self.tr('New'))
    self.import_button.setText(self.tr('Import'))
    self.info_text.setText(self.tr('Welcome to Password Manager!'))
    self.search_field.setPlaceholderText(self.tr('Search...'))
```

Рисунок 1.55. Метод changeEvent класу Ui_MainWindow

Цей метод було переписано наступним чином. Спочатку перевіряється яка подія відбулася, в нашому випадку перевіряється чи була змінена мова. Ця подія автоматично викликається завжди коли змінюється завантажений перекладач (видаляється або додається новий). Таким чином, коли було завантажено новий перекладач, всі необхідні елементи інтерфейсу відреагували на цю подію, та повторно ініціалізували весь текст видимий для користувача.

Для того щоб це працювало ми маємо створити файли перекладу для англійської та української мови, Qtranslator підтримує спеціальний тип файлів .qm, для того щоб отримати ці файли потрібно провести ряд етапів підготовки. Для початку всі елементи тексту які будуть підтримувати переклад, мають бути огорнуті у спеціальний тег .tr(), це спеціальний тег який позначає текст для перекладу. Коли весь потрібний текст підготовлено, можна перейти до наступного етапу. Тепер необхідно спарсити весь текст для перекладу з усіх файлів програми, для цього разом з PyQt6 поставляється ряд інструментів для використання у командному рядку, для того щоб спарсити весь потрібний текст у файл, використовується утиліта pylupdate6 де через кому можна перерахувати всі необхідні файли (рис. 1.56).

```
(venv) D:\PasswordManager > pylupdate6.exe .\ui\new_import.py .\ui\password_card.py .\ui\ui_
dialog.py .\ui\ui_main.py  main.py -ts test.ts --verbose
Reading .\ui\new_import.py...
Parsing .\ui\new_import.py...
Reading .\ui\password_card.py...
Parsing .\ui\password_card.py...
Reading .\ui\ui_dialog.py...
Parsing .\ui\ui_dialog.py...
Reading .\ui\ui_main.py...
Parsing .\ui\ui_main.py...
Reading main.py...
Parsing main.py...
Updating test.ts from .\ui\new_import.py...
```

```
Writing test.ts...
Summary of changes to test.ts:
    120 new messages were added (and 4 duplicates)
```

Рисунок 1.56. Використання інструменту pylupdate6

Як бачимо, програма успішно спарсила 120 елементів тексту які були позначені тегом .tr(), та створила новий файл формату .ts, це відкритий файл у який нам тепер потрібно додати сам переклад, для цього використовується програма Qt Linguist. Це окремий додаток з графічним інтерфейсом який можна завантажити з офіційного гітхабу Qt. Через цей додаток можна написати весь необхідний переклад, приклад на рис. 1.57.

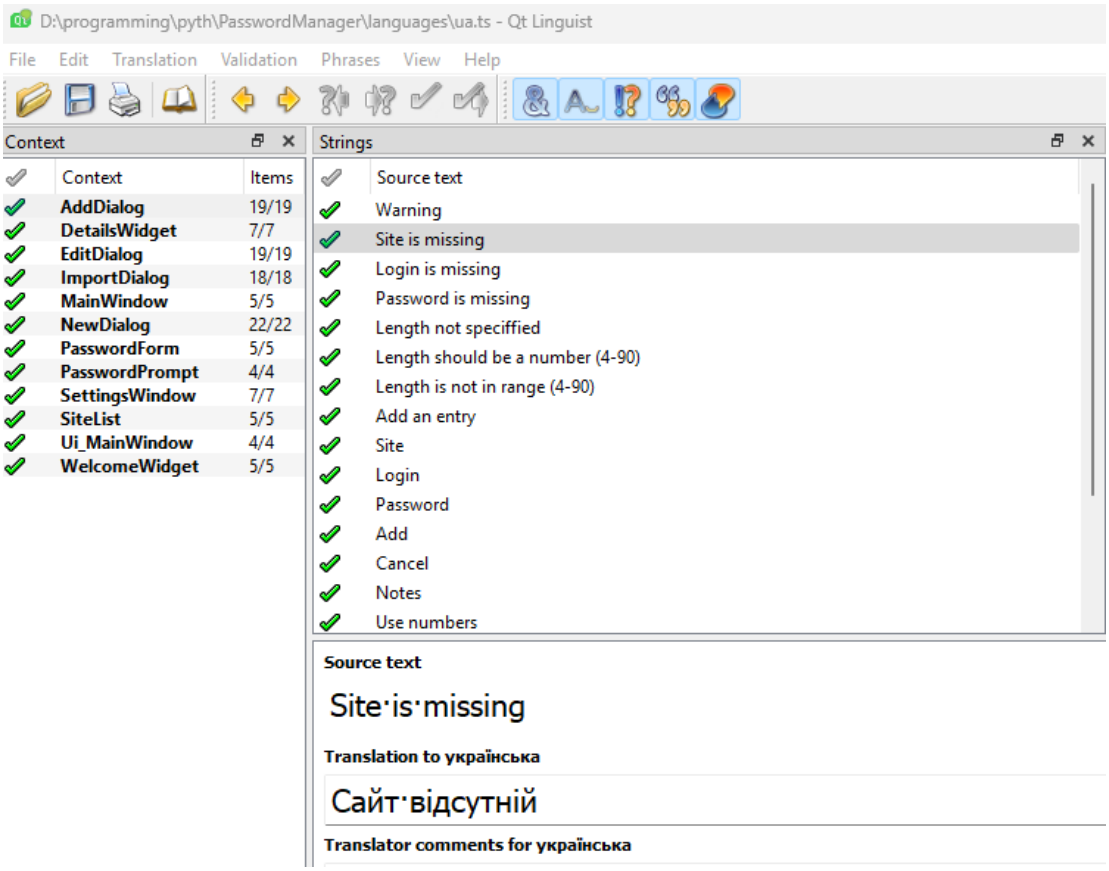


Рисунок 1.57. Приклад перекладу для української мови

Було додано український варіант тексту для кожного елементу тексту. Далі цей файл компілюється щоб його можна було використовувати для перекладачу, для цього разом з PyQt6 постачається інструмент lrelease для використання в командному рядку, достатньо тільки передати файл перекладу для отримання компільованого файлу .qm (рис. 1.58)

```
(venv) D:\[redacted]\languages > lrelease.exe .\en.ts
Updating '.\en.qm'...
    Generated 120 translation(s) (120 finished and 0 unfinished)
```

Рисунок 1.58. Компіляція файлу перекладу

Було створено відповідні файли перекладу на англійську та українську мови, на цьому логіка для динамічної зміни мови реалізована. Для динамічної зміни теми застосунку було реалізовано логіку яку можна побачити на рис. 1.59.

```
if current_theme_settings != selected_settings['theme']:
    app_instance = cast(QApplication, QApplication.instance())
    app_instance.setStyleSheet(read_theme())
    self.ui.password_list.details_widget.refresh_icon_theme()
```

Рисунок 1.59. Логіка для динамічної зміни теми

Логіка динамічної зміни теми була реалізована з використанням файлів .qss, це аналог формату .css але з підтримкою спеціальних властивостей Qt. Для зміни теми достатньо отримати доступ до ядра програми та встановити новий файл теми, для того щоб зручніше читати необхідний файл було створено функцію read_theme (рис. 1.60).

```
def read_theme():
    theme_settings = load_settings(type='GENERAL', parameter='theme')

    with open(f'themes\\{theme_settings}\\style.qss') as new_theme:
        style = new_theme.read()

    return style
```

Рисунок 1.60. Функція read_theme

Дана функція інкапсулює всю логіку, пов'язану із завантаженням візуальних стилів. Вона спочатку звертається до поточних налаштувань програми, щоб визначити, яка саме тема є активною на даний момент (наприклад, "світла" чи "темна"). На основі цієї інформації вона формує шлях до відповідного файлу стилів (наприклад, у форматі QSS або CSS), зчитує його вміст, обробляючи можливі винятки, як-от відсутність файлу. В кінцевому підсумку, готовий результат у вигляді текстового рядка повертає у місце виклику, де ці дані вже використовуються для безпосереднього застосування стилів до графічного інтерфейсу програми.

На цьому метод для зміни налаштувань реалізовано. Далі було створено

допоміжні методи для ініціалізації пошуку, оновлення інформаційного тексту, та зміни назви головного вікна програми, їх можна побачити на рис. 1.61.

```
def search_passwords(self) → None:
    filter: str = self.ui.search_field.text()
    self.ui.password_list.fuzzy_search(filter=filter)
    self.ui.password_list.update_info.connect(self.set_info)

def set_info(self, info: str):
    self.ui.info_text.setText(info)

def update_title(self, path):
    if vault_exists(path=path):
        vault_name = Path(path).name
        title = f'Password Manager - {vault_name}'
        self.setWindowTitle(title)
```

Рисунок 1.61. Допоміжні методи

Також було створено метод для відображення інформації користувачу після кліку на кнопку допомоги (рис. 1.62).

```
def show_help(self):
    help_window = QMessageBox(self)
    help_window.setContentsMargins(2, 2, 5, 5)
    help_window.setWindowTitle(self.tr('Help / Info'))
    help_window.setText(
        self.tr(
            "Welcome to Password Manager!\n\n"
            "To start, you need to:\n"
            "Create New or Import existing Vault\n\n"
            "You can use two key options:\n"
            "- Key (AES) - safer.\n"
            "- Password (sha256) - simpler.\n\n"
            "You can add entries when vault is loaded\n"
            "You can either edit or delete any entry\n\n"
            "Random password generation support"
        )
    )
    help_window.setWindowIcon(QIcon('assets\\info_window.ico'))
    help_window.exec()
```

Рисунок 1.62. Метод show_help

Для відображення інформації використано QMessageBox, вказано розмір, назву, створено сам інформаційний текст та відображення вікна методом .exec().

Клас з основною логікою програми реалізовано. На цьому написання коду завершено, було створено всі зумовлені системи, для виконання всіх необхідних функцій.

Було описано технічну реалізацію менеджера паролів. Було продемонстровано використання мови Python та обраного стеку технологій, зокрема PyQt6 для побудови графічного інтерфейсу та бібліотеки cryptography для криптографічних операцій, з метою створення функціонального та безпечного додатку. Розглянуто архітектурні рішення та ключові алгоритми, що лежать в основі роботи програми, включаючи шифрування AES-256, механізми автентифікації за допомогою файлу-ключа або майстер-пароля, генерацію паролів, пошук з допущеннями та інтуїтивний інтерфейс. Вибір цих інструментів та підходів був зумовлений прагненням досягти балансу між надійністю, продуктивністю та зручністю для кінцевого користувача. Розроблений код є основою для функціонування всіх можливостей програми, які будуть детально продемонстровані в наступному розділі.

1.5 Тестування функціоналу додатку

Інтерфейс користувача було створено з акцентом на простоту та інтуїтивність використання. При першому запуску програма вітає користувача та пропонує два основні шляхи: створити нове сховище або імпортувати вже існуюче, це вікно можна побачити на рис. 1.63.

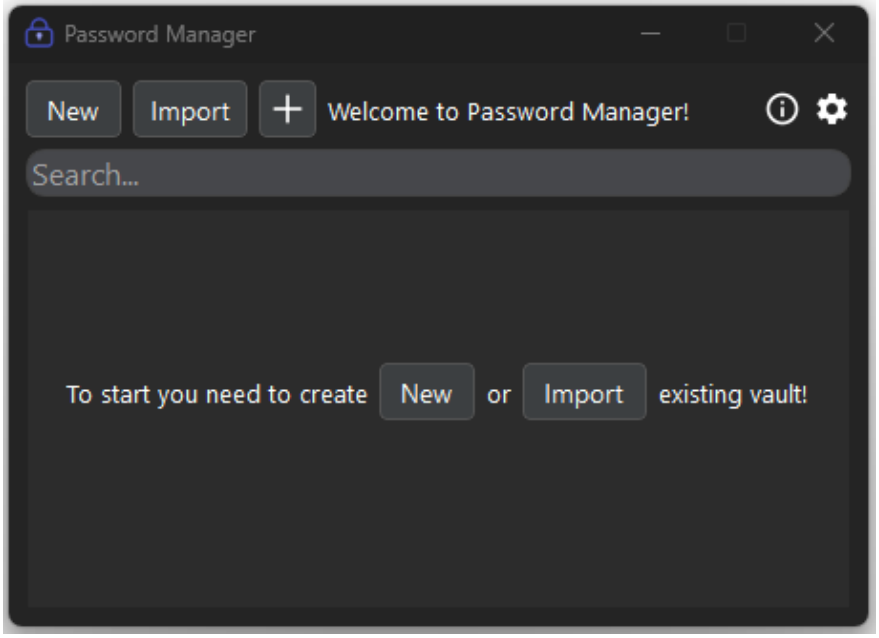


Рисунок 1.63. Вигляд вікна на першому запуску програми

Для початку потрібно створити нове сховище, після кліку на кнопку New,

з'явиться наступне вікно для створення сховища (рис. 1.64).

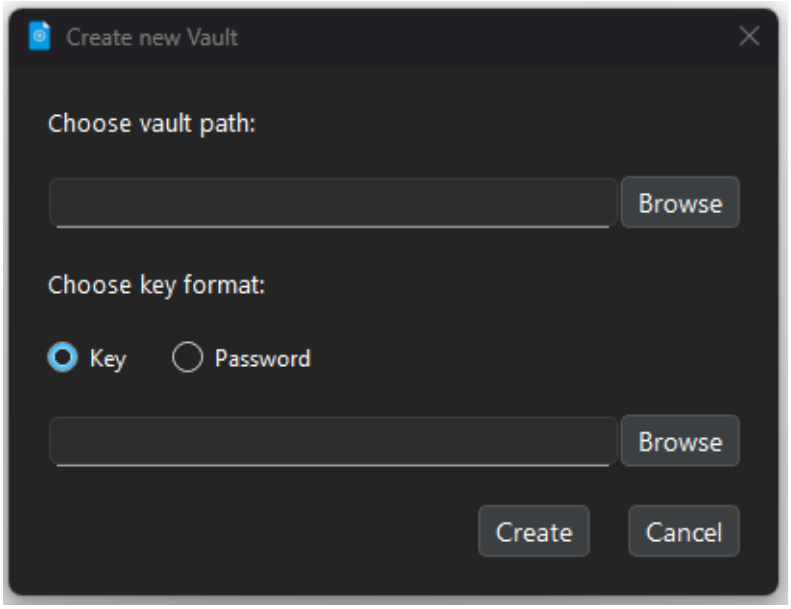


Рисунок 1.64. Вікно для створення сховища

Обов’язковим параметром являється введення шляху до нового сховища, і надається вибір для двох типів ключа: key – це ключ у вигляді файлу, password – це варіант з майстер-паролем. Також для варіанту з паролем було додано додатковий шар безпеки у вигляді перевірки складності паролю на рис. 1.65.

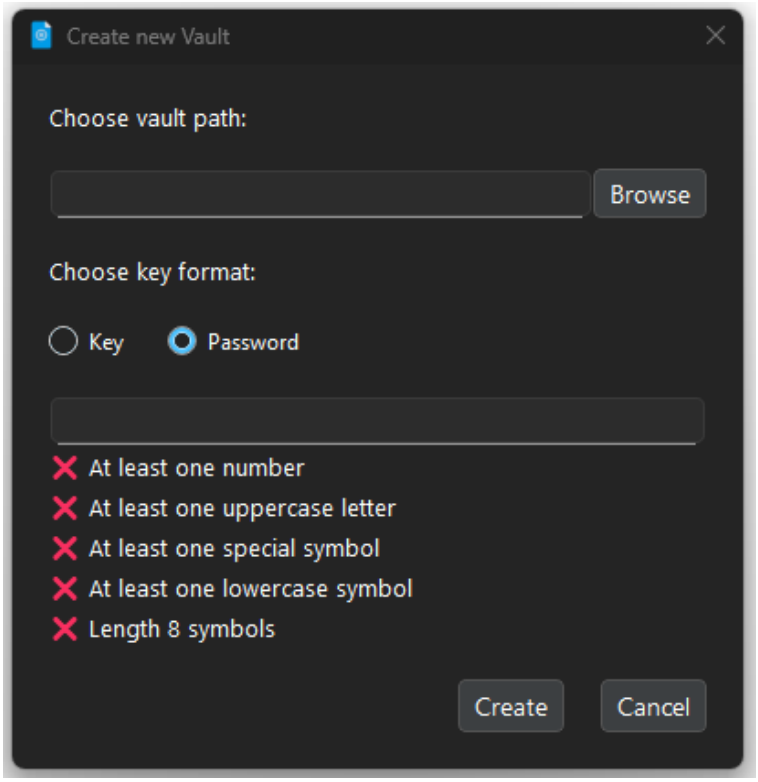


Рисунок 1.65. Перевірка складності майстер-паролю

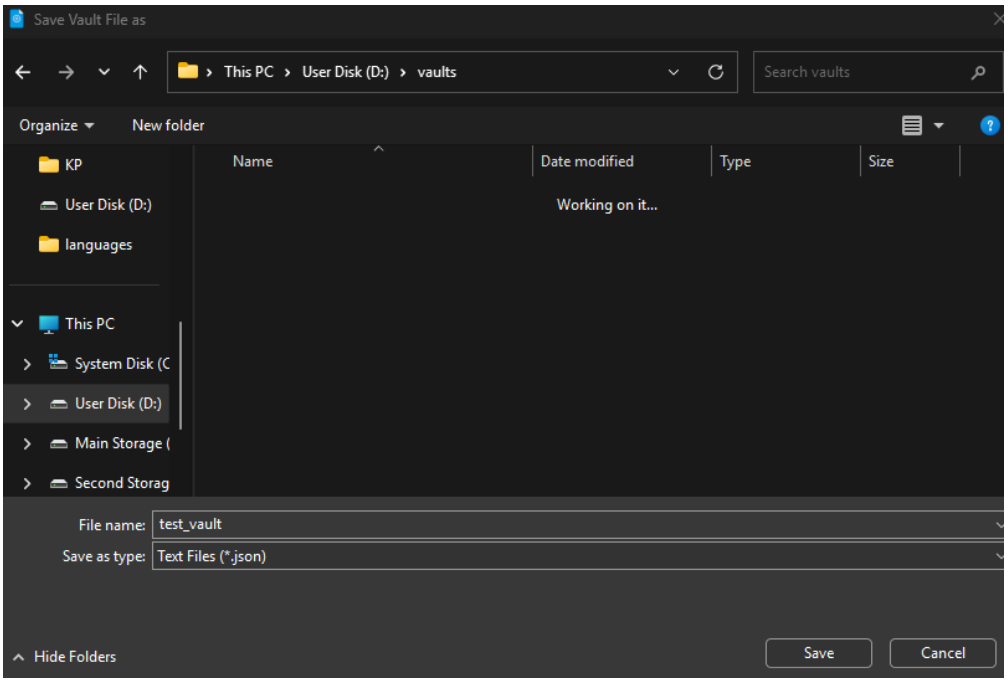


Рисунок 1.66. Провідник зі створенням файлу сховища

Необхідно вказати шлях до нового сховища, для цього використовується функціонал PyQt6 який надає доступ до файлового провідника системи, залишається лише перейти до необхідної директорії та вказати назву сховища (рис. 1.66).

Для тесту було створено сховище з ключем у вигляді файлу, для цього вказується шлях тим самим чином, після чого потрібно натиснути Create, в результаті можна побачити наступні зміни в програмі (рис. 1.67).

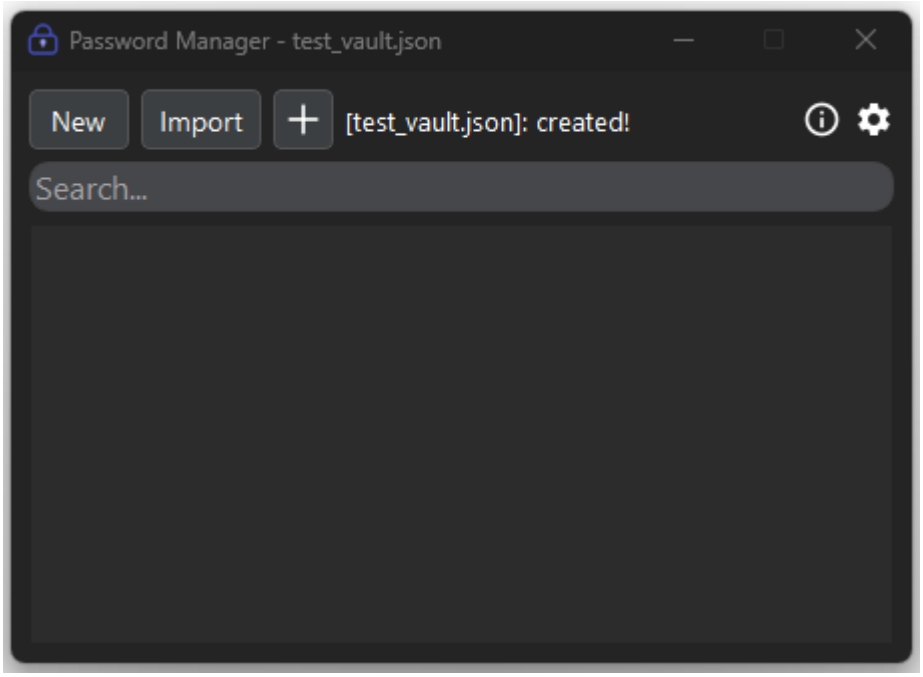


Рисунок 1.67. Вигляд вікна після створення нового сховища

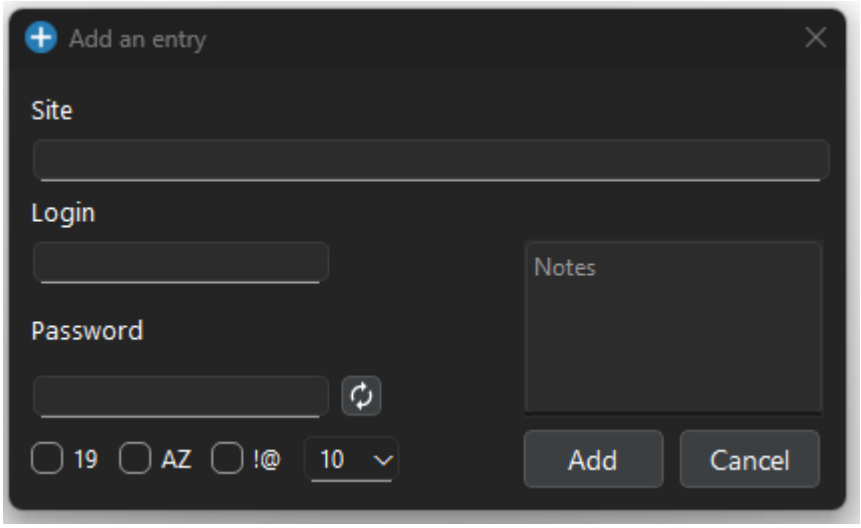


Рисунок 1.68. Вікно додавання нового запису

Назва вікна змінилась, відобразилась назва створеного сховища, був відображений невеличкий інформаційний текст, зник віджет початкового запрошення у центрі та розблокувалася кнопка для додавання нових записів, тепер можна додати новий запис до сховища, для цього треба натиснути на кнопку з плюсиком, відкривається наступне вікно для додавання запису на рис. 1.68.

У цьому вікні користувач вже додає власну інформацію таку як назву ресурсу, логін, пароль та замітки за бажанням, перші три поля вважаються обов'язковими, також видно генератор паролів під полем паролю на рис. 1.69.

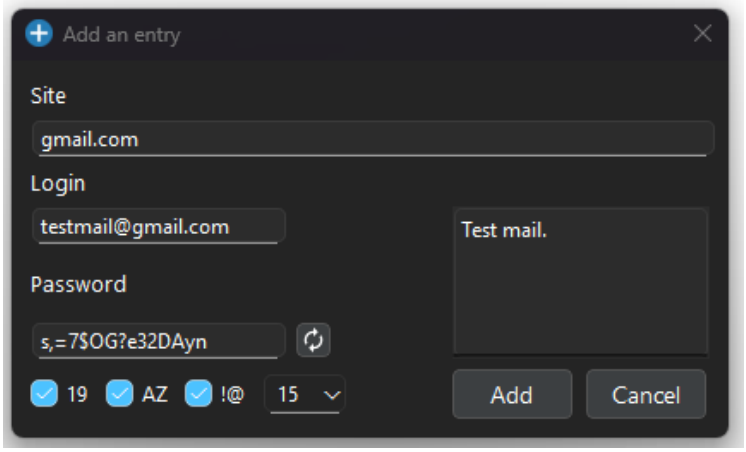


Рисунок 1.69. Заповнене вікно для додавання запису

Генератор паролів доволі гнучкий, можна вказати використання цифр, великих літер, спеціальних символів та звісно довжину пароля. Після чого треба натиснути на кнопку Add, в результаті видно зміни на головному вікні (рис. 1.70)

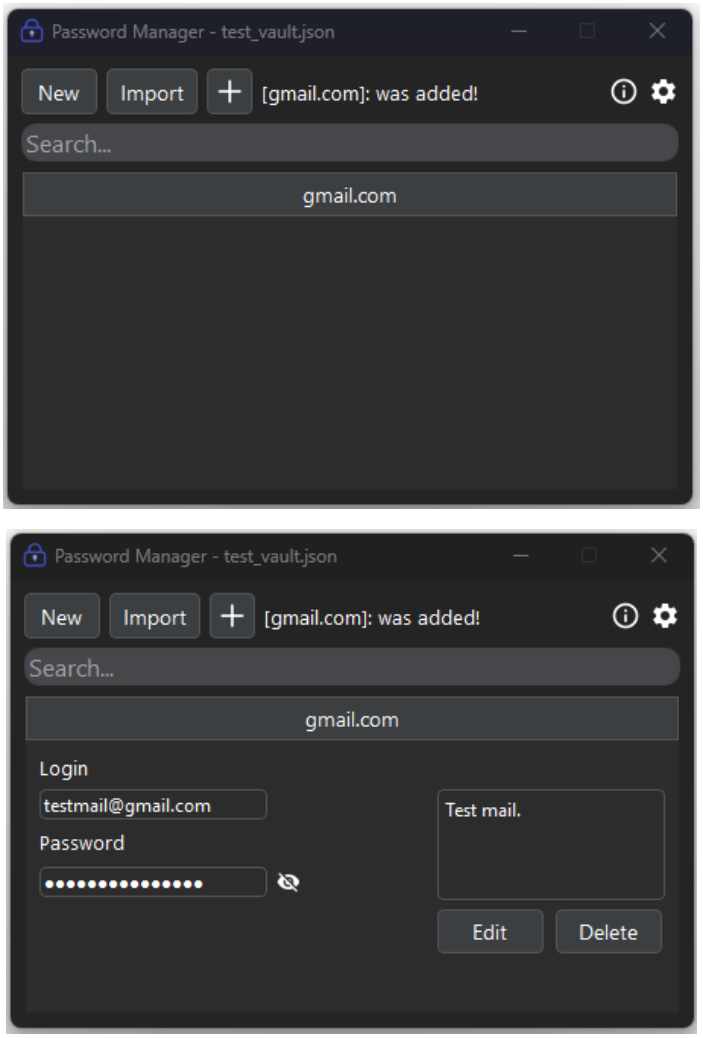


Рисунок 1.70. Вигляд нового запису в прихованому та розкритому вигляді

Було додано ряд допоміжних функцій, таких як перемикач видимості для паролю та можливість копіювати пароль або логін натиском на відповідне поле, для копіювання не потрібно робити пароль видимим (рис. 1.71).

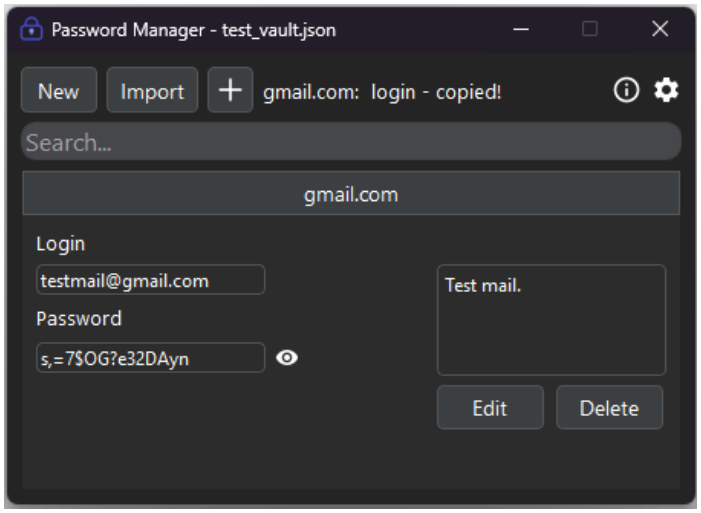


Рисунок 1.71. Функції для копіювання та перемикач видимості для паролю

Для того щоб змінити щойно доданий запис, у розкритому вигляді необхідно натиснути кнопку Edit та відкриється відповідне вікно для редагування запису, що можна побачити на рис. 1.72.

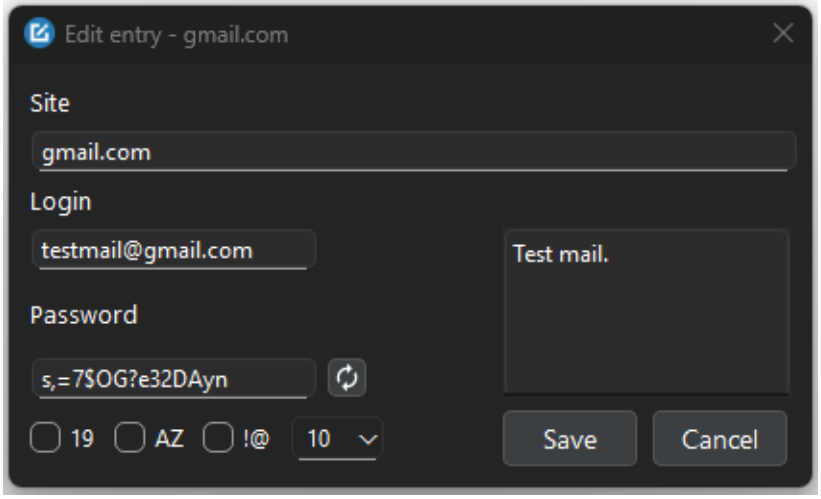


Рисунок 1.72. Вікно для редагування запису

Це вікно схоже на вікно для додавання нових записів, але всі дані запису автоматично введено.

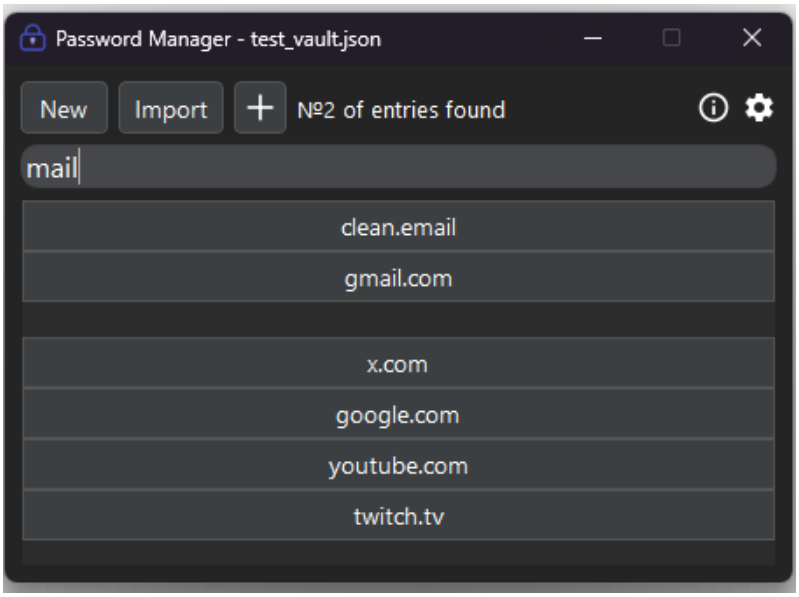


Рисунок 1.73. Демонстрація пошуку

Для того, щоб наочно продемонструвати ефективність пошуку, було додано більше записів. Ця функція особливо корисна, коли база даних паролів стає великою. Для того щоб знайти необхідний запис, користувачеві достатньо почати вводити назву ресурсу в поле пошуку. Алгоритм в режимі реального часу аналізує введення, і найбільш підходящі, релевантні результати будуть динамічно

переміщуватися нагору списку, навіть якщо у запиті є друкарська помилка, цей процес можна побачити на рис. 1.73.

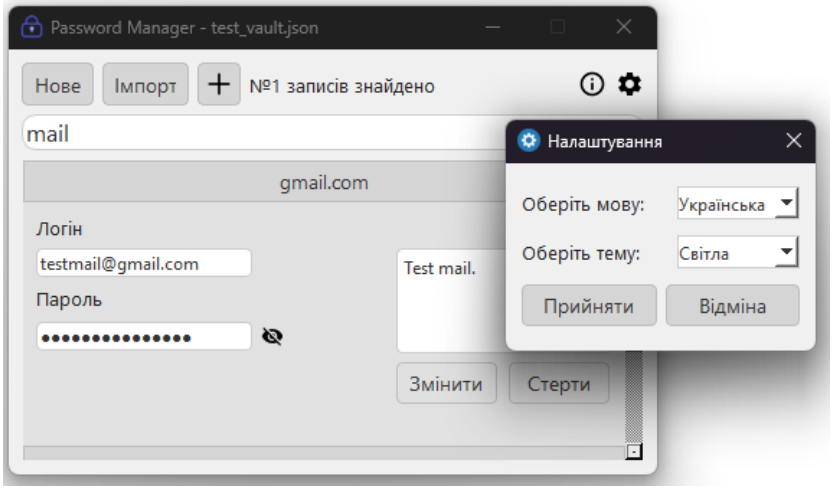


Рисунок 1.74. Зміна налаштувань мови та теми

Для динамічної зміни налаштувань потрібно натиснути на кнопку з шестернею, відкриється невеличке вікно на рис. 1.74, де можна змінити налаштування мови та теми застосунку, підтримується англійська/українська мови та світла/темна теми.

Було додано невеличку довідку, щоб її побачити потрібно натиснути на кнопку з інформаційною іконкою, з’явиться повідомлення (рис. 1.75).

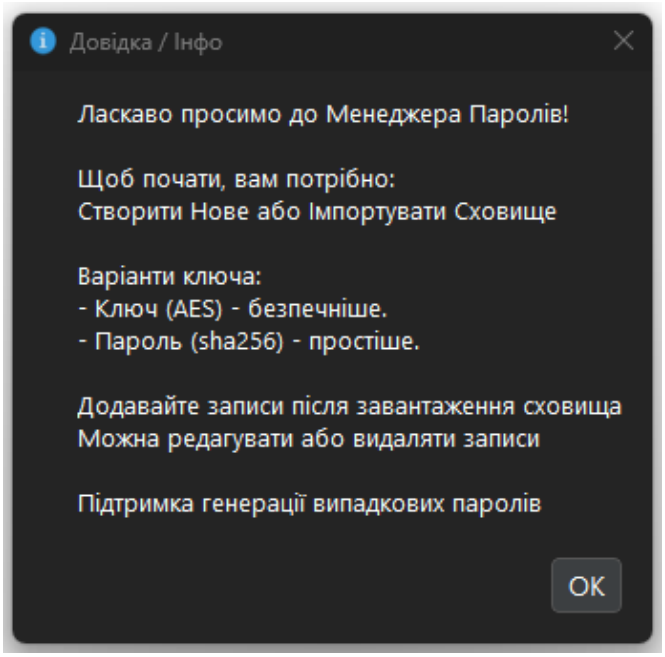


Рисунок 1.75. Вікно з довідкою

Загалом алгоритм взаємодії користувача з додатком на першому запуску для створення нового сховища можна побачити на рис. 1.76.

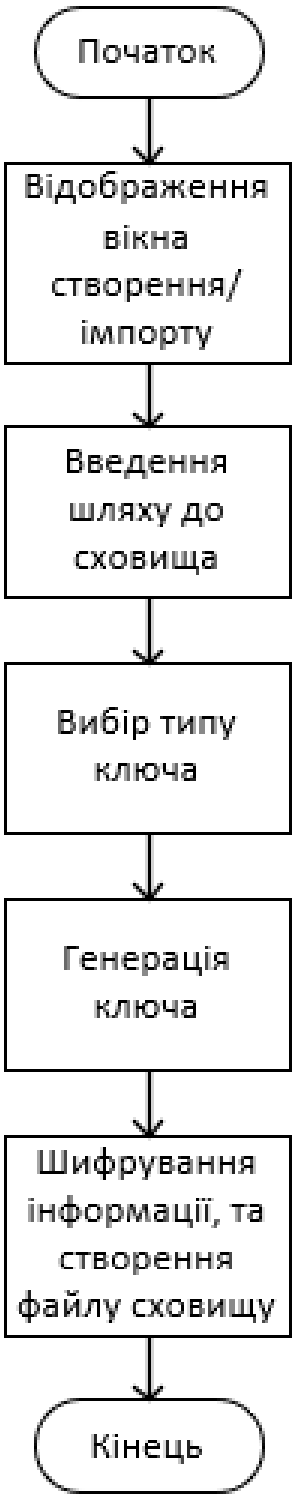


Рисунок 1.76. Блок-схема алгоритму створення нового сховища

Якщо сховище вже було створено раніше, то під час наступних запусків логіка роботи програми буде трохи відрізнятися. Замість екрана привітання з пропозицією створити або імпортувати сховище, програма одразу відобразить вікно для автентифікації. Це зроблено для зручності, щоб користувач міг негайно перейти до розблокування своїх даних за допомогою майстер-пароля (рис. 1.77).

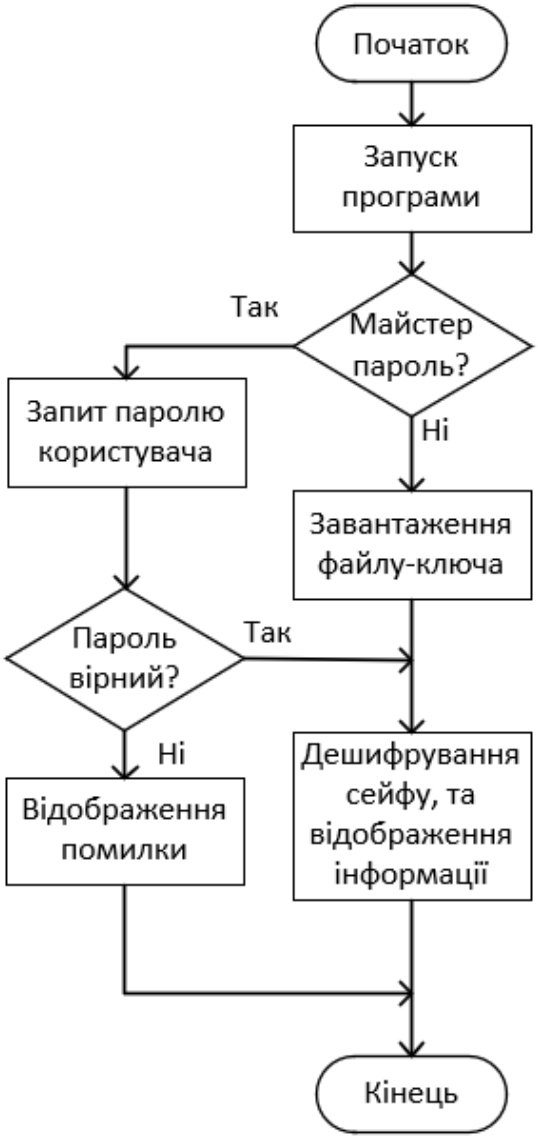


Рисунок 1.77. Блок-схема алгоритму взаємодії зі створеним сховищем

Було створено менеджер паролів з графічним інтерфейсом, який дозволяє створювати/імпортувати сховище з важливими даними користувача які шифруються/дешифруються алгоритмом AES-256. Було реалізовано два типи ключа, у вигляді файлу та у вигляді майстер-паролю (з якого виводиться ключ на 256 біт). Було додано нечіткий пошук. Динамічну зміну мови між англійською та українською, динамічну зміну теми між темною та світлою. З огляду на вимоги до роботи та розроблений застосунок можна заключити, що всі зумовлені функції та системи були успішно реалізовані.

2 РОЗДІЛ ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

Сучасний прогрес у технічному та технологічному секторах виробництва вимагає постійної автоматизації та оптимізації виробничих процесів. У зв'язку з широким застосуванням комп’ютерів працівниками для виконання своїх завдань, законодавство України чітко регулює вимоги та стандарти щодо використання комп’ютерної техніки на підприємстві, зокрема стосовно безпеки праці під час роботи з комп’ютером.

Питання охорони праці працівника необхідно вирішувати на всіх стадіях трудового процесу незалежно від виду професійної діяльності.

У кваліфікаційній роботі у розділі «охорона праці» розглянуті основні небезпечні та шкідливі виробничі фактори що можуть мати місце під час аналізу методів для генерації та керування паролями з використанням AES, та розробці застосунку – менеджера паролів на Windows 11 з використанням мови програмування Python, розробка заходів забезпечення безпечної роботи програміста.

2.1 Аналіз небезпечних та шкідливих факторів, що впливають на програміста під час розробки додатку

До основних критеріїв забезпечення гігієни робочого середовища належать інтенсивність освітлення, температура повітря, вологість, рівень шумового забруднення, ступінь вібраційного впливу, токсичність, загазованість, а також обмеження загальної м'язової активності (гіподинамія). Крім цього, враховується дія електростатичного поля та вплив як неіонізуючих, так і іонізуючих електромагнітних випромінювань.

2.2 Виробниче приміщення

Для приміщень з персональними комп'ютерами встановлено чіткі норми: мінімальна площа на одне робоче місце — 6,0 м², об'єм — 20,0 м³. Облаштування таких робочих місць у підвалах чи на цокольних поверхах заборонено. Оздоблювальні матеріали не повинні виділяти шкідливих речовин, а покриття підлоги має бути матовим, неслизьким та антистатичним. З міркувань безпеки

заземлені конструкції мають бути ізольовані. Приміщення також оснащується аптечкою, автоматичною пожежною сигналізацією, вогнегасниками та потребує щоденного вологого прибирання. Для стін рекомендовано використовувати світлі пастельні тони.

2.3 Вимоги до робочого середовища під час роботи з ПК

До причин, які зумовлюють виникнення професійних захворювань користувачів ВДТ, значне місце мають умови праці. Головними з них є ті, що створюються під впливом випромінювання з ВДТ, освітлювання, шуму, вмісту шкідливих речовин у повітрі робочої зони, іонного складу повітря, електростатичного поля.

2.4 Організація робочого місця с ПК

Організація робочого місця передбачає правильне розміщення меблів, обладнання та врахування особливостей праці. Робочі місця розташовуються на відстані не менше 1 м від стіни з вікном, з проходами між рядами не менше 1 м. Відстань між бічними поверхнями комп'ютерів має бути не меншою за 1,2 м, а між тильною стороною одного монітора та екраном іншого – не менше 2,5 м.

Конструкція меблів має забезпечувати оптимальну робочу позу (рис. 2.1). Робочий стіл повинен мати достатню площу та висоту в межах 680–800 мм. Ключовим елементом є підйомно-поворотне крісло, що регулюється за висотою, кутом нахилу сидіння та спинки, а також наявність підставки для ніг з можливістю регулювання.

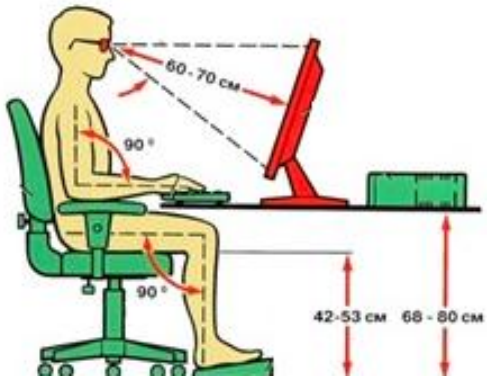


Рисунок 2.1. Конструкція робочого місця, норми робочого простору

2.4.1 Освітлення робочого місця

Для забезпечення належного освітлення приміщення, де працює програміст, застосовується комбінована система, що поєднує природне освітлення із додатковим штучним світлом. Загальне оздоблення простору виконується за допомогою газорозрядних ламп типу ЛД. Згідно з встановленими нормами, для робочого місця, на якому здійснюються високоточні операції (де мінімальний розмір об'єкта розрізнення становить 0,3–0,5 мм), необхідна освітленість рівномірно має досягати 300 лк. В цілому, ці вимоги щодо освітлення забезпечені.

2.4.2 Мікроклімат

Приміщення для роботи з персональним комп'ютером мають бути обладнані системами опалення, кондиціонування повітря, або припливно-витяжною вентиляцією. У приміщеннях на робочих місцях мають забезпечуватися оптимальні значення параметрів мікроклімату: температури, відносної вологості й рухливості повітря у відповідності до ГОСТ 12.1.005-88, СН 4088-86. Згідно санітарних норм мікроклімату виробничих приміщень ДСН 3.3.6.042 99 нормування параметрів проводиться в залежності від періоду року та категорії важкості виконуваних робіт.

Робота оператора ПК за енерговитратами відноситься до категорії легких робіт Іа, Іб. У таблиці 2.1. наведені оптимальні параметри мікроклімату в приміщеннях, де виконуються роботи операторського типу.

Таблиця 2.1. Параметри мікроклімату для приміщень з ПК

Період року	Параметр мікроклімату	Величина
Холодний	Температура повітря в приміщенні; відносна вологість; швидкість руху повітря	22...24°С; 40... 60%; до 0,1 м/с
Теплий	Температура повітря в приміщенні; відносна вологість; швидкість руху повітря	23...25 °С; 40...60%; 0,1...0,2 м/с

2.4.3 Шум

Джерелами шуму при роботі з комп'ютерною технікою є жорсткий диск, вентилятор блока живлення мережі, вентилятор розташований на процесорі,

швидкісні CD-ROM, механічні сканери, пересувні механічні частини принтера.

При роботі матричних голкових принтерів шум виникає при переміщенні головки принтеру і в процесі удару голок головки по паперу. При роботі вентиляційної системи ПК, яка забезпечує оптимальний температурний режим електронних блоків ПК і вмонтована в задню панель, створюється аеродинамічний шум. Окрім того діють і інші зовнішні джерела шуму, не пов’язані з роботою ПК.

Для офісів та приміщень, обладнаних персональними комп'ютерами або технікою для бізнесу допустимий рівень шуму цілодобово - 50 дБА, а максимальний- 65 дБА

2.4.4 Електробезпека

Персональні комп’ютери, периферійні пристрої, інше устаткування (апарати управління, контрольно-вимірювальні прилади, світильники), електропроводи та кабелі за виконанням і ступенем захисту відповідають класу зони, мають апаратуру захисту від струму короткого замикання та інших аварійних режимів.

Лінія електромережі для живлення персональних комп’ютерів та периферійних пристроїв виконана як окрема групова трипровідна мережа шляхом прокладання фазового, нульового робочого та нульового захисного провідників. Нульовий захисний провідник використовується для заземлення (занулення) електроприймачів. Персональні комп’ютери і периферійні пристрої повинні підключатися до електромережі тільки за допомогою справних штепсельних з’єднань і електророзеток заводського виготовлення.

При розміщенні в приміщенні до п’яти персональних комп’ютерів і периферійних пристроїв допускається прокладання трипровідникового захищеного проводу або кабелю в оболонці з негорючого чи важкогорючого матеріалу по периметру приміщення без металевих труб та гнучких металевих рукавів. Це виняток з загальних правил, який діє за умови, що сумарна потужність обладнання не створює підвищеного пожежного ризику.

2.5 Пожежна безпека при роботі з ВДТ

Пожежна безпека забезпечується системою запобігання та захисту, що включає обов'язковий план евакуації. Сучасні ПК створюють ризик загоряння через високу щільність елементів, що генерують тепло, здатне розплавити ізоляцію. Джерелами запалювання можуть стати самі ПК, електропроводка та системи кондиціонування.

Для гасіння пожеж на початковій стадії застосовують вуглекислотні вогнегасники, які є безпечними для електронного обладнання. Вогнегасники розташовують на видних місцях на висоті до 1,5 м, на кронштейнах або в пожежних шафах. Інструкції мають бути добре видно, а механізми запуску — опломбовані.

Рисунок 2.2. Засоби пожежогасіння

Виробничі приміщення мають запасні виходи. Двері повинні мати освітлений надпис «Запасний вихід». План евакуації вивішується на видному місці у основного виходу із приміщення.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було розроблено програмний застосунок для безпечного керування паролями. Основним елементом реалізованої системи є криптографічне ядро на основі симетричного алгоритму шифрування AES-256, яке використовується для надійного захисту бази даних користувача. Завдяки цьому підходу здійснюється повна інкапсуляція чутливої інформації у локальному файлі-сховищі, що унеможливлює несанкціонований доступ з боку третіх осіб та забезпечує повний контроль користувача над власними даними.

Було створено десктопний застосунок з використанням мови Python та фреймворку PyQt6, що забезпечує інтуїтивний графічний інтерфейс. Користувач має можливість створювати та імпортувати зашифровані сховища, автентифікація в яких відбувається за допомогою майстер-пароля або файлу-ключа. Система включає генератор паролів, здатний створювати комбінації з ентропією понад 120 біт, а також інтелектуальний пошук з допущеннями, що забезпечує точність знаходження до 95% навіть за наявності однієї-двох помилок у запиті.

Проведене тестування продемонструвало високу ефективність обраних рішень. Виявлено, що використання алгоритму AES-256 з апаратною підтримкою практично не впливає на продуктивність системи: час на повне розшифрування сховища об’ємом до 100 записів займає в районі 0.1 мілісекунд на сучасних системах. Такий підхід дозволяє поєднати максимальний рівень безпеки з високою швидкістю відгуку інтерфейсу, що підвищує загальну зручність користування та мінімізує затримки.

Отримані результати підтвердили практичну доцільність і надійність реалізованого локального підходу до зберігання паролів. Розроблене програмне рішення не лише демонструє високий рівень захисту даних завдяки відкритому коду та відсутності залежності від сторонніх серверів, але й відкриває перспективи для подальшого розширення функціоналу, як-от інтеграція двофакторної автентифікації чи синхронізація між пристроями через захищені P2P-канали.

ПЕРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Коваленко А.В. Основи сучасної криптографії та інформаційної безпеки. – Київ: Наукова думка, 2019. – 350 с.
2. Мельник І.С. Симетричні алгоритми шифрування: аналіз та застосування стандарту AES. – Львів: Видавництво Львівської політехніки, 2020. – 288 с.
3. Петренко В.О., Захарчук Т.І. Керування паролями та автентифікаційними даними в сучасних системах. – Харків: Техніка, 2021. – 312 с.
4. Бондаренко Є.П. Практична інформаційна безпека з використанням Python. – Київ: КПІ ім. Ігоря Сікорського, 2022. – 410 с.
5. Ткаченко С.М. Хеш-функції та їх застосування в криптографічних протоколах. – Одеса: Одеський національний університет, 2018. – 275 с.
6. Шевченко О.І. Захист даних у комп'ютерних системах та мережах. – Дніпро: Прогрес, 2019. – 360 с.
7. Гриценко Р.В. Розробка кросплатформенних додатків з графічним інтерфейсом на Python. – Харків: Комп’ютерна книга, 2020. – 298 с.
8. Поліщук Л.В. Генерація псевдовипадкових послідовностей у криптографії. – Львів: Світ, 2021. – 250 с.
9. Сидоренко П.М. Аналіз вразливостей програмних застосунків та методи їх захисту. – Київ: Інформаційні системи, 2022. – 320 с.
10. Міністерство цифрової трансформації України. Рекомендації щодо створення та керування надійними паролями [Електронний ресурс]. – Режим доступу: https://thedigital.gov.ua/passwords (Дата звернення: 20.04.2025).
11. Державна служба спеціального зв'язку та захисту інформації України. Стандарт шифрування AES-256 та його застосування [Електронний ресурс]. – Режим доступу: https://cip.gov.ua/aes-standard (Дата звернення: 15.05.2025).
12. Національний координаційний центр кібербезпеки. Безпека локальних застосунків для зберігання даних [Електронний ресурс]. – Режим доступу: https://ncscc.gov.ua/local-app-security (Дата звернення: 05.05.2025).
13. Cryptography Documentation: AES [Електронний ресурс]. – Режим доступу: https://cryptography.io/(Дата звернення: 15.05.2025).

ДОДАТОК А. Фрагмент коду модуля vault.py на мові Python додатку для керування сховищем паролів

```
import orjson
import os
import uuid
import base64
import time
from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC
from cryptography.hazmat.primitives import hashes, padding
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from utilities.settings import load_settings
from typing import TypedDict, Optional, Literal, Dict, cast
class VaultEntry(TypedDict):
    site: str
    login: str
    password: str
    notes: str
class EncodedVault(TypedDict):
    key_type: Literal['password', 'key']
    salt: str
    nonce: str
    data: str
Vault = Dict[str, VaultEntry]
def get_vault_path() -> Optional[str]:
    base_path = load_settings(type='ESSENTIALS', parameter='base_path')
    if not base_path:
        return None
    return base_path
def get_key_path() -> Optional[str]:
    key_path = load_settings(type='ESSENTIALS', parameter='key_path')
    if not key_path:
        return None
    return key_path
def vault_exists(path: Optional[str]) -> bool:
    return path is not None and os.path.isfile(path)
def key_exists(path: Optional[str]) -> bool:
    return path is not None and os.path.isfile(path)
class VaultManager():
    def __init__(self) -> None:
        self.base_path: Optional[str] = get_vault_path()
        self.key_path: Optional[str] = get_key_path()
        self.key = None
        self.entries: Optional[Vault] = None
        self.key_type: Optional[Literal['password', 'key']] = None
        self.salt: Optional[bytes] = None
        self.nonce: Optional[bytes] = None
        self.invalid_check = False
        if vault_exists(self.base_path):
            self.read_essentials()
        if self.key_type == 'key' and key_exists(self.key_path):
            self.read_key()
            self.read_vault()
    def create_key(self, key_path: str):
```

```
    with open(key_path, 'wb') as file:
        file.write(os.urandom(32))
    self.read_key()
def read_key(self) -> Optional[bytes | str]:
    if not self.key_path:
        self.invalid_check = True
        return 'NoKey'
    with open(self.key_path, 'rb') as file:
        out = file.read()
    if not out:
        self.invalid_check = True
        return 'Invalid'
    self.invalid_check = False
    self.key = out
def generate_salt(self):
    self.salt = os.urandom(16)
def derive_key_from_password(self, password: str):
    if self.salt is None:
        return
    kdf = PBKDF2HMAC(
        algorithm=hashes.SHA256(),
        length=32,
        salt=self.salt,
        iterations=1_500_000,
        backend=default_backend()
    )
    encoding_key = kdf.derive(password.encode("utf-8"))
    self.key = encoding_key
    return self.key
def _decode_data(self, encoded_data: bytes) -> Optional[Vault | str]:
    if not self.key_type or not self.salt or not self.nonce:
        return {}
    if self.invalid_check and self.key_type == 'key':
        return 'Invalid'
    if self.key is None:
        return {}
    # Decoding time measurement
    # Uncomment to see decoding time in terminal
    # start_time = time.perf_counter()
    cipher = Cipher(
        algorithm=algorithms.AES(self.key),
        mode=modes.CBC(self.nonce),
        backend=default_backend()
    )
    decryptor = cipher.decryptor()
    padded_data = decryptor.update(encoded_data) + decryptor.finalize()
    unpadder = padding.PKCS7(128).unpadder()
    try:
        decoded_data = unpadder.update(padded_data) + unpadder.finalize()
    except ValueError:
        self.invalid_check = True
        return 'Invalid'
    if decoded_data is not None:
        self.invalid_check = False
        # Decoding time measurement
        # Uncomment to see decoding time in terminal
```

```
        # end_time = time.perf_counter()
        # elapsed_time_ms = (end_time - start_time) * 1000
        # print(f"Decoding time: {elapsed_time_ms:.4f} ms")
        return orjson.loads(decoded_data)
    return {}
def _encode_data(self) -> Optional[bytes | str]:
    if not self.key_type or not self.salt or not self.nonce:
        return None
    if self.entries is None:
        return None
    if self.key is None:
        return None
    # Encoding time measurement
    # Uncomment to see encoding time in terminal
    # start_time = time.perf_counter()
    cipher = Cipher(
        algorithm=algorithms.AES(self.key),
        mode=modes.CBC(self.nonce),
        backend=default_backend()
    )
    prepared_data = orjson.dumps(self.entries)
    padder = padding.PKCS7(128).padder()
    padded_data = padder.update(prepared_data) + padder.finalize()
    encryptor = cipher.encryptor()
    try:
        encrypted_data = encryptor.update(padded_data) + encryptor.finalize()
    except ValueError:
        return 'Invalid'
    # Encoding time measurement
    # Uncomment to see encoding time in terminal
    # end_time = time.perf_counter()
    # elapsed_time_ms = (end_time - start_time) * 1000
    # print(f"Encoding time: {elapsed_time_ms:.4f} ms")
    return encrypted_data
def create_vault(self, key_type: Literal['password', 'key'], vault_path: str) -> bool | str:
    if not vault_path:
        return False
    self.base_path = vault_path
    self.key_type = key_type
    self.nonce = os.urandom(16)
    self.entries = {}
    return self.write_to_vault()
def read_essentials(self):
    if not self.base_path or not vault_exists(self.base_path):
        return False
    with open(self.base_path, 'rb') as file:
        raw_data = file.read()
        encoded_vault: EncodedVault = orjson.loads(raw_data)
    self.key_type = encoded_vault['key_type']
    self.salt = base64.b64decode(encoded_vault['salt'])
    self.nonce = base64.b64decode(encoded_vault['nonce'])
def read_vault(self) -> bool | str:
    if not self.base_path or not vault_exists(self.base_path):
         return False
    with open(self.base_path, 'rb') as file:
        raw_data = file.read()
```

```
            encoded_vault: EncodedVault = orjson.loads(raw_data)
        encrypted_data = base64.b64decode(encoded_vault['data'])
        if not self.key_type or not self.salt or not self.nonce:
            self.entries = None
            return False
        if encrypted_data is None:
            self.entries = {}
        else:
            data = self._decode_data(encrypted_data)
        if data == 'Invalid':
            self.entries = None
            self.invalid_check = True
            return 'Invalid'
        else:
            self.entries = cast(Vault, data)
        self.invalid_check = False
        return True
    def write_to_vault(self) -> bool | str:
        if self.base_path is None:
            return False
        if self.entries is None:
            return False
        if not self.key_type or not self.salt or not self.nonce:
            return False
        data = self._encode_data()
        if data == 'Invalid':
            return 'Invalid'
        if data is None:
            return False
        user_data = cast(bytes, data)
        encoded_vault = {
            'key_type': self.key_type,
            'salt': base64.b64encode(self.salt).decode('utf-8'),
            'nonce': base64.b64encode(self.nonce).decode('utf-8'),
            'data': base64.b64encode(user_data).decode('utf-8')
        }
        try:
            with open(self.base_path, 'wb') as file:
                file.write(orjson.dumps(encoded_vault))
            return True
        except (OSError, IOError):
            return False
    def add_entry(self, new_entry: VaultEntry) -> Optional[str]:
        if self.entries is None or not vault_exists(self.base_path):
            return None
        if self.key_type == 'key' and not key_exists(self.key_path):
            return None
        if new_entry.get('site') is None: return
        if new_entry.get('login') is None: return
        if new_entry.get('password') is None: return
        if new_entry.get('notes') is None: return
        new_id = str(uuid.uuid4())
        self.entries[new_id] = new_entry
        return new_id
    def update_entry(self, id: str, new_entry: VaultEntry) -> bool:
        if self.entries is None: return False
```

```
    if new_entry.get('site') is None: return False
    if new_entry.get('login') is None: return False
    if new_entry.get('password') is None: return False
    if new_entry.get('notes') is None: return False
    if self.entries.get(id) is not None:
        self.entries[id] = new_entry
        return True
    return False
def delete_entry(self, entry_id: str) -> bool:
    if self.entries is None:
        return False
    del self.entries[entry_id]
    return True
def get_entry_by_id(self, entry_id: str) -> Optional[VaultEntry]:
    if self.entries is None:
        return None
    entry = self.entries.get(entry_id)
    if entry is not None:
        return entry.copy()
    return None
def get_all_entries(self) -> Optional[Vault]:
    if self.entries is None:
        return None
    return self.entries.copy()
```

ДОДАТОК Б. Слайди мультимедійної презентації

АНАЛІЗ МЕТОДІВ ГЕНЕРАЦІЇ ТА КЕРУВАННЯ ПАРОЛЯМИ З ВИКОРИСТАННЯМ AES

Виконав: *Кодряну П.М., 2БКС-29*
Керівник: *Іванова Л.В.*

ВСП «ОТФК ОНТУ», Одеса, 2025р.

ТИПИ МЕНЕДЖЕРІВ ПАРОЛІВ

Хмарні Менеджери ☁️	**Локальні Менеджери** 💻🔒
+ Доступність, синхронізація	+ Повний контроль, безпека
- Залежність, довіра	- Самост. синхронізація, ризик втрати

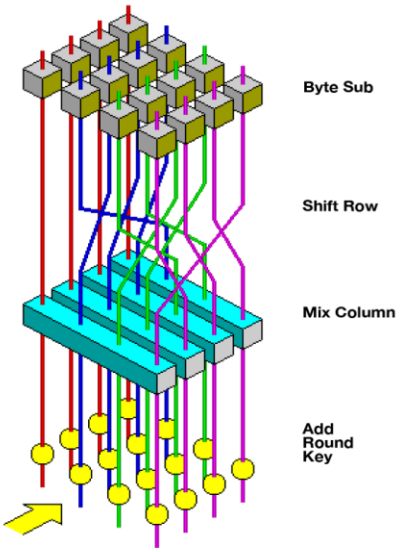
АЛГОРИТМ ШИФРУВАННЯ AES-256

🛡️ **AES** (Advanced Encryption Standard)

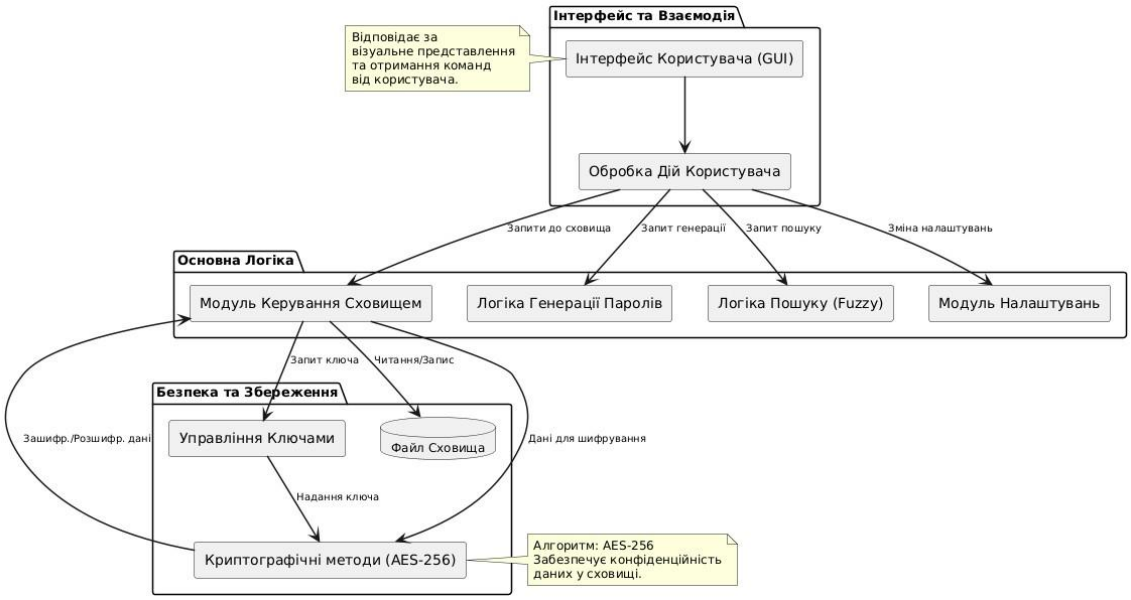
- Світовий стандарт шифрування;
- Висока стійкість.

🔑 Чому **AES-256?**

- Ключ **256** біт.
- Практично **неможливо** зламати.
- Максимальний рівень **безпеки**.



АРХІТЕКТУРА ДОДАТКУ



ІНСТРУМЕНТИДЛЯ РОЗРОБКИ

Мова програмування:

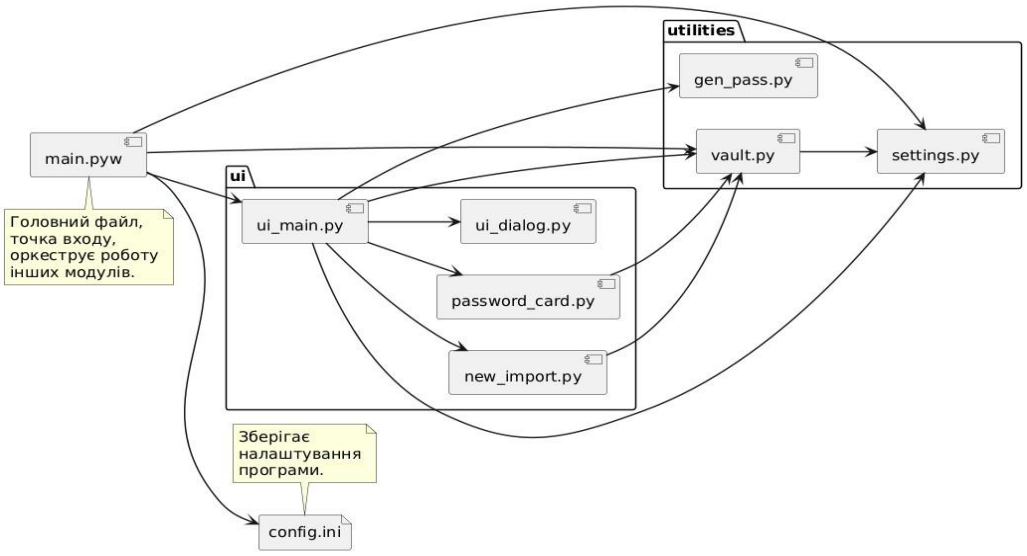
Python

Графічний інтерфейс:

PyQt6

Бібліотеки: cryptography, orjson, secrets, thefuzz.

АЛГОРИТМ ВЗАЄМОДІЇ МОДУЛІВ ПРОГРАМИ



ФУНКЦІЯ ГЕНЕРАЦІЇ ПАРОЛІВ

```python
def generate_password(length: int, digits: bool,
        capital: bool, specials: bool) -> str:

    pool = string.ascii_lowercase
    password = ''

    if digits:
        password += choice(string.digits)
        pool += string.digits
    if capital:
        password += choice(string.ascii_uppercase)
        pool += string.ascii_uppercase
    if specials:
        password += choice(string.punctuation)
        pool += string.punctuation

    pool = list(pool)
    while len(password) != length:
        rand_symbol = choice(pool)
        password += rand_symbol

    password = list(password)
    shuffle(password)
    return ''.join(password)
```

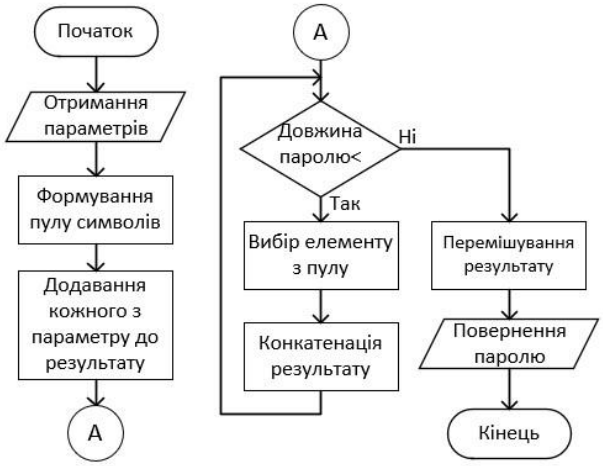
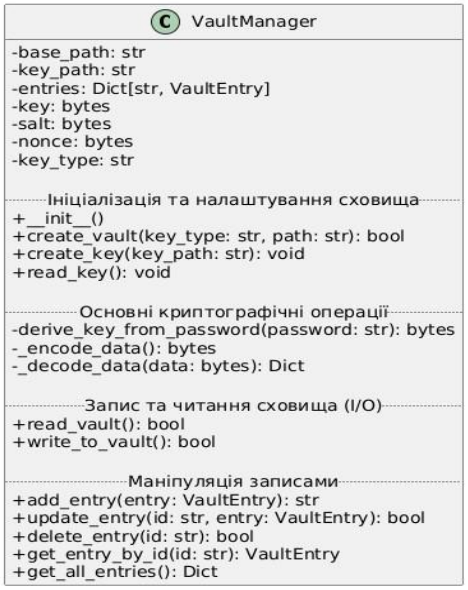


СХЕМА КЛАСУ VaultManager

```python
def __init__(self) -> None:
    self.base_path: Optional[str] = get_vault_path()
    self.key_path: Optional[str] = get_key_path()
    self.key = None
    self.entries: Optional[Vault] = None
    self.key_type: Optional[Literal['password', 'key']] = None
    self.salt: Optional[bytes] = None
    self.nonce: Optional[bytes] = None
    self.invalid_check = False

    if vault_exists(self.base_path):
        self.read_essentials()

    if self.key_type == 'key' and key_exists(self.key_path):
        self.read_key()
        self.read_vault()
```



ВИВЕДЕННЯ КЛЮЧА З МАЙСТЕР-ПАРОЛЮ

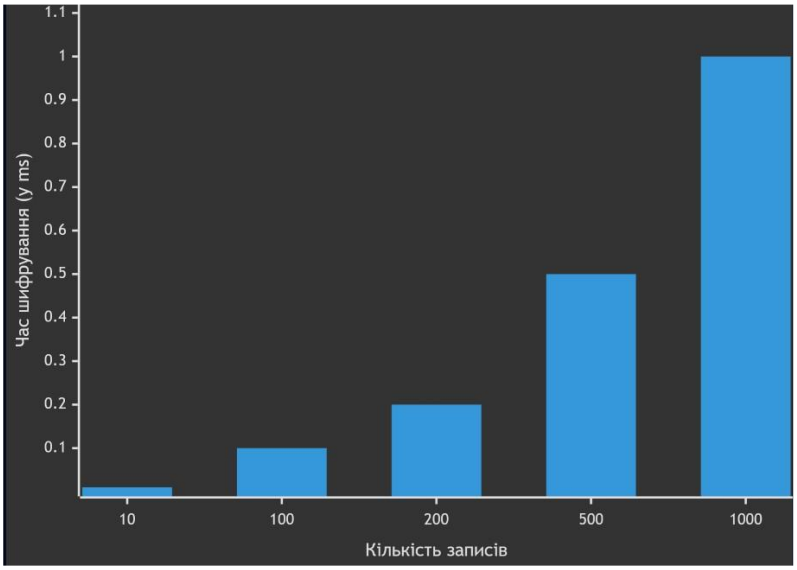
```python
def derive_key_from_password(self, password: str):

    if self.salt is None:
        return

    kdf = PBKDF2HMAC(
        algorithm=hashes.SHA256(),
        length=32,
        salt=self.salt,
        iterations=1_500_000,
        backend=default_backend()
    )

    encoding_key = kdf.derive(password.encode("utf-8"))
    self.key = encoding_key
    return self.key
```

АЛГОРИТМ ВЗАЄМОДІЇ КОРИСТУВАЧА З ДОДАТКОМ

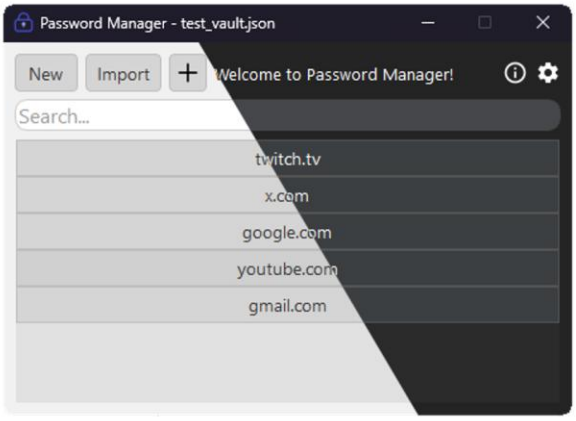
Перший запуск

Наступні запуски

АНАЛІЗ ЕФЕКТИВНОСТІ ШИФРУВАННЯ



ДЕМОНСТРАЦІЯ ЗАСТОСУНКУ



ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

РЕЦЕНЗІЯ

на кваліфікаційну роботу здобувача (здобувачки) освіти

відділення комп'ютерних систем

Кодряну Павла Михайловича

(прізвище, ім`я та по батькові)

Спеціальність *123 Комп'ютерна інженерія*

ОПП *Комп'ютерна інженерія*

Керівник кваліфікаційної роботи *Іванова Лілія Вікторівна*

(прізвище, ім`я та по батькові)

Тема кваліфікаційної роботи *«Аналіз методів генерації та керування паролями з використанням алгоритму AES»*

Обсяг розрахунково-пояснювальної записки *82* сторінок

Обсяг графічної (презентаційної) частини *10* аркушів (слайдів)

ХАРАКТЕРИСТИКА КВАЛІФІКАЦІЙНОЇ РОБОТИ

а) заключення про ступінь відповідності виконаного кваліфікаційної роботи завданню

кваліфікаційна робота у повному обсязі відповідає темі та завданню

б) характеристика виконання кожного розділу кваліфікаційної роботи

Кваліфікаційна робота складається з розділів: 1. Аналітичний огляд існуючих рішень. 2. Аналіз алгоритму AES. 3. Технічне завдання до проєкту. 4. Проєктування та розробка програми. 5. Тестування функціоналу додатку. 6.Охорона праці. 7. Висновки. 8. Список використаних джерел інформації.

в) оцінка якості виконання пояснювальної записки та графічної частини кваліфікаційної роботи

Пояснювальна записка виконана якісно, у достатньому обсязі, відповідно до індивідуального завдання та теми дипломного проекту, розділи пояснювальної записки відповідають етапам рішення завдання, поставленого у кваліфікаційній роботі. Презентація виконана якісно, у достатньому обсязі. Презентація наочно демонструє результати роботи.

г) перелік позитивних якостей кваліфікаційної роботи

1. Актуальна тематика

2. Сучасні технології реалізації програмного продукту

3. Якісне подання результатів роботи

д) основні недоліки кваліфікаційної роботи

Не визначено як додаток буде взаємодіяти з іншими системами, які використовують паролі для входу

З додатку не зрозуміло який тип сховища створюється і як забезпечується його цілісність і безпека

Оцінка розрахункової частини *Відмінно*

Оцінка графічної частини *Відмінно*

Загальна оцінка *Відмінно*

Прізвище, ім'я, по батькові рецензента *к.т.н. Шибаєва Наталя Олегівна*

Місце роботи і посада рецензента

Національний університет «Одеська політехніка», доцент кафедри інформаційних технологій

Підпис:

«___» _____ 2025 р.

Відокремлений структурний підрозділ

Одеський технічний фаховий коледж ОНТУ

ВІДГУК

Керівника про кваліфікаційну роботу бакалавра

Кодряну Павла Михайловича

(прізвище, ім`я та по батькові)

Освітньо-професійна програма *«Комп'ютерна інженерія»*

Спеціальність *123 «Комп'ютерна інженерія»*

Тема кваліфікаційної роботи

«Аналіз методів генерації та керування паролями з використанням алгоритму AES»

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ (РОБОТИ)

а) Обсяг і якість виконання роботи (розрахунково-пояснювальної записки)

Пояснювальна записка виконана якісно, у достатньому обсязі, відповідно до індивідуального завдання та теми дипломного проекту, розділи пояснювальної записки відповідають етапам рішення завдання, поставленого у дипломному проекті

Презентація виконана якісно, у достатньому обсязі. Презентація наочно демонструє результати роботи.

б) Самостійність роботи над кваліфікаційною роботою

Студент самостійно обрала напрям та тематику кваліфікаційної роботи. Провів аналіз існуючих рішень і зробив необхідні висновки для реалізації проекту. Виявив навички самостійно опрацьовувати новий матеріал та виконувати пошук необхідної літератури та інших джерел інформації

в) Теоретична підготовка бакалавра

відповідає вимогам, що надаються до бакалавра зі спеціальності «Комп'ютерна інженерія»

г) Вміння розв`язувати виробничі і конструкторські питання на базі останніх досліджень науки і техніки, передових методів виробництва

У роботі представлено результати розробки програмного застосунку для генерації та керування паролями з використанням AES-256, описаний принцип розробки програми з використанням бібліотек PyQt6, cryptography, thefuzz, secrets, orjson. У аналітичній частині приведене технічне завдання, огляд реалізації популярних комерційних менеджерів паролів, обґрунтування обраних технологій розробки. В технологічній частині описано структуру та алгоритми роботи програми, реалізація застосунку, систем та функціоналу додатку

Загальна оцінка 5(відмінно)

Прізвище, ім`я, по батькові Іванова Лілія Вікторівна

Місце роботи і посада керівника проєкту_ВСП «Одеський технічний фаховий коледж ОНТУ» к.т.н., зав. кафедрою Комп'ютерної інженерії

Підпис

«20» 00 2025р.

ДОЗВІЛ
НА РОЗМІЩЕННЯ
ВИПУСКНОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
В ЕЛЕКТРОННОМУ РЕПОЗИТАРІЇ ВСП «ОТФК ОНТУ»

Ми, що нижче підписалися,

Кодряну П.М.
здобувач освіти гр. 2БКС-29, та

Іванова Л.В.,
керівник дипломного проекту,

не заперечуємо щодо розміщення електронного варіанту пояснювальної записки до випускної кваліфікаційної роботи бакалавра на тему:

«Аналіз методів генерації та керування паролями з використанням алгоритму AES» (автор роботи – Кодряну П.М., керівник роботи – Іванова Л.В.)

виконаного у ВСП «Одеський технічний фаховий коледж Одеського національного технологічного університету» в 2025 році, у повному обсязі в електронному репозитарії ВСП «ОТФК ОНТУ» для вільного доступу через мережу Інтернет.

Несемо відповідальність за ідентичність електронного та друкованого варіантів випускної кваліфікаційної роботи і даємо згоду на обробку персональних даних.

Виконавець _____ / Кодряну П.М. /

Керівник _____ / Іванова Л.В. /

«20» червня 2025 р.

ДОВІДКА

кафедри комп'ютерної інженерії
про допуск до захисту кваліфікаційної роботи
здобувача (здобувачки) освіти II курсу
відділення комп'ютерних систем групи 2БКС-29

Кодряну Павла Михайловича

на тему *Аналіз методів генерації та керування паролями з використанням алгоритму AES*

Висновок відповідальної особи за проведення нормоконтролю:

пояснювальна записка до кваліфікаційної роботи виконана з несуттєвими порушеннями ДСТУ та оформлена відповідно до вимог Положення про дипломне проєктування

(підпис)	*20.06.2025* (дата)	*Петрашова В.І.* (П.І.Б.)

Висновок відповідальної особи за перевірку роботи на наявність академічного плагіату *згідно звіту про перевірку від 11.06.2025 р. значення коефіцієнту подібності в роботі становить 18,83%, коефіцієнт цитування – 0,61%.*

(підпис)	*20.06.2025* (дата)	*Краснокутська К.Г.* (П.І.Б.)

Попередня експертиза (малий захист) кваліфікаційної роботи

здобувача (здобувачки) освіти *Кодряну П.М.* (П.І.Б.)

проведена « *20* » *червня* 2025 р.

Висновки *Пояснювальна записка до кваліфікаційної роботи виконана у повному обсязі. Випускна кваліфікаційна робота відповідає вимогам Положення про дипломне проєктування та рекомендована до захисту.*

Зав. кафедри КІ _____ (підпис) *Іванова Л.В.* (П.І.Б.)

StrikePlagiarism.com

Дата звіту 6/11/2025
Дата редагування 6/11/2025

Документ прийнятий

Звіт подібності

метадані

Назва організації
Odesa Technical Professional College of Odesa National University of Technology
Заголовок
Аналіз методів генерації та керування паролями з використанням алгоритму AES

Автор
Кодряну Павло Михайлович
Науковий керівник / Експерт
Іванова Лілія Вікторівна

підрозділ
Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету"

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

КП 1

КЦ

25	12893	105455
Довжина фрази для коефіцієнта подібності 2	Кількість слів	Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв	ß	1
Інтервали	A→	0
Мікропробіли	·	0
Білі знаки	ß	248
Парафрази (SmartMarks)	a	90

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://card-file.ontu.edu.ua/bitstreams/aed610a6-43ef-47e0-9066-e85c89456f3e/download	121 0.94 %
2	https://card-file.ontu.edu.ua/bitstreams/8999d5af-6274-44f4-ae78-d23e08048d38/download	87 0.67 %
3	https://l.lekciya.com.ua/informatika/1992/index.html	83 0.64 %
4	https://card-file.ontu.edu.ua/bitstreams/8999d5af-6274-44f4-ae78-d23e08048d38/download	72 0.56 %
5	https://card-file.ontu.edu.ua/bitstreams/035f6436-20b4-4ee6-8e99-bede670e308b/download	71 0.55 %

6	https://card-file.ontu.edu.ua/bitstreams/aed610a6-43ef-47e0-9066-e85c89456f3e/download	67 0.52 %
7	https://card-file.ontu.edu.ua/server/api/core/bitstreams/d5a3d14f-d5cb-460f-9c49-cba3f9d50554/content	66 0.51 %
8	http://4-i-5.ru/text-3/page-3-ref-90317.php	53 0.41 %
9	https://card-file.ontu.edu.ua/bitstreams/8999d5af-6274-44f4-ae78-d23e08048d38/download	50 0.39 %
10	https://card-file.ontu.edu.ua/bitstreams/8999d5af-6274-44f4-ae78-d23e08048d38/download	50 0.39 %

з домашньої бази даних (0.21 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	Створення web-застосунку цифрового помічника з використанням Open AI [redacted] 5/28/2025 Odesa Technical Professional College of Odesa National University of Technology (Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету")	27 (3) 0.21 %

з програми обміну базами даних (0.50 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	Gurinchuk_diplom_mag_Lysytskiy_fkn_2023.pdf [redacted] 12/3/2023 V. N. Karazin Kharkiv National University (KKNU) (ННІ комп'ютерних наук та штучного інтелекту - кафедра математичного моделювання та аналізу даних)	29 (3) 0.22 %
2	abis_2018b_024 [redacted] 8/20/2024 O.M.Beketov National University of Urban Economy in Kharkiv (O.M.Beketov National University of Urban Economy in Kharkiv)	15 (1) 0.12 %
3	+ШАПОВАЛОВА LanaShape@gmail.com ВАРЕНІКОВ МП 2024 [redacted] 12/3/2024 National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute (IATE, К-ра цифрових технології в енергетиці)	11 (1) 0.09 %
4	Методи запобігання кіберзагрозам та супутниковим конфліктам у космічному просторі [redacted] 12/11/2023 State University of Telecommunications (ННІТ)	9 (1) 0.07 %

з Інтернету (18.13 %)

ПОРЯДКОВИЙ НОМЕР	ДЖЕРЕЛО URL	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	**https://card-file.ontu.edu.ua/bitstreams/8999d5af-6274-44f4-ae78-d23e08048d38/download**	**310 (7) 2.40 %**
2	**https://card-file.ontu.edu.ua/bitstreams/aed610a6-43ef-47e0-9066-e85c89456f3e/download**	**300 (6) 2.33 %**
3	https://card-file.ontu.edu.ua/bitstreams/3c34e9ea-f695-4fe4-82de-a5c9314f822a/download	266 (24) 2.06 %
4	**https://card-file.ontu.edu.ua/bitstreams/9908b7a9-6b3e-46f5-a46e-84d83787cfd4/download**	**181 (18) 1.40 %**
5	**https://card-file.ontu.edu.ua/bitstreams/361286d7-8a03-4221-ad05-db5133ab5f79/download**	**153 (8) 1.19 %**
6	https://dspace.nuft.edu.ua/bitstreams/88d0c800-bbde-4201-b59a-ae746f073c39/download	139 (19) 1.08 %
7	**https://otipb.at.ua/load/okhorona_praci_v_ofisi/24-1-0-3433**	**112 (4) 0.87 %**
8	**https://dnipr.kyivcity.gov.ua/files/2014/11/13/ohoronapr.pdf**	**90 (6) 0.70 %**

9	https://card-file.ontu.edu.ua/bitstreams/035f6436-20b4-4ee6-8e99-bede670e308b/download	88 (2) 0.68 %
10	https://cpo.stu.cn.ua/Oksana/dipl_bak/140.html	85 (4) 0.66 %
11	https://l.lekciya.com.ua/informatika/1992/index.html	83 (1) 0.64 %
12	https://card-file.ontu.edu.ua/server/api/core/bitstreams/d5a3d14f-d5cb-460f-9c49-cba3f9d50554/content	66 (1) 0.51 %
13	https://card-file.ontu.edu.ua/server/api/core/bitstreams/44c16132-5f53-48e2-b6c0-61e9a2f0fd75/content	58 (2) 0.45 %
14	http://4-i-5.ru/text-3/page-3-ref-90317.php	53 (1) 0.41 %
15	https://www.ses.lviv.ua/novyny/2021/hruden/bezpechni-umovy-pratsi-zaporuka-komfortnoyi-pratsezdatnosti-ofisnykh-pratsivnykiv	50 (2) 0.39 %
16	https://card-file.ontu.edu.ua/server/api/core/bitstreams/e86ba9fc-9135-43bb-922a-10bf0bce46b5/content	43 (1) 0.33 %
17	https://gc.ua/uk/oxorona-praci-v-ofisi-vimogi-do-robochogo-miscya-ofisnogo-pracivnika/	36 (1) 0.28 %
18	https://card-file.ontu.edu.ua/bitstreams/6046cc0e-a398-49f5-a01d-87d33230039a/download	29 (2) 0.22 %
19	http://repository.kpi.kharkov.ua/bitstream/KhPI-Press/37199/1/Book_2018_Berezutskyi_Osnovy_prof_bezpeky.pdf	26 (1) 0.20 %
20	https://www.wikiwand.com/uk/articles/Windows_11	23 (2) 0.18 %
21	https://ela.kpi.ua/bitstream/123456789/57660/1/Zevalov_bakalavr.pdf	22 (2) 0.17 %
22	http://91.198.10.12/bitstream/lib/36859/1/mag2021_Salamandra.V.I._%D0%A1%D0%9D%D0%BC-61.pdf	22 (2) 0.17 %
23	https://studfile.net/preview/5150013/	20 (2) 0.16 %
24	https://ela.kpi.ua/bitstream/123456789/34502/1/Probotiuk_bakalavr.pdf	17 (3) 0.13 %
25	https://www.yumpu.com/xx/document/view/36610022/-/215	9 (1) 0.07 %
26	https://card-file.ontu.edu.ua/bitstreams/549ee9fe-7574-4ae5-b500-9fe2711f33e6/download	8 (1) 0.06 %
27	http://eir.zp.edu.ua/bitstream/123456789/7728/1/MR_Khlyubko.pdf	8 (1) 0.06 %
28	https://www.lektsii.net/1-136225.html	8 (1) 0.06 %
29	https://nubip.edu.ua/sites/default/files/u284/123_opp_kompyuterna_inzheneriya_bak_2021.pdf	6 (1) 0.05 %
30	https://ttf.teset.sumdu.edu.ua/images/Masters/Zavdannya.doc	6 (1) 0.05 %
31	https://thepresentation.ru/obzh/elektrobezpeka-ta-pozhezhna-bezpeka-pri-robot%D1%96-z-kompyuterom	5 (1) 0.04 %
32	https://card-file.ontu.edu.ua/server/api/core/bitstreams/c63b91ba-d04f-4715-890d-b16277695c7e/content	5 (1) 0.04 %
33	https://card-file.ontu.edu.ua/server/api/core/bitstreams/6e367988-fd72-47a3-ac3d-c5748c863588/content	5 (1) 0.04 %
34	https://card-file.ontu.edu.ua/bitstreams/82a6d375-2b69-4233-b80f-fbfd149b7747/download	5 (1) 0.04 %

Список прийнятих фрагментів (немає прийнятих фрагментів)

ПОРЯДКОВИЙ НОМЕР	ЗМІСТ	КІЛЬКІСТЬ ОДНАКОВИХ СЛІВ (ФРАГМЕНТІВ)