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

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

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр
(назва освітнього ступеня)

на тему: Розробка програмної системи менеджера паролів з використанням методів криптографії

Виконав(ла): студент(ка) 6 курсу, групи СПм-61
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)

Ящук В.О.
(підпис) (прізвище та ініціали)

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

Нормоконтроль
(підпис) (прізвище та ініціали)

Завідувач кафедри
(підпис) (прізвище та ініціали)

Рецензент
(підпис) (прізвище та ініціали)

Тернопіль
2025

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

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

Кафедра програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ _____
(підпис) (прізвище та ініціали)
« » 20___ р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістр
(назва освітнього ступеня)

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

студенту Ящук Вікторії Олександрівні
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка програмної системи менеджера паролів з використанням методів криптографії

Керівник роботи Цуприк Галина Богданівна, доцент кафедри програмної інженерії
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «___» _____ 20___ року № _____

2. Термін подання студентом завершеної роботи _____

3. Вихідні дані до роботи _____

4. Зміст роботи (перелік питань, які потрібно розробити)
Проаналізувати предметну область та сучасні підходи до зберігання конфіденційної інформації,

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	Осухівська Галина Михайлівна	15.12.2025	
Безпека в надзвичайних ситуаціях	Стручок Володимир Сергійович		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			

Студент _____ Ящук В.О.
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Ящук Вікторія Олександрівна. Розробка програмної системи менеджера паролів з використанням методів криптографії. Кваліфікаційна робота освітнього ступеня «магістр». Тернопільський національний технічний університет імені Івана Пулюя, кафедра програмної інженерії, група СПм–61, спеціальність 121 «Інженерія програмного забезпечення». Тернопіль, 2025. Сторінок 96, таблиць 4, рисунків 25, додатків 3, бібліографічних посилань 45.

Метою роботи є розробка програмної системи менеджера паролів, призначеної для безпечного зберігання, обробки та керування конфіденційними даними користувачів із використанням сучасних методів криптографічного захисту.

Об’єктом дослідження є процес створення програмної системи захисту паролів для обробки і автентифікаційних даних користувачів.

Предметом дослідження є методи, моделі та технології розробки програмної системи менеджера паролів з використанням криптографічних алгоритмів шифрування та виведення ключів. Методи дослідження включають: аналіз предметної області, моделювання архітектури програмної системи, проєктування структури бази даних, реалізацію криптографічних механізмів, а також ручне та автоматизоване тестування програмного забезпечення.

У даній роботі продемонстровано повний цикл проєктування та розробки desktop-застосунку менеджера паролів. Програмну систему реалізовано у вигляді автономного desktop-застосунку без використання зовнішніх серверів або хмарних сервісів. Основна бізнес-логіка розроблена мовою Python, графічний інтерфейс користувача реалізовано з використанням Qt-фреймворку. Реалізовано механізми шифрування даних, версіонування записів та автоматичного блокування сховища.

Ключові слова: інженерія програмного забезпечення, менеджер паролів, криптографія, інформаційна безпека, шифрування даних, Python, Qt, AES, desktop-застосунок.

ABSTRACT

Yashchuk Viktoriia Oleksandrivna. Development of a Password Manager Software System Using Cryptographic Methods. Master's qualification thesis. Ternopil Ivan Pulyuj National Technical University, Department of Software Engineering, SPm–61 group, specialty 121 "Software Engineering". Ternopil, 2025. Pages 96, tables 4, figures 25, appendices 3, references 45.

The purpose of this work is to develop a password manager software system intended for the secure storage, processing, and management of users' confidential data using modern cryptographic protection methods.

The object of the study is the process of creating a software password protection system for processing and authenticating user data.

The subject of the study is methods, models, and technologies for developing a password manager software system using cryptographic encryption algorithms and key derivation. Research methods include: analysis of the subject area, modeling of the software system architecture, database structure design, implementation of cryptographic mechanisms, as well as manual and automated software testing.

This paper demonstrates the complete cycle of designing and developing a desktop password manager application. The software system is implemented as a standalone desktop application without the use of external servers or cloud services. The main business logic is developed in Python, and the graphical user interface is implemented using the Qt framework. Data encryption, record versioning, and automatic storage locking mechanisms are implemented.

Keywords: software engineering, password manager, cryptography, information security, data encryption, Python, Qt, AES, desktop application.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ І ТЕРМІНІВ

Менеджер паролів PiePass – програмна система, призначена для безпечного зберігання, обробки та управління обліковими даними користувачів.

Криптографія – галузь науки, що вивчає методи забезпечення конфіденційності, цілісності та автентичності інформації шляхом її перетворення.

Хеш-функція – криптографічна функція, яка перетворює вхідні дані довільної довжини у фіксований за розміром хеш-код.

Шифрування – процес перетворення відкритих даних у зашифрований вигляд з метою захисту інформації від несанкціонованого доступу.

Симетричне шифрування – метод шифрування, при якому для шифрування та дешифрування використовується один і той самий секретний ключ.

AES (Advanced Encryption Standard) – симетричний криптографічний алгоритм шифрування, який широко використовується для захисту даних.

OTP – одноразовий пароль

XSS (Cross Site Scripting) - міжсайтовий скриптинг

ПЗ – програмне забезпечення

Хешування паролів – метод захисту паролів, при якому зберігається не сам пароль, а результат роботи хеш-функції.

База даних – структурована сукупність даних, призначена для зберігання та обробки інформації програмною системою.

Інформаційна безпека – стан захищеності інформації від загроз випадкового або навмисного характеру.

СУТНІСТЬ-АТРИБУТ-ЗНАЧЕННЯ (Entity–attribute–value model (EAV)) – патерн для проектування бази даних.

Таблиця визначень атрибутів може містити такі стовпці: ідентифікатор атрибута, ім’я атрибута, опис, тип даних і стовпці, які допомагають перевірити вхідні дані, наприклад, максимальна довжина рядка та регулярний вираз, набір допустимих значень тощо.

ВСТУП

В умовах стрімкого прогресу інформаційно-комунікаційних технологій та зростаючої кількості онлайн-сервісів питання забезпечення безпеки аутентифікаційних даних користувачів стає надзвичайно актуальним. Збільшення обсягів конфіденційної інформації, що зберігається у цифровому середовищі, а також поширення кіберзагроз висувають високі вимоги до захисту даних. Одним із найбільш ефективних інструментів у цьому контексті є спеціалізоване програмне забезпечення для управління паролями, яке забезпечує надійний захист і спрощує роботу з аутентифікаційними даними.

Даний застосунок є актуальним і корисним, оскільки дає можливість користувачам безпечно зберігати облікові дані, генерувати надійні паролі, виконувати їх перевірку на відповідність вимогам безпеки, який дозволяє централізовано та захищено управляти конфіденційною інформацією. Це свідчить про те, що внесення певних змін є доцільним, оскільки вони можуть сприяти подальшому розвитку галузі та зробити продукт більш зручним для користувача.

Метою кваліфікаційної роботи є розробка програмної системи менеджера паролів, призначеної для безпечного зберігання, обробки та керування конфіденційними даними користувачів із використанням сучасних методів криптографічного захисту. У роботі розглядаються принципи побудови таких систем із використанням сучасних криптографічних алгоритмів та багаторівневої архітектури програмного забезпечення.

Об’єктом дослідження даної роботи є процес створення програмної системи захисту паролів для обробки і автентифікаційних даних користувачів. Предметом дослідження є методи, моделі та технології розробки програмної системи менеджера паролів з використанням криптографічних алгоритмів шифрування та виведення ключів.

Наукова новизна отриманих результатів полягає у практичній реалізації автономного менеджера паролів з використанням сучасних криптографічних

методів шифрування та багатошарової архітектури. Запропоноване рішення забезпечує безпечне локальне зберігання конфіденційних даних із підтримкою версіонування записів і мінімізацією ризиків витоку інформації. Поєднання високого рівня захисту з інтуїтивним інтерфейсом підвищує зручність та ефективність використання програмного продукту.

Результати, отримані під час виконання кваліфікаційної роботи, були апробовані та оприлюднені у вигляді наукових тез. Зокрема, за темою дослідження підготовлено та опубліковано тези доповіді «Архітектурні підходи до проєктування програмних систем для безпечного керування автентифікаційними даними», у співавторстві з науковим керівником, у матеріалах наукової конференції.

1. АНАЛІЗ ВИМОГ ДО МЕНЕДЖЕРА ПАРОЛІВ

Розвиток цифрових технологій призвів до того, що користувачам необхідно працювати з великою кількістю облікових даних у різних сервісах, таких як паролі до вебсервісів, електронної пошти, банківських систем, хмарних платформ та інших інформаційних ресурсів. Вимоги до структури, змісту та оформлення програмних рішень у межах кваліфікаційних робіт магістра регламентуються методичними рекомендаціями та нормативними документами, які визначають загальні підходи до розроблення, аналізу та документування програмного забезпечення [1].

1.1 Аналіз предметної області

У наукових роботах з програмної інженерії все частіше наголошується на необхідності використання формалізованих методів під час аналізу та перевірки програмних рішень. Зокрема, у працях, присвячених статистичній верифікації моделей та алгоритмів, обґрунтовується необхідність системного підходу до оцінювання ефективності програмних компонентів та процесів обробки даних [2]. Особливе значення це має для програмних систем, які обробляють критичну або конфіденційну інформацію.

У наукових публікаціях останніх років все частіше розглядається інтеграція алгоритмів інтелектуального аналізу даних і штучних нейронних структур у програмні рішення, що використовуються в реальних виробничих та прикладних умовах. Роботи, присвячені використанню трансформерних нейронних мереж у контексті Industry 4.0, демонструють ефективність багатокомпонентних архітектур і підтверджують доцільність чіткого розмежування відповідальностей між рівнями програмної системи [3]. Подальший розвиток цих підходів простежується у дослідженнях, пов’язаних із видобуванням даних з текстової інформації та обробкою неструктурованих даних із використанням сучасних нейромережевих моделей [4–5].

Зі зростанням кількості онлайн-сервісів зростає і кількість потенційних загроз, пов’язаних із несанкціонованим доступом до персональної інформації [6].

Однією з основних проблем є використання простих або повторюваних паролів. Аналіз матеріалів з інформаційного захисту свідчить про те, що багато користувачів повторно застосовують ті самі облікові комбінації для доступу до різних онлайн-ресурсів, у результаті цього зростає ймовірність несанкціонованого доступу до інших облікових записів у випадку порушення безпеки хоча б одного сервісу [7]. Такі витоки трапляються досить часто і можуть призводити до фінансових втрат, втрати доступу до важливих сервісів або витоку конфіденційної інформації.

Ще однією проблемою є небезпечне зберігання паролів, наприклад у текстових файлах, браузерах без належного захисту або навіть у паперовому вигляді. Такі методи зберігання не забезпечують належного рівня захисту і не відповідають сучасним вимогам інформаційної безпеки [8]. У випадку фізичного доступу до пристрою або зараження шкідливим програмним забезпеченням дані користувача можуть бути легко скомпрометовані.

Для вирішення зазначених проблем використовуються менеджери паролів — спеціалізовані програмні системи, призначені для безпечного зберігання, обробки та керування конфіденційними даними користувача. Такі системи зазвичай використовують криптографічні алгоритми для шифрування інформації та забезпечують доступ до неї лише після успішної автентифікації користувача [9].

Використання менеджерів паролів дозволяє значно зменшити ризики, пов’язані з повторним використанням паролів, а також спрощує процес керування великою кількістю облікових записів. Користувач отримує можливість зберігати складні та унікальні паролі без необхідності їх запам’ятовування, що позитивно впливає як на рівень безпеки, так і на зручність використання цифрових сервісів.

Окрім технічних загроз, важливо враховувати і організаційні аспекти захисту облікових даних. У багатьох випадках користувачі не мають чіткого

розуміння принципів безпечного зберігання інформації та не дотримуються базових рекомендацій з інформаційної безпеки. Це призводить до ситуацій, коли навіть складні паролі зберігаються у відкритому вигляді або передаються через незахищені канали зв’язку.

У контексті розвитку цифрових технологій важливу роль відіграє баланс між безпекою та зручністю використання програмних систем. Надмірно складні механізми захисту можуть ускладнювати роботу користувача та знижувати ефективність використання системи. Тому сучасні менеджери паролів повинні поєднувати надійні криптографічні методи захисту з інтуїтивно зрозумілим інтерфейсом.

Крім того, актуальним є питання захисту даних у разі втрати або компрометації пристрою користувача. У таких випадках особливо важливим є застосування механізмів шифрування та автоматичного блокування доступу до конфіденційної інформації. Це дозволяє мінімізувати можливі наслідки несанкціонованого доступу та зберегти цілісність даних.

Таким чином, предметна область менеджерів паролів є актуальною та важливою в умовах зростання цифрових загроз. Розробка власної програмної системи в цій сфері дозволяє глибше дослідити принципи захисту інформації, організацію безпечного зберігання даних та реалізацію зручного і надійного інтерфейсу користувача.

1.2 Постановка завдання та цілей

Під час розробки програмної системи менеджера паролів необхідно враховувати як вимоги інформаційної безпеки, так і зручність використання для кінцевого користувача. Основним завданням такої системи є забезпечення безпечного зберігання конфіденційної інформації з мінімальним ризиком її компрометації. Особливу увагу слід приділяти правильній обробці введених даних, оскільки помилки на цьому етапі можуть призвести до втрати інформації або порушення цілісності сховища [10].

Якісний програмний засіб повинен характеризуватися стабільною роботою та достатньою швидкодією. Усі операції, пов’язані з додаванням, редагуванням або отриманням даних, мають виконуватися за короткий проміжок часу, незалежно від кількості збережених записів. Також важливо забезпечити коректну обробку помилок та захист від несанкціонованого доступу до даних [6].

Під час створення програмної системи менеджера паролів необхідно передбачити можливість виконання таких основних функцій:

- створення та зберігання записів із конфіденційною інформацією (паролі, ключі доступу, приватні ключі, seed-фрази);
- редагування наявних записів без втрати попередніх даних;
- шифрування секретних значень перед збереженням у базі даних;
- пошук записів за назвою або іншими відкритими параметрами;
- зберігання історії змін записів із можливістю перегляду попередніх версій;
- видалення або відновлення неактуальних записів.

Окрім базового функціоналу, програмна система повинна підтримувати низку додаткових можливостей, які підвищують зручність та безпеку:

- зрозумілий і зручний для сприйняття користувацький інтерфейс;
- чітку навігацію між різними типами збережених даних;
- автоматичне блокування сховища у разі бездіяльності користувача;
- захист від витоку даних через буфер обміну;
- можливість розширення функціональних можливостей без суттєвих змін у структурі програми;
- коректну роботу програми незалежно від рівня технічної підготовки користувача.

Важливим завданням при розробці програмної системи є забезпечення цілісності та надійності даних. Програмний продукт має забезпечувати збереження даних і недопущення їх пошкодження або втрати у випадках збоїв у

роботі застосунку, некоректних дій користувача чи виникнення непередбачених ситуацій. Для цього необхідно передбачити механізми контролю стану даних та коректного завершення операцій збереження.

Важливим елементом є забезпечення надійної перевірки користувача під час входу до системи. Доступ до конфіденційної інформації має надаватися лише після успішного підтвердження особи користувача за допомогою головного пароля. Також необхідно, щоб система належним чином реагувала на неправильне введення даних та не допускати розкриття будь-якої службової інформації, яка могла б полегшити несанкціонований доступ.

Також важливим є організація зручної структури зберігання даних. Користувач повинен мати можливість логічно впорядковувати записи, швидко знаходити необхідну інформацію та працювати з нею без зайвих складнощів. Це дозволяє підвищити ефективність використання системи та зменшити кількість помилок під час роботи. У майбутньому кількість збережених записів або типів даних може значно зрости, тому архітектура програмного продукту має передбачати можливість додавання нового функціоналу без суттєвого втручання в основну логіку його роботи.

Не менш потрібним є забезпечення незалежності програмної системи від зовнішніх сервісів. Використання локального зберігання даних дозволяє користувачеві зберігати повний контроль над власною інформацією та зменшує ризики, пов’язані з передачею конфіденційних даних через мережу. Це особливо актуально для користувачів, які приділяють підвищену увагу питанням приватності.

Основною ціллю даної роботи є розробка програмної системи менеджера паролів, яка поєднує високий рівень безпеки з простотою використання. У результаті виконання роботи має бути створений програмний продукт, готовий до практичного використання та подальшого розвитку.

1.3 Пошук акторів та варіантів використання

Для формування вимог до програмної системи менеджера паролів було проведено аналіз існуючих рішень на ринку програмних продуктів у сфері зберігання та захисту облікових даних. У результаті проведеного аналізу було встановлено, що на даний момент найбільшу популярність мають такі менеджери паролів: Bitwarden, 1Password, LastPass та KeePass [11–12]. Дані програмні продукти активно використовуються як приватними користувачами, так і організаціями для зберігання конфіденційної інформації.

Аналіз відкритих джерел та відгуків користувачів показав, що значна частина існуючих рішень має певні недоліки. Зокрема, користувачі часто відзначають складність інтерфейсу, обмежені можливості налаштування структури даних, залежність від хмарних сервісів або недостатній контроль над процесами шифрування [13]. Окрім цього, в окремих випадках мали місце суттєві інциденти, пов’язані з порушенням конфіденційності даних, що призвело до зниження рівня довіри до подібних рішень [14].

За результатами виконаного аналізу було виокремлено ключових акторів взаємодії з програмною системою менеджера паролів. Головним актором є кінцевий користувач, який безпосередньо взаємодіє із системою з метою зберігання, перегляду та редагування своїх конфіденційних даних. Саме користувач виконує такі дії, як створення сховища, автентифікація за допомогою головного пароля, додавання нових записів та керування ними.

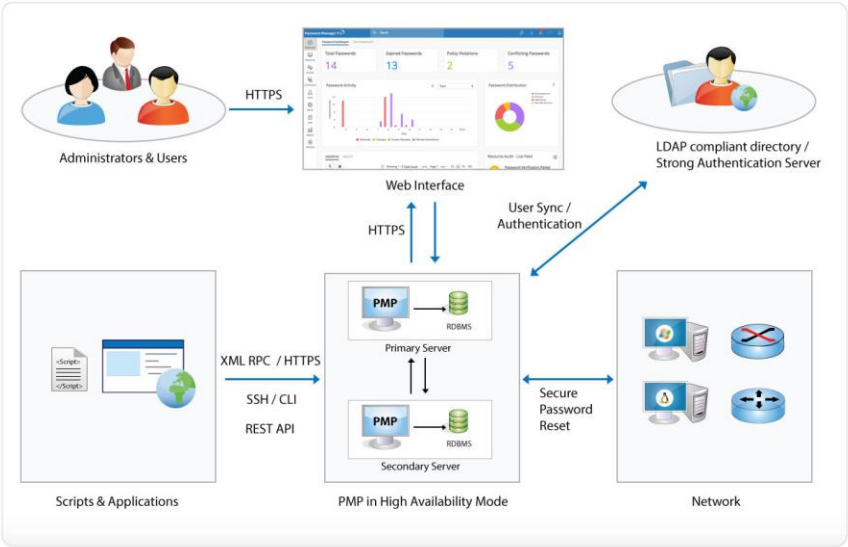

Рисунок 1.1 – Візуалізація діаграми сценаріїв використання

Окрім основного актора, важливу роль відіграє сама програмна система, яка відповідає за обробку даних, їх шифрування, зберігання у базі даних та контроль доступу. У рамках цієї роботи система розглядається як окремий логічний актор, що автоматично виконує криптографічні операції та забезпечує захист інформації без прямого втручання користувача.

Під час аналізу акторів також було враховано різні сценарії використання програмної системи залежно від контексту роботи користувача. Зокрема, менеджер паролів може використовуватися як у домашньому середовищі, так і в робочих умовах, де зберігається значна кількість службових облікових даних. Цей фактор визначає підвищені вимоги до стабільності роботи системи, її продуктивності та зручності отримання даних.

Поряд із типовими сценаріями використання, було розглянуто ситуації, пов’язані з помилками користувача або нестандартною поведінкою системи. Наприклад, введення некоректного головного пароля, переривання роботи програми або завершення сеансу без коректного блокування сховища. У таких випадках система повинна коректно обробляти подібні події та забезпечувати збереження конфіденційності даних. Програмний продукт має бути зручним у використанні як для підготовлених користувачів, так і для людей без технічного досвіду. Тому робота з системою повинна бути побудована таким чином, щоб основні дії виконувалися на інтуїтивному рівні й не вимагали детального

розуміння її внутрішніх механізмів.

Варто окремо підкреслити усі визначені варіанти використання орієнтовані на мінімізацію ризику випадкового розголошення конфіденційної інформації. Наприклад, доступ до секретних значень здійснюється лише за ініціативою користувача, а сама система не відображає такі дані без необхідності. Завдяки цьому зменшується ймовірність помилок з боку користувача, що позитивно впливає на безпеку системи.

Виходячи з виділених ролей акторів, було сформовано перелік ключових сценаріїв роботи з програмною системою, зокрема: створення нового захищеного сховища, розблокування сховища за допомогою головного пароля, додавання та редагування записів, перегляд збережених даних, пошук необхідної інформації, а також блокування сховища у разі завершення роботи або бездіяльності користувача.

Дані варіанти використання дозволяють охопити основні сценарії взаємодії користувача з програмною системою та слугують основою для подальшого проєктування її функціоналу.

1.4 Опис ключових варіантів використання

Після виокремлення ключових ролей акторів програмної системи менеджера паролів доцільно перейти до наступного етапу роботи, а саме — опис ключових варіантів використання, які відображають основні сценарії взаємодії користувача з програмним продуктом. Дані варіанти використання описують послідовність дій, які виконує користувач для досягнення певного результату, а також поведінку системи у відповідь на ці дії.

Основним актором у програмній системі є користувач, який використовує менеджер паролів для зберігання та керування власними конфіденційними даними. Головні сценарії взаємодії з системою зосереджені навколо забезпечення безпечного доступу до сховища, роботи з записами та підтримання цілісності даних.

Першим і базовим варіантом використання є створення захищеного сховища. Даний сценарій реалізується під час першого запуску програмної системи. Користувач задає головний пароль, який використовується для подальшого доступу до сховища. Після введення пароля система перевіряє його коректність та складність, генерує необхідні криптографічні параметри та створює зашифроване сховище для подальшого зберігання даних. У результаті виконання даного варіанту використання система переходить у стан готовності до роботи.

Наступним ключовим варіантом використання є розблокування сховища. Даний сценарій виконується під час кожного запуску програми або після автоматичного блокування. Для отримання доступу до збережених відомостей користувач вводить секретну комбінацію, після чого система аналізує її правильність. Якщо перевірка завершується успішно, відкривається можливість роботи з даними, тоді як у разі помилкового введення доступ до сховища залишається недоступним, а користувач інформується про виниклу помилку.

Ініціалізація та розблокування сховища є базовими варіантами використання менеджерів паролів, що підтверджується сучасними підходами до зберігання облікових даних [15–16].

Серед основних сценаріїв використання є додавання нового запису. У межах цього сценарію користувач створює новий запис із конфіденційними даними, такими як логін, пароль, ключ доступу або інша секретна інформація. Після отримання введеної інформації програмний продукт здійснює її обробку, виконує захист конфіденційних елементів за допомогою криптографічних механізмів та фіксує результати у сховищі даних. Реалізація цього сценарію надає користувачу можливість зосередити всі важливі відомості в одному безпечному середовищі.

Ще одним важливим сценарієм є редагування існуючого записуСистема надає користувачеві інструменти для редагування раніше створених записів, зокрема для зміни паролів або додавання нових відомостей. Після збереження правок оновлена інформація фіксується, а попередній варіант може зберігатися

окремо з метою подальшого перегляду. Це дозволяє уникнути втрати даних у разі помилкового редагування.

Наведені сценарії дають можливість проаналізувати роботу системи в умовах практичного використання. При цьому програмне рішення має забезпечувати стабільне функціонування не тільки під час типових дій, але й у ситуаціях, коли користувач взаємодіє з програмою непослідовно або перериває процес роботи. Це дозволяє зробити систему більш надійною та знизити ймовірність виникнення помилок під час її повсякденного використання.

Важливим варіантом використання є перегляд та пошук записів. Користувач може переглядати список збережених даних та здійснювати пошук за назвою або іншими відкритими параметрами. При цьому секретні значення залишаються прихованими та відображаються лише за необхідності, що зменшує ризик випадкового розголошення інформації.

Окремим сценарієм є копіювання конфіденційних даних. Користувач може скопіювати пароль або інше захищене значення, а система автоматично очищає буфер обміну через певний час, підвищуючи рівень безпеки. Сценарії додавання, редагування та пошуку записів є типовими для desktop-менеджерів паролів [17].

Завершальним ключовим варіантом використання є блокування сховища. Даний сценарій виконується за ініціативою користувача або автоматично у разі бездіяльності. Після блокування доступ до зашифрованих даних стає неможливим до моменту повторного введення головного пароля, що забезпечує додатковий рівень захисту інформації. Процеси автентифікації та блокування доступу відповідають рекомендаціям з безпеки інформаційних систем [18].

Варіанти використання менеджера паролів спрямовані на максимальне спрощення процесу зберігання складних комбінацій символів, зменшуючи потребу в їх запам’ятовуванні. Користувач взаємодіє із системою через зрозумілі дії, тоді як всі складні операції, пов’язані з обробкою та захистом даних, виконуються програмою у фоновому режимі. Завдяки цьому користувач може приділяти увагу безпосередній роботі з даними, не вникаючи у технічні особливості їх збереження.

Важливим елементом сценаріїв роботи системи є обмеження доступу до

інформації з урахуванням часових параметрів. У ситуаціях, коли користувач залишає робоче місце або припиняє взаємодію з програмою, система повинна самостійно переходити у захищений стан. Такий механізм зменшує ризик несанкціонованого доступу до сховища та підвищує рівень безпеки у повсякденних умовах використання. Також слід враховувати сценарії, пов’язані з тривалим накопиченням інформації. З часом кількість збережених записів може значно збільшуватися, що потребує зручних інструментів для навігації та роботи з даними. Передбачені варіанти використання повинні забезпечувати швидкий доступ до потрібної інформації незалежно від обсягу сховища.

Узагальнюючи, можна зазначити, що визначені ключові варіанти використання формують цілісну модель взаємодії користувача з менеджером паролів.

2 ПРОЄКТУВАННЯ ТА РОЗРОБКА МЕНЕДЖЕРА ПАРОЛІВ

Проєктування програмних систем відіграє важливу роль у розробці, основним є вибір архітектури, оскільки саме вона визначає структуру програмного продукту, спосіб взаємодії його компонентів та можливості подальшого розвитку. Архітектура програмного забезпечення описує високорівневу організацію системи, її основні складові та зв’язки між ними. Від правильного вибору архітектурного підходу залежить надійність, масштабованість і зручність супроводу програмного продукту [19].

2.1 Вибір процесу розробки

Під час проєктування сучасних програмних систем застосовуються різні архітектурні моделі. До найбільш поширених належать монолітна архітектура, клієнт-серверна архітектура, багатошарова архітектура, мікросервісна архітектура та подієво-орієнтовані підходи. Кожна з цих архітектур має специфічні особливості, переваги та обмеження, тому вибір конкретного підходу визначається характеристиками програмного продукту та вимогами до його функціонування і масштабування [20].

Для програмних систем, орієнтованих на локальне зберігання даних та підвищені вимоги до безпеки, зокрема для desktop-застосунків, доцільним є використання архітектур, що забезпечують чітке розмежування відповідальності між компонентами. Це дозволяє зменшити зв’язаність між частинами системи та підвищити рівень захисту критичних даних.

З урахуванням особливостей розроблюваного менеджера паролів було обрано багатошарову архітектуру у поєднанні з ітеративним процесом розробки (див. рисунок 2.1). Обрання такого підходу зумовлене прагненням відокремити механізми роботи з конфіденційною інформацією від користувацького інтерфейсу, а також забезпечити зручні умови для тестування й подальшого розвитку можливостей системи.

Перевагами цієї архітектури є:

- Чітке розмежування відповідальності між рівнями програмної системи.
- Посилений захист інформації завдяки відокремленню обробки даних і прикладної логіки від користувацького рівня взаємодії.
- Спрощення супроводу та підтримки програмного коду.
- Можливість незалежного розвитку окремих компонентів системи.
- Полегшення тестування окремих рівнів програми.
- Зручність розширення функціональних можливостей без зміни загальної структури системи.
- Зменшення ризику помилок при роботі з конфіденційними даними.

Недоліки архітектури:

- Підвищена складність проєктування порівняно з простими монолітними рішеннями.
- Збільшення кількості компонентів та файлів у проєкті.
- Вищі вимоги до планування взаємодії між рівнями системи.
- Можливе зниження продуктивності через додаткові рівні обробки даних.

Багатошарова архітектура передбачає поділ програмної системи на логічні рівні, кожен з яких відповідає за окремий аспект роботи програми. У межах даного підходу система поділяється на окремі рівні, серед яких зазвичай розрізняють користувацький рівень, логіку обробки та компонент роботи з даними. Така структура забезпечує можливість оновлення або зміни окремих елементів без необхідності втручання в роботу всієї системи [21].

У цьому проєкті рівень представлення відповідає за відображення інтерфейсу та забезпечення взаємодії користувача з програмою. На цьому рівні здійснюється введення даних, відображення інформації та обробка дій користувача. Важливою особливістю є відсутність прямого доступу цього рівня

до бази даних або криптографічних механізмів, що зменшує ризик порушення безпеки.

На сервісному рівні зосереджена логіка роботи менеджера паролів, включно з керуванням сховищем, доступом до даних та їх криптографічним захистом. Зосередження критичної логіки на одному рівні дозволяє забезпечити єдині правила обробки інформації та спростити контроль безпеки [22].

Компонент роботи з даними забезпечує обмін інформацією з локальним сховищем. Збереження відомостей здійснюється виключно в захищеному форматі, а будь-які операції з ними виконуються опосередковано через логічний рівень системи. Така організація відповідає сучасним підходам до створення безпечного програмного забезпечення та знижує ймовірність несанкціонованого отримання доступу до конфіденційних даних [23].

Використання багатошарової архітектури також сприяє підвищенню підтримуваності програмного коду. Кожен рівень може розвиватися незалежно, що є важливим у разі подальшого вдосконалення системи або додавання нових функцій. Крім того, така архітектура добре поєднується з ітеративним процесом розробки, дозволяючи поступово розширювати можливості програмного продукту без порушення його загальної структури.

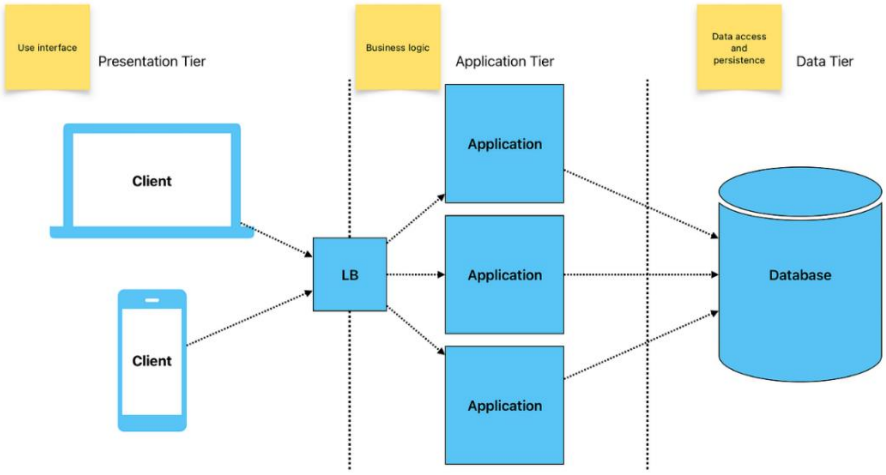

Рисунок 2.1 – Схема роботи багатошарової архітектури

Отже, обраний архітектурний підхід є обґрунтованим для реалізації менеджера паролів як desktop-застосунку з підвищеними вимогами до безпеки.

Застосування багаторівневої архітектури допомагає розділити функції системи, покращити безпеку даних і забезпечити можливість її подальшого розвитку.

2.2 Проектування архітектури системи

Розробка архітектурної моделі програмної системи менеджера паролів виконувалася на основі попередньо сформульованих функціональних вимог і визначених сценаріїв її використання. На даному етапі роботи ключовим завданням стало визначення способів взаємодії між складовими системи та розмежування функцій, за які відповідає кожен компонент. Метою проєктування було створення такої архітектури, яка забезпечує захист конфіденційних даних, зручність реалізації логіки та можливість подальшого розширення системи.

Першим кроком проєктування стало логічне розділення системи на незалежні компоненти відповідно до виконуваних функцій. Було прийнято рішення відокремити інтерфейс користувача від логіки обробки даних та механізмів зберігання. Такий підхід дозволяє змінювати або вдосконалювати інтерфейс без впливу на критичні частини системи, що відповідають за безпеку.

На етапі проєктування було визначено, що всі операції, пов’язані з конфіденційними даними, мають виконуватися виключно в межах сервісних компонентів. Інтерфейс користувача не зберігає і не обробляє секретні значення безпосередньо, а лише ініціює відповідні запити. Це допомагає підвищити безпеку даних і зробити обробку конфіденційної інформації більш контрольованою.

Окрему увагу під час проєктування архітектури було приділено організації криптографічної логіки. Криптографічні операції були винесені в окремі компоненти, що дозволяє централізовано керувати алгоритмами шифрування, параметрами ключів та процесами автентифікації. Такий підхід спрощує перевірку коректності реалізації криптографії та зменшує ймовірність помилок у різних частинах системи.

При проєктуванні механізму збереження даних було прийнято рішення

використовувати локальне сховище з чітко визначеним інтерфейсом доступу. Побудова системи організована таким чином, що взаємодія з базою даних здійснюється виключно через окремий модуль роботи з даними, тоді як інші компоненти не мають безпосереднього доступу до її внутрішньої структури. Завдяки цьому стає можливим оновлення способу зберігання або зміна формату даних без необхідності втручання в логіку функціонування системи.

Також під час проєктування архітектури були враховані сценарії, пов’язані з безпекою сеансу роботи користувача. Архітектура передбачає наявність механізмів автоматичного блокування доступу, очищення тимчасових даних та контроль часу активності. Дані механізми інтегровані у загальну структуру системи та взаємодіють із сервісними компонентами без участі користувача.

Результатом проєктування архітектури стала структурована модель системи, у якій кожен компонент має чітко визначене призначення та межі відповідальності. Така архітектура дозволяє реалізувати менеджер паролів як надійний та безпечний програмний продукт, а також створює основу для подальшої деталізації структури даних і побудови UML-діаграм на наступних етапах роботи.

На рисунку представлено багатошарову архітектуру менеджера паролів PiePass, що реалізує чітку трирівневу структуру, яка включає користувацький інтерфейс, рівень обробки логіки та компонент взаємодії зі сховищем даних (див. рисунок 2.2). Такий поділ забезпечує чітке розмежування відповідальностей між компонентами, підвищує підтримуваність коду та підсилює безпеку, оскільки доступ до конфіденційних даних і криптографічних механізмів контролюється централізовано.

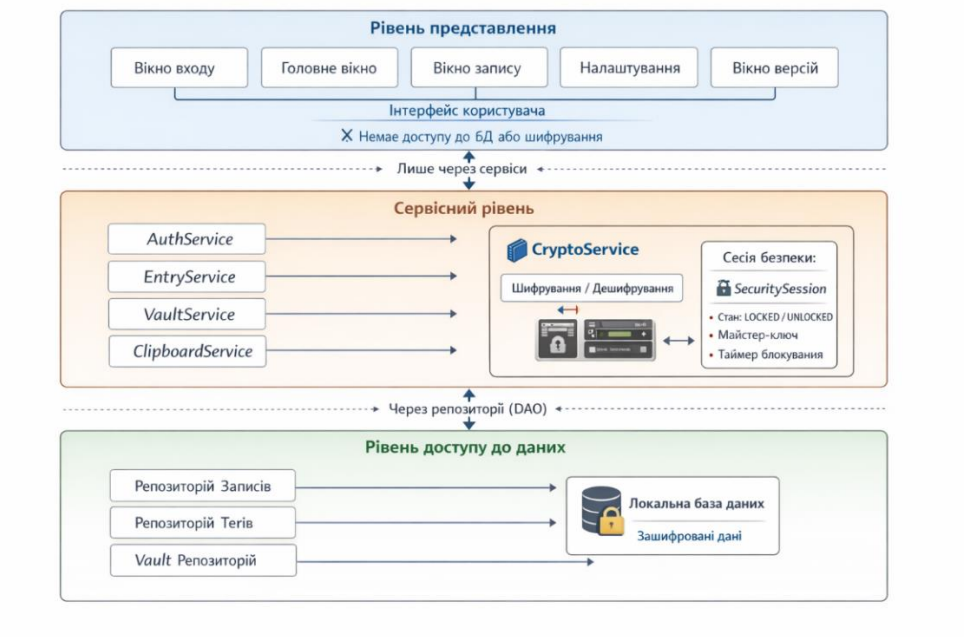

Рисунок 2.2 – Багатошарова архітектура менеджера паролів PiePass

Рівень представлення відповідає за взаємодію з користувачем і реалізацію графічного інтерфейсу. До цього рівня належать основні UI-модулі: вікно входу, головне вікно, вікно створення/редагування запису, вікно налаштувань та вікно версій. Цей рівень відповідає за роботу з введенням даних, показ інформації та реакцію системи на дії користувача.

Однією з основних вимог безпеки є відсутність прямої взаємодії користувацького рівня з базою даних, а також винесення операцій шифрування й дешифрування за межі рівня представлення. Інтерфейс виконує лише роль ініціатора запитів, а вся чутлива логіка передається на сервісний рівень. На схемі це підкреслено позначкою відсутності доступу до БД або шифрування та правилом «лише через сервіси».

Сервісний рівень є центральною частиною системи та реалізує основну бізнес-логіку. До нього входять модулі:

AuthService — відповідає за автентифікацію користувача, розблокування/блокування сховища, а також керування станом доступу.

VaultService — реалізує операції, пов’язані зі сховищем: ініціалізація, керування параметрами та загальною конфігурацією.

EntryService — відповідає за керування записами (створення, читання,

редагування, видалення) та контроль доступу до секретних полів;

ClipboardService — забезпечує копіювання секретних даних у буфер обміну та їх автоматичне очищення після завершення встановленого часу.

Окремо в межах сервісного рівня виділено компонент CryptoService, який централізовано виконує криптографічні операції шифрування та дешифрування. Винесення криптографії в окремий сервіс спрощує контроль коректності реалізації, уніфікує правила роботи з ключами і зменшує ризик помилок у різних частинах системи.

Також у сервісному рівні використовується SecuritySession — компонент, який підтримує параметри сеансу безпеки: стан доступу (LOCKED/UNLOCKED), тимчасове зберігання майстер-ключа в оперативній пам’яті під час активного сеансу та механізми таймера автоматичного блокування. Це дозволяє реалізувати політику безпеки, коли за відсутності активності доступ до сховища блокується без участі користувача, а тимчасові дані очищуються.

Рівень доступу до даних відповідає за взаємодію з локальною базою даних та інкапсулює всі операції читання/запису. На схемі цей рівень представлено через репозиторії (DAO), зокрема:

- репозиторій записів — операції з даними облікових записів та їх атрибутами;
- репозиторій тегів — збереження та зв’язки тегів/категорій;
- vault-репозиторій — збереження службових параметрів сховища.

Локальне сховище даних містить інформацію у формі структурованих записів, при цьому всі чутливі відомості зберігаються з використанням криптографічного захисту. Звернення до таких даних здійснюється виключно через сервісний рівень системи, який відповідає за виконання операцій шифрування та розшифрування за допомогою модуля CryptoService. Таким чином, навіть у разі прямого доступу до файлу бази даних зловмисник не зможе прочитати секрети без ключа.

У результаті сформовано безпечну та підтримувану архітектуру, у якій кожен рівень має чітко визначену відповідальність, а критичні операції з конфіденційними даними виконуються централізовано у сервісному шарі.

2.3 Побудова схем бази даних

Для забезпечення зручності використання програмної системи менеджера паролів користувачами без спеціальних технічних знань, сховище даних має бути організоване у вигляді зрозумілої та впорядкованої структури, яка функціонує незалежно від зовнішніх впливів. Проєктування бази даних виконувалося з урахуванням вимог до безпеки, гнучкості структури та можливості подальшого розширення функціоналу системи.

На основі вимог, визначених у ході аналізу предметної області та розглянутих сценаріїв використання, було сформовано концептуальне подання структури даних. На її основі визначено основні сутності системи та зв’язки між ними. Аналіз функціональних вимог дозволив виділити такі ключові сутності бази даних менеджера паролів:

- сховище;
- тип запису;
- опис поля;
- запис;
- значення поля запису;
- версія запису.

Інформацію щодо основних типів сутностей подано в таблиці 2.1. З урахуванням сформульованих користувацьких вимог для кожної з них було визначено набір характеристик, що описують ключові об’єкти системи. Сукупність виділених сутностей разом із відповідними атрибутами формує опис предметної області. Деталізовані відомості про атрибути та їх належність до конкретних сутностей наведено в таблиці 2.2

Таблиця 2.1 – Перелік типів сутностей системи

Ім’я сутності	**Опис сутності**	**Псевдонім**	**Особливості використання**
Сховище	Містить службову та криптографічну інформацію сховища	vault	Є кореневою сутністю системи
Тип запису	Визначає структуру записів	item_type	Дозволяє створювати різні типи даних
Опис поля	Описує структуру полів типу запису	field_definition	Задає тип і властивості полів
Запис	Містить метадані запису	item	Представляє конкретний об’єкт
Значення поля	Зберігає значення окремих полів	item_field_value	Містить зашифровані дані
Версія запису	Зберігає історію змін	item_version	Забезпечує версіонування

Таблиця 2.2 – Характеристика атрибутів сутностей

Сутність	**Атрибут**	**Тип даних**	**Обмеження**	**Псевдонім**	**NULL**
Vault (Сховище)	Ідентифікатор	Ціле число	Первинний ключ	id	Ні
	Salt	Текст	—	salt	Ні
	Параметри KDF	Текст	—	kdf_params	Ні
	Зашифрований ключ	Текст	—	encrypted_key	Ні
	Дата створення	Дата	—	created_at	Ні

Продовження таблиці 2.2

ItemType (Тип запису)	Ідентифікатор	Ціле число	Первинний ключ	id	Ні
	Назва типу	Текст	—	name	Ні
	Опис	Текст	—	description	Так
	Іконка	Текст	—	icon	Так
FieldDefinition (Опис поля)	Ідентифікатор	Ціле число	Первинний ключ	id	Ні
	Тип запису	Посилання	Зовнішній ключ	item_type_id	Ні
	Назва поля	Текст	—	name	Ні
	Тип даних	Текст	—	field_type	Ні
	Обов’язкове	Логічний	—	required	Ні
Item (Запис)	Ідентифікатор	Ціле число	Первинний ключ	id	Ні
	Тип запису	Посилання	Зовнішній ключ	item_type_id	Ні
	Назва	Текст	—	title	Ні
	Дата створення	Дата	—	created_at	Ні
	Дата редагування	Дата	—	updated_at	Ні
ItemFieldValue (Значення поля)	Ідентифікатор	Ціле число	Первинний ключ	id	Ні
	Запис	Посилання	Зовнішній ключ	item_id	Ні
	Поле	Посилання	Зовнішній ключ	field_definition_id	Ні
	Значення	Текст	—	encrypted_value	Ні

Сутність Vault є ключовою для всієї бази даних, оскільки містить криптографічні параметри, необхідні для розблокування сховища. Атрибут id забезпечує унікальну ідентифікацію сховища, а поля salt та kdf_params використовуються для виведення ключа шифрування.

Сутність ItemType відповідає за формування логічної організації записів і надає можливість створювати різні типи даних без внесення змін до структури бази даних.

Сутність FieldDefinition описує набір полів, які належать до певного типу запису. Завдяки цьому система підтримує гнучку модель даних.

Сутність Item представляє конкретний запис користувача та містить лише метадані, без збереження секретних значень.

Сутність ItemFieldValue зберігає зашифровані значення полів, що забезпечує захист конфіденційної інформації.

Сутність ItemVersion використовується для збереження історії змін записів та дозволяє переглядати попередні стани даних.

Наступним етапом є розгляд атрибутів сутності Сховище (Vault), яка є основоположною для всієї бази даних менеджера паролів. Стовпець під назвою «Ідентифікатор» (id) забезпечує сховищу унікальний номер, за яким його можна однозначно визначити у системі без додаткових даних. Стовпець «Salt» (salt) використовується як криптографічна сіль для формування ключа шифрування і є необхідним для правильного розблокування сховища. Стовпець «Параметри KDF» (kdf_params) зберігає налаштування алгоритму виведення ключа (наприклад кількість ітерацій або режим роботи), що дозволяє відтворити процес отримання ключа під час входу. Стовпець «Зашифрований ключ» (encrypted_key) містить зашифрований ключ сховища, який використовується для роботи з конфіденційними даними після успішної автентифікації. Стовпець «Дата створення» (created_at) фіксує момент ініціалізації сховища і може використовуватися як службова інформація.

Атрибути сутності Тип запису (ItemType) виглядають наступним чином. Атрибут «Ідентифікатор» (id) є первинним ключем і забезпечує унікальність

кожного типу запису. Стовпець «Назва типу» (name) використовується для відображення типу користувачеві (наприклад “Пароль”, “Нотатка”, “Ключ доступу”) і є обов’язковим. Стовпець «Опис» (description) містить коротке пояснення призначення типу і може бути відсутнім, якщо користувач не заповнює його. Стовпець «Іконка» (icon) використовується для зручності в інтерфейсі — він дозволяє візуально відрізняти типи записів один від одного.

Атрибути сутності Опис поля (FieldDefinition) формують структуру полів для кожного типу запису. Атрибут «Ідентифікатор» (id) забезпечує унікальність конкретного поля. Стовпець «Тип запису» (item_type_id) є посиланням на таблицю типів і визначає, до якого саме типу належить поле. Стовпець «Назва поля» (name) задає зрозуміле ім’я поля (наприклад “Логін”, “Пароль”, “URL”), яке буде показане користувачу. Стовпець «Тип даних» (field_type) визначає, як саме поле повинно відображатися та редагуватися (текст, пароль, багаторядкове поле тощо). Стовпець «Обов’язкове» (required) позначає, чи повинно поле бути заповненим; це потрібно для контролю введення даних та запобігання створенню “порожніх” записів.

Далі розглянемо сутність Запис (Item), яка представляє конкретний об’єкт у сховищі користувача. Стовпець «Ідентифікатор» (id) є первинним ключем і дозволяє однозначно звертатися до запису. Стовпець «Тип запису» (item_type_id) визначає структуру цього запису, тобто які поля він має містити. Стовпець «Назва» (title) використовується для відображення запису в загальному списку та для пошуку — зазвичай це назва сервісу або коротка мітка. Стовпці «Дата створення» (created_at) і «Дата редагування» (updated_at) дозволяють відслідковувати життєвий цикл запису, а також можуть використовуватися в сортуванні або історії змін.

Сутність Значення поля (ItemFieldValue) використовується для збереження конкретних значень полів, які належать певному запису. Атрибут «Ідентифікатор» (id) забезпечує унікальність кожного збереженого значення. Стовпець «Запис» (item_id) вказує, до якого саме запису належить значення. Стовпець «Поле» (field_definition_id) визначає, яке це поле за структурою

(наприклад саме “Пароль”, а не “Логін”). Стовпець «Значення» (encrypted_value) зберігає безпосередньо дані, причому у зашифрованому вигляді, щоб у базі даних не було відкритих конфіденційних значень.

Останньою сутністю є Версія запису (ItemVersion), яка потрібна для збереження історії змін. Атрибут «Ідентифікатор» (id) є первинним ключем. Стовпець «Запис» (item_id) вказує, до якого об’єкта належить версія. Стовпець «Дата версії» (created_at) фіксує момент створення версії і дозволяє зрозуміти, коли саме були внесені зміни. Стовпець «Дані версії» (snapshot) містить збережений стан запису у вигляді структури, яка дозволяє відновити попередні значення у разі потреби.

Виявлено всі ключові сутності та визначено їхні атрибути. Унаслідок цього сформовано чітку структуру основних таблиць у базі даних менеджера паролів.

2.4 Побудова UML-діаграм класів

ML (Unified Modeling Language) — це стандартизований спосіб візуально описувати та моделювати системи під час розробки програмного забезпечення. Її застосовують для об’єктного моделювання, проєктування систем, опису бізнес-процесів, а також для наочного подання структури організації чи взаємодії компонентів. UML належить до універсальних мов моделювання: це відкритий стандарт, який використовує набір графічних позначень, щоб сформувати узагальнену (абстрактну) модель системи — тобто UML-модель.

UML не призначена для безпосереднього написання програмного коду, однак створені з її допомогою моделі можуть слугувати основою для подальшої автоматичної або напівавтоматичної генерації програмних компонентів. На основі технології UML будується єдина інформаційна модель. UML підтримує етапи життєвого циклу інформаційної системи і для цього надає графічні засоби – діаграми.

Серед різновидів UML-діаграм важливе місце займає діаграма прецедентів.

Вона використовується для наочного відображення взаємодії між користувачами (акторами) та функціональними можливостями системи.

Діаграма прецедентів використовується для відображення функціональних можливостей системи та ролей користувачів, які з ними взаємодіють. Зазвичай вона доповнюється іншими UML-діаграмами для детальнішого опису роботи програмного продукту. Функціональні сценарії на такій діаграмі позначаються у вигляді овальних елементів, тоді як користувачі системи відображаються за допомогою умовних графічних символів [24].

З метою наочного відображення взаємодії між ролями користувачів і функціональними можливостями системи доцільно використовувати діаграму варіантів використання. Таке графічне представлення дозволяє чітко окреслити набір функцій, які мають бути реалізовані в програмному продукті (див. рисунок 2.4).

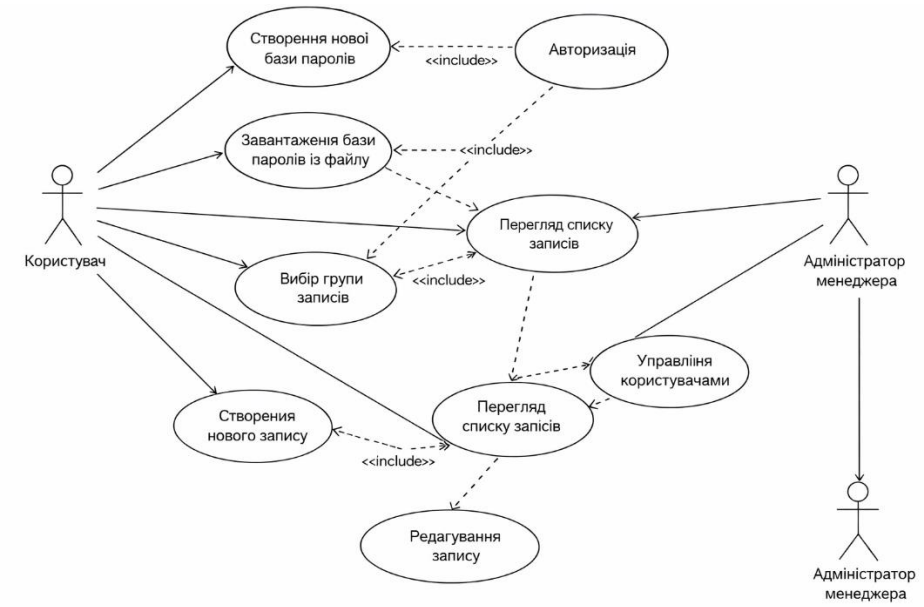

Рисунок 2.4 – Діаграма варіантів використання системи менеджера паролів

Під час розробки програмної системи менеджера паролів було визначено одну основну категорію користувачів — кінцевий користувач. Такий вибір зумовлений тим, що система є локальним desktop-застосунком і не передбачає розподілу ролей між декількома типами користувачів. Усі операції зі створення, редагування та керування записами виконуються безпосередньо користувачем.

Для детальнішого аналізу взаємодії користувача з програмною системою були розглянуті основні потоки подій, що відповідають варіантам використання, представленим на діаграмі варіантів використання. Наступним етапом моделювання є побудова UML-діаграм діяльності [25]. Для їх формування попередньо виконується аналіз можливих сценаріїв перебігу подій у системі . Узагальнені результати такого аналізу подано у таблицях, де відображено основні потоки подій, що використовуються при створенні відповідних діаграм (див. табл. 2.5).

Таблиця 2.5 – Аналіз потоків подій системи менеджера паролів

№	Потоки подій	Опис
1	Створення нової бази паролів	• користувач ініціює створення нової бази паролів; • система пропонує задати головний пароль; • на основі введеного пароля генеруються криптографічні параметри; • створюється порожнє зашифроване сховище; • користувачу відображається повідомлення про успішне створення бази.
2	Завантаження бази паролів з файлу	• користувач вибирає файл існуючої бази паролів; • система зчитує службові дані та параметри шифрування; • користувач вводить головний пароль; • у разі успішної автентифікації база паролів відкривається.
3	Вибір групи записів	• користувач обирає групу записів у графічному інтерфейсі програми; • система формує запит на отримання списку записів обраної групи; • список відповідних записів відображається користувачу.
4	Створення нового запису	• користувач ініціює створення нового запису; • система відображає форму введення даних; • користувач вводить необхідну інформацію; • отримана інформація піддається криптографічному захисту та зберігається у сховищі даних; • новий запис додається до загального списку.

Продовження таблиці 2.5

5	Редагування запису	• користувач вибирає існуючий запис; • система відображає поточні дані запису; • користувач вносить зміни до інформації; • оновлені дані шифруються та зберігаються; • за необхідності фіксується нова версія запису.

Такий опис потоків подій дозволяє детально проаналізувати логіку роботи програмної системи менеджера паролів та є основою для побудови UML-діаграм діяльності, а також подальшої реалізації програмних модулів.

2.5 Вибір мови та середовища розробки

У процесі створення програмної системи менеджера паролів важливе значення мав вибір відповідних засобів розробки та мови програмування, адже саме вони впливають на рівень захищеності, надійність функціонування та простоту подальшого обслуговування програмного продукту. Підбір інструментів і технологій здійснювався з урахуванням необхідності застосування об’єктно-орієнтованого підходу, створення настільних програм, роботи з локальними базами даних та впровадження актуальних засобів криптографічного захисту інформації.

Як основний інструмент розробки програмної системи було використано мову програмування Python. Вона є популярним рішенням для створення прикладних програм завдяки зрозумілій структурі коду, зручності у супроводі та широкому вибору готових програмних бібліотек. Застосування Python дає можливість ефективно реалізовувати логіку роботи системи та знижує ймовірність виникнення помилок, що можуть з’являтися через складність програмного коду [26].

Важливою перевагою Python є його підтримка об’єктно-орієнтованого програмування, що дозволяє природно реалізувати архітектурні рішення, закладені на етапі проєктування системи. Програмні класи, що реалізують сутності бази даних і функціональні сервіси, побудовані відповідно до структури предметної області, що полегшує підтримку програмного продукту та дає змогу без ускладнень розширювати його можливості [27].

Для створення графічного інтерфейсу користувача у програмній системі було використано Qt-фреймворк через Python-бібліотеки. Qt надає потужні засоби для розробки кросплатформних desktop-застосунків із сучасним та інтуїтивно зрозумілим інтерфейсом. Використання Qt дозволяє відокремити логіку роботи програми від її візуального представлення, що відповідає обраній багатошаровій архітектурі [28].

Для зберігання даних у менеджері паролів використовується локальна реляційна база даних, що дозволяє зберігати всю конфіденційну інформацію без передачі її через мережу. Python забезпечує зручні можливості для роботи з базами даних, що робить його придатним для автономних систем із підвищеними вимогами до приватності [29].

Окрему роль у програмній системі відіграють криптографічні бібліотеки Python, які використовуються для реалізації шифрування, виведення ключів та захисту конфіденційних даних. Наявність перевірених та добре задокументованих криптографічних бібліотек дозволяє реалізувати сучасні алгоритми захисту без необхідності створення власних криптографічних механізмів, що зменшує ризик помилок у реалізації безпеки [30].

Для створення програмного продукту було використано сучасне інтегроване середовище розробки, яке забезпечує повноцінну підтримку мови Python, засоби налагодження та ефективну роботу з проєктами різного рівня складності. Застосування такого інструментарію сприяє підвищенню швидкості розробки, полегшує процес перевірки коректності роботи програми та забезпечує зручну організацію і перегляд структури програмної системи [31-34].

Отже, використання мови програмування Python у поєднанні з

фреймворком Qt для побудови користувацького інтерфейсу, локальним сховищем даних і засобами криптографічного захисту є доцільним рішенням. Сформований технологічний набір дозволяє створити кросплатформний програмний продукт із підвищеним рівнем безпеки, спрощує процес розробки та створює умови для подальшого розширення функціональних можливостей менеджера паролів.

2.6 Реалізація основних класів та методів

Після визначення функціональних можливостей менеджера паролів і побудови діаграми варіантів використання доцільно перейти до опису внутрішньої структури системи. Для цього використовується діаграма класів, яка дозволяє відобразити основні елементи програмної системи та характер їх взаємодії між собою.

Діаграми класів у UML застосовуються для формування статичного уявлення про структуру програмного продукту. Вони відображають ключові поняття системи у вигляді класів, кожен з яких містить набір властивостей і відповідних методів. У графічному поданні UML клас зображається у формі прямокутного блоку. Взаємозв’язки типу спадкування показують механізм передавання атрибутів і функцій від базових класів до похідних, що дозволяє повторно використовувати логіку без дублювання її опису в кожному окремому класі [34].

Уніфікована мова моделювання (UML) використовує поняття класу для представлення сукупності об'єктів з однаковою структурою, поведінкою та відношеннями.

У процесі створення діаграми класів були визначені ключові класи системи та наведено їх характеристику:

- Password
- MasterKey
- AccountEntry

- GroupEntry
- FileManager
- FileEncryptor
- MainForm

1. Клас "Password" (див. рис. 2.6)

Для представлення паролів у системі було розроблено клас Password, який містить наступні поля:

- password_masterKey: зберігає майстер-ключ пароля.
- password_masterKey_len: вказує довжину майстер-ключа.
- encryptedPassword: зберігає зашифрований пароль.
- encryptedPassword_len: вказує довжину зашифрованого пароля.
- password_salt: зберігає сіль пароля.
- password_salt_len: вказує довжину солі.
- password_IV: зберігає ініціалізаційний вектор пароля.
- password_IV_len: вказує довжину ініціалізаційного вектора.

Поля для зберігання довжини даних використовуються для забезпечення сумісності з функцією ProtectMemory, яка запобігає несанкціонованому доступу до області оперативної пам’яті. Функція вимагає, щоб розмір блоку був кратний 16 байтам.

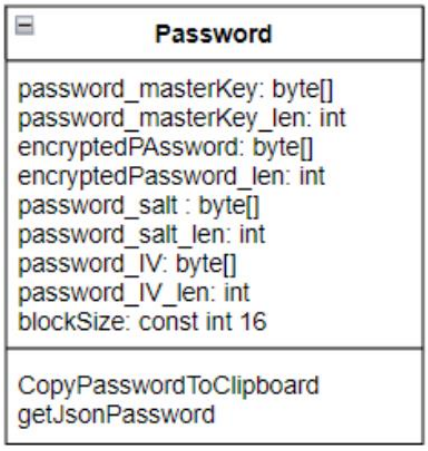

Рисунок 2.6 – Поля і методи класу Password

2. Клас "MasterKey" (див. рис. 2.7)

MasterKey забезпечує роботу з майстер-ключем, який використовується під час шифрування та дешифрування паролів.

Поля класу:

• _key: приватне поле, що містить майстер-ключ у вигляді масиву байтів.

• _keyLen: публічне поле, що вказує довжину майстер-ключа.

• blockSize: константа, яка визначає розмір блоку для вирівнювання довжини ключа, необхідний для коректної роботи методу ProtectedMemory.Protect().

Функція Protect() класу ProtectedMemory застосовується для забезпечення безпеки майстер-ключа під час його зберігання в оперативній пам’яті, запобігаючи доступу до нього з боку сторонніх процесів.

MasterKey тісно взаємодіє з класом Password, забезпечуючи безпечне зберігання та використання майстер-ключа для шифрування та розшифрування паролів користувача.

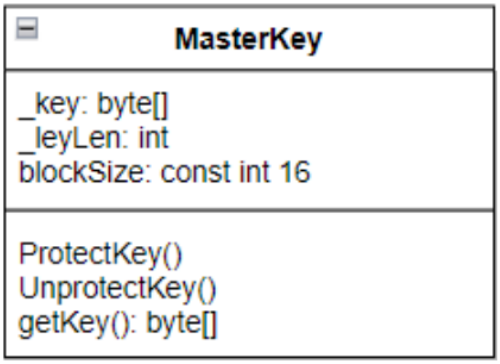

Рисунок 2.7 – Поля і методи класу MasterKey

3. Клас "AccountEntry" (див. рис. 2.8)

Клас AccountEntry використовується для представлення облікового запису користувача та зберігання пов'язаної з ним інформації, такої як назва облікового запису, логін, пароль та додаткові властивості.

Поля класу:

- accountName: назва облікового запису (рядок).
- login: логін облікового запису (рядок).
- password: об'єкт типу Password, що містить зашифрований пароль.
- properties: словник для зберігання додаткових властивостей облікового запису.

SerializedPassword властивість, яка серіалізує пароль у формат JSON за допомогою методу password.getJsonPassword().

Клас AccountEntry також містить індексатор, який дозволяє звертатися до властивостей у словнику properties за допомогою ключа. Якщо властивість з таким ключем існує, індексатор повертає її значення, інакше повертає null. Індексатор також дозволяє встановлювати значення властивостей.

Також реалізується інтерфейс IDisposable, що дозволяє звільняти ресурси, які використовуються об'єктом, коли він більше не потрібний. Це забезпечує ефективне управління пам'яттю та запобігає витоку ресурсів.

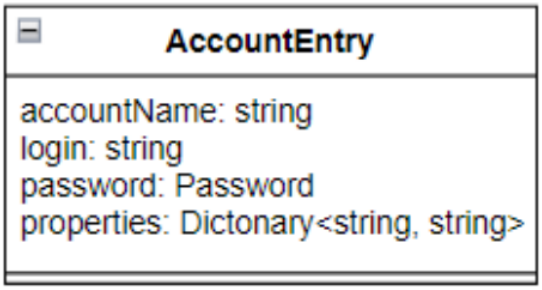

Рисунок 2.8 – Поля і методи класу AccountEntry

4. Клас "GroupEntry" (див. рис. 2.9)

Клас GroupEntry використовується для представлення групи в менеджері паролів. Він містить наступні основні елементи:

- groupList: список об'єктів GroupEntry, що представляють вкладені

групи в поточній групі. Цей список використовується для зберігання та управління підгрупами.

• accountEntries: список об'єктів AccountEntry, що представляють облікові записи, пов'язані з поточною групою. Цей список використовується для зберігання та управління обліковими записами.

• name: рядок, що містить назву поточної групи.

Цей клас надає зручне представлення інформації у вигляді дерева, що дозволяє організовувати облікові записи та групи в ієрархічну структуру. Кожен об'єкт GroupEntry представляє окрему групу і може містити як підгрупи, так і облікові записи.

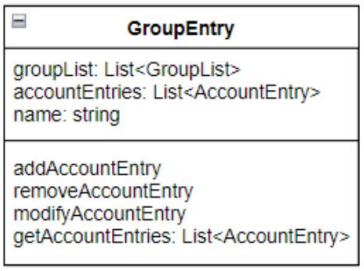

Рисунок 2.9 – Поля і методи класу GroupEntry

5. Класи для роботи з файловими даними (див. рис. 2.10)

З метою забезпечення ефективної роботи з файлами в рамках проекту було розроблено два спеціалізовані класи: FileManager та FileEncryptor.

У програмній системі для виконання операцій із файлами передбачено два окремі класи — FileManager і FileEncryptor, кожен з яких відповідає за власний набір функцій.

FileEncryptor надає статичні методи для шифрування та розшифрування JSON-об'єктів, що використовуються для зберігання даних у файлах. Завдяки статичній природі методів, немає необхідності створювати екземпляр класу для їх виклику.

FileManager відповідає за завантаження, збереження та керування

файлами бази паролів та конфігурації.

Основні методи класу:

- LoadDb: завантажує та розшифровує файл бази паролів, використовуючи переданий шлях до файлу та об'єкт MasterKey.
- SaveDbToFile: шифрує та зберігає кореневий об'єкт GroupEntry (базу паролів) у файл за вказаним шляхом.
- Методи для завантаження та збереження файлу конфігурації.

Усі методи класу FileManager реалізовані як статичні, що дозволяє зручно використовувати їх у різних модулях програмної системи без створення окремих екземплярів класу.

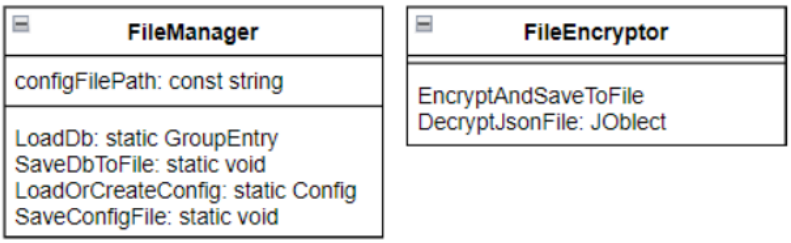

Рисунок 2.10 – Поля і методи класів FileManager та FileEncryptor

6. Клас "MainForm"

MainForm виконує роль основної форми програми, через яку користувач взаємодіє з системою та її компонентами.

Для ефективної організації інтерфейсу та управління даними в класі MainForm реалізовано два вбудовані класи:

- TreeViewManager використовується для організації та управління елементами у TreeView, що відображає ієрархічну структуру груп та облікових записів. Він відповідає за додавання, видалення та оновлення вузлів дерева, а також обробляє дії користувача, пов'язані з вибором та розкриттям вузлів.
- ListViewManager: керує елементом ListView, що відображає список облікових записів в обраній групі. Він забезпечує додавання, видалення та оновлення елементів списку, а також обробляє дії користувача зі списком.

Ключові поля класу:

- MasterKey: призначений для керування майстер-ключем і виконання криптографічних операцій над захищеними даними.
- Config: об'єкт для зберігання та управління налаштуваннями програми.
- GroupEntry: використовується як головний елемент структури, що об’єднує групи та відповідні облікові записи.

Після детального опису класів системи та їх взаємозв'язків на рівні екземплярів було створено діаграму класів, яка відображає статичну структуру програмного забезпечення. Дана діаграма наочно відображає архітектурну побудову системи та може слугувати основою для подальшого створення або вдосконалення програмних рішень у наступних проєктах (див. рис. 2.11).

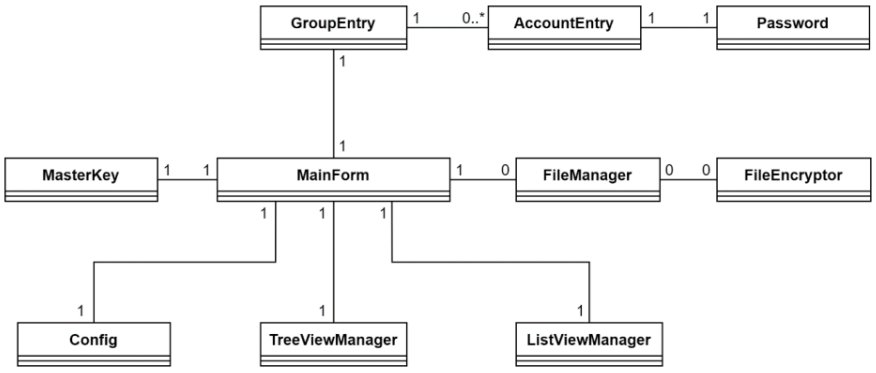

Рисунок. 2.11 – Діаграма класів менеджера паролів

2.7 Розробка інтерфейсу користувача

У програмних рішеннях, що працюють із чутливою інформацією, особливе значення має спосіб взаємодії користувача із системою. Хоча може здатися, що користувачі менеджерів паролів володіють необхідними знаннями для їх використання, саме зрозумілий та логічно побудований інтерфейс визначає, наскільки безпечно й ефективно буде застосовуватися програмний продукт у

повсякденній роботі. Саме тому під час розробки інтерфейсу особлива увага приділялася простоті навігації, зрозумілості елементів керування та мінімізації кількості дій, необхідних для виконання основних операцій.

Після відкриття програми користувач бачить головне вікно системи, показане на рисунку 2.12. Дане вікно є основною робочою областю менеджера паролів і містить перелік усіх збережених записів, панель навігації та елементи керування пошуком і сортуванням.

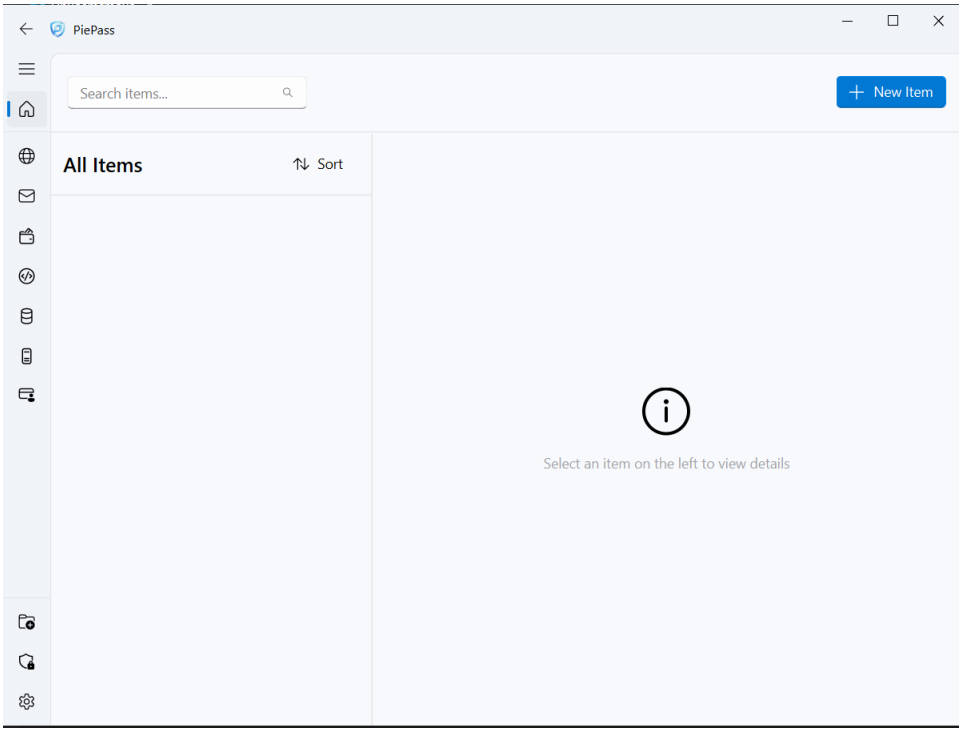

Рисунок 2.12 – Початкове вікно застосунку менеджера паролів

Ліва бічна панель призначена для навігації між різними типами записів, такими як облікові дані для вебсайтів, електронна пошта, криптогаманці, API-ключі, серверні доступи та банківські картки (див. рисунок 2.13). Такий підхід дозволяє швидко фільтрувати записи за їх призначенням і спрощує пошук необхідної інформації.

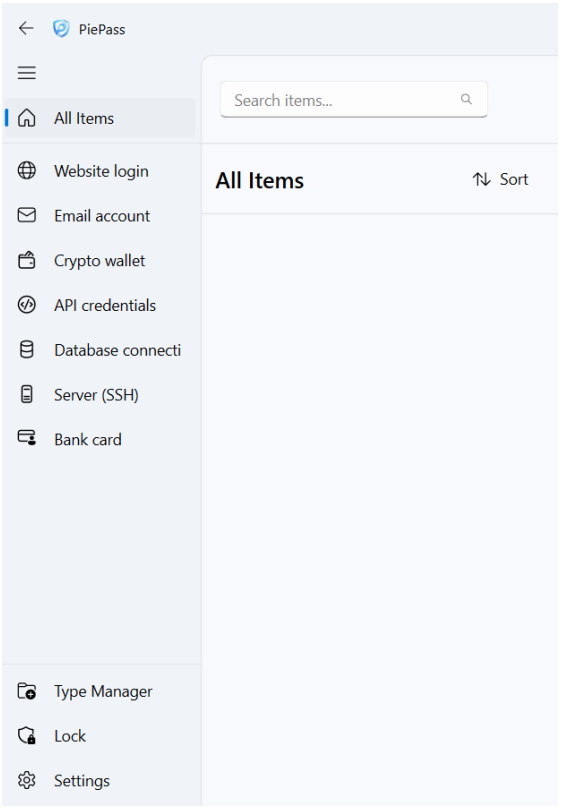

Рисунок 2.13 – Панель навігації та групування записів

У верхній області інтерфейсу розміщені засоби для пошуку й сортування інформації, що спрощує роботу з великими обсягами даних. Користувач може виконувати пошук за назвою запису або іншими відкритими параметрами, не розкриваючи конфіденційні значення.

Програмний продукт дозволяє створювати нові записи, змінювати наявні та видаляти непотрібні дані. Створення нового запису реалізовано за допомогою окремого діалогового вікна, яке зображене на рисунку 2.14. У цьому вікні користувач вводить назву запису, обирає його тип та заповнює необхідні поля, такі як логін, пароль, URL або додаткові нотатки.

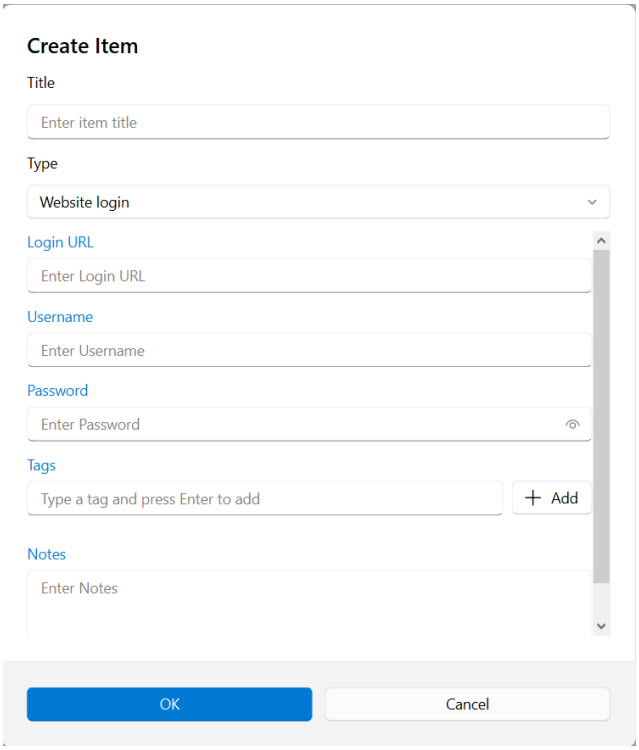

Рисунок 2.14 – Вікно створення нового запису

Після введення всієї необхідної інформації користувач завершує операцію шляхом підтвердження збереження. Якщо дані було опрацьовано без помилок, система повідомляє про успішне виконання дії відповідним інформаційним повідомленням.

Для перегляду детальної інформації про збережений запис використовується панель відображення запису, представлена на рисунку 2.15. У даній панелі відображаються всі поля запису, а також кнопки керування, які дозволяють редагувати, переглядати історію змін або видаляти запис.

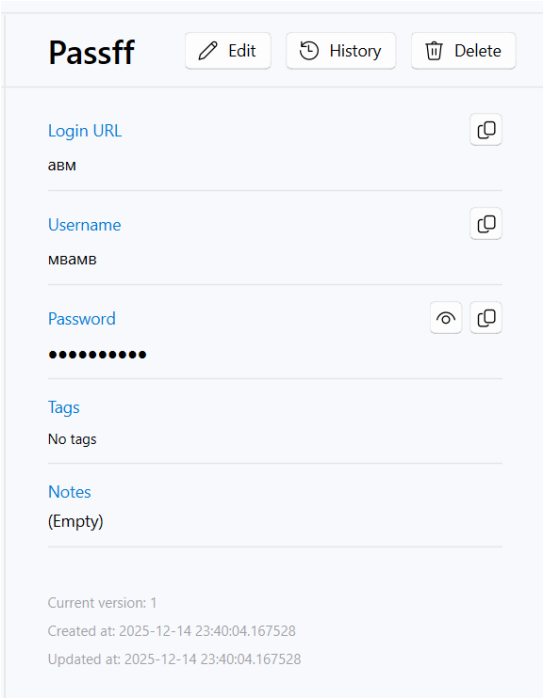

Рисунок 2.15 – Вікно перегляду детальної інформації запису

Окрему увагу приділено захисту конфіденційних даних. Пароль та інші секретні значення за замовчуванням приховані і можуть бути відображені лише за ініціативою користувача. Також реалізована можливість копіювання значень із автоматичним очищенням буфера обміну через заданий проміжок часу.

У разі видалення запису система додатково запитує підтвердження дії, щоб запобігти випадковій втраті даних (див. рисунок 2.16).

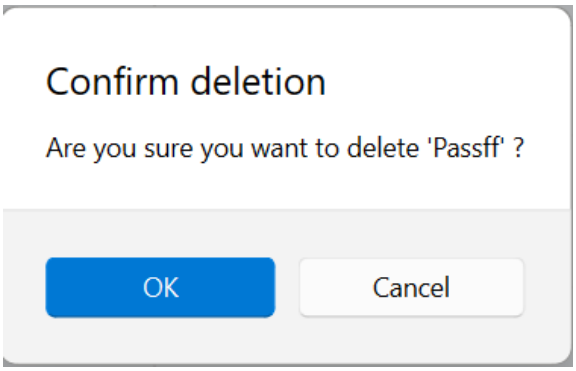

Рисунок 2.16 – Діалог підтвердження видалення запису

Після успішного виконання операцій редагування або видалення користувач отримує відповідне інформаційне повідомлення, яке підтверджує

результат виконаної дії (див. рисунок 2.17).

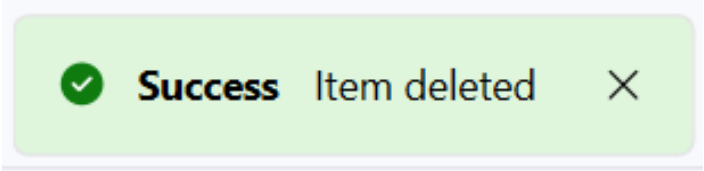

Рисунок 2.17 – Повідомлення про успішне виконання операції

Окремим компонентом інтерфейсу є вікно налаштувань програмної системи, зображене на рисунку 2.18. У ньому користувач може змінювати параметри персоналізації, зокрема тему оформлення та акцентний колір. Також у розділі безпеки доступні налаштування автоматичного блокування сховища та очищення буфера обміну.

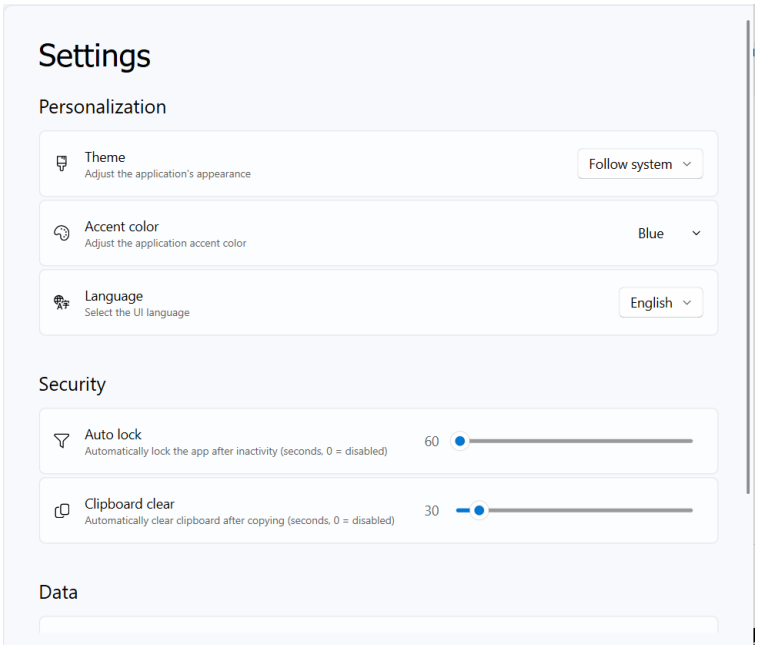

Рисунок 2.18 – Вікно налаштувань програмної системи

У результаті можна стверджувати, що візуальну частину програмного продукту повністю проілюстровано, а всі ключові елементи інтерфейсу та доступні користувачеві можливості були розглянуті.

3 ТЕСТУВАННЯ, ВПРОВАДЖЕННЯ ТА ПІДТРИМКА

Перевірка працездатності програмного продукту є невід'ємною частиною процесу його створення, оскільки саме на цьому етапі визначається правильність роботи реалізованого функціоналу, надійність системи та ступінь відповідності результату початково визначеним вимогам. Для менеджера паролів тестування має особливе значення, оскільки помилки у роботі з конфіденційними даними можуть призвести до втрати інформації або зниження рівня безпеки.

3.1 Тестування менеджера паролів

Процес тестування полягає у перевірці реакції програмної системи на різні сценарії використання, включаючи як стандартні дії користувача, так і нестандартні або помилкові ситуації. У загальному випадку тестування охоплює два основні етапи: ініціювання виконання програмного коду та аналіз результатів його роботи. У разі автоматизованого тестування ці дії виконуються програмними засобами, тоді як при ручному тестуванні перевірка здійснюється безпосередньо розробником або тестувальником.

3.1.1 Види та план тестування

Для розробленої програмної системи менеджера паролів було використано поєднання ручних та автоматизованих методів тестування. Основний акцент було зроблено на перевірці коректності виконання операцій зі створення, збереження, редагування та видалення записів, а також на стабільності роботи механізмів шифрування та розблокування сховища. Оскільки система є desktop-застосунком, значну увагу було приділено тестуванню графічного інтерфейсу користувача.

У ході тестування було виділено декілька основних видів перевірок, які є доцільними для даного типу програмної системи:

- ad hoc-тестування;
- функціональне тестування;
- модульне тестування.

На початкових етапах створення програмної системи, а також у процесі об’єднання її окремих модулів, використовувалося неформальне тестування типу Ad hoc. Даний вид тестування не передбачає наявності формалізованих тестових сценаріїв і базується на досвіді розробника та припущеннях щодо можливих помилок. У процесі такого тестування перевірялися нестандартні дії користувача, наприклад введення некоректних даних, переривання роботи програми або повторне виконання одних і тих самих операцій.

Перевірка функціональності програмної системи проводилася з метою встановлення відповідності роботи менеджера паролів визначеним вимогам до його можливостей. У межах цього виду тестування перевірялися основні сценарії використання системи: створення нового сховища, розблокування доступу, додавання записів різних типів, перегляд і редагування збережених даних, а також видалення записів. Окремо тестувалася робота механізмів пошуку та сортування, що є важливими для зручної роботи з великою кількістю записів.

Модульне тестування застосовувалося для перевірки окремих логічних компонентів системи, зокрема класів, відповідальних за роботу з базою даних, шифрування даних та обробку введеної інформації. Застосування цього методу дало змогу виявити потенційні помилки ще на початкових стадіях розробки, а також перевірити правильність роботи взаємозв’язків між окремими модулями програмної системи. Модульне тестування є особливо корисним при подальшому розвитку системи, оскільки дозволяє перевіряти нові зміни без ризику порушення вже реалізованого функціоналу.

Окрему увагу було приділено тестуванню інтерфейсу користувача. Перевірялася коректність відображення елементів інтерфейсу, логічність розташування кнопок і полів, а також узгодженість візуальних компонентів між різними вікнами програми. Такий підхід дозволив переконатися, що інтерфейс є

інтуїтивно зрозумілим і не викликає труднощів у користувачів без спеціальної технічної підготовки.

3.1.2 Розробка тестових сценаріїв

Розробка тестових сценаріїв є важливим етапом перевірки програмної системи, оскільки дозволяє систематизувати процес тестування та оцінити відповідність реалізованого функціоналу поставленим вимогам. Тестові сценарії описують послідовність дій користувача або системи, умови виконання тесту та очікуваний результат, що дає змогу виявити помилки ще на етапі експлуатаційної перевірки програмного продукту.

Функціональне тестування дозволяє перевірити, наскільки повно менеджер паролів реалізує основні функціональні вимоги за різних умов використання. Функціональні вимоги визначають, які саме задачі повинна виконувати програмна система та яким чином вона має реагувати на дії користувача. Для менеджера паролів до ключових функціональних вимог належать:

- функціональна придатність;
- коректність та точність обробки даних;
- стабільність взаємодії між компонентами;
- відповідність вимогам інформаційної безпеки;
- захищеність конфіденційних даних.

Під час тестування програмної системи менеджера паролів особливу увагу було приділено сценаріям, пов’язаним зі створенням та розблокуванням сховища, додаванням нових записів, редагуванням існуючих даних та обробкою помилкових дій користувача. Зокрема, було виявлено проблему з некоректною обробкою окремих форм введення даних у разі їх повторного відкриття. Після аналізу програмного коду з’ясувалося, що в деяких випадках не виконувалося коректне очищення стану інтерфейсних компонентів після закриття діалогових

вікон. Виявлений дефект було зафіксовано у Bug-report та усунено на етапі доопрацювання системи.

У межах функціонального тестування було перевірено відповідність реалізованих функцій вимогам, визначеним у попередніх розділах роботи. Основні результати функціонального тестування наведені у таблиці 3.1.

Таблиця 3.1 – Аналіз функціональної відповідності менеджера паролів

№	Функціональна вимога	Опис тестового сценарію	Очікуваний результат	Фактичний результат
1	Створення сховища	Створення нового сховища з головним паролем	Сховище створюється та зберігається	Відповідає
2	Розблокування сховища	Введення коректного головного пароля	Надано доступ до даних	Відповідає
3	Додавання запису	Створення нового запису з обов’язковими полями	Запис збережено	Відповідає
4	Редагування запису	Зміна значень існуючого запису	Дані оновлено	Відповідає
5	Видалення запису	Видалення вибраного запису	Запис видалено	Відповідає
6	Обробка помилок	Введення некоректних даних	Відображено повідомлення про помилку	Відповідає

Окрім функціонального тестування, було проведено модульне тестування окремих компонентів програмної системи. Модульне тестування передбачає перевірку працездатності окремих класів і методів без урахування повної взаємодії з іншими частинами системи. Застосування цього підходу дає можливість своєчасно знаходити недоліки в роботі системи та знижувати ймовірність появи серйозних помилок під час об’єднання окремих компонентів.

У межах даної роботи модульне тестування застосовувалося для перевірки логіки роботи з базою даних, механізмів шифрування та обробки введених користувачем даних. Тести виконувалися ізольовано від інтерфейсу користувача, що дозволило оцінити коректність роботи окремих методів та класів. Основною перевагою модульних тестів є їхня швидкість виконання та можливість багаторазового використання під час подальшого розвитку програмної системи.

Використання підходу розробки, орієнтованої на тестування, дає змогу

вибудувати систему автоматизованих тестів і сформувати практичні вміння у сфері тестування програмного забезпечення [35–36].

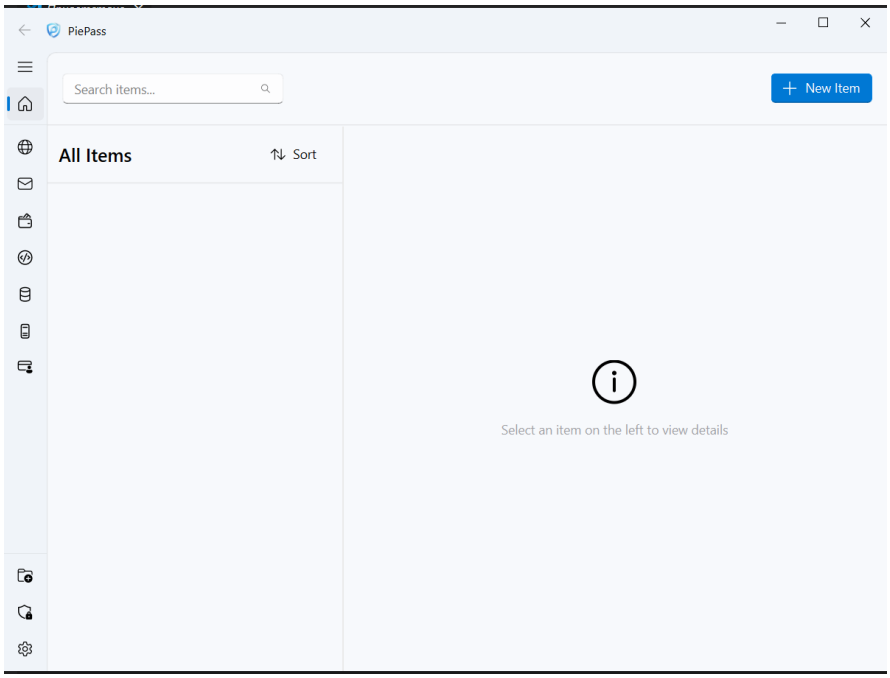

Рисунок 3.1 – Загальний вигляд інтерфейсу менеджера паролів

Одним із важливих аспектів проєктування початкового інтерфейсу програмної системи є логічна узгодженість основних елементів керування та їх візуальне сприйняття користувачем. На початковому екрані застосунку, проілюстрованому на відповідному рисунку, використано єдиний стиль оформлення, що поєднує іконку програми, заголовок вікна та елементи введення даних. Наявність фірмової іконки у верхній частині вікна забезпечує однозначне розпізнавання програмного продукту та підкреслює його належність до єдиного дизайнерського концепту. Використання цього графічного елемента позитивно впливає на сприйняття застосунку користувачем і підтримує узгодженість усіх складових інтерфейсу.

Так як розроблений застосунок є менеджером паролів із криптографічним захистом даних, було прийнято рішення у цьому підрозділі зосередитися саме на реалізації тих класів, які формують відмінні особливості системи: захист майстер-паролем, виведення ключів, шифрування даних, керування життєвим

циклом сховища та безпечна робота з базою даних. Усі фрагменти коду ключових компонентів наведено у додатку В.

Першим ключовим компонентом, який визначає безпеку застосунку, є модуль керування сховищем Vault, що реалізований у сервісі VaultService. Його основним завданням є створення сховища, ініціалізація криптографічних параметрів, розблокування сховища за майстер-паролем та підготовка ключа даних для подальшого шифрування/розшифрування записів.

У першу чергу необхідно забезпечити наявність бази даних та механізму керування транзакціями, оскільки сховище зберігається у SQLite. Для цього реалізовано клас DatabaseManager, який відповідає за ініціалізацію двигуна SQLAlchemy, створення фабрики сесій та безпечне виконання операцій у транзакційному контексті. Структуру ініціалізації підключення та контекстний менеджер сесії показано на рис. 3.2.

```
class DatabaseManager:
    _instance = None
    engine = None
    _session_factory = None

    def __new__(cls):
        if cls._instance is None:
            cls._instance = super().__new__(cls)
        return cls._instance

    def initialize(self, db_path: str = None):

        if self.engine is not None:
            return

        if db_path is None:

            app_data_dir = Path.home() / ".pipass"
            app_data_dir.mkdir(parents=True, exist_ok=True)
            db_path = str(app_data_dir / DATABASE_NAME)
```

Рисунок 3.2 – Клас DatabaseManager

Далі для реалізації криптографічного захисту потрібно сформувати ключ шифрування на основі майстер-пароля. З огляду на можливість несанкціонованого доступу до файлу бази даних у автономному режимі, у програмному рішенні реалізовано механізм формування ключа на основі алгоритму scrypt. Використання цього підходу значно підвищує стійкість до атак повного перебору за рахунок високих вимог до обчислювальних ресурсів і обсягу пам’яті. За генерацію солі та отримання ключа відповідає клас

KdfManager. Відповідні фрагменти наведено на рис. 3.3.

```python
class KdfManager:


    @staticmethod
    def generate_salt() -> bytes:

        return secrets.token_bytes(SCRYPT_SALT_LENGTH)

    @staticmethod
    def derive_key(password: str, salt: bytes,
                   n: int = SCRYPT_N,
                   r: int = SCRYPT_R,
                   p: int = SCRYPT_P) -> bytes:

        password_bytes = password.encode('utf-8')

        kdf = Scrypt(
            salt=salt,
            length=SCRYPT_KEY_LENGTH,
            n=n,
            r=r,
            p=p,
            backend=default_backend()
        )

        key = kdf.derive(password_bytes)
        return key
```

Рисунок 3.3 – Клас KdfManager

Наступним етапом є реалізація симетричного шифрування даних. У програмі використано режим AES-GCM, який забезпечує не лише конфіденційність, а й перевірку цілісності даних (завдяки тегу автентифікації). За операції шифрування та розшифрування відповідає клас CryptoManager, який формує результуючий запис у форматі version:nonce:ciphertext та дозволяє розширювати формат у майбутньому. Основну частину реалізації наведено на рис. 3.4.

```python
class CryptoManager:

    FORMAT_VERSION = "v1"

    def __init__(self, vdk: Optional[bytes] = None):

        self._vdk = vdk
        self._aesgcm = AESGCM(vdk) if vdk else None

    def set_vdk(self, vdk: bytes):

        if len(vdk) != AES_KEY_LENGTH:
            raise ValueError(f"VDK length must be{AES_KEY_LENGTH}bytes")
        self._vdk = vdk
        self._aesgcm = AESGCM(vdk)

    def clear_vdk(self):
        self._vdk = None
        self._aesgcm = None

    @property
    def is_unlocked(self) -> bool:

        return self._vdk is not None
```

Рисунок 3.4 – Клас CryptoManager

Після реалізації криптографічних примітивів можна описати основний процес створення сховища. У VaultService застосовується така схема: генерується сіль KDF, обчислюється ключ шифрування ключа (KEK) із майстер-пароля, після чого генерується ключ даних (VDK). Важливо, що VDK не зберігається у відкритому вигляді — він шифрується KEK та зберігається в таблиці метаданих сховища (VaultMeta). Це забезпечує принцип: без майстер-пароля неможливо відновити ключ даних і розшифрувати секрети. Фрагмент процесу створення сховища наведено на рис. 3.5.

```python
def create_vault(self, master_password: str) -> Tuple[bool, str]:
    if not master_password:
        return False, "Master password cannot be empty"

    try:
        score, description = self._kdf_manager.check_password_strength(master_password)
        if score < 30:
            return False, f"Password strength is insufficient: {description}"

        self.initialize_database()

        kdf_salt = self._kdf_manager.generate_salt()

        kek = self._kdf_manager.derive_key(
            master_password, kdf_salt,
            SCRYPT_N, SCRYPT_R, SCRYPT_P
        )

        vdk = crypto_manager.generate_vdk()

        enc_vdk = crypto_manager.encrypt_vdk(vdk, kek)
```

Рисунок 3.5 – Генерування ключа даних (VDK)

Отже, у розробленому менеджері паролів ключові класи (DatabaseManager, KdfManager, CryptoManager, VaultService) утворюють базову безпекову підсистему: вони забезпечують транзакційну роботу з БД, стійке виведення ключа з майстер-пароля та шифрування конфіденційних даних.

Наступним необхідним функціональним блоком розробленого менеджера паролів є модуль керування записами сховища, який забезпечує створення, редагування, видалення та версіонування облікових даних. Наявність гнучкої системи типів і полів зустрічається порівняно рідко серед локальних менеджерів паролів, що дозволяє виділити розроблений застосунок серед аналогів.

Першим ключовим елементом цього модуля є сервіс ItemService, який

відповідає за CRUD-операції над записами сховища. Даний сервіс координує взаємодію між рівнем бізнес-логіки, репозиторіями доступу до даних та криптографічним модулем. Структуру основних методів сервісу наведено на рис. 3.6.

```python
class ItemService:

    def __init__(self):
        pass

    def create_item(
        self,
        item_type_id: int,
        title: str,
        field_values: Dict[int, str]
    ) -> Tuple[bool, str, Optional[int]]:

        if not crypto_manager.is_unlocked:
            return False, "Vault is locked", None

        if not title or not title.strip():
            return False, "Title cannot be empty", None

        title = title.strip()
        if len(title) > 200:
            return False, "Title is too long (max 200 characters)", None
```

Рисунок 3.6 – Сервіс ItemService

Для забезпечення гнучкості структури даних у системі реалізовано окремий сервіс ItemTypeService, який відповідає за створення та редагування типів записів (наприклад, облікові записи сайтів, електронна пошта, банківські картки тощо) та пов’язаних із ними полів. Основна логіка ініціалізації типів наведена на рис. 3.7.

```python
class ItemTypeService:

    def __init__(self):
        pass

    def initialize_builtin_types(self) -> bool:
        try:
            builtin_types_data = [
                {
                    "name": ITEM_TYPE_WEBSITE_LOGIN,
                    "icon": "GLOBE",
                    "order_index": 1,
                    "fields": [
                        {"display_name": "Login URL", "key": "login_url", "field_type": "url", "is_secret": False, "order_index": 0},
                        {"display_name": "Username", "key": "username", "field_type": "text", "is_secret": False, "required": True, "orde
                        {"display_name": "Password", "key": "password", "field_type": "password", "is_secret": True, "required": True,
                         "copy_behavior": COPY_BEHAVIOR_NORMAL, "order_index": 2},
                        {"display_name": "Tags", "key": "tags", "field_type": "tag_list", "is_secret": False,
                         "copy_behavior": COPY_BEHAVIOR_DISABLED, "order_index": 3},
                        {"display_name": "Notes", "key": "notes", "field_type": "multiline", "is_secret": False,
                         "copy_behavior": COPY_BEHAVIOR_DISABLED, "order_index": 4},
                    ]
                },
                {
```

Рисунок 3.7 – Сервіс ItemTypeService

Для підвищення надійності та захисту від втрати даних у системі реалізовано механізм версіонування записів. Кожна зміна облікового запису може бути збережена як окрема версія у зашифрованому вигляді. За даний функціонал відповідає сервіс ItemVersionService. Основну логіку створення версії наведено на рис. 3.8.

```python
class ItemVersionService:

    def __init__(self):
        pass

    def create_version_snapshot(
        self,
        item_id: int,
        field_values: Dict[int, str]
    ) -> Tuple[bool, str]:

        if not crypto_manager.is_unlocked:
            return False, "Vault is locked"

        try:
            with db_manager.session_scope() as session:
                version_repo = ItemVersionRepository(session)
                item_repo = ItemRepository(session)

                item = item_repo.get_by_id(item_id)
                if not item:
                    return False, "Item not found"
```

Рисунок 3.8 – Сервіс ItemVersionService

У результаті проведення комплексу тестів було здійснено всебічну перевірку коректності роботи менеджера паролів щодо заявлених функцій, що дало змогу оперативно ідентифікувати недоліки та усунути їх на ранніх етапах. Отримані результати також засвідчили надійність і стабільність програмного рішення. Архітектура сервісного рівня реалізованого менеджера паролів орієнтована на зручне управління даними, підтримує механізми версійності та забезпечує єдину точку обробки бізнес-логіки системи. Запропонований підхід дозволяє легко розширювати функціональність системи та інтегрувати нові типи облікових даних без порушення цілісності архітектури.

3.2 Розгортання менеджера паролів та системні вимоги

Розгортання програмної системи менеджера паролів є завершальним етапом розробки, який передбачає підготовку програмного продукту до практичного використання кінцевими користувачами. Поточний етап передбачає перевірку правильності встановлення програмного продукту, його безперебійного функціонування в межах визначених операційних платформ, а також підтвердження відповідності заявленим мінімальним апаратним і програмним характеристикам.

Розроблений менеджер паролів є desktop-застосунком, що не потребує складного серверного середовища або підключення до зовнішніх хмарних сервісів. Інформація, що належить користувачу, розміщується безпосередньо на його власному пристрої, що зменшує ризики несанкціонованого доступу та сприяє збереженню приватності. Такий підхід також спрощує процес розгортання, оскільки система не залежить від доступності мережі або сторонніх серверів.

Процес розгортання програмної системи включає встановлення середовища виконання мови програмування Python та необхідних бібліотек, що використовуються у проєкті. Для коректної роботи менеджера паролів також необхідно забезпечити наявність бібліотек для роботи з графічним інтерфейсом користувача та криптографічними алгоритмами. Після встановлення всіх залежностей користувач може запускати програму без додаткових налаштувань.

Для стабільної роботи менеджера паролів необхідно, щоб апаратне та програмне забезпечення користувача відповідало мінімальним системним вимогам, наведеним нижче.

Базові вимоги до системи:

- операційна система: Windows 10 / Linux / macOS;
- процесор: двоядерний з тактовою частотою від 1,6 ГГц;
- оперативна пам’ять: не менше 4 ГБ;
- вільне місце на диску: від 200 МБ;

Рекомендовані системні вимоги:

- операційна система: Windows 11 / сучасні версії Linux або macOS;
- процесор: чотириядерний з тактовою частотою від 2,0 ГГц;
- оперативна пам’ять: 8 ГБ і більше;
- вільне місце на диску: від 500 МБ;
- екран з роздільною здатністю не менше 1920×1080.
- встановлене середовище виконання Python версії 3.10 або вище.

Окрему увагу під час розгортання програмної системи було приділено безпеці збережених даних. Усі конфіденційні значення шифруються перед записом у базу даних, а доступ до сховища можливий лише після успішного введення головного пароля. Таким чином, навіть у разі копіювання файлу бази даних сторонніми особами доступ до інформації без пароля є неможливим.

Після встановлення та первинного запуску менеджера паролів користувач має можливість одразу створити нове сховище або відкрити вже існуючу базу даних. Надалі програмна система не потребує складного обслуговування та може використовуватися автономно протягом тривалого часу.

У межах даної роботи верифікація менеджера паролів виконувалася шляхом зіставлення реалізованого функціоналу з вимогами, сформульованими у розділі аналізу предметної області та постановки завдання. Особлива увага приділялася перевірці відповідності механізмів зберігання даних, автентифікації користувача та захисту конфіденційної інформації вимогам інформаційної безпеки.

Одним із ключових аспектів верифікації стала перевірка логіки роботи зі сховищем. Було підтверджено, що доступ до конфіденційних даних можливий лише після успішного введення головного пароля, а всі секретні значення зберігаються виключно у зашифрованому вигляді. Це відповідає поставленій меті забезпечення захисту персональної інформації користувача.

Також у процесі верифікації було проаналізовано відповідність реалізованої архітектури проєктним рішенням, визначеним на етапі

проєктування системи. Було підтверджено, що програмна система побудована за багатошаровою архітектурою, у якій логіка роботи з даними, інтерфейс користувача та механізми доступу до бази даних є логічно відокремленими. Такий підхід забезпечує зручність супроводу та можливість подальшого розширення функціоналу.

Верифікація інтерфейсу користувача полягала у перевірці відповідності реалізованих вікон та елементів керування описаним варіантам використання. Було підтверджено, що всі основні сценарії взаємодії користувача з програмною системою реалізовані коректно, а інтерфейс є інтуїтивно зрозумілим і не потребує спеціальної підготовки для роботи з ним.

Окрім цього, було перевірено відповідність програмної системи нефункціональним вимогам, зокрема вимогам до стабільності, швидкодії та автономності роботи. Менеджер паролів продемонстрував коректну роботу при багаторазовому виконанні основних операцій, а також відсутність збоїв під час тривалого використання.

Отже, результати верифікації підтверджують, що розроблена програмна система менеджера паролів відповідає поставленим вимогам та може вважатися коректно реалізованою. Це дозволяє зробити висновок про готовність програмного продукту до практичного використання та подальшого розвитку.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

Охорона праці є важливою складовою забезпечення безпечних умов професійної діяльності, оскільки спрямована на збереження життя, здоров’я та працездатності працівників у процесі виконання ними виробничих завдань.

4.1 Охорона праці

Неналежний рівень організації охорони праці може негативно впливати на фізичний та психоемоційний стан працівника, що, у свою чергу, позначається на якості виконуваних робіт, продуктивності праці та загальному стані здоров’я. Саме тому дотримання вимог охорони праці є необхідною умовою ефективної та безпечної діяльності фахівців у галузі інформаційних технологій. Для того щоб запобігти цьому, потрібно дотримуватись ряду вимог та стандартів. В НПАОП 0.00-7.15-18 [44] описуються вимоги, яких потрібно дотримуватись при роботі з екранними пристроями.

У межах даного розділу розглядаються основні небезпечні та шкідливі чинники, які можуть виникати під час роботи з програмною системою менеджера паролів у процесі її розробки та використання.

Мікроклімат виробничих приміщень має підтримуватись на постійному рівні та відповідати вимогам Санітарних норм мікроклімату виробничих приміщень ДСН 3.3.6.042-99 [45]. У процесі розробки програмного забезпечення, зокрема програмної системи менеджера паролів, працівник тривалий час перебуває у закритому приміщенні та працює за персональним комп’ютером, що зумовлює підвищені вимоги до параметрів мікроклімату.

Для приміщень, у яких здійснюється робота з електронно-обчислювальною технікою, повинні забезпечуватися такі умови мікроклімату, які створюють комфортне теплове відчуття для працівника та не викликають перенапруження систем терморегуляції організму. У випадках, коли з технічних або економічних причин забезпечення оптимальних параметрів мікроклімату є неможливим,

допускається встановлення гранично допустимих значень показників мікроклімату, які не перевищують безпечних для здоров’я людини меж та регламентуються відповідними нормативними документами [37].

Для приміщення, у якому здійснюється розробка та використання спеціалізованої програмної системи керування паролями, характер виконуваних робіт належить до категорії робіт легкої важкості (Іб) за фізичним навантаженням. Така класифікація обумовлена тим, що основна діяльність працівника пов’язана з розумовою працею, роботою за персональним комп’ютером та не передбачає значних фізичних зусиль або активних рухів.

Відповідно до вимог чинних санітарних норм, повітря робочої зони в приміщеннях, де виконуються роботи за допомогою електронно-обчислювальної техніки, не повинно містити шкідливих речовин у концентраціях, що перевищують встановлені гранично допустимі концентрації (ГДК) [38]. Дотримання цих вимог є необхідною умовою забезпечення безпечних і комфортних умов праці, а також збереження здоров’я працівників під час тривалої роботи за комп’ютером.

Контроль за станом повітряного середовища робочої зони повинен здійснюватися систематично з метою запобігання можливому перевищенню гранично допустимих концентрацій шкідливих речовин.

При розробці та використанні програмної системи менеджера паролів враховувалися вимоги охорони праці та пожежної безпеки. Основна діяльність пов’язана з експлуатацією електронно-обчислювальної техніки, що зумовлює необхідність дотримання правил безпечної організації робочого місця, норм мікроклімату, освітлення та допустимих рівнів шуму.

Також було враховано вимоги пожежної безпеки, оскільки персональні комп’ютери, периферійне обладнання та електричні мережі можуть бути потенційними джерелами займання. Робочі приміщення повинні бути оснащені справною електромережею, засобами первинного пожежогасіння, а також забезпечувати вільний доступ до шляхів евакуації.

При використанні електронно-обчислювальної техніки (ЕОМ) джерелом забруднення повітря може бути також іонізація молекул повітря, спричинена

електричними полями, які виникають в процесі роботи комп’ютерної техніки. Цей процес може впливати на склад повітря в робочій зоні, тому важливо враховувати такі чинники під час організації робочого місця програміста.

Згідно з вимогами до зорових робіт, в приміщенні, де проводяться роботи зі спеціалізованою системою керування паролями на підприємстві, визначено, що ці роботи відповідають IV розряду зорових робіт.

Контраст об’єкта з фоном має бути середнім, що дозволяє чітко бачити інформацію на екрані, не створюючи зайвого напруження для очей.

Характеристика фону має бути середньою, що гарантує оптимальні умови для сприйняття тексту та зображень на екрані комп’ютера.

Під час експлуатації електронно-обчислювальної техніки та супутнього обладнання можливе виникнення віброакустичних коливань, зокрема шуму, який негативно впливає на працездатність та концентрацію уваги працівника.

Встановлено, що основними джерелами шуму на робочих місцях є вентилятори блоку живлення персонального комп’ютера, системи охолодження центрального процесора та відеоадаптера, а також елементи вентиляційної системи приміщення.

Для зменшення негативного впливу віброакустичних коливань рекомендується використовувати сучасне комп’ютерне обладнання з низьким рівнем шуму, забезпечувати справний технічний стан систем охолодження, а також раціонально розміщувати робочі місця з урахуванням джерел шуму.

З метою забезпечення нормованих показників рівня шуму в приміщенні, де здійснюється робота зі спеціалізованою програмною системою керування паролями, передбачено реалізацію таких організаційно-технічних заходів:

Оздоблення стін та елементів інтер’єру шумопоглинальними матеріалами, зокрема використання перфорованих плит.

Періодичний контроль рівня шуму в робочій зоні з періодичністю не рідше одного разу на рік.

Розміщення робочих місць, оснащених електронно-обчислювальною технікою, рекомендується виконувати у приміщеннях з однобічним

розташуванням вікон. Вікна приміщень повинні бути обладнані засобами захисту від прямого сонячного випромінювання, зокрема шторами або жалюзі, що дозволяє регулювати рівень освітленості залежно від часу доби та погодних умов [39].

У разі розміщення робочих місць у приміщеннях, де наявні потенційно шкідливі або небезпечні виробничі чинники, такі робочі місця повинні бути організовані в ізольованих кабінетах, які обладнані системами природного освітлення та ефективної вентиляції.

Робочі місця, оснащені відеодисплейними терміналами, необхідно розміщувати на відстані не менше 1,5 м від стіни з віконними прорізами, не менше 1 м від інших стін, а також на відстані не менше 1,5 м одне від одного.

Раціональна організація робочих місць сприяє зниженню зорового навантаження, підвищенню комфорту під час роботи та забезпеченню безпечних умов праці для працівників, діяльність яких пов’язана з розробкою програмної системи менеджера паролів.

4.2 Планування заходів цивільного захисту на об’єкті у випадку надзвичайних ситуацій

Ефективність планування та реалізації заходів цивільного захисту у разі виникнення надзвичайних ситуацій безпосередньо впливає на рівень безпеки персоналу, збереження їх життя та здоров’я, а також на масштаби можливих матеріальних збитків. Для підприємств, діяльність яких пов’язана з експлуатацією електронно-обчислювальної техніки та розробкою програмного забезпечення, своєчасна підготовка до дій у надзвичайних ситуаціях є необхідною умовою стабільного та безпечного функціонування.

Відповідно до Кодексу цивільного захисту України, підготовка працівників підприємств незалежно від форми власності до дій у разі виникнення надзвичайних ситуацій здійснюється на основі заздалегідь розробленої системи організаційних, технічних та інформаційних заходів, спрямованих на захист населення і територій [40]. Ці заходи мають враховувати специфіку діяльності об’єкта,

можливі загрози та умови праці персоналу.

Для підприємств малого та середнього типу, зокрема тих, у яких здійснюється розробка та використання програмної системи менеджера паролів, система заходів цивільного захисту у випадку надзвичайних ситуацій включає:

- планування та реалізацію заходів щодо захисту працівників і об’єкта господарювання, з урахуванням можливих техногенних і природних загроз;
- розроблення планів дій у разі аварійних ситуацій, зокрема пожеж, відключення електропостачання та задимлення приміщень, з урахуванням рекомендацій Державної служби України з надзвичайних ситуацій;
- підтримання у постійній готовності сил і засобів, призначених для запобігання виникненню надзвичайних ситуацій та ліквідації їх наслідків;
- створення та підтримання необхідних матеріальних резервів, зокрема засобів пожежогасіння та аварійного освітлення;
- забезпечення своєчасного оповіщення персоналу про загрозу або виникнення надзвичайної ситуації та інформування щодо порядку дій.

Відповідно до статті 130 Кодексу цивільного захисту України, на підприємствах із чисельністю працівників 50 осіб і менше повинні розроблятися та затверджуватися інструкції щодо дій персоналу у разі загрози або виникнення надзвичайних ситуацій.

Розроблена інструкція не повинна суперечити положенням і вимогам Кодексу цивільного захисту України та інших чинних нормативно-правових актів у сфері цивільного захисту. Інструкція складається уповноваженою посадовою особою підприємства з питань цивільного захисту, погоджується та затверджується керівником підприємства і доводиться до відома всіх працівників під особистий підпис.

Окрім інструкцій, на малому підприємстві обов’язково розробляється план евакуації у разі пожежі або загрози вибуху, який розміщується на видимих місцях у приміщенні. Наявність такого плану дозволяє забезпечити швидку та впорядковану евакуацію персоналу у разі виникнення надзвичайної ситуації, що є

особливо важливим для приміщень із постійним перебуванням працівників [43].

Усі працівники підприємства повинні бути навчені діям у надзвичайних ситуаціях, чітко знати свої обов’язки та неухильно їх виконувати. Це в повній мірі стосується і адміністрації підприємства, яка в умовах надзвичайної ситуації повинна діяти злагоджено, приймати обґрунтовані рішення та забезпечувати координацію дій персоналу з метою збереження життя і здоров’я людей [42].

Одним з ключових аспектів є своєчасне оповіщення про загрозу або виникнення надзвичайної ситуації. При виявленні ознак пожежі, задимлення, витоку газу або інших небезпечних явищ працівник зобов’язаний негайно повідомити про це своєму безпосередньому керівнику або спеціально призначеній особі, відповідальній за цивільний захист. Це дозволяє оперативно ініціювати заходи щодо захисту персоналу та мінімізації наслідків події, включаючи виклик відповідних служб та початок евакуації [41].

У разі оголошення сигналу про надзвичайну ситуацію персонал підприємства повинен діяти відповідно до заздалегідь розробленого плану евакуації та інструкцій з цивільного захисту, що затверджені на підприємстві та доведені до відома всіх працівників під підпис. Працівники повинні чітко знати маршрути евакуації, місця збору за межами приміщення, а також правила користування засобами первинного пожежогасіння.

Розуміння персоналом своїх обов’язків, послідовність дій у надзвичайній ситуації, а також регулярне відпрацювання практичних навичок сприяють мінімізації негативних наслідків та створенню безпечного робочого середовища, що є необхідною умовою ефективної діяльності будь-якого підприємства, у тому числі і такого, що займається розробкою програмного забезпечення.

ВИСНОВКИ

У ході виконання магістерської роботи було досліджено підходи до розробки програмних систем для безпечного зберігання конфіденційної інформації та створено програмний продукт у вигляді desktop-менеджера паролів PiePass. Основною метою роботи було проєктування та реалізація програмної системи, яка поєднує високий рівень інформаційної безпеки з простотою та зручністю використання для кінцевого користувача.

На стартовому етапі роботи було проведено дослідження середовища застосування системи, у результаті якого виявлено основні потенційні загрози, пов’язані з використанням ненадійних паролів і небезпечними методами їх зберігання.

У межах даної роботи для менеджера паролів було сформульовано функціональні та нефункціональні вимоги, визначено основних акторів і ключові варіанти використання. Це дозволило побудувати логічну модель взаємодії користувача з програмною системою та закласти основу для подальшого проєктування архітектури.

Для реалізації було обрано багатошарову архітектуру, яка забезпечує чітке розмежування між інтерфейсом користувача, бізнес-логікою та рівнем доступу до даних. У процесі проєктування бази даних менеджера паролів PiePass було визначено ключові сутності та їх атрибути, що дозволило реалізувати гнучку модель зберігання даних. Усі секретні значення зберігаються у зашифрованому вигляді, а механізм версіонування записів забезпечує збереження історії змін і зменшує ризик втрати даних у разі помилкових дій користувача.

У роботі також було побудовано UML-діаграми, зокрема діаграму варіантів використання та діаграму класів, які наочно відображають структуру програмної системиьта взаємодію її основних компонентів. Це дозволило формалізувати архітектурні рішення та спростити реалізацію програмного продукту. Під час створення менеджера паролів застосовано мову програмування Python у поєднанні з фреймворком Qt для реалізації графічного

інтерфейсу користувача. Реалізований інтерфейс є інтуїтивно зрозумілим, логічно структурованим та орієнтованим на роботу з різними типами записів, такими як облікові дані, ключі доступу, нотатки та інші конфіденційні дані.

У процесі тестування програмної системи було проведено ad hoc-тестування, функціональне та модульне тестування, що дозволило перевірити коректність реалізації основних сценаріїв використання, виявити та усунути помилки, а також оцінити стабільність роботи системи. Результати тестування підтвердили відповідність менеджера паролів поставленим вимогам та його готовність до практичного використання.

Під час розгортання було встановлено, що менеджер паролів не потребує складного серверного середовища та може використовуватися автономно на персональному комп’ютері користувача. Це підвищує рівень приватності та робить програмний продукт зручним для щоденного використання. Проведена верифікація підтвердила відповідність реалізованої системи функціональним і нефункціональним вимогам.

Варто зазначити, що розроблений менеджер паролів є локальним програмним продуктом, який не залежить від хмарних сервісів і сторонніх серверів. Цей підхід має особливе значення для українських користувачів у сфері захисту персональних даних, оскільки забезпечує зберігання всієї конфіденційної інформації без її передачі за межі пристрою.

Отже, у результаті виконання кваліфікаційної роботи було створено повноцінний програмний продукт — менеджер паролів, який забезпечує надійне зберігання конфіденційної інформації, зручний доступ до неї та можливість подальшого розвитку та який є актуальним та корисним в умовах сучасного цифрового розвитку України. Створена система придатна для застосування як у щоденному користуванні, так і в професійній сфері, забезпечуючи підвищений рівень цифрового захисту для користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Методичні вказівки до виконання кваліфікаційної роботи магістра для здобувачів спеціальності 121 – Інженерія програмного забезпечення, всіх форм навчання / укладачі: Михалик Д. М., Цуприк Г. Б., Бревус В. М., Мудрик І. Я. Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2024. 44 с. URL: https://elartu.tntu.edu.ua/handle/lib/50316 (дата звернення: 05.12.2025).
2. Yavorskyy B., Yavorska E., Tsupryk H., Kinash R. Methods of constructing algorithms for comparative test statistical verification of mathematical models of bioobject responses to low-intensity stimuli. Scientific Journal of TNTU. Ternopil: TNTU, 2023. Vol. 112, No. 4. 82–90 p.
3. Олянін, Д., Цуприк, Г. (2025) Transformer Neural Networks in Industry 4.0 / Д. Олянін, Г. Цуприк, Т. Говорущенко, О. Багрій-Заяць, І. Андрущак // Computer Information Technologies in Industry 4.0: proceedings of the 3rd International Workshop (CITI-2025), Ternopil, Ukraine, 11–12 June 2025. – Ternopil : Ternopil Ivan Puluj National Technical University, 2025 (Scopus) https://ceur-ws.org/Vol-4057/ (дата звернення: 05.12.2025).
4. Tsupryk, H., Olianin, D. (2025). Vydobuvannia danyh z tekstu vykorystovuiuchy transformerni neironni merezhi [Data extraction from text using Transformer Neural Networks]. Information Technology: Computer Science, Software Engineering and Cyber Security, 125–130, DOI: https://doi.org/10.32782/IT/2025-2-13 (дата звернення: 05.12.2025).
5. ОЛЯНІН D., & ЦУПРИК Н. (2025). Огляд ролі трансформерних нейроних мереж у видобувані інформації із неструктурованих даних. Measuring and computing devices in technological processes, 82(2), 360–364. https://doi.org/10.31891/2219-9365-2025-82-52 (дата звернення: 05.12.2025).
6. Tsupryk H. LLM-based Extraction from Resumes / D. Olianin, H. Tsupryk // Advanced Technologies in Scientific Research: collection of scientific papers with proceedings of the 1st International Scientific and Practical Conference,

Rotterdam, Netherlands, 20–22 August 2025. – International Scientific Unity, 2025. – 72-76 p.

7. Massive password leaks expose billions of accounts worldwide. Associated Press, 2024. URL: https://apnews.com/article/2a758a40c398b0a68fb2371a522f70ed (дата звернення: 07.12.2025)
8. Internet users urged to change passwords after billions of logins exposed. The Guardian, 2025. URL: https://www.theguardian.com/technology/2025/jun/21/internet-users-advised-to-change-passwords-after-16bn-logins-exposed (дата звернення: 08.12.2025).
9. Stay safe in the digital era: password security. Institute of Chartered Accountants in England and Wales (ICAEW), 2025. URL: https://www.icaew.com/insights/viewpoints-on-the-news/2025/sep-2025/stay-safe-in-the-digital-era-password-security (дата звернення: 08.12.2025).
10. Password manager. Wikipedia. URL: https://en.wikipedia.org/wiki/Password_manager (дата звернення: 10.12.2025).
11. Stallings W. Cryptography and Network Security: Principles and Practice. 7th Edition. — Pearson Education, 2017. 41–48 p.
12. Anderson R. Security Engineering: A Guide to Building Dependable Distributed Systems. 3rd Edition. — Wiley, 2020. 87–95 p.
13. Password managers are an essential way to protect yourself from hackers — here's how they work. Business Insider, Price, Rob, 2017. 272 p.
14. How password managers work. 1Password. URL: https://1password.com/product/password-manager (дата звернення: 10.12.2025).
15. KeePass Password Safe: user documentation and limitations. URL: https://keepass.info/help/base/security.html (дата звернення: 10.12.2025).
16. Heartbleed bug: What you need to know. BBC. URL: https://www.bbc.com/news/technology-26969629 (дата звернення: 10.12.2025).
17. Authentication Cheat Sheet. OWASP Foundation. URL: https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet (дата

звернення: 10.12.2025).

18. SP 800-53 Rev. 5. Security and Privacy Controls. NIST, 2020. URL: https://csrc.nist.gov/publications/detail/sp/800-53/rev-5/final (дата звернення: 11.12.2025)
19. Dennis A., Wixom B. H., Roth R. M. Systems Analysis and Design. 7th Edition. — Wiley, 2021. 205–218 p.
20. What is a Use Case Diagram? Lucid Software Inc. URL: https://www.lucidchart.com/pages/uml-use-case-diagram (дата звернення: 11.12.2025).
21. Pressman R. S., Maxim B. R. Software Engineering: A Practitioner's Approach. 9th ed. — McGraw-Hill Education, 2020. 13–30 p.
22. Sommerville I. Software Engineering, 2016. 100–120 p.
23. Richards M. Software Architecture Patterns. — O'Reilly Media, 2015. 1–15 p.
24. An introduction to web applications architecture. The Open University, OpenLearn. URL: https://www.open.edu/openlearn/science-maths-technology/an-introduction-web-applications-architecture/content-section-1.2 (дата звернення: 11.12.2025).
25. Dubaj A. Multilayer architecture. URL: https://www.open.edu/openlearn/science-maths-technology/an-introduction-web-applications-architecture/content-section-1.2 (дата звернення: 11.12.2025).
26. Діаграма прецедентів. URL: https://www.wikiwand.com/uk/articles/Діаграма_прецедентів (дата звернення: 11.12.2025).
27. Як будувати UML-діаграми. URL: https://dou.ua/forums/topic/40575/ (дата звернення: 11.12.2025).
28. The Python Language Reference. Python Software Foundation. URL: https://docs.python.org/3/reference/ (дата звернення: 11.12.2025).
29. Lutz M. Learning Python. — O'Reilly Media, 2013. 68–69 p.
30. Qt for Application Development. Qt Documentation. URL: https://doc.qt.io/ (дата звернення: 11.12.2025).

31. SQLite Overview. SQLite Documentation. URL: https://www.sqlite.org/docs.html (дата звернення: 11.12.2025).
32. Cryptography Documentation. Python Cryptographic Authority. URL: https://cryptography.io/en/latest/ (дата звернення: 11.12.2025).
33. PyCharm Documentation. JetBrains. URL: https://www.jetbrains.com/pycharm/documentation/ (дата звернення: 11.12.2025).
34. Діаграма класів. URL: https://uk.wikipedia.org/wiki/Діаграма_класів (дата звернення: 11.12.2025)
35. Beck K. Test-Driven Development: By Example. — Boston: Addison-Wesley Professional, 2003. 240 p.
36. The Three Rules of TDD. Clean Coder Blog. URL: https://blog.cleancoder.com/uncle-bob/2014/12/17/TheCyclesOfTDD.html (дата звернення: 11.12.2025).
37. Охорона праці та безпека в надзвичайних ситуаціях. URL: https://elartu.tntu.edu.ua/bitstream/123456789/17983/1/Konspekt_lekcij_z_Ohorony_praci_u_galuzi_2015.pdf (дата звернення: 15.12.2025).
38. Наказ про затвердження Методичних вказівок «Критерії обґрунтування необхідності і визначення черговості розробки гігієнічних нормативів шкідливих речовин у повітрі робочої зони, атмосферному повітрі населених місць». 2004. URL: https://zakon.rada.gov.ua/rada/show/v0369282-04#Text (дата звернення: 15.12.2025).
39. Наказ про затвердження Загальних вимог стосовно забезпечення роботодавцями охорони праці працівників. 2012. URL: https://zakon.rada.gov.ua/laws/show/z0226-12#Text (дата звернення: 15.12.2025).
40. Кодекс цивільного захисту України : Закон України від 02.10.2013 № 5403-VI (ред. від 12.09.2025). URL: https://zakon.rada.gov.ua/laws/show/4574-20#n1163 (дата звернення: 15.12.2025).
41. Як діяти персоналу підприємства в надзвичайній ситуації. 2013. URL:

https://oppb.com.ua/content/yak-diyati-personalu-pidpriiemstva-v-nadzvichayniysituaciyi (дата звернення: 15.12.2025).

42. Стручок В.С. Методичний посібник для здобувачів освітнього ступеня «магістр» всіх спеціальностей денної та заочної (дистанційної) форм навчання «БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ». Тернопіль: ФОП Паляниця В. А., 156 с. URL: https://elartu.tntu.edu.ua/handle/lib/39196 (дата звернення: 15.12.2025).
43. Стручок В.С. Навчальний посібник «ТЕХНОЕКОЛОГІЯ ТА ЦИВІЛЬНА БЕЗПЕКА. ЧАСТИНА «ЦИВІЛЬНА БЕЗПЕКА». Тернопіль: ФОП Паляниця В. А., 156 с. URL: http://elartu.tntu.edu.ua/handle/lib/39424 (дата звернення: 15.12.2025).
44. Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями. Верховна Рада України – 2018. URL: https://zakon.rada.gov.ua/laws/show/z0508
45. Санітарні норми мікроклімату виробничих приміщень ДСН 3.3.6.042-99. Верховна Рада України – 1999. – URL: https://zakon.rada.gov.ua/rada/show/va042282-99

ДОДАТКИ

Додаток А

ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ
УНІВЕРСИТЕТ ІМЕНІ ІВАНА ПУЛЮЯ
КАФЕДРА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

ТЕХНІЧНЕ ЗАВДАННЯ

на розробку проекту
«Розробка програмної системи менеджера паролів з використанням методів криптографії»

Виконаве
ць: ст. гр.
СПм-61
Ящук В.О.

_____

Керівник проекту:
Цуприк Г.Б

_____

Тернопіль 2025

ЗМІСТ

1 ПІДСТАВИ ДО РОЗРОБКИ

Розробка проводиться у відповідності до графіку навчального плану підготовки магістрів за спеціальністю 121 «Інженерія програмного забезпечення», та згідно наказу на виконання дипломної роботи студента-магістра.

Тема проекту: «Розробка програмної системи менеджера паролів з використанням методів криптографії».

Термін виконання: до ....._____р.

2 ПРИЗНАЧЕННЯ ПРОГРАМНОГО ПРОДУКТУ

Програмний продукт призначений для безпечного зберігання, обробки та управління обліковими даними користувачів. Розроблена програмна система буде корисною для осіб та організацій, які потребують надійного захисту паролів та конфіденційної інформації. Використання сучасних криптографічних методів дозволяє підвищити рівень інформаційної безпеки та зменшити ризик несанкціонованого доступу до даних. Система забезпечує зручний інтерфейс для роботи з обліковими записами, що значно спрощує процес управління паролями та підвищує ефективність їх використання.

3 ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

3.1 Функціональні вимоги

У системі повинні бути реалізовані наступні можливості:

- реєстрація та автентифікація користувачів з використанням захищених механізмів перевірки облікових даних;
- створення, збереження та редагування облікових записів користувача (логінів і паролів);
- шифрування конфіденційних даних перед їх збереженням у базі даних;

- хешування паролів користувачів із використанням криптографічних хеш-функцій та солі;
- перегляд збережених облікових записів у розшифрованому вигляді після успішної автентифікації;
- видалення облікових записів та керування збереженими даними;
- пошук і фільтрація облікових записів за заданими параметрами;
- забезпечення захисту доступу до програмної системи від несанкціонованих дій.

3.2 Технічні вимоги

Для забезпечення коректної роботи програмної системи менеджера паролів серверна частина повинна відповідати таким вимогам:

- операційна система Windows;
- обсяг оперативної пам’яті – не менше ніж 8 ГБ;
- накопичувач SSD обсягом не менше 120 ГБ.

Клієнтська частина програмної системи повинна працювати на таких платформах:

- Windows;
- macOS;
- Linux.

3.3 Програмні вимоги

Для реалізації та функціонування програмної системи менеджера паролів використовуються такі програмні засоби:

- система керування базами даних – SQLite / MySQL;
- мова програмування – Python (версія 3.9 і вище);
- графічна бібліотека – Qt (PyQt / PySide);
- криптографічні бібліотеки – hashlib, cryptography;
- інструменти для керування середовищем виконання – virtualenv / venv.

4 ЕТАПИ РОЗРОБКИ

Розробка інформаційної системи проводиться в наступному порядку:

- аналіз предметної області, виявлення акторів та варіантів використання системи;
- вибір засобів розробки та архітектури системи;
- проектування архітектури;
- розробка програмного забезпечення системи;
- тестування інформаційної системи на реальних даних;
- оформлення супровідної документації;
- здача проекту.

Результати виконання кожного етапу проекту погоджуються з керівником проекту.

5 СУПРОВІДНА ДОКУМЕНТАЦІЯ

Для інформаційної системи повинні бути розроблені наступні документи:

- пояснювальна записка до проекту;
- презентація проекту;
- рецензія на проект;
- диск з проектом.

Пояснювальна записка до проекту оформляється згідно діючих вимог до нормоконтролю проектів.

6 ПОРЯДОК ЗДАЧІ ПРОЕКТУ

Розроблена система повинна відповідати вимогам, що складаються з перерахованих у п.3.1 цього документу характеристик.

Для здачі проекту необхідно підготувати весь перелік документів зазначений у п.5 цього документу.

Технічне завдання оформляється у відповідності з загальними вимогами до текстових конструкторських документів за ГОСТ 2.105-95 на аркушах формату А4.

Приймання проекту проводиться спеціально створеною комісією в термін зазначені в п.1 цього документу.

7 ВІДМІТКИ ПРО ВИКОНАННЯ ЕТАПІВ ТА ЗМІНИ В ПРОЕКТІ

Назва етапу	Відмітка*
Аналіз предметної області	
Архітектура системи	
Проектування база даних	
Використання системи	
Супровідна документація	

* відмітки про виконання етапу ставляться керівником проекту

Додаток Б

Тези

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

М А Т Е Р І А Л И
XIII НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

17-18 грудня 2025 року

ТЕРНОПІЛЬ
2025

УДК 004.4
Вікторія Ящук, Галина Цуприк, канд. техн. наук, доц.
Тернопільський національний технічний університет імені Івана Пулюя

АРХІТЕКТУРНІ ПІДХОДИ ДО ПРОЄКТУВАННЯ ПРОГРАМНИХ СИСТЕМ ДЛЯ БЕЗПЕЧНОГО КЕРУВАННЯ АВТЕНТИФІКАЦІЙНИМИ ДАНИМИ

Viktoriia Yashchuk, Halyna Tsupryk, PhD, Assoc.Prof.

ARCHITECTURAL APPROACHES TO DESIGNING SOFTWARE SYSTEMS FOR SECURE MANAGEMENT OF AUTHENTICATION DATA

У сучасних умовах розвитку цифрових технологій питання безпечного керування автентифікаційними даними стає особливо актуальним. Значне зростання кількості кібератак, пов'язаних із компрометацією користувацьких паролів, підкреслює необхідність створення систем, які забезпечують високий рівень захисту та надійності [1]. Дослідження у сфері програмної інженерії та прикладної криптографії демонструють, що більшість порушень безпеки виникає через недоліки архітектури та слабкі механізми перевірки автентифікаційних даних [2].

Архітектура програмних систем для керування автентифікаційними даними включає низку компонентів, що забезпечують зберігання, обробку та перевірку паролів у захищеному середовищі. Одним із ключових елементів є модуль шифрування, який відповідає за криптографічні перетворення даних. Сучасні підходи передбачають використання симетричних алгоритмів AES, а також асиметричних алгоритмів для розмежування прав доступу [3]. Застосування надійних алгоритмів дозволяє суттєво зменшити ризики несанкціонованого доступу навіть у випадку часткової компрометації системи.

Модуль зберігання автентифікаційних даних базується на зашифрованих сховищах, які унеможливлюють доступ до інформації у відкритому вигляді. Особлива увага приділяється структурі сховища, яка повинна бути оптимізована для швидкого доступу до записів та захищена від атак на метадані. У сучасних дослідженнях підкреслюється, що добре структурована архітектура програмної системи дозволяє мінімізувати площину атаки та зменшити вплив внутрішньої складності на загальну стійкість до загроз [4].

Дослідження трансформерних моделей у контексті Industry 4.0 демонструють їх здатність адаптивно працювати з великими потоками даних, що може бути використано й для оптимізації процесів обробки автентифікаційної інформації [5].

Надзвичайно важливим компонентом архітектури є модуль виявлення слабких та скомпрометованих паролів. Серед сучасних методів особливе місце займає фільтр Блума — ймовірнісна структура даних, що дозволяє ефективно визначати належність пароля до великої множини хешів з мінімальними ресурсними витратами [6]. Перевагою цього підходу є можливість локальної перевірки без передавання даних на сторонні сервери, що значно підвищує рівень конфіденційності. Показано, що моделі Transformer можуть ефективно виявляти структурні залежності та витягувати патерни зі складних неструктурованих даних, що робить їх перспективними для аналізу поведінкових характеристик користувачів у системах автентифікації [7].

Сучасні програмні системи для керування автентифікаційними даними також включають інструменти генерації надійних паролів, системи контролю повторного використання, механізми моніторингу спроб доступу та захисту від автоматизованих атак. Поєднання цих механізмів у рамках єдиної архітектури сприяє підвищенню рівня безпеки користувачів та зменшенню ризиків витоку інформації. У роботах, присвячених data mining, підкреслюється, що трансформерні моделі здатні забезпечувати високу якість аналізу великих масивів даних, що є важливим для локальних систем перевірки автентифікації без значного навантаження на мережеву інфраструктуру [8].

Перспективи розвитку таких систем пов'язані з інтеграцією адаптивних алгоритмів машинного навчання, здатних аналізувати поведінкові характеристики користувача, а також з

розширенням можливостей локальної обробки великих наборів даних. Використання цих технологій дозволить підвищити точність виявлення небезпечних дій, оптимізувати процес автентифікації та зменшити навантаження на центральні сервери [9].

Практичні результати застосування LLM-моделей свідчать про їх здатність виконувати обробку чутливих даних локально, забезпечуючи точне вилучення релевантних ознак, що може бути корисним для автономних модулів автентифікації [10].

Таким чином, архітектурні підходи до проєктування систем керування автентифікаційними даними відіграють важливу роль у забезпеченні загальної безпеки інформаційних систем. Використання поєднання криптографічних алгоритмів, структур даних та модульних архітектурних рішень дає змогу створювати ефективні, стійкі та масштабовані програмні засоби, здатні протидіяти сучасним кіберзагрозам.

Література

1. Verizon. Data Breach Investigations Report 2023. Verizon Enterprise Solutions, 2023. URL: https://www.verizon.com/business/resources/reports/dbir/

2. National Institute of Standards and Technology. NIST Special Publication 800-63B: Digital Identity Guidelines — Authentication and Lifecycle Management. U.S. Department of Commerce, 2017 (update 2023). URL: https://pages.nist.gov/800-63-3/sp800-63b.html

3. Anderson R. Security Engineering. 3rd ed. Wiley, 2020. С. 118.

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

5. Ke Y., Liang Y., Sha Z., Shi Z., Song Z. DPBloomfilter: Securing Bloom Filters with Differential Privacy. arXiv, 2025. URL: https://arxiv.org/abs/2502.00693

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

7. Tsupryk, H., Olianin, D. (2025). Overview of transformers role indata mining from unstructured data. International Scientific-technical journal «Measuring and computing devices in technological processes» 2025, Issue 2, 125–130, DOI: https://doi.org/10.31891/2219-9365-2025-82-52

8. Naldi M., Flamini M. Cybersecurity and behavioral authentication models. Journal of Information Security, 2023.

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

Додаток В

Лістинг 1.1 – crypro.py

```
import base64
import secrets
from typing import Optional

from cryptography.hazmat.primitives.ciphers.aead import AESGCM

from app.utils.constants import AES_KEY_LENGTH, AES_NONCE_LENGTH


class CryptoManager:

    FORMAT_VERSION = "v1"

    def __init__(self, vdk: Optional[bytes] = None):

        self._vdk = vdk
        self._aesgcm = AESGCM(vdk) if vdk else None

    def set_vdk(self, vdk: bytes):

        if len(vdk) != AES_KEY_LENGTH:
            raise ValueError(f"VDK length must
 be{AES_KEY_LENGTH}bytes")
        self._vdk = vdk
        self._aesgcm = AESGCM(vdk)

    def clear_vdk(self):
        self._vdk = None
        self._aesgcm = None

    @property
    def is_unlocked(self) -> bool:

        return self._vdk is not None

    @staticmethod
    def generate_vdk() -> bytes:

        return secrets.token_bytes(AES_KEY_LENGTH)

    @staticmethod
    def generate_nonce() -> bytes:

        return secrets.token_bytes(AES_NONCE_LENGTH)

    def encrypt(self, plaintext: str, aad: Optional[bytes] = None)
 -> str:
```

```
        if not self.is_unlocked:
            raise RuntimeError("Crypto manager is locked; set VDK
first")


        nonce = self.generate_nonce()


        plaintext_bytes = plaintext.encode('utf-8')
        ciphertext_with_tag = self._aesgcm.encrypt(nonce,
plaintext_bytes, aad)

        version_b64 =
base64.b64encode(self.FORMAT_VERSION.encode()).decode()
        nonce_b64 = base64.b64encode(nonce).decode()
        ciphertext_b64 =
base64.b64encode(ciphertext_with_tag).decode()

        return f"{version_b64}:{nonce_b64}:{ciphertext_b64}"

    def decrypt(self, encrypted: str, aad: Optional[bytes] = None)
-> str:

        if not self.is_unlocked:
            raise RuntimeError("Crypto manager is locked; set VDK
first")

        try:

            parts = encrypted.split(':')
            if len(parts) != 3:
                raise ValueError("Invalid ciphertext format")

            version_b64, nonce_b64, ciphertext_b64 = parts


            version = base64.b64decode(version_b64).decode()
            nonce = base64.b64decode(nonce_b64)
            ciphertext_with_tag = base64.b64decode(ciphertext_b64)


            if version != self.FORMAT_VERSION:
                raise ValueError(f"Unsupported ciphertext version:
{version}")


            plaintext_bytes = self._aesgcm.decrypt(nonce,
ciphertext_with_tag, aad)
            return plaintext_bytes.decode('utf-8')

        except Exception as e:
            raise ValueError(f"Decryption failed: {str(e)}")
```

```
    def encrypt_vdk(self, vdk: bytes, kek: bytes) -> str:

        aesgcm = AESGCM(kek)
        nonce = self.generate_nonce()
```

Лістинг 1.2 – kdf.py

```
import secrets
from typing import Tuple

from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives.kdf.scrypt import Scrypt

from app.utils.constants import (
    SCRYPT_N, SCRYPT_R, SCRYPT_P,
    SCRYPT_KEY_LENGTH, SCRYPT_SALT_LENGTH
)


class KdfManager:


    @staticmethod
    def generate_salt() -> bytes:

        return secrets.token_bytes(SCRYPT_SALT_LENGTH)

    @staticmethod
    def derive_key(password: str, salt: bytes,
                   n: int = SCRYPT_N,
                   r: int = SCRYPT_R,
                   p: int = SCRYPT_P) -> bytes:

        password_bytes = password.encode('utf-8')

        kdf = Scrypt(
            salt=salt,
            length=SCRYPT_KEY_LENGTH,
            n=n,
            r=r,
            p=p,
            backend=default_backend()
        )

        key = kdf.derive(password_bytes)
        return key

    @staticmethod
    def verify_password(password: str, salt: bytes, expected_key:
 bytes,
                        n: int = SCRYPT_N,
                        r: int = SCRYPT_R,
                        p: int = SCRYPT_P) -> bool:
```

```
            try:
                derived_key = KdfManager.derive_key(password, salt, n,
    r, p)

                return secrets.compare_digest(derived_key,
    expected_key)
            except Exception:
                return False

       @staticmethod
       def check_password_strength(password: str) -> Tuple[int, str]:

            if not password:
                return 0, "Empty password"

            score = 0
            feedback = []


            length = len(password)
            if length < 8:
                feedback.append("Too short (at least 8 characters)")
            elif length < 12:
                score += 20
                feedback.append("Consider using 12+ characters")
            elif length < 16:
                score += 30
            else:
                score += 40


            has_lower = any(c.islower() for c in password)
            has_upper = any(c.isupper() for c in password)
            has_digit = any(c.isdigit() for c in password)
            has_special = any(not c.isalnum() for c in password)

            char_types = sum([has_lower, has_upper, has_digit,
    has_special])
```

Лістинг 1.3 – item_service.py

```
import json
from typing import List, Optional, Tuple, Dict, Any

from app.config.config import cfg
from app.models.item import Item
from app.repositories.field_definition_repository import
FieldDefinitionRepository
from app.repositories.item_field_value_repository import
ItemFieldValueRepository
```

```
from app.repositories.item_repository import ItemRepository
from app.repositories.item_version_repository import
ItemVersionRepository
from app.security.crypto import crypto_manager
from app.utils.database import db_manager


class ItemService:

    def __init__(self):
        pass

    def create_item(
        self,
        item_type_id: int,
        title: str,
        field_values: Dict[int, str]
    ) -> Tuple[bool, str, Optional[int]]:

        if not crypto_manager.is_unlocked:
            return False, "Vault is locked", None

        if not title or not title.strip():
            return False, "Title cannot be empty", None

        title = title.strip()
        if len(title) > 200:
            return False, "Title is too long (max 200 characters)",
None

        try:
            with db_manager.session_scope() as session:
                item_repo = ItemRepository(session)
                field_value_repo = ItemFieldValueRepository(session)
                field_def_repo = FieldDefinitionRepository(session)

                item = item_repo.create(
                    item_type_id=item_type_id,
                    title=title,
                    version=1,
                    is_deleted=False
                )

                for field_def_id, value in field_values.items():
                    if not value:
                        continue

                    field_def =
field_def_repo.get_by_id(field_def_id)
                    if not field_def:
                        continue

                    if field_def.is_secret:
```

```
                        encrypted_value =
crypto_manager.encrypt(value)
                        field_value_repo.create(
                            item_id=item.id,
                            field_def_id=field_def_id,
                            value_plain=None,
                            value_encrypted=encrypted_value
                        )
                    else:
                        field_value_repo.create(
                            item_id=item.id,
                            field_def_id=field_def_id,
                            value_plain=value,
                            value_encrypted=None
                        )

                version_repo = ItemVersionRepository(session)
                snapshot_data = {
                    "title": item.title,
                    "item_type_id": item.item_type_id,
                    "field_values": field_values,
                    "created_at": item.created_at.isoformat(),
                    "updated_at": item.updated_at.isoformat()
                }
                snapshot_json = json.dumps(snapshot_data,
ensure_ascii=False)
                snapshot_encrypted =
crypto_manager.encrypt(snapshot_json)

                version_repo.create_version(
                    item_id=item.id,
                    version_number=item.version,
                    snapshot_encrypted=snapshot_encrypted
                )

                return True, "Item created successfully", item.id

        except Exception as e:
            return False, f"Failed to create item: {str(e)}", None

    def update_item(
        self,
        item_id: int,
        title: str = None,
        field_values: Dict[int, str] = None
    ) -> Tuple[bool, str]:

        if not crypto_manager.is_unlocked:
            return False, "Vault is locked"

        if title is not None:
            if not title.strip():
                return False, "Title cannot be empty"
```

```
            if len(title.strip()) > 200:
                return False, "Title is too long (max 200
characters)"

        try:
            with db_manager.session_scope() as session:
                item_repo = ItemRepository(session)
                field_value_repo = ItemFieldValueRepository(session)
                field_def_repo = FieldDefinitionRepository(session)

                item = item_repo.get_by_id(item_id)
                if not item:
                    return False, "Item not found"

                if title is not None:
                    item.title = title

                if field_values:
                    for field_def_id, value in field_values.items():
                        field_def =
field_def_repo.get_by_id(field_def_id)
                        if not field_def:
                            continue

                        if field_def.is_secret:
                            encrypted_value =
crypto_manager.encrypt(value) if value else None
                            field_value_repo.create_or_update_value(
                                item_id=item_id,
                                field_def_id=field_def_id,
                                value_plain=None,
                                value_encrypted=encrypted_value
                            )
                        else:
                            field_value_repo.create_or_update_value(
                                item_id=item_id,
                                field_def_id=field_def_id,
                                value_plain=value,
                                value_encrypted=None
                            )

                item_repo.increment_version(item_id)

                updated_item = item_repo.get_by_id(item_id)
                if updated_item:
                    current_field_values =
updated_item.get_field_values_dict()
                    current_field_values_int = {int(k): v for k, v
in current_field_values.items()}

                    version_repo = ItemVersionRepository(session)
                    snapshot_data = {
                        "title": updated_item.title,
```

```
                        "item_type_id": updated_item.item_type_id,
                        "field_values": current_field_values_int,
                        "created_at":
updated_item.created_at.isoformat(),
                        "updated_at":
updated_item.updated_at.isoformat()
                    }
                    snapshot_json = json.dumps(snapshot_data,
ensure_ascii=False)
                    snapshot_encrypted =
crypto_manager.encrypt(snapshot_json)

                    version_repo.create_version(
                        item_id=item_id,
                        version_number=updated_item.version,
                        snapshot_encrypted=snapshot_encrypted
                    )

                    max_versions = cfg.get(cfg.maxVersionsPerItem)
                    version_repo.delete_oldest_versions(item_id,
max_versions)

                return True, "Item updated successfully"

        except Exception as e:
            return False, f"Failed to update item: {str(e)}"
```

Лістинг 1.4 – data.base.py

```
from contextlib import contextmanager
from pathlib import Path

from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker, scoped_session
from sqlalchemy.pool import StaticPool

from .constants import DATABASE_NAME


class DatabaseManager:
    _instance = None
    engine = None
    _session_factory = None

    def __new__(cls):
        if cls._instance is None:
            cls._instance = super().__new__(cls)
        return cls._instance

    def initialize(self, db_path: str = None):
```

```
        if self.engine is not None:
            return

        if db_path is None:

            app_data_dir = Path.home() / ".pipass"
            app_data_dir.mkdir(parents=True, exist_ok=True)
            db_path = str(app_data_dir / DATABASE_NAME)



        self.engine = create_engine(
            f"sqlite:///{db_path}",
            poolclass=StaticPool,
            connect_args={
                "check_same_thread": False,
                "timeout": 30
            },
            echo=False
        )

        self._session_factory = scoped_session(
            sessionmaker(
                bind=self.engine,
                autocommit=False,
                autoflush=False
            )
        )

    def get_session(self):
        if self._session_factory is None:
            raise RuntimeError("Database is not initialized. Call
initialize() first.")
        return self._session_factory()

    def get_engine(self):
        if self.engine is None:
            raise RuntimeError("Database is not initialized. Call
initialize() first.")
        return self.engine

    def close(self):
        if self._session_factory:
            self._session_factory.remove()
        if self.engine:
            self.engine.dispose()

    @property
    def database_exists(self) -> bool:
        app_data_dir = Path.home() / ".pipass"
        db_path = app_data_dir / DATABASE_NAME
        return db_path.exists()
```

```
    @contextmanager
    def session_scope(self):
        session = self.get_session()
        try:
            yield session
            session.commit()
        except Exception:
            session.rollback()
            raise
        finally:
            session.close()


db_manager = DatabaseManager()
```