НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ
Завідувач кафедри
Комп'ютерних наук
(назва кафедри)
_____ Голуб Б. Л.
(підпис) (ПІБ)
"_2_" __червня__ 2025 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему

"Програмне забезпечення системи для шифрування та дешифрування даних"

Спеціальність 121 – «Інженерія програмного забезпечення»

Гарант освітньої програми

к.н.т., доцент _____ Вайганг Г.О.
(наукова ступінь та вчене звання) (підпис) (ПІБ)

Керівник Бакалаврської кваліфікаційної роботи

ст. викладач _____ Міловідов Ю. О.
(наукова ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ Старовіт Андрій Віталійович
(підпис) (ПІБ студента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
Комп'ютерних наук
(назва кафедри)
Голуб Б. Л.
(підпис) (ПІБ)
"16" грудня 2024 р.

З А В Д А Н Н Я
на виконання бакалаврської кваліфікаційної роботи студенту
Старовіту Андрію Віталійовичу
(прізвище, ім'я, по батькові)
Спеціальність 121 – «Інженерія програмного забезпечення»
Тема бакалаврської кваліфікаційної роботи Програмне забезпечення системи для шифрування та дешифрування даних
Затверджена наказом ректора НУБіП України від "16" грудня 2024 р. № 2249 "С"
Термін подання завершеної роботи на кафедру 2025.06.02
(рік, місяць, число)
Вихідні дані до бакалаврської кваліфікаційної роботи
Опис предмету дослідження, опис програмного забезпечення
Перелік питань, які потрібно розробити:
Системний аналіз предметної області
Аналіз предметної області
Розробка програмного забезпечення
Тестування програмного забезпечення

Дата видачі завдання "16" грудня 2024 р.

Керівник Бакалаврської кваліфікаційної роботи
ст. викладач Міловідов Ю. О.
(наукова ступінь та вчене звання) (підпис) (ПІБ)

Завдання прийняв до виконання
Старовіт Андрій Віталійович
(підпис) (ПІБ студента)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AES – симетричний алгоритм шифрування, що є міжнародним стандартом.

C++ – високорівнева, об’єктно-орієнтована мова програмування.

Figma – онлайн сервіс для проєктування графічних інтерфейсів користувача.

GUI – графічний інтерфейс користувача.

Qt – фреймворк для створення графічного інтерфейсу користувача.

SQLite – вбудована база даних.

UI – інтерфейс користувача.

UML – уніфікована мова моделювання.

XOR – симетричний алгоритм для шифрування.

БД – база даних.

Інтерфейс авторизації – частина програмного забезпечення, де користувач вводить свої облікові дані.

Інтерфейс функціоналу – частина програмного забезпечення, яка дозволяє здійснювати основні дії над файлами.

СУБД – програмне забезпечення що дозволяє створювати, зберігати та керувати базами даних.

Файл – одиниця даних, з якою виконується операція шифрування або дешифрування.

ВСТУП

У сучасному світі безпека інформації має ключове значення. Щодня відбувається обмін конфіденційними даними, починаючи з особистих повідомлень та фінансових транзакцій, закінчуючи державними документами. У зв’язку з цим виникає гостра необхідність в ефективних методах захисту інформації від несанкціонованого доступу до них. Одним з найефективніших засобів збереження інформації від сторонніх осіб є їх шифрування.

Шифрування – це процес зміни символів відкритої інформації шляхом їх перестановки, заміни тощо, що робить неможливим їх прочитання без відповідного ключа. Зворотнім процесом до шифрування є дешифрування. Дешифрування – дозволяє повернути зашифровану інформацію назад у первісний вигляд за наявності необхідного ключа.

Окрім технічних аспектів варто також враховувати практичну цінність шифрувальних систем для широкого кола користувачів. Система має бути зручною як користувачам, що використовуватимуть систему для персональних потреб так і користувачам що зберігатимуть робочу інформацію. Надійне збереження особистих файлів вимагає ефективних і доступних засобів шифрування, тому актуальним є створення програмного забезпечення який не лише використовуватиме сучасні алгоритми захисту, а й буде інтуїтивно зрозумілим, легким у використанні й адаптованим під реальні потреби користувачів у різних галузях.

Метою даної бакалаврської кваліфікаційної роботи є аналіз сфери застосування, проєктування системи та розробка програмного забезпечення системи шифрування та дешифрування даних. Система має поєднувати в собі зручність у використанні з надійністю алгоритмів шифрування. В роботі розглядатимуться теоретичні основи криптографії, порівнюватимуться методи шифрування, а також розроблено програмний застосунок для захисту інформації на рівні користувача.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

В теперішніх умовах цифрового обміну інформацією, коли обробка, передача та зберігання даних здійснюється переважно в електронному вигляді, захист цієї інформації набуває критичного значення. Особливе значення набуває питання безпеки в контексті передачі особистих даних, фінансової інформації, комерційних та державних таємницях. Для захисту інформації від сторонніх осіб необхідно використовувати сучасні та надійні алгоритми їх шифрування.

Основу захисту інформації від третіх осіб становлять алгоритми симетричного та асиметричного шифрування, методи генерації та зберігання ключів, та засоби обробки й перетворення текстової та бінарної інформації. При захисті інформації необхідно враховувати не лише стійкість алгоритмів до їх зламу, а й продуктивність, масштабованість, зручність і доцільність їх використання у програмних застосунках.

Програмне забезпечення системи для шифрування та дешифрування даних має дозволяти перетворювати відкриту інформацію у зашифрований вигляд, який неможливо прочитати без наявності спеціального ключа шифрування. Водночас, дешифрування дає змогу відновити початковий вигляд даних для уповноважених користувачів. Така система є важливою складовою в сферах кібербезпеки, електронного документообігу, банківських систем, захищеного зберігання тощо.

Розробка програмного забезпечення системи для шифрування та дешифрування даних має на меті створення програмного застосунка , здатного надати користувачеві надійний захист інформації від несанкціонованого доступу до них використовуючи сучасні рішення. Цей застосунок може

використовуватися як для особистих потреб користувачів, так і для корпоративних чи державних.

1.2 Аналіз вимог до програмної системи

Аналіз вимог це один з основних етапів розробки програмного забезпечення. Програмне забезпечення системи для шифрування та дешифрування даних повинно містити наступні вимоги:

Функціональні вимоги:

- Можливість реєстрації та авторизації для користувачів.
- Користувач повинен мати змогу власноруч вибирати файли підтримуваного розширення.
- Можливість шифрувати файли користувачів.
- Можливість дешифрувати файли користувачів.
- Програмне забезпечення має автоматично зберігати зашифровані чи дешифровані файли.
- Застосунок повинен містити графічний інтерфейс користувача з можливістю відображення повідомлень про успішні та помилкові операції.

Нефункціональні вимоги:

- Програмне забезпечення повинно працювати на операційній системі Windows.
- Повинен використовуватися сучасний алгоритм шифрування.
- Має використовуватися мова програмування C++ для розробки системи.
- Для розробки GUI повинен використовуватися фреймворк Qt.
- Повинна використовуватися база даних SQLite.
- Покриття тестами повинно бути не менше 80% критичних функцій.
- Без правильного ключа відновлення даних неможливо.

- Великий розмір файлів може впливати на швидкість роботи системи.

1.3 Моделювання предметної області

Моделювання предметної області це наступний етап розробки програмного забезпечення, він дозволяє представити структуру, логіку та взаємозв’язки об’єктів, що беруть участь у роботі системи. Основною метою моделювання є побудова абстрактного уявлення про сутності предметної області, які згодом складатимуть архітектуру програмного забезпечення. Також моделювання дозволяє на ранньому етапі зрозуміти логіку функціонування системи, основні сценарії використання, вхідні та вихідні дані, а також ролі користувачів.

Моделювання дає уявлення про роботу системи, якими є її межі та які функції вона повинна виконувати. Ця інформація важлива як для складних, так для критично важливих програмних рішень. Оскільки безпека, цілісність і коректність роботи програми мають ключове значення, то на етапі моделювання варто детально приділити увагу механізму авторизації, вибору файлів, шифруванню інформації та її збереженню.

Під час процесу моделювання використовуються різні методи та інструменти. Найпоширенішими з них є мова візуального моделювання UML. Ця мова дозволяє представити логіку роботи системи у вигляді діаграм таких як: діаграма прецедентів, діаграма класів, діаграма діяльності та інші. Кожна з цих діаграм дозволяє візуалізувати найрізноманітніші аспекти системи від ролей користувачів і їх варіантів використання до внутрішньої структури та послідовності використання операцій.

Задіяння моделювання предметної області допомагає уникнути непорозумінь між замовником та розробником. Ще одним плюсом моделювання є полегшення планування архітектури застосунка та надання розуміння у виборі інструментів розробки. Також моделі можуть бути використані як основа для тестування та подальшого масштабування системи.

Отже моделювання предметної області це важливий етап розробки системи. Він дозволяє більш глибоко розуміти потреби користувача та забезпечує структурований підхід до побудови системи. Зменшення ризиків під час побудови програмного забезпечення також завдячує моделюванню, адже на цьому етапі відловлюються ризики пов’язані з логічними помилками, або невідповідністю функціональним очікуванням.

На цьому етапі спроєктовано діаграму прецедентів Рис. 1.1 програмного забезпечення системи шифрування та дешифрування даних. Дана діаграма відображає взаємодію із системою через дії або сценарії, які виконує система у відповідь на запити користувачів. Вона допомагає чітко визначити які саму функції повинне містити програмне забезпечення.

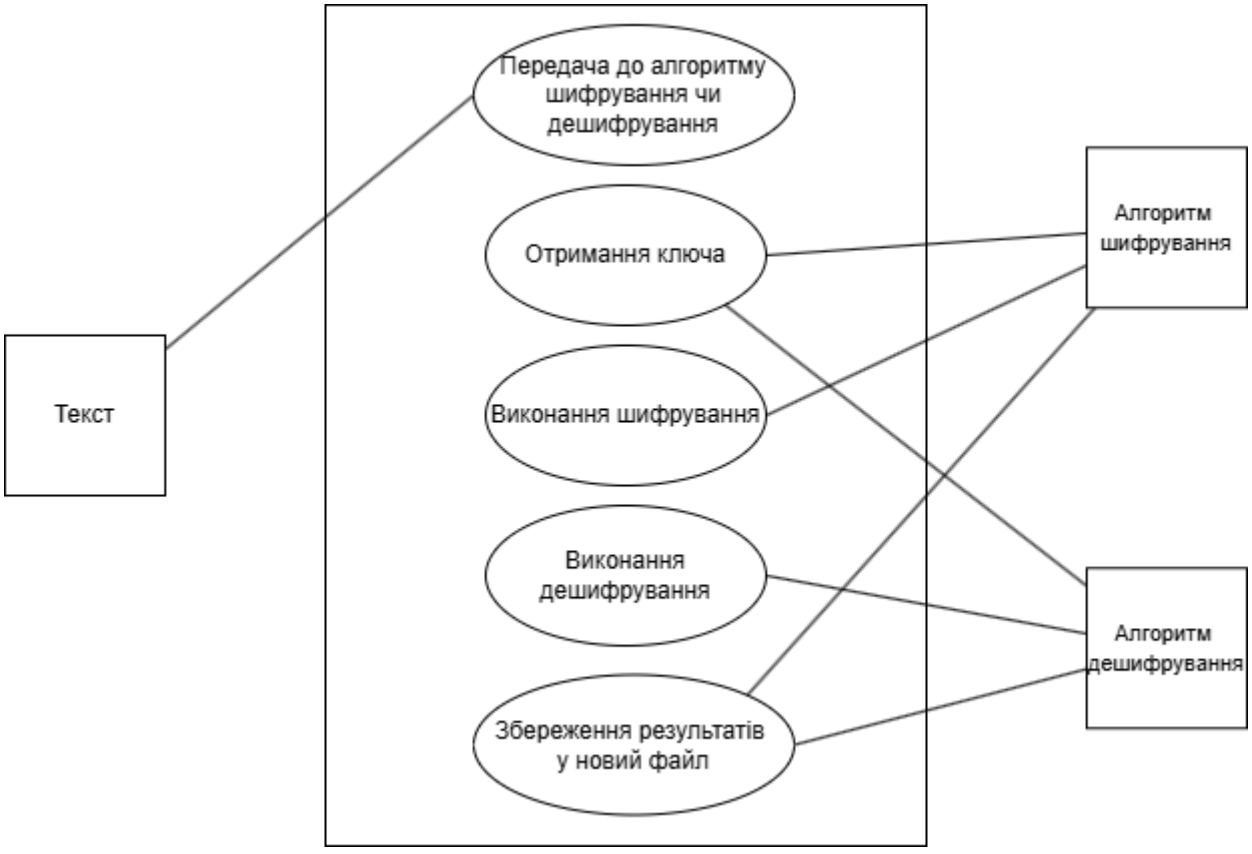

Рис. 1.3.1 – Діаграма прецедентів

Ця діаграма відображає акторів що взаємодіють із системою та перелік дій які ці актори можуть використати.

Актори:

- Текст – набір символів над якими виконуватимуть операцію шифрування.
- Алгоритм шифрування – метод яким шифруватиметься текст.
- Алгоритм дешифрування – метод яким дешифруватиметься текст.

Прецеденти:

1. Текст
   - Передача до алгоритму шифрування чи дешифрування.
2. Алгоритм шифрування
   - Отримання ключа
   - Виконання шифрування
   - Збереження результатів у новий файл
3. Алгоритм дешифрування
   - Отримання ключа
   - Виконання дешифрування
   - Збереження результатів у новий файл

Діаграма послідовності в UML призначена для демонстрації взаємодії між об’єктами системи. На ній відображено які об’єкти прийматимуть участь у процесі виконання певного функціоналу, яку інформацію вони передають між собою та як відповідає система на певні дії користувача. Ця діаграма дає змогу зрозуміти динаміку роботи системи, що має ключове значення в роботі інтерактивних застосунків із обробкою даних.

Для програмного забезпечення системи для шифрування та дешифрування даних змодельовано діаграму послідовності Рис. 1.2. На ній відображено хід роботи системи, взаємодію між компонентами та взаємодію з базою даних. Діаграма дозволяє виявити можливі проблеми, зайві рішення або невраховані обробки помилок.

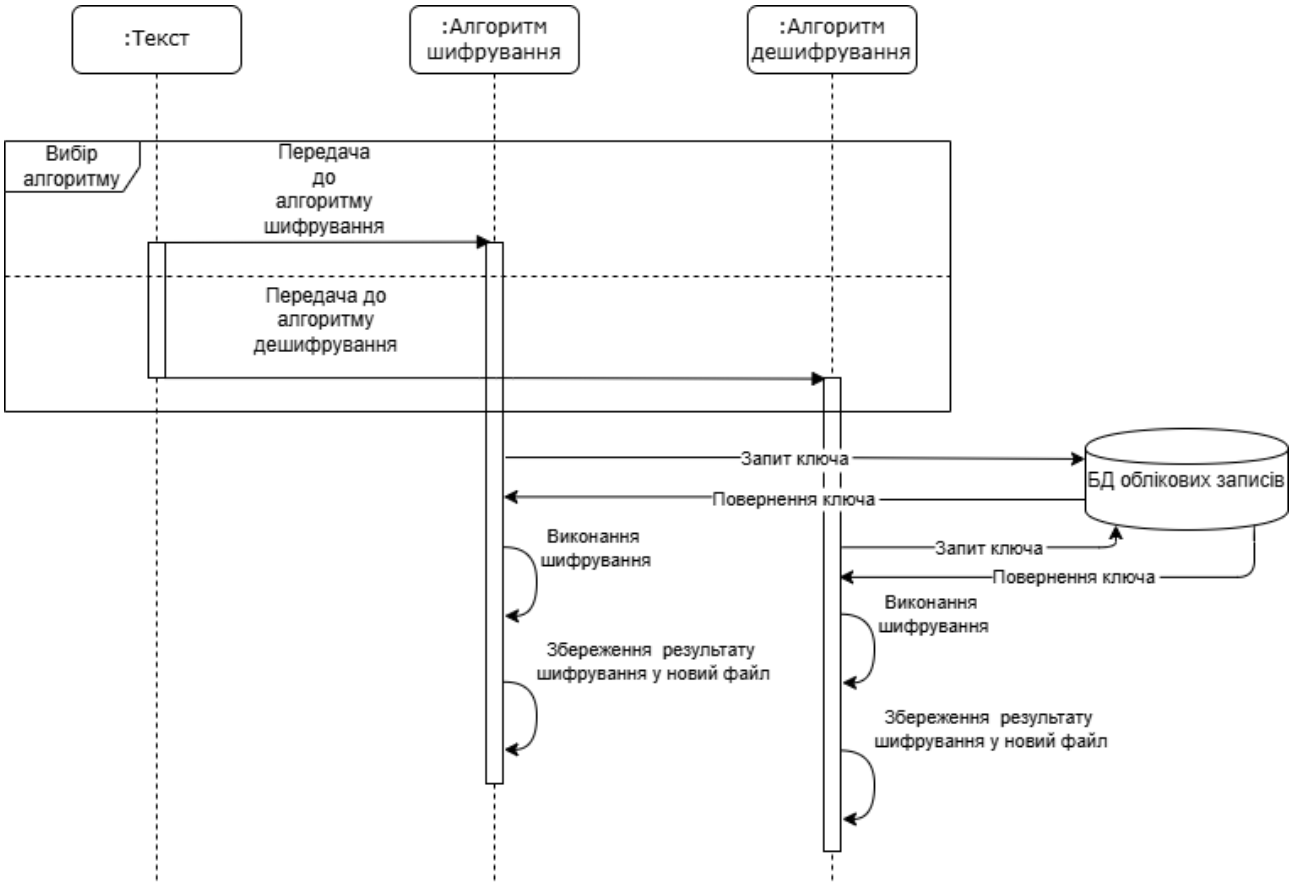

Рис. 1.3.2 – Діаграма послідовності

Сценарій що відображений на діаграмі послідовності це процес шифрування та дешифрування тексту. Ця діаграма дає уявлення про взаємодію інформації з алгоритмами шифрування, розуміння роботи алгоритмів з базою даних та процес виконання методів.

Сценарій роботи системи для шифрування або дешифрування даних:

1. Текст передається в алгоритм шифрування чи дешифрування.
2. Алгоритм отримує ключ з бази даних.
3. Алгоритм виконує операцію над інформацією.
4. Алгоритм зберігає результат виконаної операції у новий файл.

Для моделювання логіки роботи програмного забезпечення у вигляді послідовності дій, умов переходу та паралельних процесів використовують діаграму активності. Дана діаграма допомагає візуалізувати алгоритми, бізнес-

процеси та користувацькі сценарії. Вона наочно демонструє рух даних в системі, а також місця де відбувається розгалуження чи об’єднання потоків.

Розроблена діаграма активності Рис. 1.3 для програмного забезпечення демонструє послідовність виконання дій під час роботи системи.

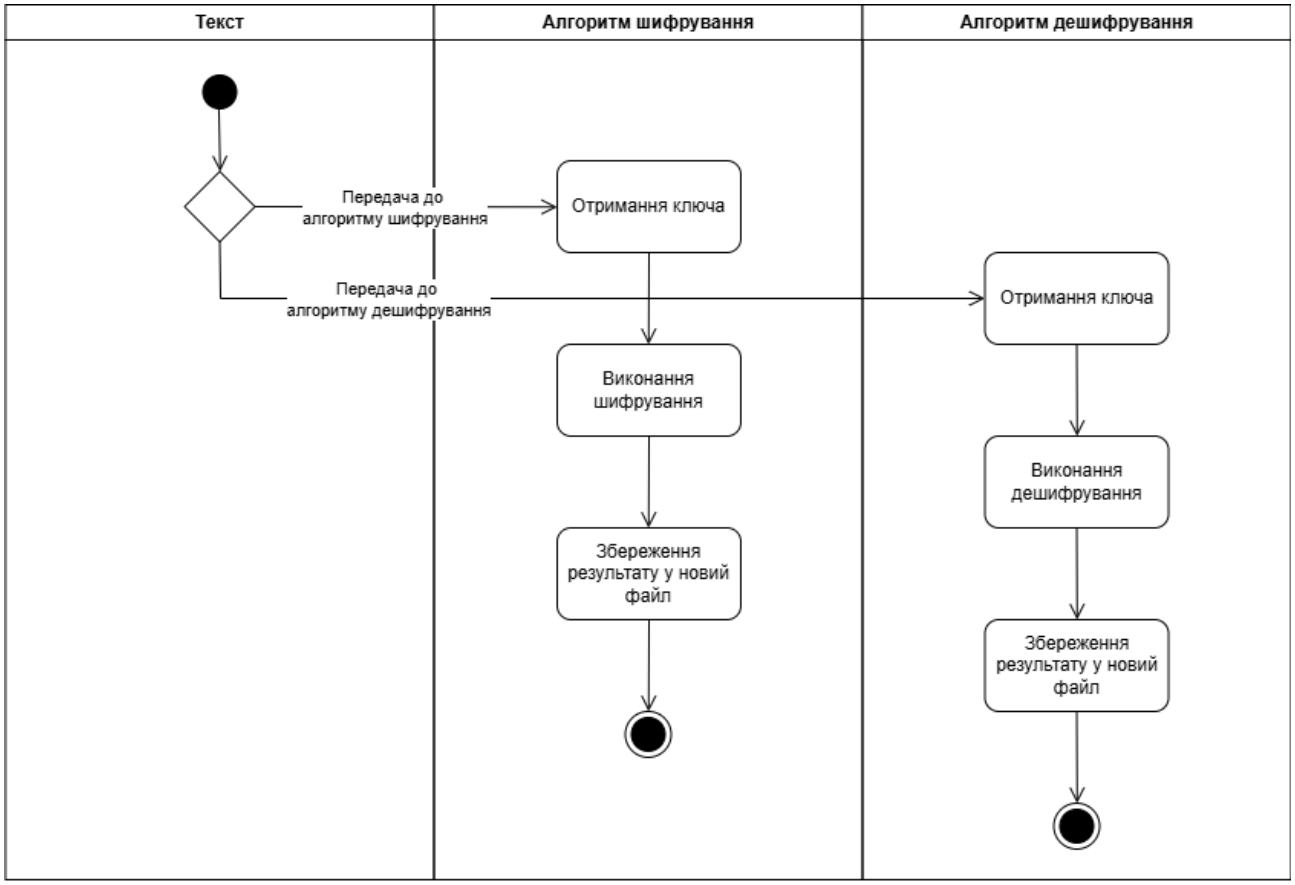

Рис. 1.3.3 – Діаграма активності

Розроблена діаграма активності має таку послідовність дій для виконання:

1. Початок.
2. Вибір алгоритму.
3. Отримання ключа.
4. Виконання шифрування або дешифрування.
5. Збереження результату у новий файл.
6. Кінець.

Основною частиною об’єктно-орієнтованого аналізу та проєктування, що дозволяє виділити найважливіші характеристики об’єктів є абстракції. Абстракції дозволяють зменшити складність системи, підвищити її розуміння та забезпечити гнучкість для повторного використання коду.

Виконуючи моделювання предметної області було реалізовано ключові абстракції Рис. 1.4, які відображають основні сутності та процеси. Створення абстракцій дає змогу відокремити логіку роботи системи від конкретної реалізації та полегшити процес розробки.

Рис. 1.3.4 – Абстракції

На даній діаграмі продемонстровано такі абстракції з їх властивостями та обов’язками:

1. Текст

   Важливі властивості:

   - Зміст
   - Довжина

   Обов’язки:

   - Передача до алгоритму
   - Збереження оригінального вигляду

2. Алгоритм шифрування

   Важливі властивості:

   - Тип шифрування
   - Ключ

   Обов’язки:

   - Обробка ключа
   - Шифрування тексту
   - Мінімізація помилок

3. Алгоритм дешифрування

   Важливі властивості:

   - Тип шифрування
   - Ключ

   Обов’язки:

   - Обробка ключа
   - Дешифрування тексту
   - Мінімізація помилок

У процесі моделювання предметної області розроблено низку UML діаграм, кожна з яких має важливу роль у створенні цілісного уявлення про програмну систему. Діаграма прецедентів дала змогу визначити основні сценарії взаємодії в системі. Діаграма послідовності продемонструвала порядок взаємодії об’єктів під час виконання ключових сценаріїв.

Діаграма активності допомогла продемонструвати динаміку виконання алгоритмів шифрування та дешифрування. Окрім цього розробка і застосування абстракцій дозволило узагальнити основні сутності системи, спростити її архітектуру та забезпечити зручність для подальшої розробки.

1.4 Дослідження існуючих рішень

Програмне забезпечення системи для шифрування та дешифрування даних – це система основним призначенням якої є забезпечення конфіденційності, цілісності та безпеки інформації шляхом її перетворення у недоступну для сторонніх осіб форму (зашифрований вигляд) та зворотного відновлення (дешифрування) при потребі. Такі системи переважно використовують один або кілька криптографічних алгоритмів, підтримають управління ключами та автентифікацію користувачів.

Якісна система шифрування повинна надійно захищати інформацію від третіх осіб, бути зручною та зрозумілою у використанні та гарантувати захист даних навіть у випадку компрометації частини середовища. Також система повинна гарантувати цілісність вхідних даних після обробки та забезпечити швидку роботу.

Систем для приховування інформації є безліч та кожна з них є унікальною. Вони використовують відмінні алгоритми кожен з яких в чомусь кращий, а в чомусь гірший, призначення систем також різне. На цьому етапі знайдено та досліджено різні системи для шифрування інформації, вивчено їхні переваги та недоліки.

VeraCrypt – це одна з найпопулярніших безкоштовних систем, яка створює зашифровані віртуальні диски, а також може шифрувати фізичні носії. Ця система використовує стійкі до зламу методи, а також для кращого захисту інформації їх поєднання. Має версії для різних операційних систем як то Windows, Linux та macOS. Перевагами даної системи є гнучкість та надійність яку вона надає користувачеві, проте взамін вона очікує що користувач який її використовує технічно обізнаний. Початківцям необхідно приділити час та увагу для вивчення даної системи, адже процес створення томів та їх шифрування складний.

BitLocker – це вбудований в операційну систему Windows інструмент, який надає можливість повністю шифрувати диски. Цей інструмент використовує алгоритм шифрування AES з ключем довжиною в 128 або 256 біт та забезпечує прозоре шифрування, що означає що дані автоматично шифруються та розшифровуються без втручання користувача. Інструмент є зручним та має високу продуктивність, він інтегрований в операційну систему Windows, що дозволяє користувачам легко його застосовувати. Проте ця система не підходить для шифрування окремих файлів і є недоступною в домашній версії Windows, що звужує коло її користувачів.

Cryptomator – це безкоштовна система з відкритим кодом, призначена для безпечного зберігання файлів у хмарі. Дана система створює зашифровану віртуальну скриньку в якій користувач зберігаю всі необхідні дані. Всі дані що зберігаються в скрині автоматично шифруються за допомогою вище згадуваного алгоритму шифрування AES. Найбільш корисною дана система є для користувачів хмарних сервісів, адже дає можливість зберігати файли локально та синхронізувати їх у хмарі без ризику їх можливого витоку. Програма доступна на операційних системах Windows, Linux, macOS та Android, а головним її недоліком є обмежений контроль над ключами шифрування та відсутність розширених можливостей, таких як шифрування назви файла.

Таблиця 1.4.1

Критерії	VeraCrypt	BitLocker	Cryptomator
Тип шифрування	Повне або часткове	Повне шифрування диска	Файлове
Алгоритми	AES, Serpent, Twofish	AES-128 або AES-256	AES
Підтримка	Ні	Ні	Так

хмарних сервісів			

Продовження Таблиці 1.4.1

Інтеграція з ОС	Обмежена	Повна	Помірна
Відкритий код	Так	Ні	Так
Складність для користувача	Висока	Низька	Середня
Основна перевага	Висока безпека та гнучкість	Інтеграція з Windows	Шифрування хмарних файлів
Недоліки	Складне налаштування, не для початківців	Лише для певних версій Windows	Менше можливостей у мобільній версії

Дані системи шифрування відіграють важливу роль у захисті цифрової інформації, кожна з них має свої унікальні особливості, алгоритми шифрування та підходи до забезпечення конфіденційності інформації. Один з ний пропонує найвищий рівень гнучкості та безпеки завдяки підтримці складних шифрувальних схем, тоді як інший націлений на зручність у використанні для початківців. Одна з цих систем повністю інтегрується з операційною системою даючи надійне повне шифрування дисків, а інша спеціалізується на захисті у хмарних середовищах.

Порівнюючи ці системи можна зробити висновок, що вибір конкретного рішення залежить від потреб користувача, таких як максимальна безпека та надійність алгоритмів, або зручність для початківців. Незалежно від вибраної системи, їх використання у сучасному світі значно підвищує рівень безпеки чутливої інформації від сторонніх осіб.

1.5 Постановка завдання

Метою даної бакалаврської роботи є розробка програмного забезпечення системи для шифрування та дешифрування даних з графічним інтерфейсом користувача, що дає можливість авторизованим користувачам обирати файли заданих форматів, шифрувати або дешифрувати їх із використанням симетричного алгоритму шифрування, а також автоматично зберігати результати операцій в новому файлі.

Система має бути зручною у використанні, надійною, підтримувати обробку різноманітних помилок та повідомлень про виконані дії. В межах розробки також передбачено створення UML-діаграм для ширшого уявлення про вигляд майбутньої системи та проведення тестування програмного забезпечення для надання надійності критично важливим функціям.

Враховуючи все вищезгадане в даній роботі мають бути виконані такі завдання:

1. Реалізувати систему реєстрації та авторизації. Кожен користувач системи повинен мати власний обліковий запис для захисту доступу до зашифрованих даних.
2. Розробити механізм вибору файлів для шифрування. Користувач повинен мати змогу обрати файли з файлової системи, що підтримуються в системі.
3. Реалізувати шифрування файлів. Зашифрована інформація повинна зберігатися в новому файлі, файли повинні шифруватися за допомогою симетричного алгоритму.
4. Реалізувати дешифрування файлів. Дешифрована інформація повинна зберігатися в новому файлі, файли повинні дешифруватися за допомогою симетричного алгоритму.

5. Забезпечити автоматизацію збереження результатів. Зашифрована інформація повинна автоматично зберігатися в новому файлі та у визначеній директорії без втручання користувача.

6. Розробити графічний інтерфейс користувача. Інтерфейс має бути зручним у використанні, простим у розумінні та повинен містити повідомлення з попередженнями у разі помилок, або повідомлення про успіх у разі успішного виконання завдань.

7. Забезпечити роботу системи на операційній системі Windows. Програмне забезпечення має підтримуватися на будь якому приладі з операційною системою Windows.

8. Забезпечити мінімальні вимоги до персональних комп’ютерів. Система має вимагати якомога менше ресурсів для коректної та швидкої роботи.

9. Відсутність залежностей від інтернет з’єднання. Система має однаково працювати не залежно від того підключений персональний комп’ютер до інтернету чи ні.

10. Інструменти для реалізації. Програмне забезпечення повинно розроблятися мовою програмування C++ за допомогою фреймворку Qt, для зберігання інформації про облікові записи користувачів та їх ключі шифрування необхідно використовувати SQLite.

2 Проєктування програмного забезпечення

2.1 Діаграма класів

Діаграма класів є однією з UML-діаграм, вона відображає класи, їх атрибути, методи та зв’язки між ними. У контексті виконання даної роботи ця діаграма дозволяє представити основні компоненти застосунка. Кожен клас представляє тип об’єкта, який має властивості та функціонал. Діаграма класів виступає основою для створення класів у реальному коді та полегшує розуміння архітектури програмного забезпечення.

Дана діаграма дозволяє візуалізувати логічну структуру системи ще до початку її розробки. Це особливо важливо для роботи в команді, коли кілька розробників повинні мати спільне уявлення про майбутню систему. Також діаграма допомагає виявити дублювання логіки, слабкі місця системи та надмірні залежності між компонентами це на ранніх етапах розробки.

Окрім структурного уявлення діаграма може виявити порушення принципів об’єктно-орієнтованого проєктування. Наприклад один клас матиме надто багато відповідальностей, або багато де залежитиме від інших класів, це означатиме, що порушено принцип єдиної відповідальності. Саме тому діаграма класів не лише демонструє вигляд системи, а й сприяє її поліпшенню та дотримання стандартів розробки.

У контексті системи для шифрування та дешифрування даних діаграма класів Рис. 2.1 демонструє, які компоненти відповідають за отримання інформації, виконання операцій над текстом та отримання ключів які в подальшому використовуватимуться в процесах шифрування та дешифрування файлів.

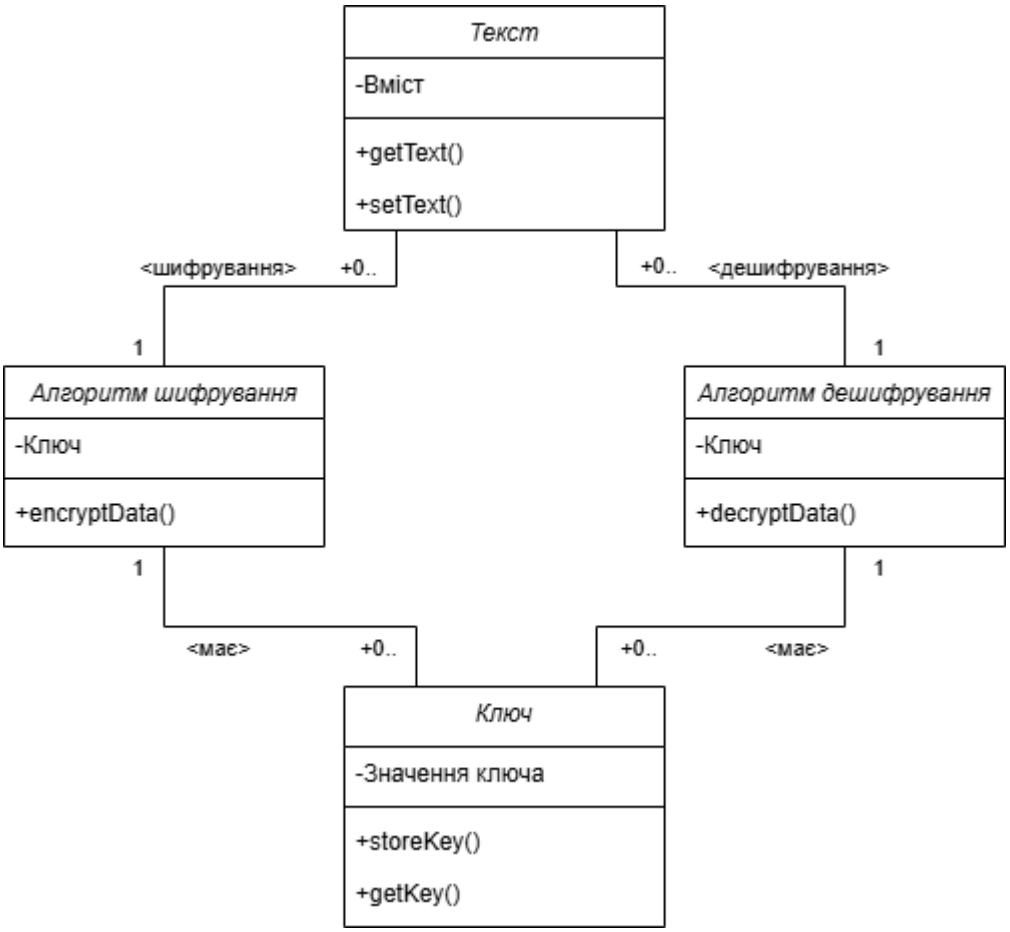

Рис. 2.1.1 – Діаграма класів

Розроблена діаграма класів демонструє основні елементи та взаємозв’язки між ними у системі. Вона дозволяє візуалізувати логіку роботи компонентів, що сприяє ефективній розробці та підтримці програмного забезпечення системи для шифрування та дешифрування даних.

Діаграма кооперацій є однією з поведінкових діаграм в UML, вона описує взаємодію між об’єктами системи. Ця діаграма демонструє які саме об’єкти приймають участь у виконанні певної дії, та які зв’язки вони мають. На відмінну від діаграми послідовностей де фокус уваги на послідовності виконання дій, діаграма класів зосереджується на організації об’єктів та їх структурі зв’язків.

Дана діаграма надає уявлення про щільність взаємодії об’єктів які пов’язані один з одним та допомагає виявити можливі перевантаження. Кожне повідомлення що передається між об’єктами підписується для кращого уявлення про виконання операцій між ними.

В програмному забезпеченні системи для шифрування та дешифрування даних, діаграма кооперацій Рис. 2.2 зображує взаємодію об’єктів під час виконання шифрування або дешифрування інформації. Ця діаграма допомагає зручно виявити місця, де потрібна логіка перевірки помилок або обробник виняткових ситуацій.

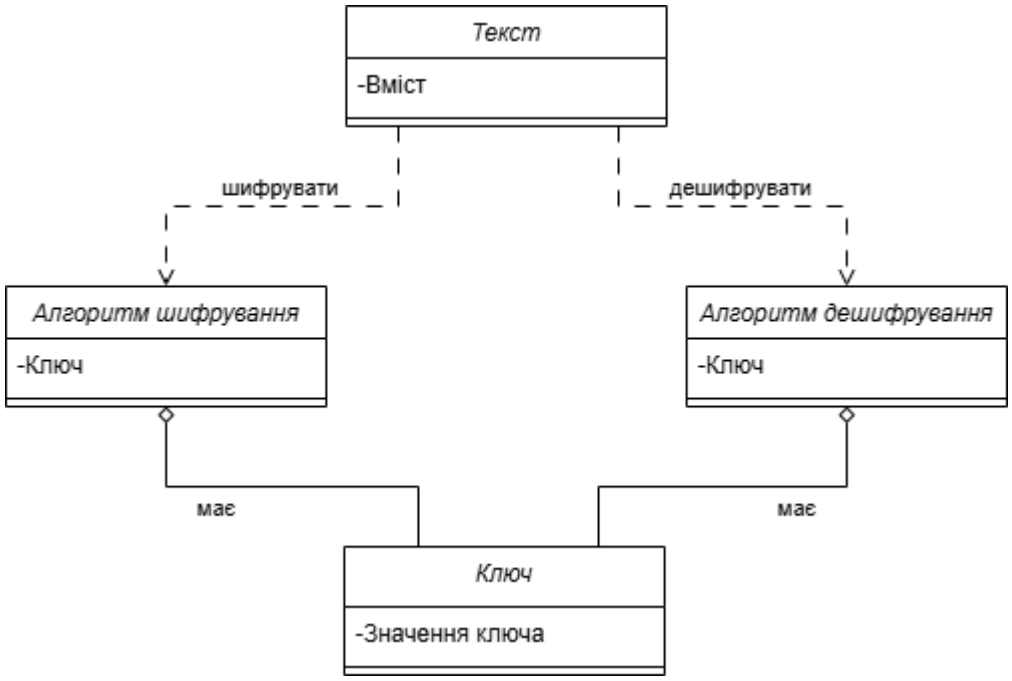

Рис. 2.1.2 – Діаграма кооперацій

Проаналізувавши діаграму класів створено діаграму кооперацій, що дає змогу виділити основні взаємодії між класами та продемонструвати їхню спільну роботу в рамках певного функціоналу системи.

2.2 Діаграма пакетів

Діаграма пакетів є структурною діаграмою UML, вона використовується для організації моделей системи у вигляді ієрархічних груп. Кожен з пакетів може містити як класи, інтерфейси, так і інші пакети або будь які інші елементи UML. Основною метою цієї діаграми є представлення структури великих програм і візуалізувати залежності між продемонстрованими модулями або частинами коду.

Пакети допомагають розробникам згрупувати класи та інші елементи між собою, що значно полегшує навігацію по системі, забезпечує кращу організацію коду та сприяє повторному використанню модулів. Цей підхід дозволяє зробити модулі більш самостійними зменшивши кількість їхніх залежностей та зробити їх гнучкішими для повторного використання.

Також в діаграмі пакетів продемонстровані залежності між пакетами, вони дозволяють зрозуміти який пакет використовує інший при певних сценаріях. Дана властивість діаграми пакетів допомагає уникнути циклічних залежностей, що можуть ускладнити компіляцію, або оновлення коду. Якщо один пакет залежить від багатьох інших це може свідчити про порушення принципів модульності, і тоді слід розглянути можливість його декомпозиції.

У процесі проєктування програмного забезпечення системи для шифрування та дешифрування даних, діаграма пакетів Рис. 2.3 виконує роль архітектурної карти, що показує, як саме поділена система як між собою взаємодіють її частини. Це зручно коли необхідно зобразити загальну структуру програмного забезпечення, для майбутнього масштабування системи та в рази спрощує процес адаптації і підтримки коду.

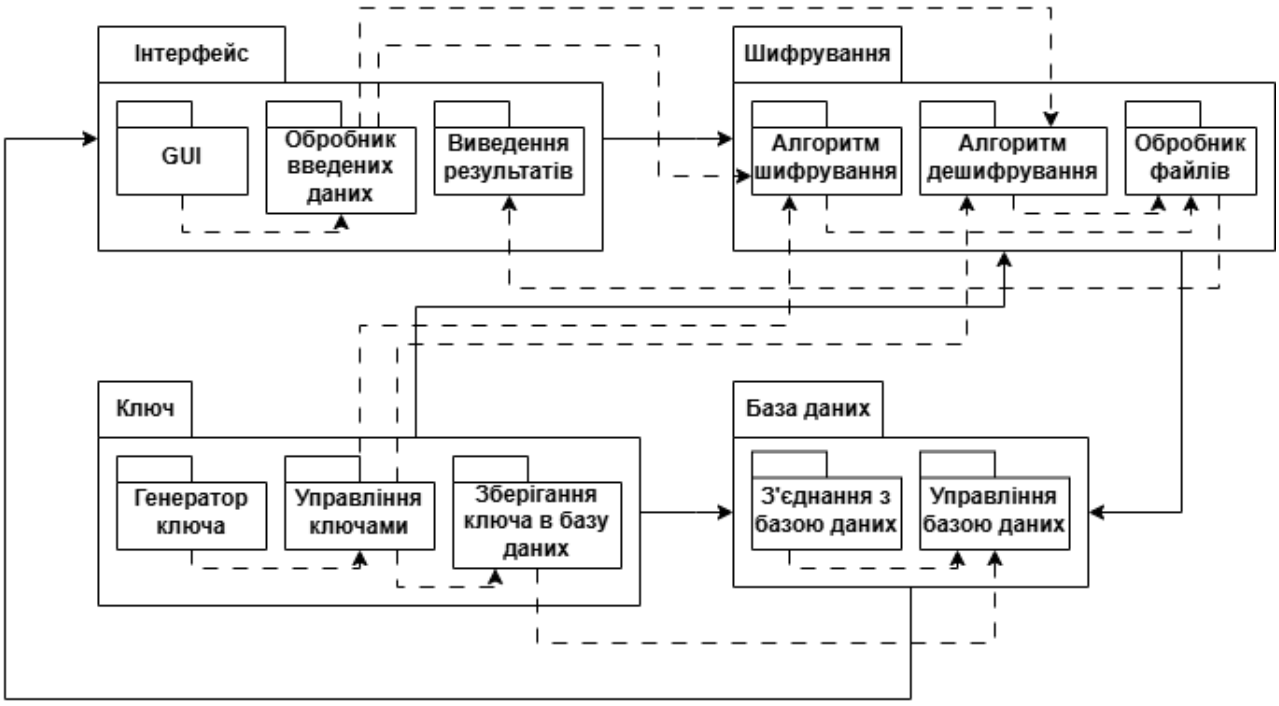

Рис. 2.2.1 – Діаграма пакетів

В рамках створення діаграми пакетів проаналізовано принципи зберігання та управління ключами шифрування, визначено основні пакети системи та їх взаємодію між собою. На даній діаграмі продемонстровано механізм створення та збереження ключів у базі даних та їх застосування в методах шифрування та дешифрування.

Окрема увага приділяється розподілу функціоналу між пакетами, що дозволило створити гнучку, масштабовану та безпечну систему. На діаграмі визначено основні пакети системи, а саме: інтерфейс, шифрування, ключ та база даних. Кожен з них містить відповідні елементи що реалізують їх функціональність.

Створена діаграма пакетів дозволяє ефективно організувати структуру програмного забезпечення, покращити його масштабування, підтримку коду та надати розуміння про безпечне керування ключами шифрування.

2.3 Логічна модель даних

Логічна модель даних дозволяє візуалізувати структуру збереження даних та описує які об’єкти прийматимуть участь в цьому процесі та їх характеристики. На відмінну від фізичної моделі, логічна не прив’язана до конкретної СУБД, тому вона більше орієнтована на пояснення структури збереження інформації. Створення логічної моделі є одним із ключових етапів при розробці програмного забезпечення, вона допомагає продемонструвати, як саме організовані дані в системі.

Логічна модель даних зазвичай представляється у вигляді ER-діаграми, де продемонстровано зв’язки між сутностями. Ця модель дозволяє уникнути дублювання інформації, забезпечує цілісність інформації і формалізувати бізнес-правила ще до етапу реалізації.

В контексті системи для шифрування та дешифрування даних логічна модель Рис. 2.4 зображує таблицю в якій міститиметься інформація про облікові записи користувачів логін, пароль та ключ для шифрування. Кожен обліковий запис міститиме унікальний ідентифікатор та унікальний ключ для шифрування.

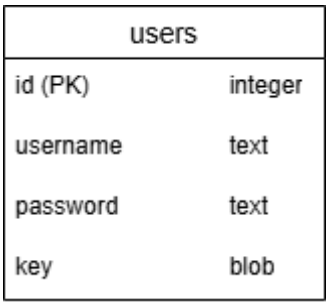

Рис. 2.3.1 – Логічна модель

Використання логічної моделі є основою для подальшої побудови фізичної моделі бази даних. Вона забезпечує уніфікацію даних у рамках системі, допомагає уникнути помилок на етапі реалізації та спрощує командну роботу. Також у майбутньому логічна модель може бути корисною при масштабуванні.

2.4 Діаграма компонентів

Діаграма компонентів також є однією з UML-діаграм. Вона використовується для моделювання фізичної структури програмного забезпечення, візуалізує компоненти які складають систему, їх взаємодію між собою та інтерфейси для зв’язку. Основною метою даної діаграми є візуалізація архітектури програмного забезпечення на рівні модулів або підсистем, що особливо корисно для розробки складних систем.

Кожен компонент діаграми представляє окрему функціональну частину системи. Компоненти зображаються у вигляді прямокутників із позначенням вкладених елементів і зв’язуються між собою залежностями, які ілюструють, які компоненти використовують інші. За допомогою цих залежностей розробники мають уявлення про те які саме блоки взаємодіють і наскільки сильно вони пов’язані.

Загалом діаграма компонентів для архітекторів програмного забезпечення та розробників, адже вона дозоляє заздалегідь визначити структуру взаємозалежних частин проєкту. Ця діаграма сприяє розумінню того, які модулі мають бути протестовані або змінені, також дозволяє зрозуміти які елементи можуть бути повторно використані. Окрім того, подібна діаграма є невід’ємною частиною документації програмного забезпечення, що використовуються для створення спільного уявлення про систему між розробниками.

Для програмного забезпечення системи для шифрування та дешифрування даних діаграма компонентів Рис. 2.5 допомагає візуалізувати які саме компоненти беруть участь у цілісній побудові програмного забезпечення та як вони взаємодіють між собою. Також ця діаграма допомагає зрозуміти як саме структуровані компоненти системи для шифрування такі як інтерфейс, алгоритми шифрування та дешифрування та як працює робота з файлами. Ця модель дозволяє краще зрозуміти архітектуру програмного забезпечення.

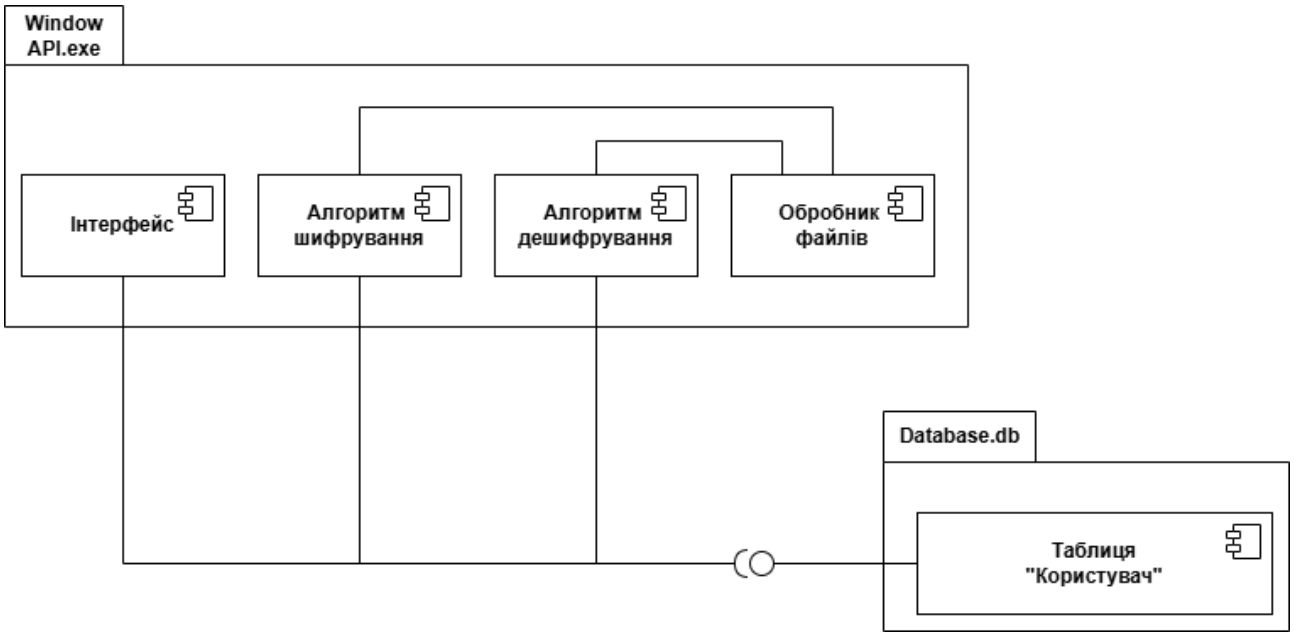

Рис. 2.4.1 – Діаграма компонентів

Результатом розробки діаграми компонентів було зображено структуру програмного забезпечення для шифрування. Використання цієї діаграми дозволяє створити спільне бачення взаємодії між модулями системи, що є важливим для спілкування замовника з розробниками. Також дана модель дозволяє створити безпечне та ефективне програмне забезпечення що неодмінно важливо для системи для шифрування.

Користь діаграми компонентів складно переоцінити оскільки користь яку отримують від чіткого уявлення, з яких саме функціональних частин складається система та як ці частини взаємодіють між собою, велика. Ця модель полегшує розподіл обов’язків між розробниками, оскільки кожен з них може працювати над окремими компонентами, не порушуючи роботу всієї системи.

Крім вищесказаного діаграма компонентів допомагає заздалегідь виявити слабкі місця в архітектурі, наприклад, занадто сильну залежність одного компонента від іншого, або відсутність чітко визначеного інтерфейсу для взаємодії з компонентами.

2.5 Діаграма розгортання

Діаграма розгортання використовується для моделювання фізичного розміщення програмних компонентів на апаратній частині програмного забезпечення. Вона показує які програмні модулі розміщуються та в яких вузлах, а також як саме ці вузли будуть взаємодіяти між собою через мережеві з'єднання. Ця діаграма є ключовою для надання розуміння про архітектуру системи у реальному середовищі виконання.

Також дана модель демонструє, які канали зв'язку використовуються між вузлами, наприклад: HTTPS для безпечної передачі даних між клієнтом та сервером, або протоколи доступу до бази даних. Це дає можливість потенційні загрози для безпеки, що в контексті системи для шифрування є дуже важливим. Також дана діаграма дозволяє оптимізувати маршрути передачі даних та забезпечити відповідність системи до вимог захищеності.

Застосування діаграми розгортання також дозволяє моделювати масштабованість системи, наприклад надає можливість додавання нових сервісів для обробки шифрування або розподілу навантаження між вузлами. Таке рішення дає можливість ще на етапі проєктування системи передбачити, яка саме поведінка буде під час зростання кількості користувачів, або кількості даних для обробки. Завдяки цьому діаграма сприяє створенню більш надійного, гнучкого та ефективного рішення для системи.

Загалом, діаграма розгортання є невід'ємною частиною проєктування програмного забезпечення та складає основу технічної документації. Таке рішення дозволяє всім учасникам розробки від технічного персоналу до самих розробників отримати єдине уявлення про те, як система буде реалізована в подальшому. Ця модель слугує як основа для розгортання, тестування, підтримки та оновлень програмної системи в майбутньому.

У контексті програмного забезпечення системи для шифрування та дешифрування даних діаграма розгортання Рис. 2.6 дозволяє створити чітке

уявлення про клієнтську частину з графічним інтерфейсом який розміщуватиметься на персональному комп'ютері користувача. Також база даних з усіма обліковими записами теж розміщуватиметься на персональному комп'ютері, таке рішення дозволяє уникнути можливості доступу до акаунтів з пристроїв інших користувачів навіть якщо вони якимось чином дізналися дані для входу в обліковий запис.

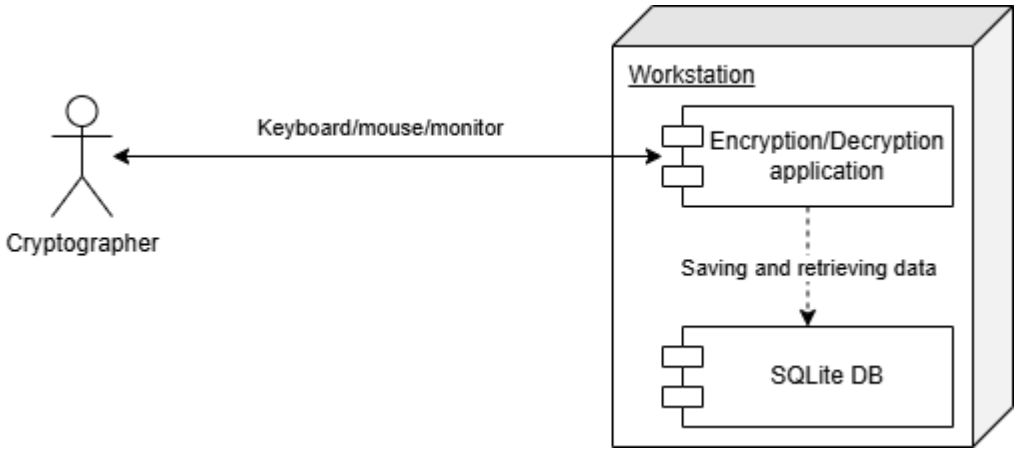

Рис. 2.5.1 – Діаграма розгортання

Діаграма розгортання в контексті даної системи є важливим інструментом для опису фізичної архітектури програмного забезпечення, взаємодії між компонентами системи, а також для візуалізації розміщення програмних компонентів на фізичних пристроях. Чітке розуміння як саме компоненти розміщені на пристроях покращує її ефективність та робить систему стійкішою до помилок.

Моделювання системи для шифрування допомагає виявити потенційні проблеми, слабкі місця, структуру та логіку взаємодії між компонентами. Крім того модель допомагає в подальшому супроводі та масштабуванні програмного забезпечення. Для системи, що працює з конфіденційною інформацією, важливою перевагою діаграми компонентів є чітке планування механізмів безпеки.

3 Розробка програмного забезпечення

3.1 Проєктування графічного інтерфейсу користувача

Проєктування графічного інтерфейсу користувача є важливим етапом розробки, адже він має визначити потреби користувача, зробити його досвід роботи із застосунком цікавим та не надто складним, особливо для таких систем як програмне забезпечення системи для шифрування та дешифрування даних. Метою графічного інтерфейсу є забезпечення зручного, інтуїтивно зрозумілого та ефективного способу виконання дій користувачем, наприклад, авторизацію, вибір файлів, запуск процесів тощо.

Під час проєктування графічного інтерфейсу користувача важливо враховувати принципи зручності взаємодії із системою, логіку розташування елементів у вікні, а також візуальну привабливість. Кожен елемент інтерфейсу має відповідати своїм функціям, наприклад кнопки мають бути підписані відповідно до того що вони виконують, поля для вводу мають дати користувачеві розуміння що в них потрібно вводити та який саме це має бути тип даних, повідомлення мають бути інформативними та не надто складними, а кольорові рішення мають бути послідовними та не втомлювати очі користувача. Також інтерфейс має інформувати користувача про будь які несправності або успішне виконання певних дій.

Особливу увагу варто приділити ергономіці роботи з файлами, забезпечити зручний вибір файлів, відображення інформації про них, а також повідомлення про збереження результатів. Також важливим є створення інтерфейса який адаптується під різні розширення екранів та їх роздільну здатність.

У випадку з програмним забезпеченням системи для шифрування та дешифрування даних, графічний інтерфейс користувача повинен демонструвати рівень безпеки та довіри до нього. Довіра до застосунку досягається шляхом створення чистого, стриманого дизайну, уникнення зайвих

елементів та шляхом інформування користувача у разі успішного чи не успішного виконання обробки дій користувача.

Загалом якісно спроєктований та розроблений графічний інтерфейс значно підвищує користувацький досвід, знижує ймовірність помилок при використанні програмного забезпечення та забезпечує комфортну і зрозумілу взаємодію із системою, що є критичним для подібних чутливих задач які виконують взаємодію із конфіденційною інформацією.

Для програмного забезпечення системи для шифрування та дешифрування даних прототипи інтерфейсу розроблявся у онлайн сервісі Figma Рис 3.1 – 3.6. Розроблено інтерфейси для входу в акаунт, реєстрації облікового запису та інтерфейс з функціоналом для шифрування файлів. Програмне забезпечення міститиме можливість змінювати режими інтерфейсу зі світлого на темний та навпаки, тому для кожного інтерфейсу розроблено два варіанти, а саме зі світлими та темними кольорами.

Дизайн:

- У світлому режимі використано світло-сірий колір для фону, колір тексту чорний, акценти чорно-білі.
- У темному режимі використано темно-сірий колір для фону, колір тексту світло-сірий, акценти чорно-білі.

На кожному інтерфейсі присутні такі об’єкти:

- Кнопка закриття програми.
- Кнопка розширення програми.
- Кнопка згортання програми.
- Кнопка зміни режиму з темного на світлий та навпаки.
- Логотип програмного забезпечення.
- Версія системи.

На сторінці входу присутні такі елементи:

- Напис “Вхід в акаунт”.
- Поля для введення логіна та пароля.
- Пункт для вибору відображення пароля.
- Кнопка для переходу в меню реєстрації.
- Кнопка “Вхід”.

Інтерфейс входу в обліковий запис є першою точкою взаємодії користувача з програмним забезпеченням, тому його дизайн має бути максимально зручним, простим, зрозумілим та візуально привабливим. На спроєктованих Рис. 3.1 та Рис. 3.2 зображено дві версії цього інтерфейсу, один демонструє як має виглядати інтерфейс в світлій темі, інший демонструє інтерфейс в темній темі. Різноманіття цих кольорової гами цих інтерфейсів надає користувачеві ширший вибір роботи з програмним забезпеченням. На центрі екрана чітко розташований заголовок “Вхід в акаунт”, який одразу інформує про призначення сторінки. Такий підхід проєктування створює позитивне перше враження і демонструє увагу до зручності користувача.

Основними елементами є поля для введення логіна та пароля. Вони мають зрозумілі підписи, які вказують на тип даних для введення. Також виділено окремий елемент, котрий допомагає відобразити пароль який по замовчуванню прихований, це покращує взаємодію користувача із системою та робить її простішою завдяки зменшенню можливості помилок під час введення тексту. Розміщення елементів є логічним, у межах зони погляду користувача, що прискорює процес авторизації. Також присутні кнопки переходу до меню реєстрації, що дає змогу неавторизованим користувачам створити обліковий запис, та кнопка для входу, що надає доступ до функціоналу системі вже авторизованим користувачам.

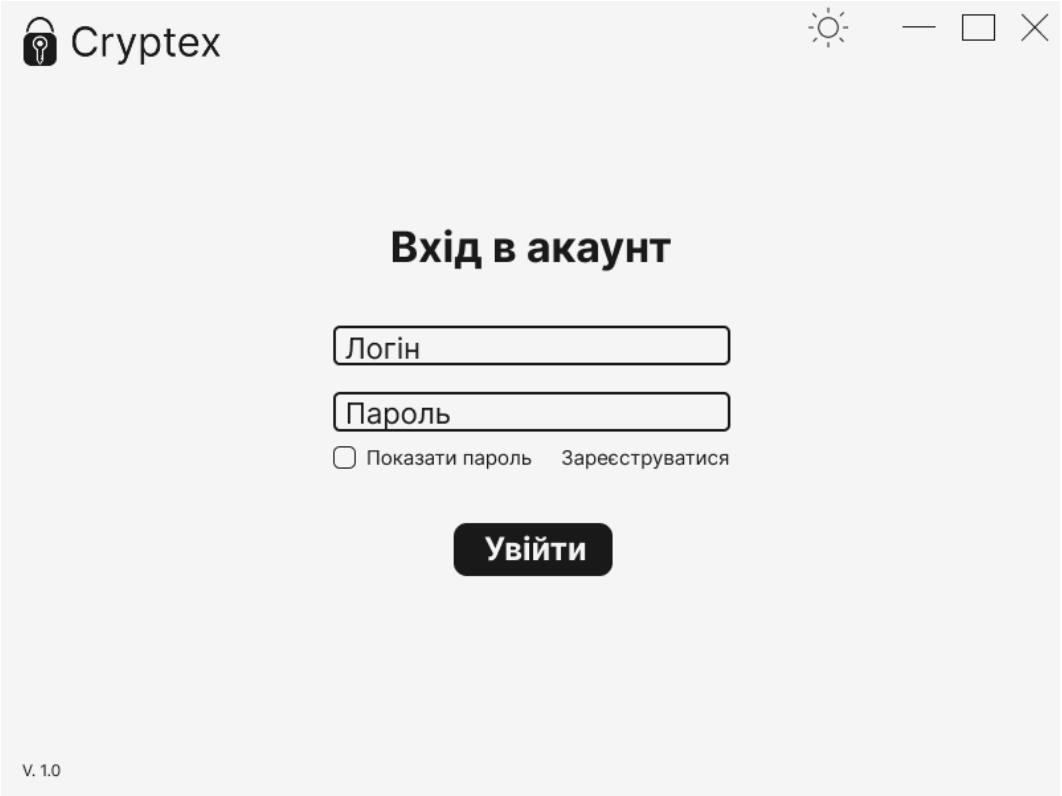

Рис. 3.1.1 – Інтерфейс входу в акаунт світлої теми

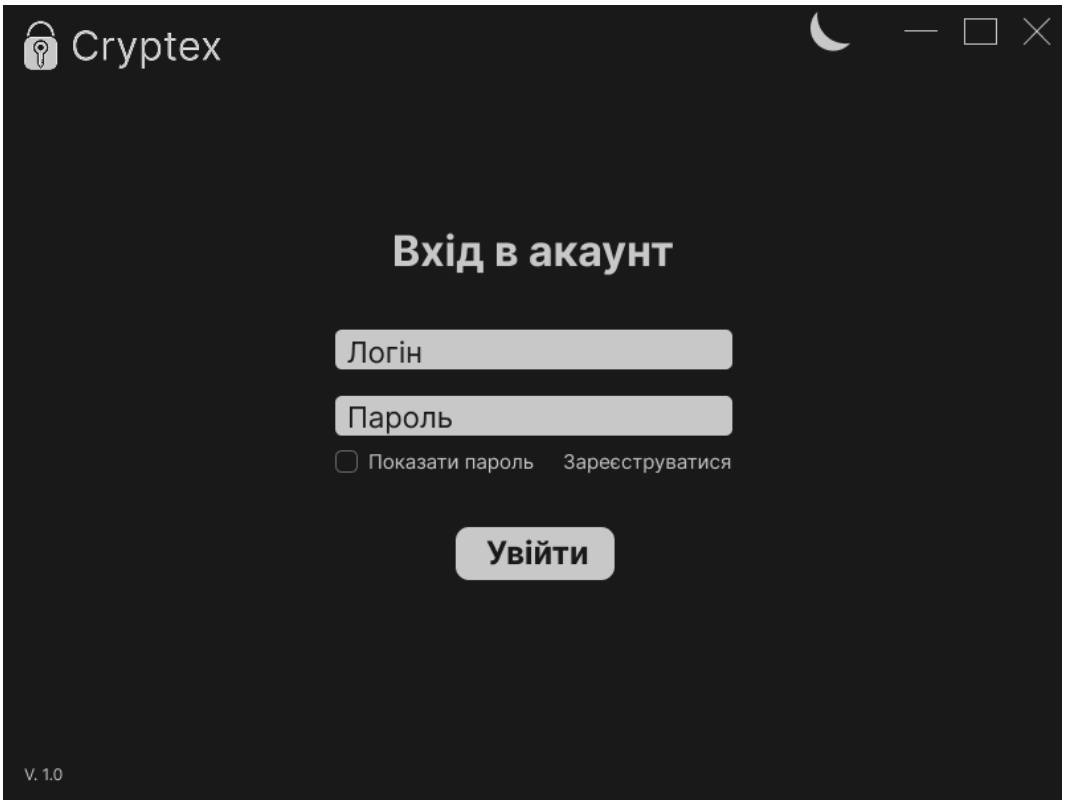

Рис. 3.1.2 – Інтерфейс входу в акаунт темної теми

На сторінці для реєстрації присутні такі елементи:

- Напис “Реєстрація”.
- Поля для введення логіна, пароля та пароля повторно.
- Пункт для вибору відображення пароля.
- Кнопка “Зареєструватися”.
- Кнопка “Назад” для повернення на попередню сторінку.

Інтерфейс для реєстрації облікового запису є важливим етапом у процесі взаємодії користувача з програмним забезпеченням, оскільки саме через нього створюється новий профіль. Проєктуючи інтерфейс створено дві візуальні версії цієї системи, світлу та темну. У центрі екрана розміщено заголовок “Реєстрація”, який відразу описує призначення сторінки. Таке рішення дозволяє користувачу швидко зорієнтуватися в контексті взаємодії з інтерфейсом.

Форма реєстрації містить три поля для вводу тексту, а саме поле для логіну, пароля і повторне введення пароля для підтвердження правильності введених даних. Така структура дозволяє мінімізувати ймовірність помилок під час створення облікового запису. Передбачено також опцію перегляду введених паролів, що забезпечує зручність для користувача та підвищує точність введених даних. Також інтерфейс містить кнопку “Зареєструватися”, щоб створити новий обліковий запис та кнопку “Назад” для повернення на сторінку входу в обліковий запис.

Темна та світла версії інтерфейсу виконані в одному стилі, проте мають різні кольорові схеми. Світла має сіруватий фон з темного кольору шрифтами, в той час коли темна версія має навпаки темного кольору фон, а шрифти сірого кольору. Обидві версії зберігають високу контрастність і доступність, також відповідають принципам зручності та інтуїтивності. Цей підхід до реалізації інтерфейсу забезпечує позитивний досвід користувача та підвищує загальну привабливість застосунка.

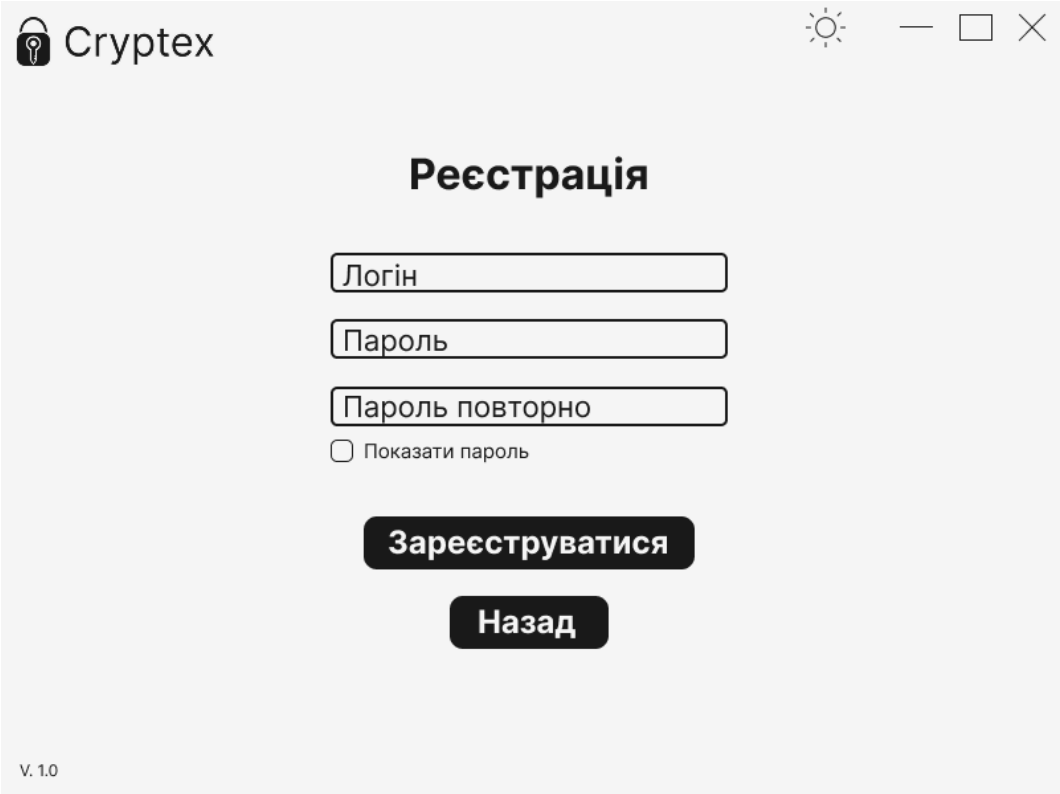

Рис. 3.1.3 – Інтерфейс реєстрації облікового запису світлої теми

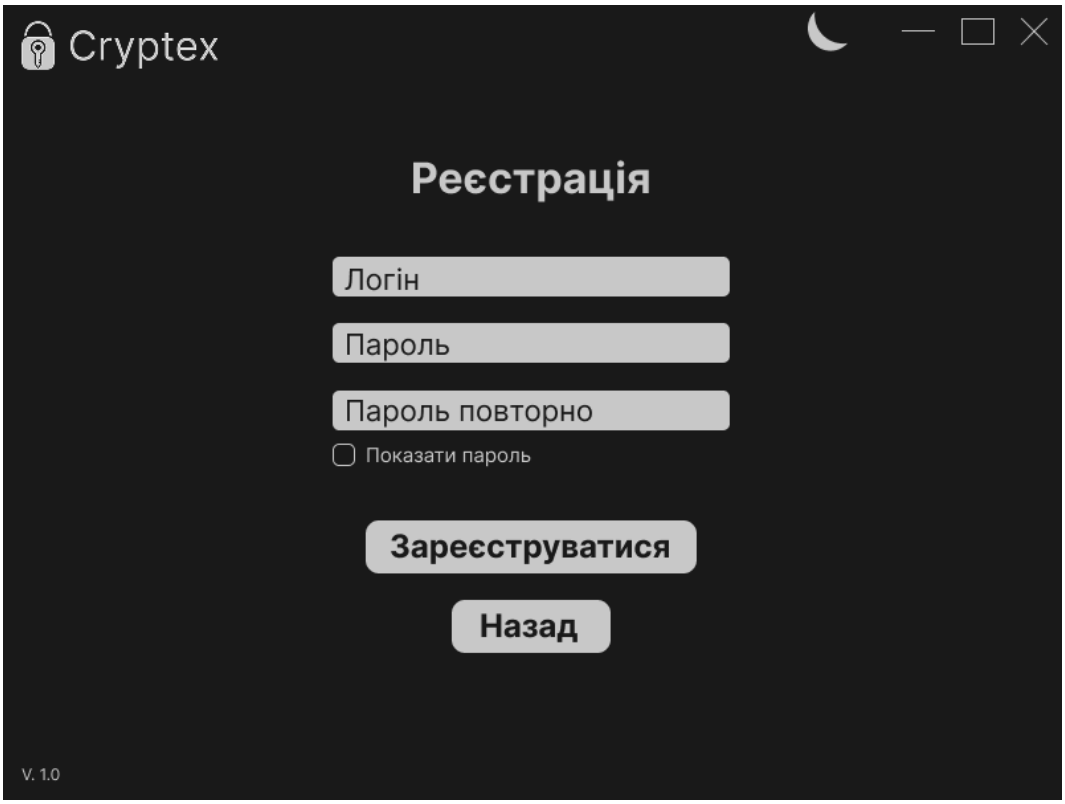

Рис. 3.1.4 - Інтерфейс реєстрації облікового запису темної теми

На сторінці з функціоналом присутні такі елементи:

- Кнопка “Вибрати файл”.
- Кнопка “Вийти”, для виходу зі свого облікового запису.
- Таблиця для відображення доданих в систему файлів.
- Таблиця для відображення файлів над якими проведено операції.
- Кнопка “Шифрувати”.

Інтерфейс сторінки з функціоналом є центральним елементом взаємодії користувача з програмним забезпеченням системи для шифрування та дешифрування даних. Цю сторінку спроєктовано у двох версіях – світлій та темній, що задовольняє різні вподобання користувачів і забезпечує комфортну роботу за будь яких умов освітлення. Основними елементами сторінки є таблиця на яких продемонстровані всі файли які були додані до системи для проведення над ними операцій шифрування або дешифрування.

На сторінці передбачено також допоміжну таблицю в якій відображаються створені файли що містять зашифровану або дешифровану інформацію. Одними з основних елементів керування є кнопки, кнопка “Вибрати файл” запускає файлову систему, що дозволяє користувачеві обрати файл над яким потрібно провести операції. Кнопка “Вийти” дає змогу користувачеві вийти з облікового запису щоб ніхто не міг отримати доступ до його персональної інформації. Кнопка “Шифрувати” виконує операції над вибраним в першій таблиці файлом в залежності від його вмісту це буде або шифрування, або дешифрування, адже використовуватиметься симетричний алгоритм шифрування.

Рис. 3.1.5 – Інтерфейс з функціоналом для шифрування світлої теми

Рис. 3.1.6 – Інтерфейс з функціоналом для шифрування темної теми

3.2 Розробка графічного інтерфейсу користувача

Для розробки графічного інтерфейсу користувача обрано мову програмування C++ через свою високу продуктивність, контроль над пам’яттю та широкою можливістю роботи з файлами та алгоритмами. У контексті криптографічних операцій, де важливими є швидкість обробки та ефективне використання ресурсів, мова C++ є найкращим вибором. Крім того ця мова дозволяє легко реалізувати складну бізнес-логіку та дає гнучкість при роботі з різними структурами даних, що є важливим для підтримки й стабільної роботи програмного забезпечення.

Як основу для побудови графічного інтерфейсу користувача обрано фреймворк Qt. Qt забезпечує багаті можливості для створення сучасних, адаптивних і кросплатформних GUI-рішень, що дозволяє запускати застосунок на різних операційних системах без значних змін у коді. Крім того, Qt має зручну систему сигналів та слотів, яка суттєво полегшує взаємодію між елементами інтерфейсу та логікою програмного забезпечення. Інструменти які надає фреймворк Qt дозволяють швидко та ефективно створити інтуїтивно зрозумілий вікна та форми.

Для зберігання даних обрано реляційну базу даних SQLite, яка ідеально підходить для застосунків з помірним обсягом даних та локальним зберіганням інформації. SQLite не потребує окремого сервера баз даних, що спрощує розгортання системи та знижує вимоги до середовища її роботи. Окрім цього, SQLite підтримується безпосередньо в фреймворку Qt, що дозволяє легко інтегрувати зберігання та обробку даних у загальний застосунок. Це рішення забезпечує надійне збереження інформації про облікові записи користувачів та ключі шифрування для алгоритмів в одному зручному та ефективному форматі.

Якщо коротко описати вибір інструментів для розробки програмного забезпечення системи для шифрування та дешифрування даних то можна виділити пункти:

1. Чому мова програмування C++?
    - Продуктивність: C++ забезпечує високу швидкодію, що є важливим для системи шифрування даних.
    - Гнучкість: Підтримка процедурного, об’єктно-орієнтованого та узагальненого програмування дозволяє створювати масштабні та ефективні рішення.
    - Сумісність: C++ легко взаємодіє з іншими мовами та бібліотеками, що може бути корисним при розширенні функціоналу.
2. Чому фреймворк Qt?
    - Розширені можливості GUI: Qt містить потужні засоби для створення інтерфейсів, включаючи готові компоненти, підтримку тем та стилів.
    - Зручність у роботі з базою даних: Qt має вбудовану підтримку баз даних через Qt SQL Module, що спрощує інтеграцію з SQLite.
    - Кросплатформеність: Qt підтримує розробку під операційні системи Windows, Linux, macOS та мобільні платформи, що забезпечує широку підтримку користувачів.
3. Чому база даних SQLite?
    - Легка інтеграція: SQLite є вбудованою базою даних, що не потребує окремого серверного середовища.
    - Проста у використанні: Відсутність складних налаштувань та легкість у розгортанні робить її ідеальною для настільних застосунків.
    - Підтримка Qt: Qt містить вбудовану підтримку SQLite для невеликих або середніх програмних систем, що не потребують складної мережевої бази даних.

За допомогою вищезгаданих інструментів розроблено графічний інструмент користувача, який повністю відповідає раніше розробленим прототипам інтерфейсу Рис. 3.7 – 3.12.

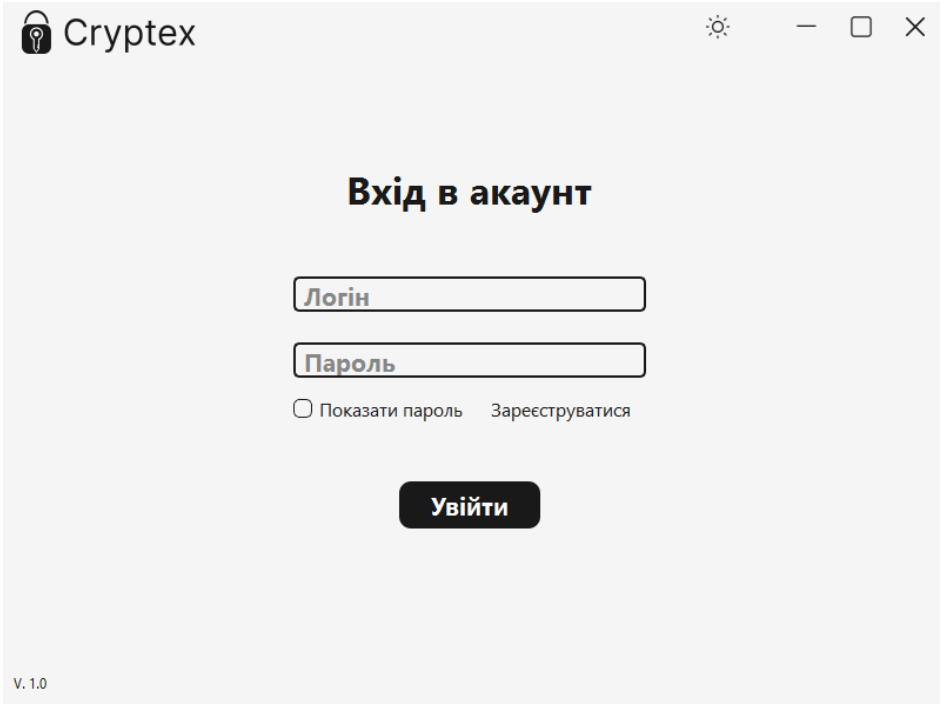

Рис. 3.2.1 – Розроблена сторінка входу в обліковий запис у світлій темі

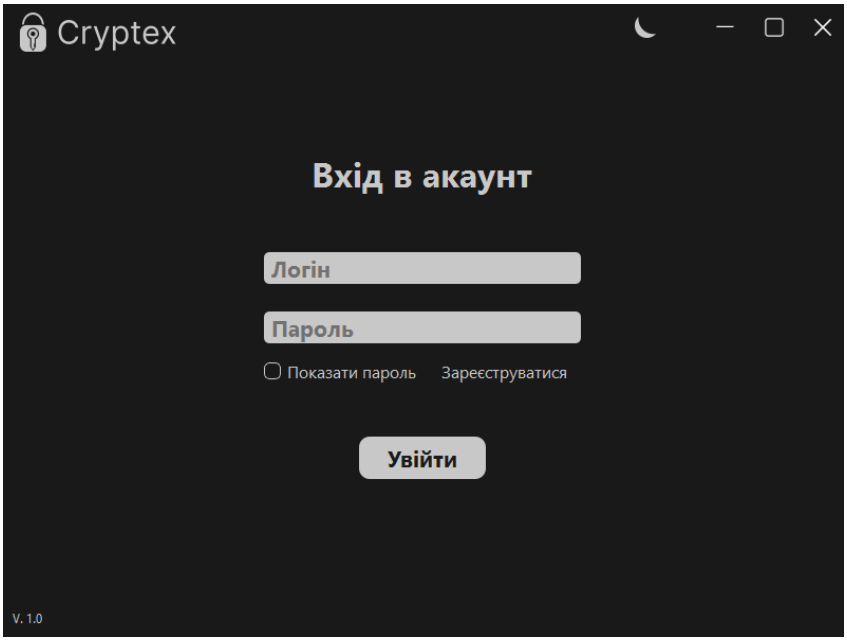

Рис. 3.2.2 – Розроблена сторінка входу в обліковий запис у темній темі

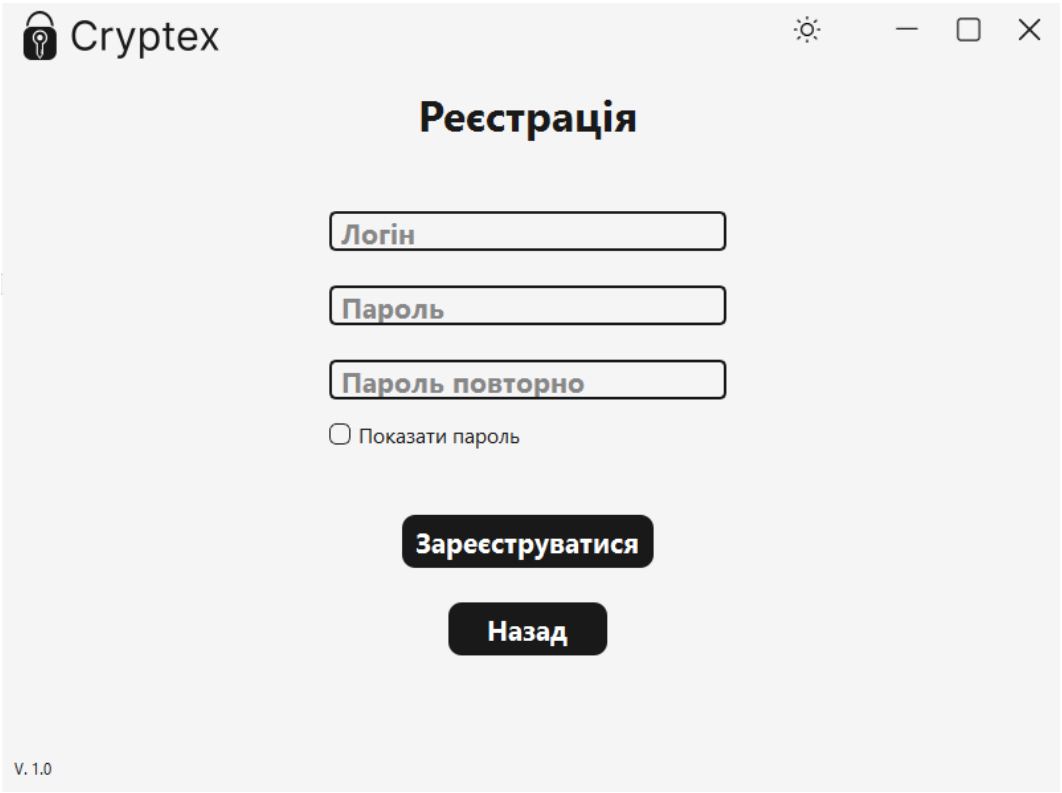

Рис. 3.2.3 – Розроблена сторінка реєстрації облікового запису у світлій темі

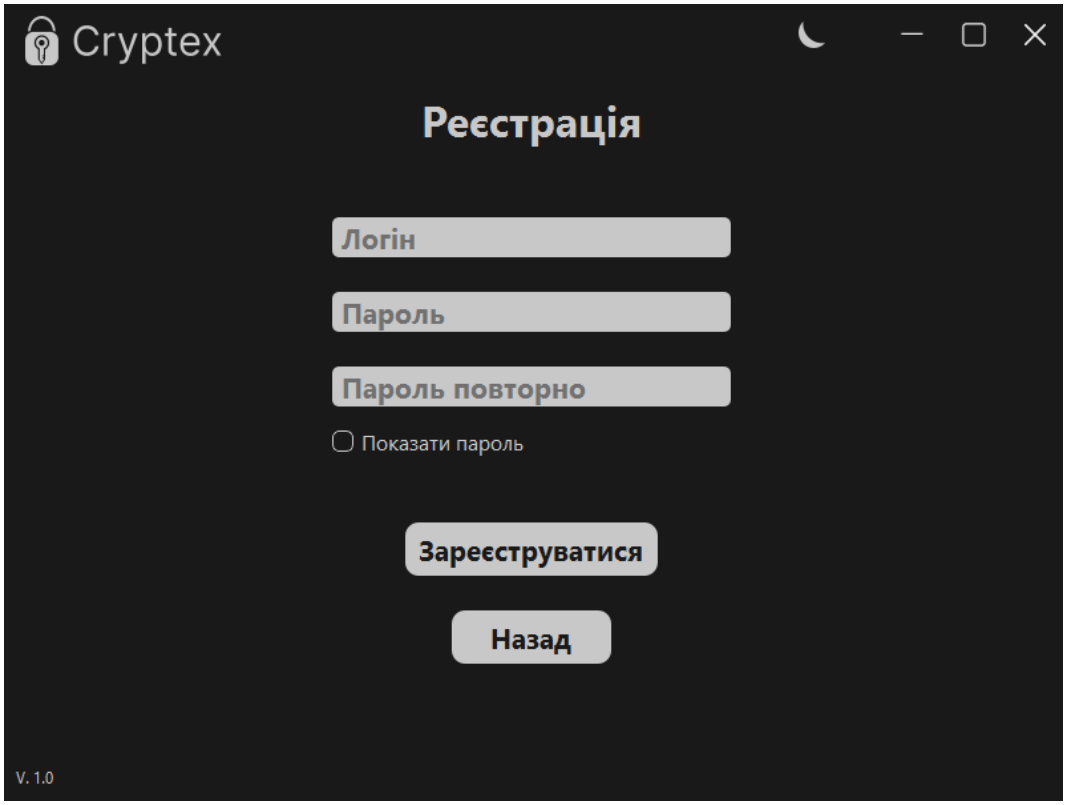

Рис. 3.2.4 – Розроблена сторінка реєстрації облікового запису у темній темі

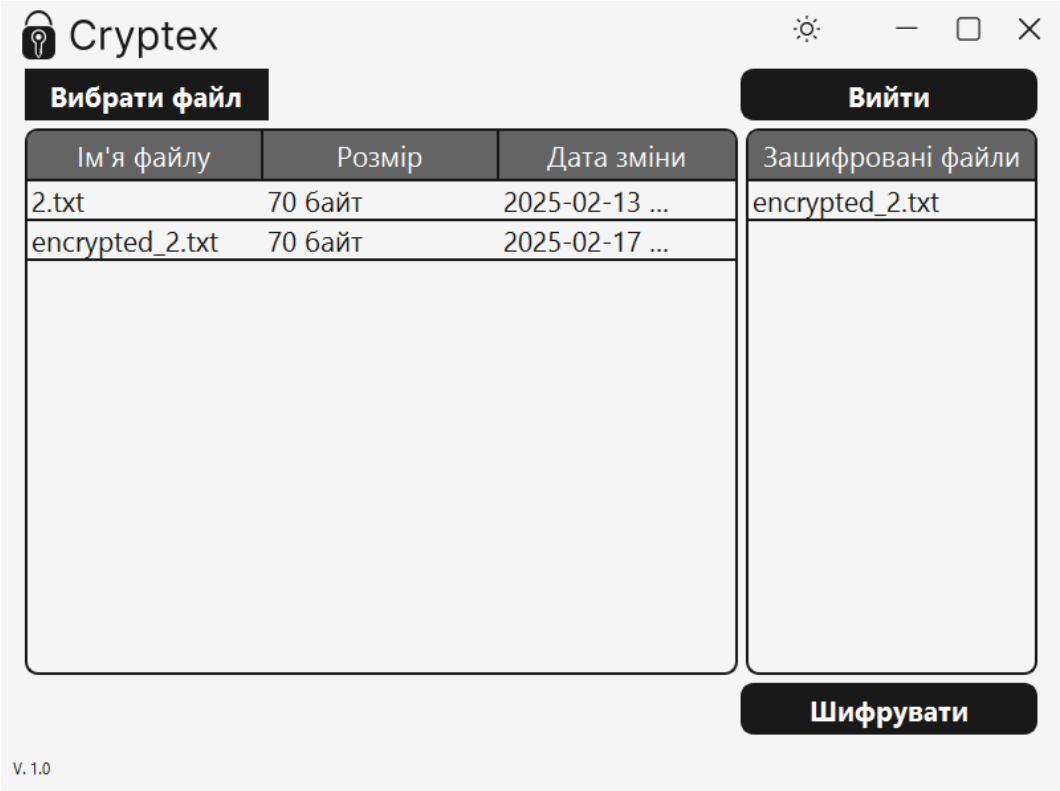

Рис. 3.2.5 – Розроблена сторінка з функціоналом у світлій темі

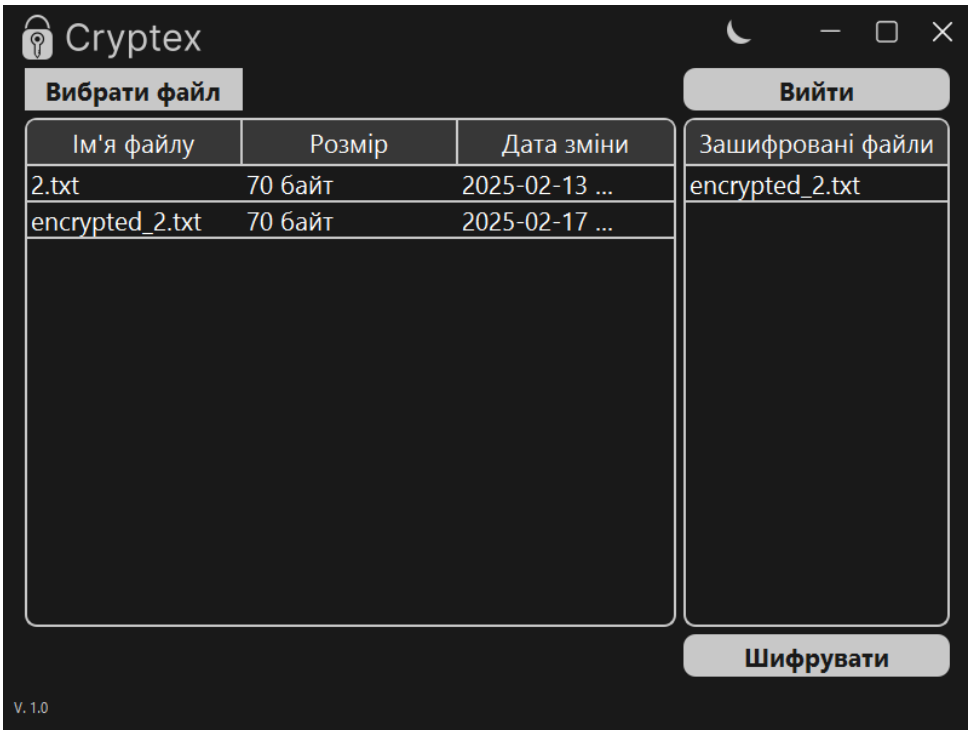

Рис. 3.2.6 – Розроблена сторінка з функціоналом у темній темі

3.3 Дослідження та вибір алгоритму шифрування

Сучасний світ складно уявити без цифрових технологій, адже вони займають значну частину нашого життя, тому питання безпеки передання інформації за їх допомогою набуває не аби якого значення. Щодня великі потоки інформації отримують та надсилають, а отже є ті хто намагаються цю інформацію перехопити, задля запобігання потрапляння інформації в чужі руки використовують алгоритми що їх шифрують, тим самим приховуючи від небажаних очей. Шифрування інформації це найефективніший метод її захисту, для перегляду зашифрованої інформації використовують алгоритми її дешифрування.

Програмне забезпечення системи для шифрування та дешифрування даних надає функціонал для шифрування файлів тим самим приховуючи її від небажаних очей. Постає питання про вибір алгоритму шифрування, який краще використати, який буде стійкіший до зламів, який краще підійде для користувачів та який буде найшвидший.

В світі існує безліч алгоритмів для приховування інформації які люди здавна використовують. Для вибору підходящого алгоритму розглянуто кілька варіантів, проведено їх порівняння та можливості інтеграції в систему.

Алгоритми та їх принципи роботи:

1. XOR алгоритм поєднує з кожен біт з ключем шифрування за допомогою логічної операції XOR (^).
2. AES алгоритм поділяє дані на блоки по 128 бітів, кожен блок проходить через перестановки, зміщування стовпців, підстановки та поєднання з ключем.
3. RSA алгоритм шифрує дані за допомогою відкритого ключа, але розшифровує їх за допомогою приватного, він побудований на основі математичної складності факторизації великих простих чисел.

4. Ceasar метод при якому кожну літеру алфавіту зсувають на фіксовану кількість позицій.

5. DES та 3DES алгоритми шифрують дані блоками по 64 біти з ключем по 56 бітів, вони застосовують 16 серій перестановок та підстановок.

Таблиця 3.3.1

Алгоритм	XOR	AES	RSA	Ceasar	DES та 3DES
Складність реалізації	Дуже проста	Складна	Дуже складна	Дуже проста	Середня
Швидкість роботи	Дуже висока	Висока	Низька	Висока	Середня
Безпека	Низька або середня, залежить від ключа	Висока	Дуже висока	Дуже низька	Середня або висока
Ресурсомісткість	Мінімальна	Середня або висока	Висока	Мінімальна	Середня

Переваги цих алгоритмів:

1. XOR алгоритм легкий в реалізації та потребує мінімум ресурсів.
2. AES алгоритм дуже надійний та широко застосовується.
3. RSA алгоритм використовує асиметричне шифрування та має високу безпеку.
4. Ceasar алгоритм має просту реалізацію.
5. DES та 3DES алгоритми є стандартизованими.

Недоліки кожного з алгоритмів:

1. XOR алгоритм є небезпечним при застосування постійного або слабкого ключа.
2. AES алгоритм потребує більше ресурсів відповідно час реалізації теж збільшується.
3. RSA алгоритм повільний особливо при роботі з великими обсягами інформації.
4. Ceaser алгоритм легко зламується через що є непридатним для серйозного використання.
5. DES та 3DES алгоритми є застарілими та менш безпечними порівняно з AES.

Виходячи з результатів досліджених алгоритмів найкращим вибором для програмного забезпечення системи для шифрування та дешифрування даних є алгоритм XOR через його простоту реалізації, швидкодію, мінімальну потребу ресурсів, симетричність та гнучкість у роботі з ключем. Даний алгоритм був би не найкращим вибором для програмних забезпечень, що вимагають високий рівень криптографічного захисту, наприклад банкові системи, застосунки що містять обмін інформації в публічних мережах тощо. Та для локальної системи, де основною метою є базовий захист від звичайного користувача, або випадкового прочитання алгоритм XOR цілком є виправданим.

Ще однією з переваг алгоритму XOR є його симетричність, тобто можливість використання як шифрування так і дешифрування без зміни алгоритму, достатньо повторно використати той самий метод, з тим самим ключем. Це значно спрощує реалізацію логіки програмного забезпечення, не потрібно створювати окремі функції для шифрування та дешифрування, що знижує ризик помилок і скорочує кількість коду. З технічного погляду на систему алгоритм XOR дозволяє уникнути затримок при обробці великих обсягів даних.

3.4 Інтеграція алгоритму шифрування в систему

Створення алгоритму шифрування XOR на мові програмування C++ із застосуванням фреймворку Qt є одним з найважливіших етапів розробки програмного забезпечення системи для шифрування та дешифрування даних. Алгоритм шифрування XOR є симетричним через використання побітового оператора XOR(^) Рис. 3.4.1 в мові програмування C++, цей оператор працює за таким принципом: бере по біту з кожної змінної та порівнює їх, якщо обидва біти є однаковими то записується результат 0, якщо вони є різними то результатом буде 1. Такий підхід дозволяє здійснити як шифрування так і дешифрування використовуючи один і той самий метод, що робить алгоритм XOR ефективним та простим для реалізації.

```
for (int i = 0; i < fileData.size(); ++i)
{
    fileData[i] = fileData[i] ^ key;
}
```

Рис. 3.4.1 – Виконання шифрування за допомогою оператора XOR

У межах проєкту в Qt алгоритм XOR інтегрується в логіку обробки файлів. Реалізація передбачає відкриття вхідного файла в двійковому режимі Рис. 3.4.2, отримання інформації його вмісту у буфер, та застосування XOR операцій з використанням унікального ключа який прив’язаний до даного облікового запису, а потім збереження результату у новий файл. Фреймворк Qt надає зручні класи QFile, QByteArray та QTextStream, що значно покращують роботу з файлами та байтовими масивами.

```
QFile file(filePath);
if (!file.open(QIODevice::ReadOnly))
{
    QMessageBox::warning(nullptr, "Попередження", "Не вдалося відкрити файл!");
    return;
}

QByteArray fileData = file.readAll();
file.close();
```

Рис. 3.4.2 – Відкриття файла та збереження інформації у буфер

Ключ для операції XOR створюється випадковим чином на етапі створення облікового запису та зберігається в базі даних SQLite, він генерується без втручання користувача, так що той навіть не підозрює про його створення. Для забезпечення безпеки важливо, щоб для кожного користувача генерувався унікальний ключ.

Основний шифрувальний метод реалізований як окрема функція яка працює з базою даних для отримання ключа шифрування, з файлами для отримання інформації та зберігання її зашифрованого вигляду. Цей підхід дозволяє обробляти файли будь-якого розміру, а Qt забезпечує стабільність.

Щоб реалізувати зручну взаємодію з алгоритмом, на головному вікні знаходиться кнопка “Шифрувати”, під час натискання на яку запускається функція для проведення операції шифрування чи дешифрування. Після вибору з таблиці необхідного файла, він автоматично зчитується та виконується операція, результат автоматично зберігається у новий файл в якому вказана операція що проводилася, в імені файла Рис. 3.4.3.

```
QString encryptedName = QFileInfo(filePath).fileName();
QString encryptedFilePath;
if (encryptedName.startsWith("encrypted_"))
{
    // Якщо є приставка "encrypted_", створюємо файл з приставкою "decrypted_"
    encryptedFilePath = QFileInfo(filePath).absoluteDir().filePath("decrypted_"
        + QFileInfo(filePath).fileName().mid(10));
}
else
{
    if (encryptedName.startsWith("decrypted_"))
    {
        // Якщо є приставка "decrypted_", створюємо файл з приставкою "encrypted_"
        encryptedFilePath = QFileInfo(filePath).absoluteDir().filePath("encrypted_"
            + QFileInfo(filePath).fileName().mid(10));
    }
    else
    {
        // Якщо приставки немає, створюємо файл з приставкою "encrypted_"
        encryptedFilePath = QFileInfo(filePath).absoluteDir().filePath("encrypted_"
            + QFileInfo(filePath).fileName());
    }
}
QFile encryptedFile(encryptedFilePath);
```

Рис. 3.4.3 – Створення нового файла з відповідним іменем та збереження зашифрованих даних в нього

Qt також дозволяє обробляти виключення та відображати повідомлення у разі якщо не знайдено ключ для шифрування, або не вдалося відкрити файл чи створити новий. Завдяки цьому користувач завжди бачить, що саме відбувається з системою і що треба зробити для забезпечення її коректної роботи. Це дуже важливо при роботі з чутливими даними.

Також передбачено автоматичне визначення, чи вже файл був зашифрований, чи ні. За допомогою додавання спеціального префікса у створений системою новий файл користувач має змогу зрозуміти без відкриття файла чи є він дешифрованим, чи зашифрованим. Це дозволяє уникнути проведення повторної операції над файлом, яке б призвело б до втрати коректності даних.

Загалом розробка алгоритму XOR у Qt за допомогою мови програмування C++ забезпечила просту, зрозумілу, надійну та ефективну реалізацію програмного забезпечення системи для шифрування та дешифрування даних. Вона легко інтегрується у весь програмний комплекс, має підтримку роботи з базою даних, має інтуїтивний та простий у використанні інтерфейс та забезпечує чітке логічне розділення між інтерфейсною та функціональною частинами системи.

3.5 Реалізація бази даних

Реалізація бази даних є ключовим етапом у розробці програмного забезпечення системи для шифрування та дешифрування даних., оскільки вона забезпечує зберігання важливої інформації, такої як облікові записи користувачів та ключі для шифрування що до них прив’язані. Для даної системи обрано базу даних SQLite через її легкість, вбудованість, відсутність потреби у встановленні окремого серверного ПЗ та повну інтеграцію з фреймворком Qt.

У SQLite всі дані зберігаються у вигляді одного файлу на диску, що робить її ідеальною для десктопної системи. Крім того дана база даних не вимагає налаштування прав доступу або складних підключень, файл бази даних створюється автоматично при першому запуску програми, якщо його не існує.

Першим етапом роботи з базою даних стало створення структури таблиці яка міститиме всю інформацію про користувача: логін, пароль, ключ для шифрування. Кожен обліковий запис містить унікальний первинний ключ, таке рішення гарантує що кожен користувач буде унікально ідентифікований та гарантуватиме безпеку зберігання пароля. Паролі не зберігаються у відкритому вигляді, вони хешуються, що запобігає компрометації у разі доступу сторонніх осіб до файлу бази даних.

Ключі для шифрування зберігаються разом з обліковими записами користувачів і є випадково згенерованими під час створення нових акаунтів. Це рішення дозволяє системі автоматично використовувати ключ для проведення операцій. В базі даних він зберігається у вигляді двійкового значення що зменшує ризики того, що сторонні особи зможуть отримати інформацію із зашифрованих файлів.

Реалізація бази даних здійснюється в коді за допомогою модуля QtSql. У момент запуску системи здійснюється перевірка, чи існує база даних, якщо ні то виконується SQL-скрипт і вона автоматично створюється Рис. 3.5.1. Дане рішення дозволяє зробити програмне забезпечення автономним та здатним самостійно налаштовувати середовище.

```
QSqlQuery query;
query.exec("CREATE TABLE IF NOT EXISTS users ("
           "id INTEGER PRIMARY KEY, "
           "username TEXT UNIQUE NOT NULL, "
           "password TEXT NOT NULL,"
           "key BLOB NOT NULL)"
);
```

Рис. 3.5.1 – SQL запит на створення таблиці

Для кожного запиту таких як реєстрація, вхід в обліковий запис, збереження ключа використовуються підготовлені SQL-запити Рис. 3.5.1 -, таке рішення забезпечує захист від SQL-ін’єкцій. Дані користувачів перевіряються на коректність, чи немає облікових записів з таким логіном, а тільки потім заносяться до бази даних, що сприяє стабільній роботі програми.

```
QSqlQuery queryCheck;
queryCheck.prepare("SELECT COUNT(*) FROM users WHERE username = :username");
queryCheck.bindValue(":username", username);

if (queryCheck.exec() && queryCheck.next() && queryCheck.value(0).toInt() > 0)
{
    ui->label_13->setText("Цей логін вже використовується!");
    return;
}
```

Рис. 3.5.2 – Запит для перевірки на наявність облікового запису з подібним логіном

```
QSqlQuery query;
query.prepare("INSERT INTO users (username, password, key) "
              "VALUES (:username, :password, :key)");
query.bindValue(":username", username);
query.bindValue(":password", hashPassword);
query.bindValue(":key", bKey);
```

Рис. 3.5.3 – Запит для додавання даних про новий обліковий запис в таблицю

```
QSqlQuery query;
query.prepare("SELECT * FROM users WHERE username = :username");
query.bindValue(":username", m_login);

QByteArray bKey;

if (!query.exec()) {
    qDebug() << "Помилка виконання запиту:" << query.lastError().text();
    return;
}

// Отримання ключа з бази даних у форматі QByteArray
if (query.next()) {
    bKey = query.value(3).toByteArray();
}
```

Рис. 3.5.4 – Запит для отримання ключа для шифрування з бази даних

Структура бази даних реалізована таким чином, щоб бути максимально простою, але розширюваною в майбутньому. За необхідності до бази даних можна легко додати нові таблиці для зберігання додаткової інформації пов’язаною з безпекою.

Крім основної реалізації бази даних важливим етапом було забезпечення її ефективної інтеграції з усім інтерфейсом системи. Наприклад під час входу в обліковий запис SQL- запит порівнює введені дані з хешованими паролями, та у разі збігу надає доступ до функціоналу. Для додавання коректних даних нових облікових записів система спершу перевіряє

пароль на коректність його вводу, для цього є два поля для паролів які порівнюються між собою.

3.6 Логіка роботи програмного забезпечення

Логіка програмного забезпечення – це сукупність правил та умов, які визначають порядок та спосіб застосування функціоналу програмного забезпечення. Логіка описує, як система має реагувати на дії користувача, як обробляти дані, коли запускати певні процеси та в якій послідовності. Логіка охоплює всі рівні функціонування починаючи з обробки на коректність введених даних користувача, закінчуючи виконанням криптографічних алгоритмів. Вона є складовою програмного забезпечення , що надає передбачуваність та безпечні роботу системи.

Наявність чітко спроєктованої логіки особливо важливе значення має для програм, що пов’язані з обробкою персональних даних та безпекою, як у випадку із програмним забезпеченням системи для шифрування та дешифрування даних. Помилки в логіці можуть призвести до втрат конфіденційних даних, порушенню доступу до них або зробити систему більш вразливою до атак на неї. Саме тому логіка програмного забезпечення є основою стабільного функціонування системи.

Взаємодія користувача з програмним забезпеченням системи для шифрування та дешифрування даних відбувається за допомогою клавіатури, миші та монітору. Програмне забезпечення має бути встановленим на персональний комп’ютер та відображати свій інтерфейс для роботи з користувачем.

Алгоритм входу користувача в обліковий запис реалізований у кілька послідовних етапів які продемонстровані на блок-схемі Рис. 3.6.1, це забезпечує безпечну та коректну авторизацію. Після запуску програми користувач потрапляє на екран входу, де він має ввести логін та пароль у відповідних

полях. Ці дані перевіряються чи існує вказаний обліковий запис в базі даних,якщо так то надається доступ до функціоналу, в іншому випадку виводиться повідомлення про помилку.

Рис. 3.6.1 – Блок-схема логіки входу користувача в обліковий запис

Якщо дані коректно введені система звертається до бази даних, де зберігається інформація про зареєстрованих користувачів. Логіни порівнюються з наявними записами, а пароль проходить перевірку на відповідність. Для безпеки пароль що вводиться хешується тим самим способом що і при створенні облікового запису, а потім порівнюється з уже збереженим хешем у базі. У випадку збігу користувач успішно авторизується та переходить до головного інтерфейсу.

У разі неправильного логіна або пароля користувач отримає повідомлення про помилку входу. Система не розкриває який саме з параметрів введено некоректно для запобіганню несанкціонованого доступу. Такий підхід до авторизації забезпечує як зручність користування, так і високий рівень безпеки в системі для шифрування та дешифрування даних.

Алгоритм реєстрації нового облікового запису реалізовано з урахуванням перевірки введених даних, унікальності логіна та безпеки зберігання пароля. Користувач на сторінці реєстрації облікового запису повинен ввести логін, пароль та повторити пароль для перевірки на коректність пароля для зберігання. Усі поля є обов’язковими для заповнення, якщо одне з цих полів не буде заповнено, система виведе повідомлення з помилкою та описом.

Також система перевіряє логін та пароль на довжину символів, довжина логіна має бути принаймні 4 символи, а довжина пароля не менше восьми. Після попередньої перевірки система звіряє, чи не існує облікового запису в базі даних з таким самим логіном, якщо такий вже існує то система повідомляє про це користувача.

Якщо всі перевірки пройдено успішно, створюється випадковим чином ключ для шифрування. Пароль хешується і разом з логіном та ключем для шифрування зберігається в базі даних. Після цього з’являється повідомлення про успішну реєстрацію облікового запису і користувач має змогу повернутися на сторінку авторизації. Цей алгоритм дозволяє забезпечити унікальність облікових записів, захистити конфіденційні записи та створити безпечне середовище для користувачів системи.

На Рис. 3.6.2 зображено блок-схему, яка демонструє логіку роботи алгоритму для реєстрації облікових записів, вона слугує наочним способом представлення послідовних дій та умов які виконує система.

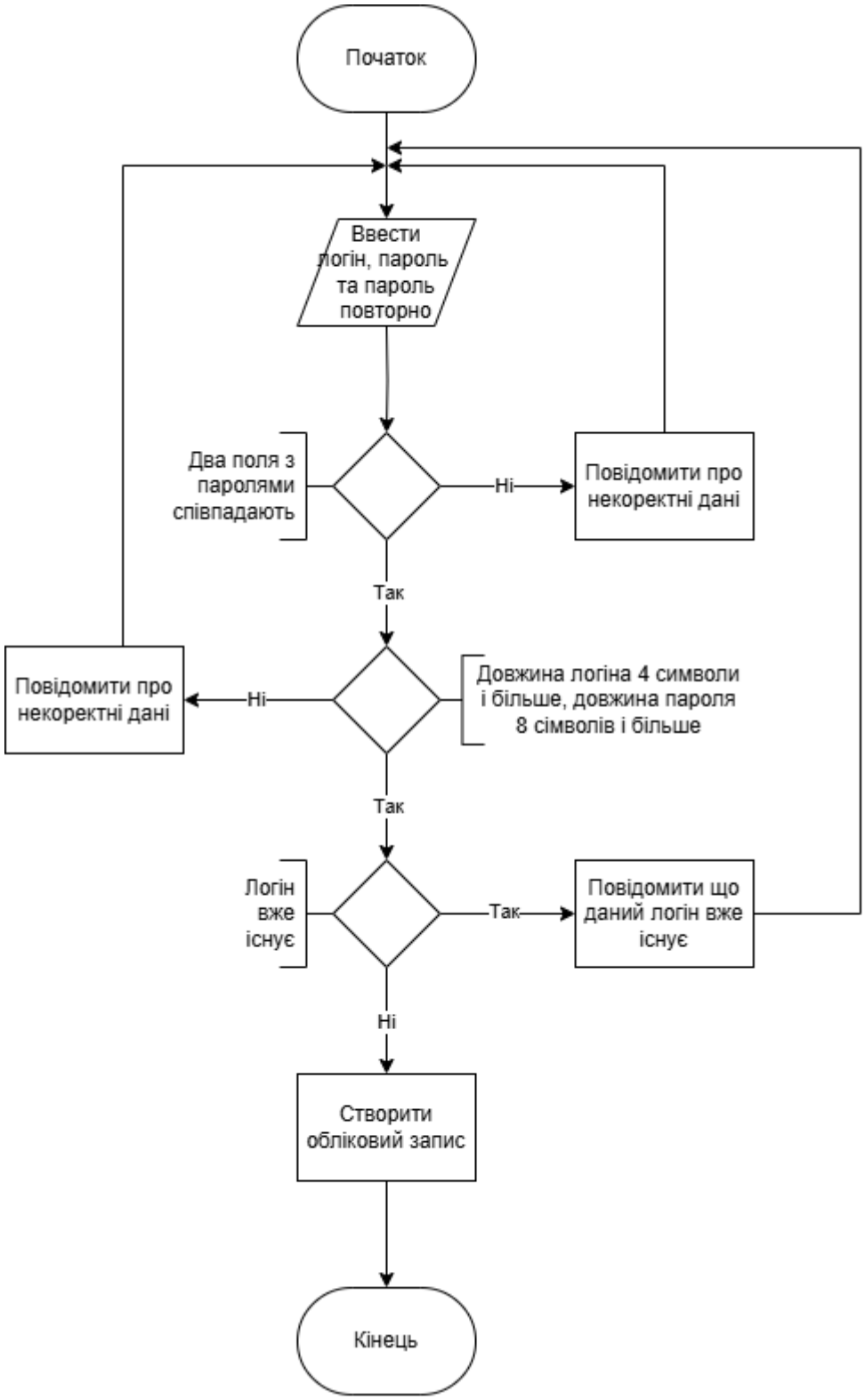

Рис. 3.6.2 – Блок-схема логіки реєстрації нового облікового запису

Алгоритм шифрування файлу в програмному забезпеченні системи для шифрування та дешифрування даних реалізується за допомогою симетричного

шифрування на основі операції XOR. Він працює шляхом поетапної обробки кожного байта вмісту файла, використовуючи ключ, що зберігається в базі даних і прив’язаний до конкретного користувача. Такий підхід дозволяє перетворити вміст файла у вигляд, непридатний для читання без відповідного ключа.

Алгоритм починається з вибору файла користувачем через графічний інтерфейс, далі файл відкривається в бінарному режимі для читання, якщо відкрити не вдалося то на екрані з’явиться повідомлення з помилкою. Основна суть шифрування полягає у тому, що кожен байт вмісту зчитується та проходить операцію XOR з відповідним байтом ключа шифрування. Якщо ключ коротший за вміст файла він повторюється циклічно, отримані зашифровані символи зберігаються у буфер після чого створюється новий файл та в нього записується зашифрована інформація.

Після завершення обробки всіх байтів, вихідний файл автоматично зберігається у заздалегідь визначену директорію, або поруч з оригіналом, але з відповідним префіксом. По завершенню алгоритм виводить користувачеві повідомлення про успішне проведення операції шифрування. Такий метод є простим у реалізації, але ефективним для шифрування персональних даних.

Блок-схема, що зображує логіку роботи алгоритму шифрування файла Рис. 3.6.3, слугує візуальним представленням послідовності кроків, які виконує користувач та система під час обробки файлу. Вона починається з початкового блоку, де користувач обирає файл для шифрування. Далі перевіряється чи файл справді вибрано, якщо ні то на екрані висвітлиться про це повідомлення. Наступним кроком є проведення методу шифрування над файлом, результат операції зберігається у новий файл та на екрані з’являється повідомлення про успішне виконання шифрування.

Рис. 3.6.3 – Блок-схема логіки шифрування файла

3.7 Тестування програмного забезпечення

Тестування програмного забезпечення є найважливішим етапом життєвого циклу розробки системи, який допомагає виявити помилки та недоліки, ще до передачі системи замовнику. Метою тестування є перевірка відповідності системи вимогам, чи працюють всі функції правильно та чи є стабільною й безпечною. Це важливий етап, що забезпечую якість та надійність програмного забезпечення.

Тестування здійснюється вручну, або автоматизовано. Ручне тестування передбачає, що тестувальник самостійно перевірить справність системи та її функцій, виловить всі помилки та відправить їх на виправлення. Автоматизоване ж тестування запускає сценарії для виявлення несправностей та задокументовує їх, воно є ефективним при великій кількості повторюваних перевірок.

Серед типів тестування основним є модульне, що перевіряє окремі частини коду, такі як функції чи класи. Для програмного забезпечення системи для шифрування та дешифрування даних важливим є тестування правильності роботи алгоритму XOR для різних вхідних даних. Модульне тестування дозволяє локалізувати помилки на ранньому етапі та забезпечує зручність у підтримці коду.

Інтеграційне тестування перевіряє коректність взаємодії модулів між собою. У контексті даної системи це перевірка взаємодії графічного інтерфейсу з функцією в якій шифруються дані, перевірка роботи з базою даних та файловою системою. Це тестування допомагає виявити помилки, що виникають на стику компонентів.

Системне тестування оцінює функціонування всієї системи як єдиного цілого. Для розробленого програмного забезпечення це означає перевірку всього процесу, від авторизації користувача в системі, до шифрування

інформації та її збереження. На цьому етапі можна оцінити поведінку системи на різноманітні дії користувача.

Свою роль також відіграє й тестування графічного інтерфейсу користувача. Воно дозволяє переконатися, що всі елементи відображаються коректно, динамічні елементи реагують на взаємодію з ними, повідомлення системи відображаються коректно та зрозуміло, а перехід між сторінками працює плавно. Інтерфейс має бути інтуїтивним та, незалежно від того чи обрано світлу тему, чи темну.

Тестування безпеки мабуть найважливіше тестування для системи шифрування, адже вона працює з чутливою інформацією. На цьому етапі відбувається перевірка чи збереження паролів в базі даних відбувається в захищеному вигляді, чи не можна отримати доступ до зашифрованих даних без авторизації у власному обліковому записі, чи система не схильна до SQL-ін’єкцій та інших атак.

Під час тестування розробленого програмного забезпечення системи для шифрування та дешифрування даних було проведено певні перевірки для виявлення коректної роботи системи, а саме: функціональні, інтерфейсні та безпекові. Основною метою було переконатися, що всі модулі працюють узгоджено, система виконує потрібні задачі без помилок, а також що користувачеві зручно працювати в системі, і вона виглядає логічною. Тестування проводилося на різних етапах розробки, включаючи модульне, інтеграційне та системне тестування.

Під час модульного тестування перевірено окремі функції: зчитування з файла, реалізацію XOR алгоритма, взаємодію з базою даних SQLite, перевірка авторизації та реєстрації облікових записів. Для кожної з функцій застосовувалися як стандарті так і граничні значення. Усі функції успішно пройшли перевірку, шифрування та дешифрування виконувалося правильно, збереження та зчитування інформації з бази даних працювало стабільно.

Інтеграційне тестування підтвердило коректну взаємодію між графічним інтерфейсом користувача, логікою програми та базою даних. Наприклад після реєстрації нового облікового запису, дані про нього з’являлися в базі даних, а під час авторизації програмне забезпечення правильно й очікувано обробляло вхід користувача в обліковий запис. У випадку введених коректних даних, користувачеві надавався доступ до функціоналу, інакше виводилося повідомлення про некоректність даних, та доступ до функціоналу не був наданий. Також було перевірено коректну обробку файлів, їхнє відображення в таблиці після додавання в систему, та після створення файла з інформацією над якою проводилася операція шифрування або дешифрування.

Також окремо протестовано інтерфейс в обох версіях системи, світлій та темній. Усі візуальні елементи відображаються відповідно до задуму, кнопки виконували відповідні дії, а підказки для користувача з’являлися у потрібні моменти. Перевірялися також повідомлення про помилки, вони відображалися коректно та зрозуміло.

Системне тестування охопило весь функціонал, від запуску системи до повного циклу обробки файлів. Результати підтвердили, що система функціонує стабільно навіть при послідовному виконанні багатьох дій. Не було виявлено збоїв, зависань чи втрат даних.

Усі результати тестування задокументовані, а виявлені незначні недоліки були виправлено. Загалом тестування підтвердило готовність програмного забезпечення до використання реальним користувачем, її стійкість, коректність обробки даних та зручність у користуванні.

ВИСНОВКИ

У результаті виконання бакалаврської кваліфікаційної роботи було розроблено програмне забезпечення системи для шифрування та дешифрування даних. Дана система використовує для шифрування інформації алгоритм XOR, система створена на мові програмування C++ з використанням фреймворку Qt та бази даних SQLite. Система забезпечує повний цикл обробки користувацьких файлів, починаючи з авторизації в обліковий запис та реєстрації нового акаунта, закінчуючи наданням функціоналу для шифрування відповідних файлів та збереження нових файлів із зашифрованою інформацією. Для даної системи реалізовано інтуїтивно зрозумілий та зручний у використанні графічний інтерфейс користувача, що має на вибір світлу та темну теми, залежно від вподобання та умов роботи користувача. Таке рішення покращує зручність взаємодії із системою при яскравому освітленні, або в темну пору дня.

У процесі розробки було виконано моделювання предметної області, побудовано необхідні UML-діаграми, що дозволило чітко структурувати систему та сформувати вимоги до функціоналу. Проведені тестування підтвердили коректність роботи всіх модулів, стабільність системи та її готовність до практичного використання. Отримані результати підтверджують доцільність використання алгоритму шифрування XOR в контексті простих, але ефективних засобів шифрування. Таким чином розроблене програмне забезпечення відповідає поставленим цілям та завданням та є функціонально завершеною, також має перспективи подальшого розвитку.

Крім основної функціональності, система демонструє гнучкість і можливість подальшого розширення. Зокрема у майбутньому можлива інтеграція складних алгоритмів, наприклад AES або RSA, вони дозволять підвищити рівень захисту даних. Таким чином, розроблене програмне забезпечення може слугувати основою для більш складних інформаційних систем із підвищеними вимогами до безпеки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bruce Schneier. (1996). Applied Cryptography: Protocols, Algorithms, and Source Code in C. Wiley.
2. Cplusplus.com [Електронний ресурс] – https://cplusplus.com/
3. Cryptomator [Електронний ресурс] – https://cryptomator.org/
4. Designing GUI applications with Qt5 [Електронний ресурс] – https://wiki.qt.io/Main
5. ISO/IEC 19505-1:2012 – Information technology – Object Management Group Unified Modeling Language (OMG UML). [Електронний ресурс] – https://www.omg.org/spec/UML/2.5.1/
6. Qt Documentation [Електронний ресурс] – https://doc.qt.io/
7. SQLite Documentation [Електронний ресурс] – https://www.sqlite.org/docs.html
8. Stallings, W. (2017). Cryptography and Network Security: Principles and Practice (7th Edition). Pearson.
9. TutorialsPoint - Cryptography [Електронний ресурс] – https://www.tutorialspoint.com/cryptography/index.htm
10. VeraCrypt [Електронний ресурс] – https://www.veracrypt.fr/en/Home.html