Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему:

«Програмне забезпечення для зберігання та обробки даних за допомогою гомоморфного шифрування»

Виконав: студент 4 курсу
групи 2ПІ-21б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки спеціальності)
Чернюк С.І.
(прізвище та ініціали)
Керівник: PhD, ст. вик. каф. ПЗ Стахов О.Я.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент: к.т.н., проф. каф. ОТ Захарченко С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Допущено до захисту
Завідувач кафедри ПЗ
д.т.н., проф. Романюк О.Н.
(ініціали та прізвище)
«09» 06 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
ПЗ О. Н. Романюк
« 21 » березня 2025 р.

З А В Д А Н Н Я

НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Чернюку Сергій Івановичу

1. Тема роботи – «Програмне забезпечення для зберігання та обробки даних за допомогою гомоморфного шифрування».

Керівник роботи: Стахов Олексій Ярославович, PhD, старший викладач кафедри ПЗ, затверджені наказом вищого навчального закладу від 20 березня 2025 р. №97.

2. Строк подання студентом роботи 2 червня 2025.

3. Вихідні дані до роботи: криптосистема Paillier, гомоморфне шифрування, алгоритми шифрування та дешифрування, мова програмування Java, середовище розробки IntelliJ IDEA, технологія JavaFX, робота з базами даних у СКБД MS Access, операційна система Windows.

4. Зміст текстової частини: вступ; аналіз предметної області та постановка задач розробки; розробка моделі та алгоритмів програмного продукту; розробка програмних модулів реалізації системи зберігання та обробки даних за допомогою гомоморфного шифрування; тестування програми; висновки; список використаних джерел; додатки.

5. Перелік ілюстративного матеріалу: титульний слайд; що таке гомоморфне шифрування; актуальність теми; мета, об'єкт та предмет; завдання дослідження; новизна отриманих результатів; практичне значення результатів; порівняльний аналіз аналогів; модель роботи програми; алгоритм роботи програми; алгоритм зберігання ключів криптосистеми Paillier; алгоритм гомоморфного віднімання на основі криптосистеми Paillier (1-2); алгоритм здійснення переказу коштів; інтерфейс застосунку (1-2); тестування застосунку (1-2); апробація та публікація матеріалів; висновки; фінальний слайд.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1-4	Стахов О.Я., доктор філософії PhD, ст. викладач	24.03.25	01.06.25

7. Дата видачі завдання 24 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

Ч.ч.	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз предметної області та постановка задач розробки	25.03.25 – 30.03.25	вик
2	Розробка моделі та алгоритмів роботи програмного застосунку	31.03.25 – 10.04.25	вик
3	Розробка програмних модулів реалізації системи зберігання та обробки даних за допомогою гомоморфного шифрування	11.04.25 – 20.04.24	вик
4	Тестування програмного застосунку	21.04.25 – 27.04.25	вик
5	Оформлення матеріалів до захисту БКР	28.04.25– 30.05.25	вик

Студент _____ Чернюк С.І.
(підпис) (ініціали та прізвище)

Керівник бакалаврської кваліфікаційної роботи _____ Стахов О.Я.
(підпис) (ініціали та прізвище)

Анотація

Бакалаврська кваліфікаційна робота складається з 54 сторінок формату А4, на яких є 41 рисунків, 3 таблиці, список використаних джерел містить 22 найменування.

У бакалаврській кваліфікаційна роботі розроблено програмний застосунок для банківської системи з реалізацією гомоморфного шифрування з використанням криптосистеми Paillier. Робота включає створення програмних модулів для безпечного зберігання, обробки та передачі зашифрованих персональних даних клієнтів, зокрема логіна, номера рахунку, ПІН-коду та залишку на рахунку. Розроблений застосунок містить функціонал для автентифікації, перегляду балансу та здійснення переказу коштів у зашифрованому вигляді, збереженні даних використовуючи СКБД MS Access.

Програмне забезпечення має графічний інтерфейс, реалізований на JavaFX, та працює у середовищі IntelliJ IDEA. Застосунок демонструє можливості використання сучасних криптографічних технологій для підвищення рівня інформаційної безпеки в банківських ІТ-системах. Розроблене рішення може бути використане як демонстраційний або навчальний приклад безпечної обробки конфіденційних даних у сфері фінансових технологій.

Ключові слова: гомоморфне шифрування, криптосистема Paillier, інформаційна безпека, зашифровані дані, банківська система, JavaFX, MS Access, фінансові технології, автентифікація, обробка даних.

Annotation

The bachelor's thesis consists of 54 pages of A4 format, on which 41 figures, 3 tables, the list of used sources contains 22 names.

In the bachelor's complex work, a software tool for a banking system with the implementation of homomorphic encryption using the Paillier cryptosystem has been developed. The work includes the creation of software modules for the secure storage, processing and transmission of encrypted personal data of clients, in particular login, account number, PIN code and account balance. The developed application contains functionality for authentication, balance viewing and transfer of funds in encrypted form, data storage using the MS Access database.

The software has a graphical interface implemented on JavaFX and runs in the IntelliJ IDEA environment. The application demonstrates the possibilities of using modern cryptographic technologies to increase the level of information security in banking IT systems. The developed solution can be used as a demonstration or training example of the secure processing of confidential data in the field of financial technologies.

Keywords: homomorphic encryption, Paillier cryptosystem, information security, encrypted data, banking system, JavaFX, MS Access, financial technologies, authentication, data processing.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Швидке зростання обсягів даних і підвищення загроз кібербезпеці вимагають постійного вдосконалення методів забезпечення конфіденційності та цілісності даних під час їхнього зберігання й обробки. З поширенням хмарних обчислень, віддаленого зберігання даних та сервісів, що обробляють конфіденційну інформацію, питання безпеки набуває особливої актуальності [1]. Захист інформації стає ключовою умовою для довіри користувачів до цифрових платформ. Традиційні підходи часто виявляються недостатніми для нових типів атак і сценаріїв використання даних. Більшість криптографічних методів забезпечують захист під час зберігання та передавання даних, однак не дозволяють виконувати обчислення безпосередньо над даними у зашифрованому вигляді, що обмежує їхнє використання у деяких сценаріях.

Зважаючи на це, гомоморфне шифрування є перспективною технологією, яка дозволяє виконувати математичні операції над зашифрованими даними без необхідності їхнього розшифровування. Це відкриває можливості для побудови безпечних обчислювальних сервісів, де навіть власник серверу не має доступу до вихідної інформації. Застосування гомоморфного шифрування може суттєво підвищити рівень захищеності інформаційних систем, особливо у фінансовій, медичній, урядовій та освітній сферах.

Незважаючи на значний науковий інтерес до тематики, на практиці гомоморфне шифрування поки що рідко використовується в реальних застосуваннях через важкість реалізації, високу обчислювальну складність та відсутність зручних інструментів для інтеграції в прикладні системи. Тому актуальною є розробка програмного забезпечення для зберігання та обробки даних за допомогою гомоморфного шифрування.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно з планом виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою бакалаврської кваліфікаційної роботи є підвищення конфіденційності даних користувачів завдяки використанню гомоморфного шифрування, що дозволяє здійснювати обчислення над зашифрованими даними без їхнього розшифрування, тим самим забезпечуючи захист інформації навіть у процесі її обробки на ненадійних серверах або у хмарних середовищах. Проєкт передбачає створення програмних компонентів, які реалізують повноцінну систему для зберігання та обробки зашифрованих даних користувача.

У межах бакалаврської роботи необхідно реалізувати такі завдання:

- аналіз основних видів гомоморфного шифрування та вибір криптосистеми для подальшої розробки;
- розробка програмних модулів для шифрування, зберігання і обробки даних користувача;
- реалізація програмного продукт з практичним використанням розроблених модулів;
- розробка зручного та адаптивного графічного інтерфейсу програмного забезпечення для роботи з модулями;
- проведення тестування розробленого функціоналу.

Об’єктом дослідження є процес зберігання та обробки конфіденційних даних користувачів у програмних системах з підвищеними вимогами до інформаційної безпеки.

Предметом дослідження є методи зберігання та обробки зашифрованих даних користувачів на основі гомоморфного шифрування, а також алгоритми і засоби забезпечення конфіденційності інформації під час гомоморфних обчислень.

Методи дослідження. У процесі виконання дослідження застосовувались такі методи: аналіз і синтез інформаційних систем, теорія криптографії для реалізації гомоморфного шифрування, методи об’єктно-орієнтованого програмування, структурного моделювання, а також методи тестування функціональності програмного забезпечення.

Новизна отриманих результатів.

1. Подальшого розвитку набув метод зберігання ключів криптосистеми Paillier, який, на відміну від існуючих, базується на багаторівневому підході до шифрування приватних компонентів ключа із використанням сучасних криптографічних геш-функцій та симетричного шифрування, що дає можливість підвищити стійкість до несанкціонованого доступу й забезпечити безпечне зберігання ключів.

2. Подальшого розвитку набув метод гомоморфного віднімання на основі криптосистеми Paillier, який, на відміну від існуючих реалізацій, відрізняється спрощеною структурою та зменшеною кількістю обчислювальних операцій, що дозволяє полегшити його інтеграцію в прикладні програмні рішення та підвищити ефективність обробки зашифрованих даних.

Практичне значення отриманих результатів полягає в тому, що на основі отриманих в бакалаврській кваліфікаційній роботі теоретичних положень запропоновано алгоритми та розроблено програмні засоби захисту даних користувачів із використанням гомоморфного шифрування.

Особистий внесок здобувача. Усі теоретичні та прикладні результати дослідження, викладені у бакалаврській кваліфікаційній роботі, отримані особисто автором. Розробка архітектури, імплементація криптографічних алгоритмів, тестування програмного забезпечення та оформлення звітної документації виконано самостійно.

Апробація матеріалів бакалаврської кваліфікаційної роботи. Основні результати дослідження були представлені на «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)» у формі доповіді та обговорені в науковому середовищі [2].

Публікація. За результатами дослідження опубліковано 1 наукову працю у збірнику тез конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)» [2].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧ РОЗРОБКИ

1.1 Аналіз стану систем гомоморфного шифрування

Швидкий розвиток інформаційних технологій та зростання обсягів оброблюваних даних вимагають постійного вдосконалення методів забезпечення безпеки інформації. Традиційні криптографічні підходи, такі як симетричне та асиметричне шифрування, хоча і ефективно захищають дані під час зберігання чи передавання, обмежують можливість виконання обчислень над зашифрованими даними без попереднього розшифрування. У відповідь на ці обмеження з'явилися методи гомоморфного шифрування, що дозволяють виконувати обчислення безпосередньо у зашифрованому вигляді.

Одним із ключових напрямів розвитку гомоморфного шифрування є створення повністю гомоморфних схем (FHE), які забезпечують підтримку довільних обчислень над зашифрованими даними. Сучасні реалізації повністю гомоморфного шифрування демонструють значні покращення в продуктивності та практичності застосування [3]. У наведеній статті автори аналізують поточний стан технології FHE, її потенціал для забезпечення конфіденційності даних під час обчислень, а також виклики, пов'язані з її впровадженням у реальні системи.

Іншою важливою тенденцією є впровадження гомоморфного шифрування у хмарні обчислення та сфери, де критичним є захист даних при передачі обробки на зовнішні сервери. Використання гомоморфного шифрування дозволяє уникати необхідності розкривати дані перед обчисленням, що значно підвищує загальний рівень безпеки систем.

Системи гомоморфного шифрування відкривають нові можливості для розробки безпечних сервісів у таких сферах, як фінансові технології, медицина, державне управління та машинне навчання. Наприклад, у фінансовому секторі гомоморфне шифрування дозволяє реалізувати системи для обробки транзакцій, не розкриваючи чутливі дані користувачів навіть для внутрішніх сервісів обробки.

Водночас, незважаючи на істотні переваги, широке впровадження гомоморфного шифрування залишається обмеженим через велику обчислювальну складність і затримки, які виникають під час роботи із зашифрованими даними. Більшість сучасних реалізацій вимагають значних обчислювальних ресурсів, що ускладнює їхнє застосування у реальному часі або на пристроях із обмеженими ресурсами. Проте завдяки активній розробці спеціалізованих бібліотек, таких як Microsoft SEAL [4], гомоморфне шифрування стає дедалі доступнішим для практичного використання.

Таким чином, аналіз стану систем гомоморфного шифрування підтверджує їхню високу перспективність у забезпеченні безпеки обробки даних без компрометації конфіденційності. Подальший розвиток цієї технології спрямований на оптимізацію продуктивності, зниження обчислювальних витрат та підвищення практичної придатності у різноманітних галузях. Розробка ефективних програмних модулів, що використовують гомоморфне шифрування, є важливим кроком до інтеграції цих технологій у реальні інформаційні системи нового покоління.

1.2 Аналіз видів гомоморфного шифрування та вибір криптосистеми

Гомоморфне шифрування – це тип асиметричного шифрування, який дозволяє виконувати обчислення над зашифрованими даними без необхідності їх розшифрування. Результат таких обчислень після розшифрування відповідає результату обчислень, проведених над незашифрованими даними. Завдяки цій властивості гомоморфне шифрування набуває особливої актуальності в задачах, де необхідно забезпечити конфіденційність даних під час їхньої обробки.

Існують 3 основних види гомоморфного шифрування [5]:

1. Частково гомоморфне шифрування (PHE) підтримує лише одну математичну операцію над зашифрованими даними (додавання або множення). Прикладами таких систем є шифрування RSA (множення) та Paillier (додавання).

2. Деякою мірою гомоморфне шифрування (SHE) дозволяє виконувати обмежену кількість обчислень (додавань і множень) над зашифрованими даними

до того, як накопичена помилка зробить розшифрування неможливим. Прикладом є системи на основі решіток.

3. Повне гомоморфне шифрування (FHE) забезпечує можливість виконувати необмежену кількість довільних обчислень над зашифрованими даними. Ці системи є найбільш універсальними, однак значно складнішими в реалізації та вимагають великих обчислювальних ресурсів. Прикладами FHE-схем є Gentry, BGV, CKKS.

У межах даної бакалаврської кваліфікаційної роботи було обрано криптосистему Paillier [6], яка реалізує адитивне гомоморфне шифрування. Причинами такого вибору є:

- простота реалізації у порівнянні з повним гомоморфним шифруванням;
- достатня безпека для демонстраційних та навчальних цілей;
- широка доступність готових бібліотек і прикладів реалізації.

Криптосистема Paillier дозволяє обчислювати суму двох зашифрованих чисел без необхідності їх попереднього розшифрування. Це робить її зручною для задач, де потрібно виконувати додавання над конфіденційними даними без розкриття їхнього вмісту. Завдяки своїй простоті реалізації та підтримці адитивної гомоморфності, система Paillier широко використовується в різних сферах, що потребують безпечної обробки зашифрованої інформації, зокрема банківські системи, електронні голосування тощо.

1.3 Порівняльний аналіз аналогів

Більшість існуючих програмних рішень у сфері гомоморфного шифрування представлені у вигляді бібліотек або платформ, що орієнтовані на інтеграцію в інші програмні продукти. Такі рішення, як правило, надають розробникам інструменти для реалізації криптографічних алгоритмів у різних середовищах програмування, забезпечуючи гнучкість і масштабованість.

Натомість дана бакалаврська робота орієнтована на створення прикладної програми, яка демонструє реальне використання модулів гомоморфного шифрування у прикладному сценарії роботи із зашифрованими даними.

Для проведення порівняльного аналізу було відібрано декілька проєктів з відкритим вихідним кодом, які також реалізують прикладне застосування гомоморфного шифрування.

Першим таким проєктом є консольна демонстраційна система голосування (див. рисунок 1.1), яка використовує бібліотеку HElib [7]. Ця розробка ілюструє, як можна забезпечити конфіденційність голосів та цілісність підрахунку без розшифрування індивідуальних голосів.

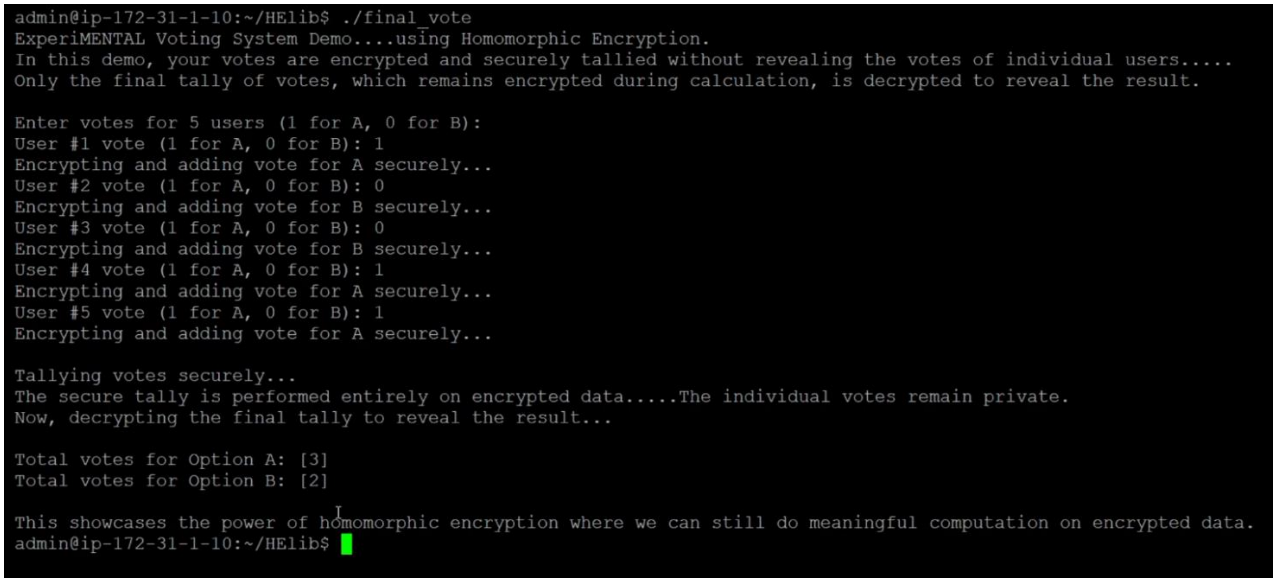

Рисунок 1.1 – Консольна система електронного голосування

Розробка реалізує схему Brakerski-Gentry-Vaikuntanathan (BGV), яка підтримує як додавання, так і множення над зашифрованими даними. Кожен голос шифрується окремо, забезпечуючи конфіденційність виборця. Підрахунок голосів здійснюється безпосередньо над зашифрованими даними, без необхідності їхнього розшифрування. Після завершення підрахунку лише загальний результат розшифровується для отримання підсумків голосування.

Наступна розробка містить приклад використання часткового гомоморфного шифрування для забезпечення конфіденційності даних у медичних дослідженнях [8]. У цьому випадку, учасники дослідження вводять свої рівні глюкози та холестерину в інтерфейсі клієнта (див. рисунок 1.2).

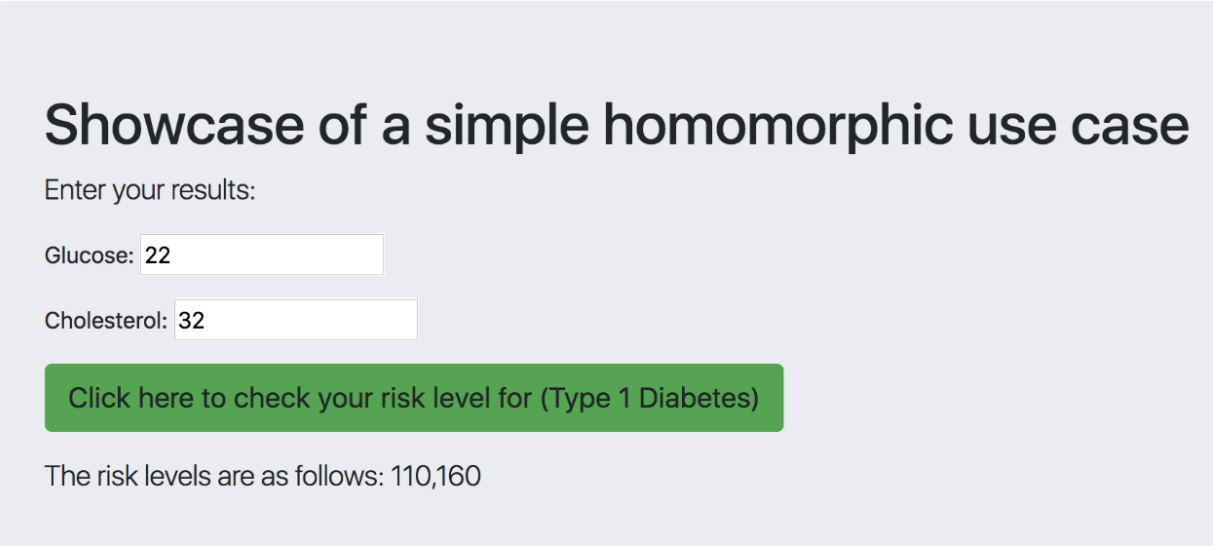

Рисунок 1.2 – Приклад використання гомоморфного шифрування в медицині

Дані зашифровуються на стороні клієнта та передаються на сервер, де проводиться обчислення (множення значення на випадкове число) над зашифрованими даними без їхньої розшифровки. Сервер повертає результат, а саме рівень ризику розвитку діабету, у зашифрованому вигляді, що гарантує збереження конфіденційності протягом усієї обробки.

Реалізація використовує бібліотеку Python для гомоморфного шифрування Paillier і побудована з використанням Docker, що дозволяє легко налаштувати середовище для запуску клієнтської та серверної частини програми.

Третій аналог це консольна програма (див. рисунок 1.3) для класифікації зашифрованих зображень [9].

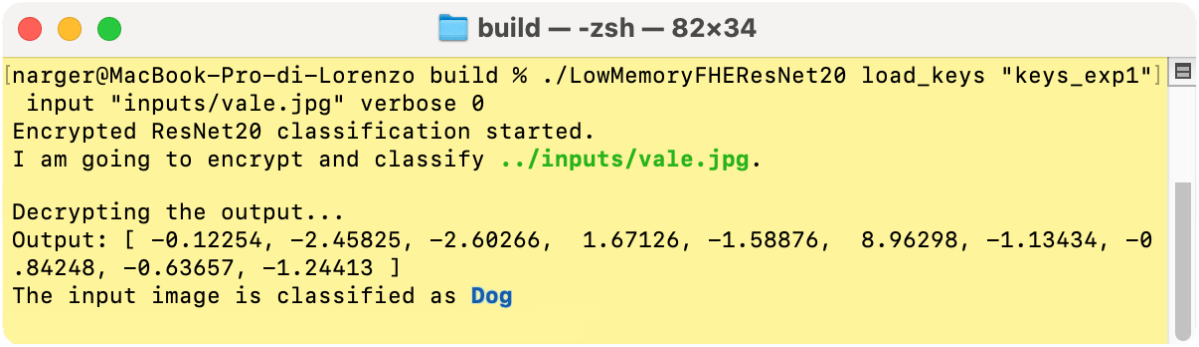

Рисунок 1.3 – Приклад роботи програми для класифікації зображень

Дана розробка є прикладом застосування повного гомоморфного шифрування (FHE) для класифікації зображень. Проєкт реалізує модель ResNet20, адаптовану для обробки зашифрованих зображень з набору даних CIFAR-10, використовуючи бібліотеку OpenFHE та схему CKKS для приблизних обчислень над зашифрованими даними. Цей проєкт демонструє можливість ефективної обробки зашифрованих зображень з використанням глибоких нейронних мереж при обмежених ресурсах

Проведений аналіз показує, що існують різноманітні розробки, які демонструють можливості використання гомоморфного шифрування у прикладних задачах – від систем голосування до обробки зашифрованих зображень. З урахуванням результатів аналізу для власної розробки було обрано реалізацію прикладної програми банківського призначення для персонального комп'ютера. Для наочності та подальшого аналізу доцільно узагальнити основні характеристики обраних аналогів і власної розробки у вигляді порівняльної таблиці (див. таблицю 1.1).

Таблиця 1.1 – Порівняльна характеристика аналогів

Розробки / Критерії	Власна розробка	Система електронного голосування	Обчислювач рівня ризику діабету	Класифікатор зашифрованих зображень
Реалізація FHE	-	+	-	+
Наявність GUI/WebUI	+	-	+	-
Доступність коду	+/-	+	+	+
Використання додаткових методів захисту даних	+	-	-	-
Можливість розширення	+	-	+	+

Відповідно до проведеного порівняльного аналізу, розробка власного прикладного програмного засобу для банку, що демонструє використання гомоморфного шифрування, є доцільною. Створена система дозволяє на практиці реалізувати обробку зашифрованих даних із мінімальними вимогами до ресурсів користувача, а також забезпечує доступний інтерфейс для взаємодії з результатами обчислень, що відрізняє її від більшості існуючих аналогів.

Отже, запропоноване рішення сприяє популяризації і практичному застосуванню гомоморфного шифрування в реальних прикладних задачах, зокрема в банківських сервісах, де критично важливо зберігати конфіденційність інформації навіть під час виконання обчислень. Розроблений програмний продукт може стати основою для подальшої розробки безпечних банківських сервісів, що потребують обробки чутливих даних без їхнього розшифрування.

1.4 Постановка задач бакалаврської кваліфікаційної роботи

Проаналізувавши усі переваги та недоліки існуючих програмних рішень, які демонструють прикладне використання гомоморфного шифрування, було визначено наступні завдання, які необхідно виконати для успішної розробки власного програмного забезпечення:

- розробити алгоритми шифрування та обробки даних на основі криптосистеми Paillier;
- реалізувати модуль взаємодії з базою даних, що зберігає інформацію про користувачів в зашифрованому вигляді;
- реалізувати базові прикладні функції (вхід та вихід користувача в систему, перегляд власного балансу на рахунку, переказ коштів на рахунок іншого користувача тощо);
- створити графічний інтерфейс користувача для зручного введення, перегляду та обробки зашифрованих даних користувачів;
- провести тестування програмного забезпечення для перевірки правильності шифрування, обробки даних та відповідності поставленим завданням.

1.5 Висновки

У першому розділі було розглянуто стан систем гомоморфного шифрування, описано види гомоморфного шифрування та вибрано криптосистеми Paillier для подальшої розробки. Були проаналізовані такі рішення, як система електронного голосування на основі бібліотеки HElib, обчислювач рівня ризику діабету, а також класифікатор зашифрованих зображень. Кожен із розглянутих застосунків має свої переваги та обмеження, що були детально проаналізовані.

Проаналізувавши переваги та недоліки існуючих програмних рішень, було сформульовано завдання для подальшої розробки власного прикладного програмного забезпечення для демонстрації роботи гомоморфного шифрування у банківській сфері. Поставлені завдання включають розробку алгоритмів шифрування та обробки даних на основі криптосистеми Paillier, створення модуля взаємодії з зашифрованою базою даних, реалізацію прикладних функцій роботи з рахунками користувачів та розробку зручного графічного інтерфейсу. Таким чином було доведено доцільність створення власної розробки для практичної демонстрації переваг гомоморфного шифрування у захисті фінансової інформації. На основі аналізу сформовано перелік конкретних задач, які необхідно реалізувати у межах бакалаврської кваліфікаційної роботи.

2 РОЗРОБКА МОДЕЛІ ТА АЛГОРИТМІВ ПРОГРАМНОГО ПРОДУКТУ

2.1 Розробка моделі системи

Розробка програмного забезпечення передбачає створення кількох основних модулів. Першим буде модуль авторизації користувачів, який дозволить здійснювати вхід до системи за допомогою пін-коду і логіну користувача. Далі буде реалізовано модуль перегляду поточного балансу рахунку користувача. Також буде створений модуль для здійснення переказу коштів між рахунками, що реалізовуватиме обчислення над зашифрованими сумами відповідно до властивостей криптосистеми Paillier.

Окрім прикладної логіки, буде створено модуль взаємодії з базою даних, де буде організоване зберігання зашифрованої інформації про користувачів – таких як номери рахунків, залишки коштів, пін-коди тощо. Всі запити до бази даних та обробка отриманої інформації будуть здійснюватися таким чином, щоб уникнути розкриття незашифрованих даних на будь-якому етапі.

Для кращого розуміння роботи модулів програмних засобів використання гомоморфного шифрування у банківських операціях було вирішено розробити модель роботи системи на прикладі UML [10] діаграми діяльності.

Як видно з діаграми (див. рисунок 2.1), спершу відбувається спроба перейти в кабінет користувача. Для цього програма шукає валідний JSON Web Token (JWT) [11]. Якщо такого токена немає, то після цього користувач може відкрити довідку, яка містить два окремих вікна: вікно «Допомога» та вікно «Про програму». У вікні «Допомога» буде написано коротку інструкцію використання програмного застосунку, а у вікні «Про програму» наведено додаткову інформацію про програму та автора.

Окрім цього, у початковому вікні користувач матиме змогу увійти в систему, ввівши власні логін та пін-код. Якщо дані збігаються зі збереженими у базі даних, то користувач успішно входить і далі йому стають доступними функції перегляду його балансу на рахунку, переказу власних коштів на рахунок

іншого користувача, перегляд історії транзакцій та вихід із системи, тобто повернення до попереднього вікна. У іншому випадку користувач отримає повідомлення про помилку з вказівками її вирішення.

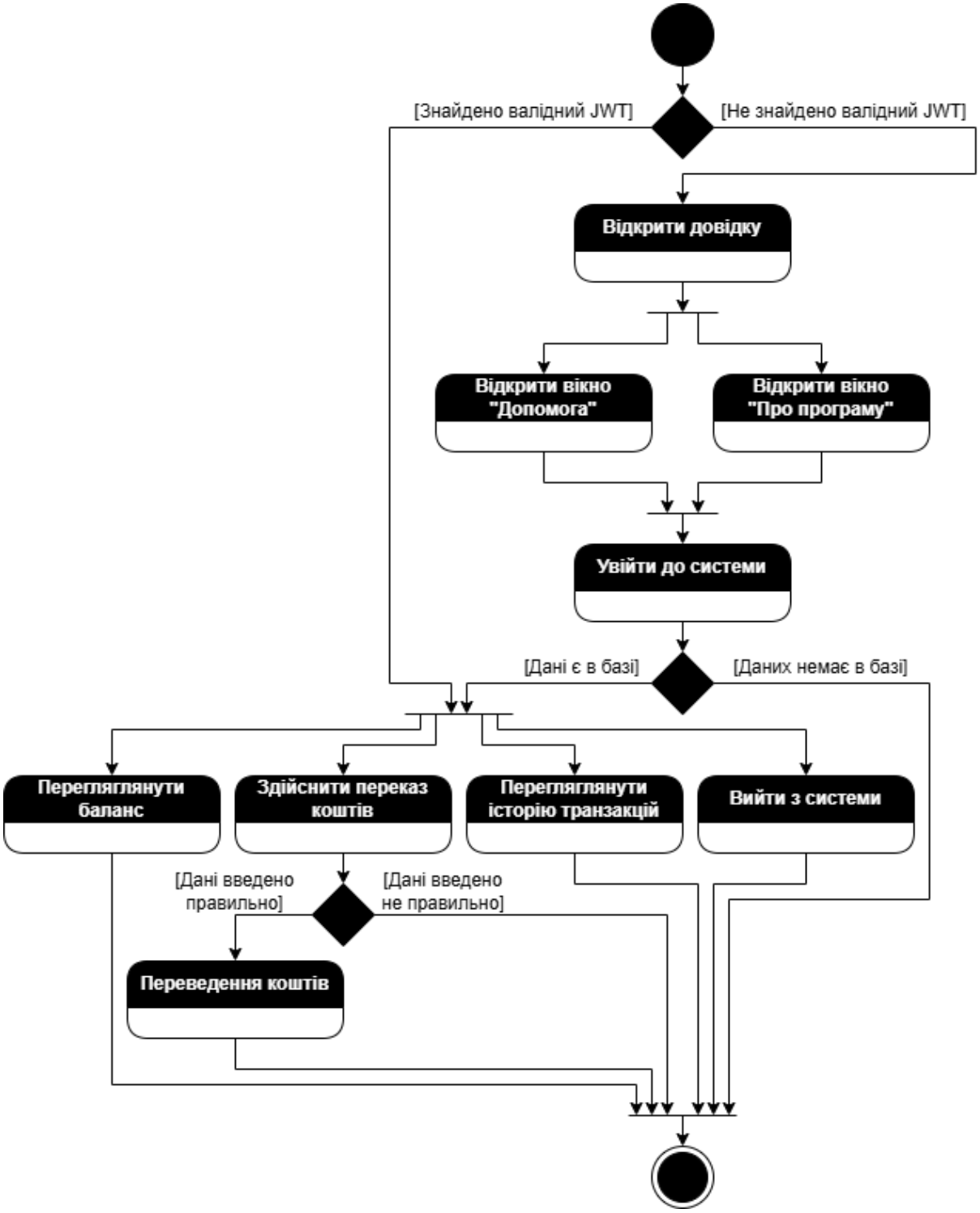

Рисунок 2.1 – Модель роботи програми у вигляді діаграми діяльності програмних засобів

Якщо користувач вибере функцію перегляду балансу, то програма відкриває додаткове вікно з розшифрованим балансом.

У випадку потреби здійснення переказу коштів, користувачу дається інше додаткове вікно, у якому йому потрібно ввести номер рахунку іншого користувача і суму переказу. Після цього, при умові, що все було введено правильно користувач побачить повідомлення про успішний переказ. Інакше користувач зіткнеться з повідомленням про помилку, у якій як і раніше буде вказуватися, що потрібно зробити, щоб позбутися її.

Якщо користувачу знадобиться перевірити історію останніх транзакцій, то натиснувши на відповідну кнопку, програма виведе додаткове вікно з таблицею, у якій відображатиметься інформація про те, кому було надіслано або від кого отримано кошти, суму переказу і дату та час здійснення переказу.

Розробка діаграми діяльності була виконана у програмному забезпеченні Draw.io [12].

2.2 Розробка методу зберігання ключів криптосистеми Paillier

Для підвищення безпеки зберігання ключів криптосистеми Paillier було розроблено метод, що базується на багаторівневому підході до шифрування приватних компонентів ключа із використанням сучасних криптографічних геш-функцій та симетричного шифрування. Загальна послідовність алгоритму збереження ключів представлена у вигляді блок-схеми (див. рисунок 2.2).

Процес починається з перевірки на наявність вже існуючого файлу зі збереженими ключами. Якщо такого файлу ще немає, то програма створює набір ключів Paillier та отримує спеціальний пароль, який буде використаний для захисту приватних ключів.

Після цього ініціалізується файл для збереження ключових даних. Публічні ключі безпосередньо записуються у файл у незашифрованому вигляді, оскільки їхній відкритий характер не потребує додаткового захисту.

Наступним кроком здійснюється гешування пароля за допомогою криптографічної геш-функції SHA-256 [13]. Це дозволяє отримати надійне

представлення пароля у вигляді фіксованої довжини бітового рядка, що підвищує безпеку.

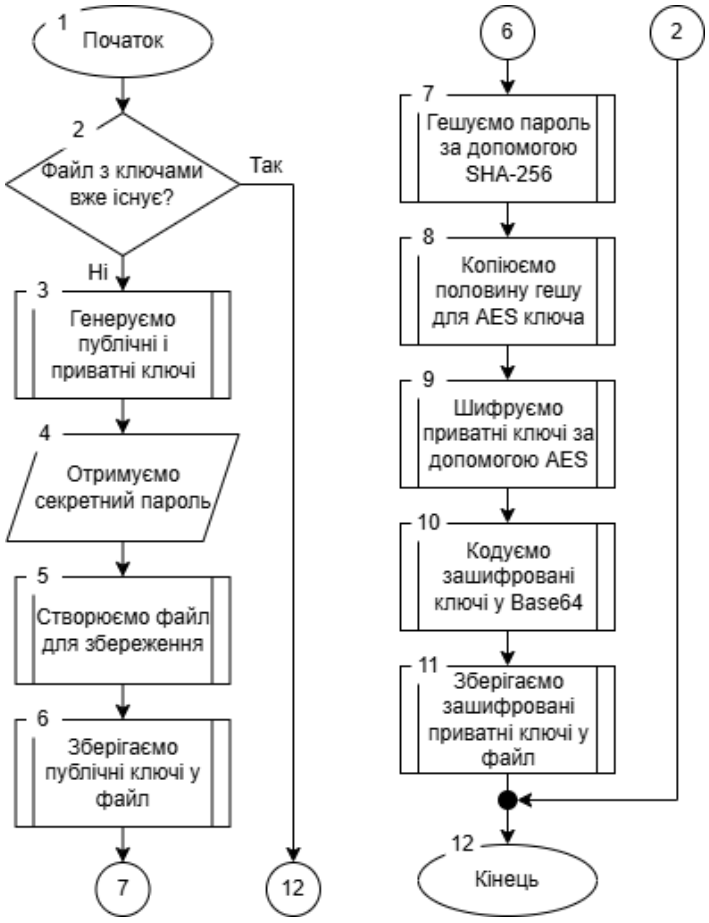

Рисунок 2.2 – Алгоритм збереження ключів криптосистеми Paillier

З отриманого гешу пароля копіюється половина, яка використовується як симетричний ключ для шифрування приватних ключів. Такий підхід дозволяє уникнути прямого використання пароля і забезпечує високу стійкість до атак.

Приватні ключі шифруються із застосуванням алгоритму AES [14], що є одним із найпоширеніших і надійних методів симетричного шифрування.

Після шифрування, отримані дані кодуються у формат Base64, що дозволяє безпечно зберігати їх у текстовому файлі, уникаючи проблем з кодуванням та передачею бінарної інформації. На завершення зашифровані та закодовані приватні ключі записуються у файл, що гарантує їхнє захищене збереження у середовищах з підвищеними вимогами до конфіденційності.

Таким чином, розроблений метод забезпечує багаторівневий захист ключових компонентів криптосистеми Paillier, підвищуючи загальну безпеку програмного застосунку.

2.3 Розробка методу гомоморфного віднімання на основі криптосистеми Paillier

Гомоморфне шифрування дозволяє виконувати базові арифметичні операції безпосередньо над зашифрованими даними. Однією з таких операцій є віднімання, що особливо важливе у фінансових розрахунках, при обліку або у логіці перевірки обмежень. Проте класичні реалізації гомоморфного віднімання на основі криптосистеми Paillier часто містять зайві проміжні обчислення, які ускладнюють інтеграцію в прикладні системи.

У класичному варіанті гомоморфного віднімання на основі криптосистеми Paillier використовується обчислення мультиплікативного оберненого зашифрованого від'ємника [15]. Нехай маємо два цілі числа: a – зменшуване та b – від'ємник. Після їхнього шифрування отримаємо вирази (2.1).

$$E(a) = g^a \cdot r_1^n (\mathrm{mod}\, n^2); E(b) = g^b \cdot r_2^n (\mathrm{mod}\, n^2) \quad (2.1)$$

де $E(x)$ – зашифроване значення x;

$n = p \cdot q$ (p та q – випадково вибрані великі прості числа);

$g = n + 1$;

r – випадкове число в межах {0, …, n-1}.

Операція віднімання a – b у гомоморфному вигляді виконується за формулою (2.2).

$$E(a - b) = E(a) \cdot E(b)^{-1} (\mathrm{mod}\, n^2) \quad (2.2)$$

де $E(b)^{-1}$ – є мультиплікативним оберненим значенням до $E(b)$ за модулем n^2.

Для кращого розуміння і подальшого порівняння було створено блок-схему даного алгоритму (див. рисунок 2.3).

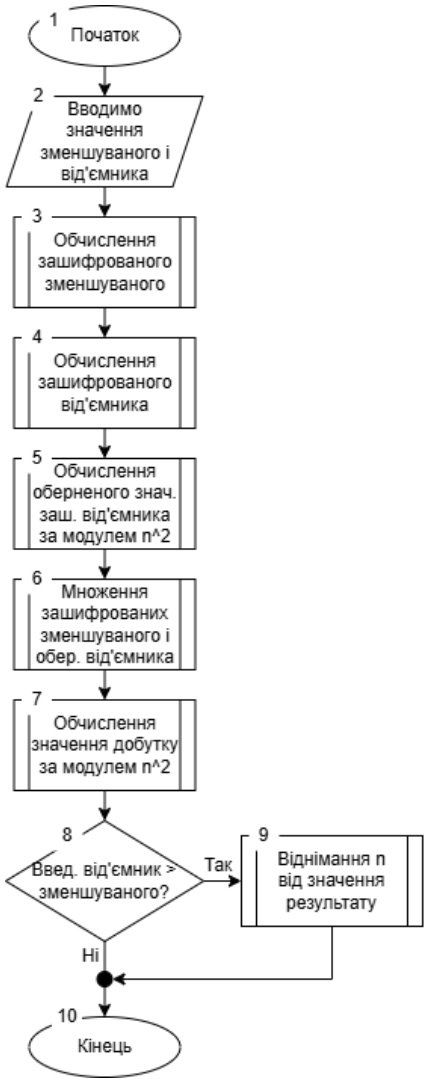

Рисунок 2.3 – Традиційний алгоритм гомоморфного віднімання

Основні етапи реалізації алгоритму включають:

- введення вихідних значень a та b;
- шифрування значень у вигляді $E(a)$ та $E(b)$;
- обчислення оберненого елемента $E(b)^{-1}$;
- множення $E(a) \cdot E(b)^{-1} (\mathrm{mod}\, n^2)$;
- перевірка, чи b > a; у разі перевищення від значення результату віднімається n для корекції негативного результату.

Даний метод можна спростити шляхом попереднього введення числа −b замість обчислення оберненого елемента. Такий підхід дозволяє уникнути в порівнянні довгого обчислення та зменшити кількість операцій. Блок-схема даного варіанту представлена на рисунку 2.4.

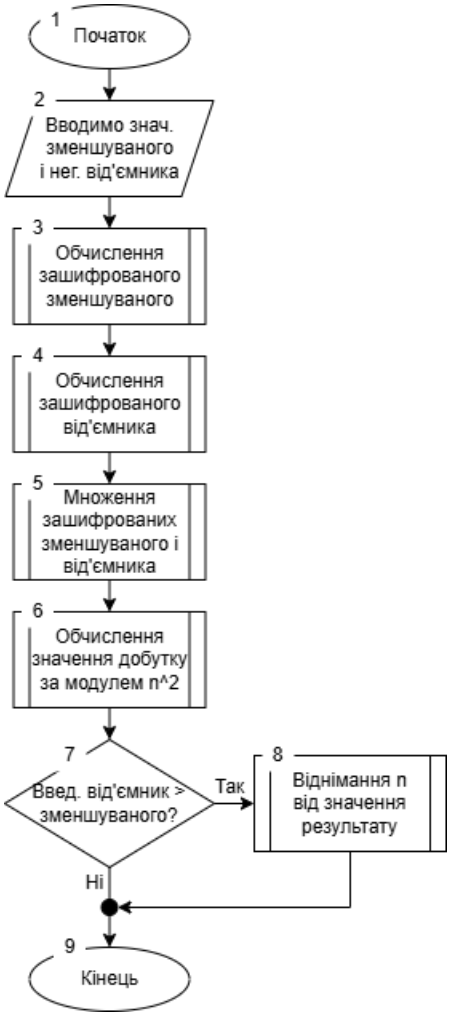

Рисунок 2.4 – Спрощений алгоритм гомоморфного віднімання

Його реалізація полягає у наступному:

- введення значень a та −b;
- шифрування обох значень: $E(a)$ та $E(-b)$;
- обчислення добутку $E(a) \cdot E(-b) (\mathrm{mod}\, n^2)$;
- перевірка, чи b > a; якщо так, то від результату віднімається n.

Доведемо, що даний метод дійсно буде еквівалентним попередньому. Для цього розглянемо перетворення (2.3, 2.4).

$$E(a) \cdot E(b)^{-1} (\mathrm{mod}\, n^2) = (g^a \cdot r_1^n) \cdot (g^b \cdot r_2^n)^{-1} (\mathrm{mod}\, n^2) =$$
$$= g^{a-b} \cdot \frac{r_1^n}{r_2^n} (\mathrm{mod}\, n^2) \quad (2.3)$$

$$E(a) \cdot E(-b) = (g^a \cdot r_1^n) \cdot (g^{-b} \cdot r_2^n) (\mathrm{mod}\, n^2) = g^{a-b} \cdot r_1^n \cdot r_2^n (\mathrm{mod}\, n^2) \quad (2.4)$$

І хоча ці вирази не рівні між собою, але при дешифруванні, вони даватимуть один результат. Для цього розглянемо формулу дешифрування (2.5).

$$D\big(E(x)\big) = \frac{L\big(E(x)^\lambda \bmod n^2\big)}{L\big(g^\lambda \bmod n^2\big)} (\mathrm{mod}\, n) \quad (2.5)$$

де $D\big(E(x)\big)$ – розшифроване значення $E(x)$;

λ – найменше спільне кратне $(p-1)$ і $(q-1)$;

$L(u) = \frac{u-1}{n}$ – допоміжна функція.

Тоді маємо:

$$E(a-b)^\lambda = \left(g^{a-b} \cdot \frac{r_1^n}{r_2^n}\right)^\lambda (\mathrm{mod}\, n^2) = g^{\lambda(a-b)} \cdot \frac{r_1^{\lambda n}}{r_2^{\lambda n}} (\mathrm{mod}\, n^2) \quad (2.7)$$

Але $r_1^{\lambda n} (\mathrm{mod}\, n^2) \equiv 1$ і $r_2^{\lambda n} (\mathrm{mod}\, n^2) \equiv 1$, що випливає з теореми Лагранжа про теорію груп [16].

Отже, запропонований метод дозволяє досягти еквівалентного результату із меншою кількістю обчислень, що позитивно впливає на продуктивність криптографічних обчислень у реальних системах.

2.4 Розробка алгоритмів роботи системи

Для розробки програмного застосунку, для банкінгу з використання гомоморфного шифрування необхідно розробити загальний алгоритм, згідно якого буде працювати застосунок (див. рисунок 2.5).

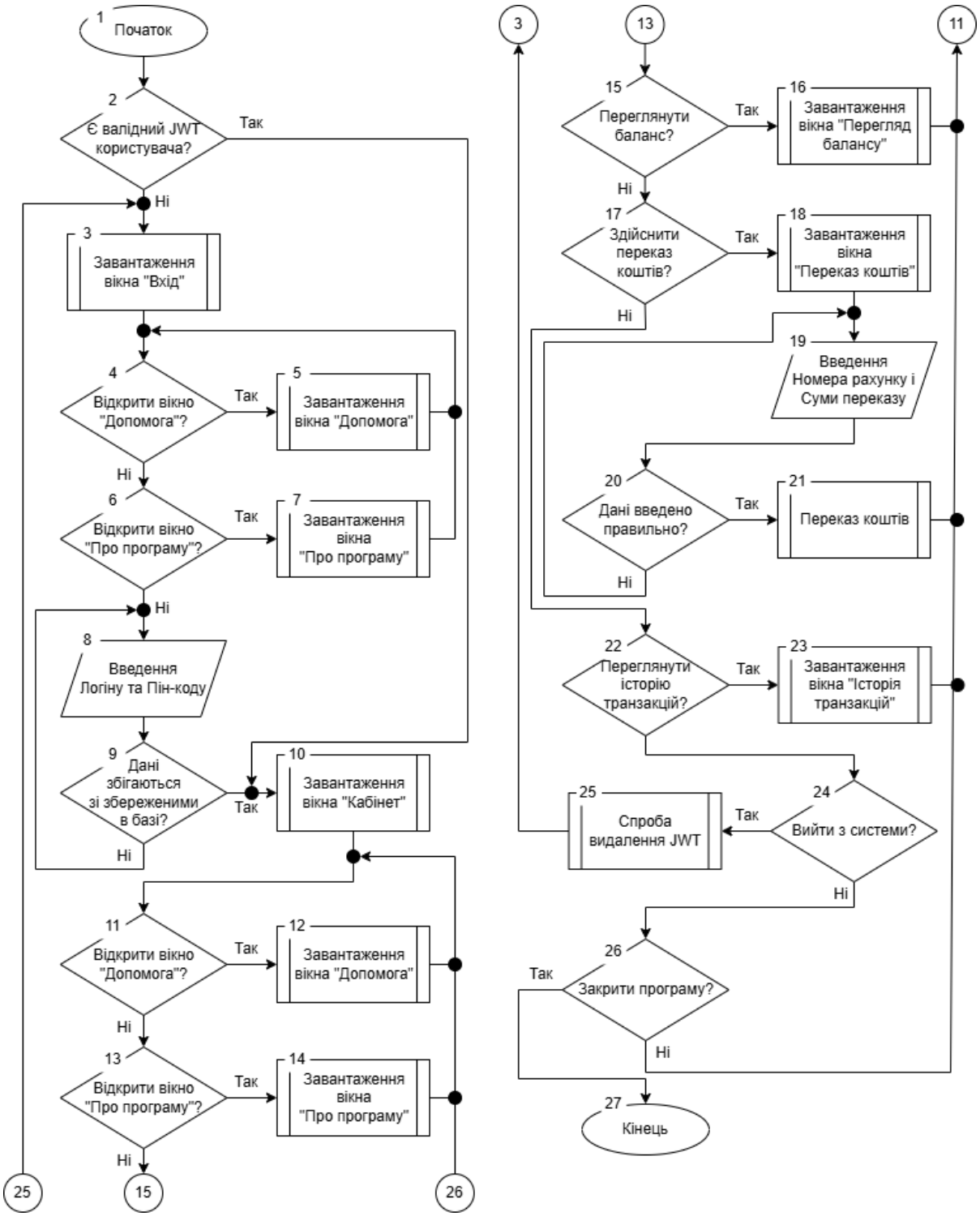

Рисунок 2.5 – Алгоритм роботи програмного застосунку

Згідно з даним алгоритмом, спочатку відбувається перевірка на наявність JWT користувача. Якщо такий файл існує, то відбувається завантаження вікна кабінету користувача. Інакше завантажується вікно для входу в систему. У ньому можна додатково відкрити вікна довідки, а саме: вікно з короткою інструкцією

використання програми і вікно з додатковою інформацією про програму та розробника.

Наступним кроком йде введення даних користувача для входу. Якщо користувач ввів дійсні дані і вони збігаються з даними, які зберігаються в базі даних, то він переходить до кабінету. Інакше користувачу потрібно буде повторити спробу, слідуючи вказівкам з повідомлення про помилку.

У кабінеті користувача також буде міститися довідка на випадок, якщо користувач зіткнувся з якоюсь проблемою при подальшому використанні програми. Окрім цього, тепер користувач матиме змогу переглядати свій баланс на рахунку, здійснювати перекази коштів і переглядати історію транзакцій. При натисканні на відповідну кнопку, програма відкриє додаткове вікно, аби користувач мав доступ до всіх попередніх функцій. Якщо користувач вирішив переглянути свій баланс, то в окремому вікні висвітлиться його поточний рахунок у гривнях.

Коли користувачу знадобиться перевести кошти на інший рахунок, йому доведеться вводити номер рахунку отримувача і суму коштів для переказу. Якщо користувач ввів все правильно, то в цьому ж вікні з’явиться повідомлення про успішний переказ коштів. Інакше з’явиться повідомлення про одну з помилок. Тоді користувач повинен буде спробувати змінити ввід відповідно до вказівок у повідомленні.

Якщо у користувача виникне потреба переглянути інформацію про свої останні транзакції, то йому буде достатньо відкрити відповідне вікно. У даному вікні буде відображатися таблиця з усією необхідною інформацією про транзації користувача.

Після виконаної роботи, додаткові вікна можна закрити і повернутися до вікна кабінету. Якщо користувачу більше не потрібна жодна з функцій додатку, то він може спокійно вийти з системи, під час чого відбудеться видалення наявних JWT і користувач повернеться до вікна входу.

Для кращого розуміння роботи модулів здійснення переказу коштів, було розроблено відповідний алгоритм (див. рисунок 2.6).

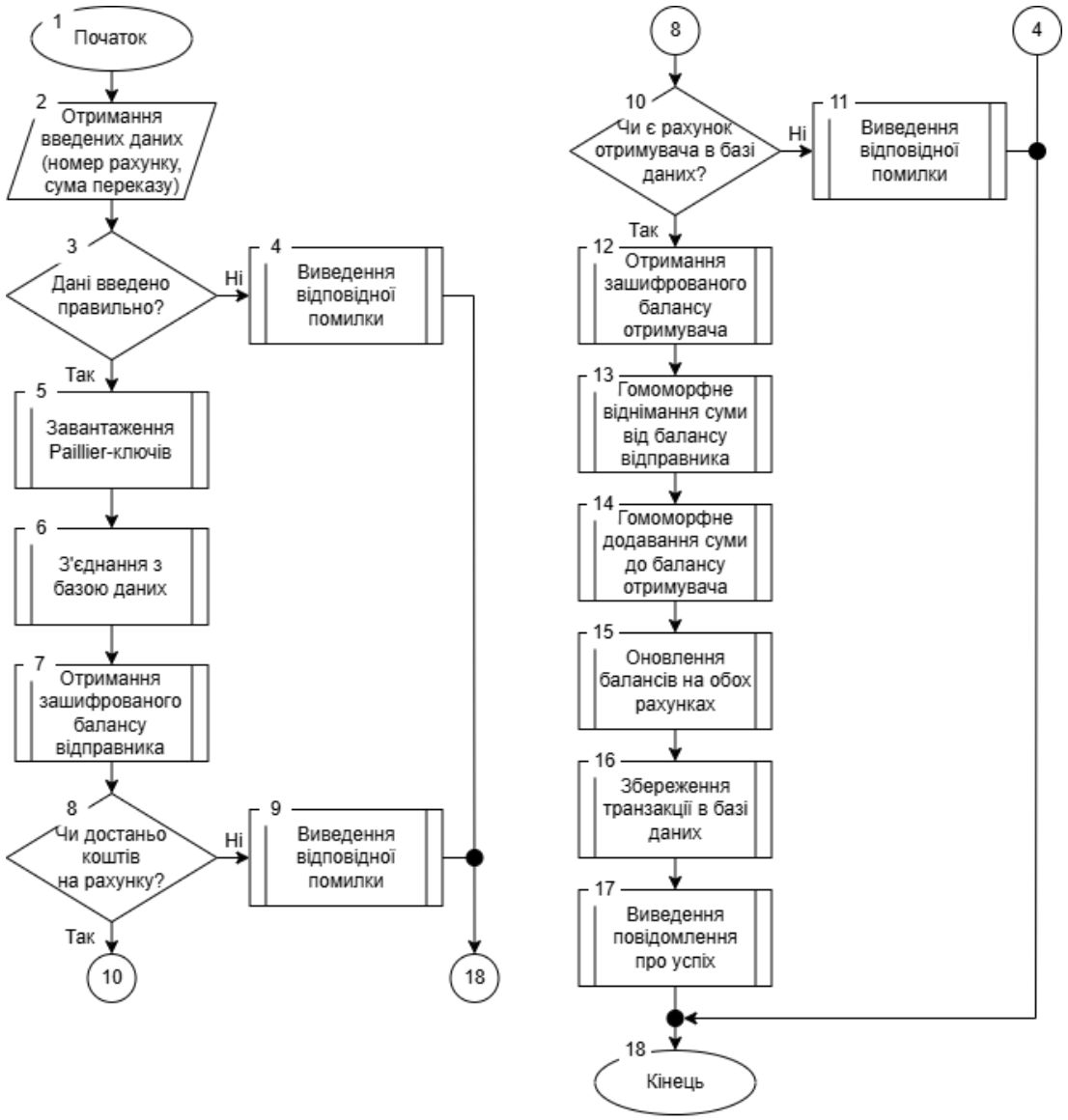

Рисунок 2.6 – Алгоритм здійснення переказу коштів

Згідно з вище наведеним рисунком, після введення номеру рахунку і суми переказу в необхідних полях, відбувається перевірка цих значень. Якщо дані введено неправильно, відбувається виведення помилки з вказівками щодо вирішення проблеми, після чого алгоритм починається з початку.

Далі відбувається завантаження відкритого і закритого ключів криптосистеми з відповідного файлу збереження. Наступним кроком є з’єднання з базою даних та отримання зашифрованого балансу відправника. На жаль, криптосистема Paillier не дає можливості порівнювати зашифровані значення без їхнього розшифрування, тому баланс відправника розшифровується для

перевірки чи не є він меншим ніж введена сума переказу. Потім іде перевірка наявності рахунку отримувача. Якщо вона проходить успішно, то програма отримує зашифрований баланс отримувача і продовжує алгоритм далі.

Останніми кроками є проведення гомоморфних операцій над зашифрованими балансами і їхнє оновлення в базі даних. Спочатку йде гомоморфне віднімання суми переказу від балансу відправника. Одразу за ним іде гомоморфне додавання суми до балансу отримувача. Після цього баланси обох користувачів перезаписуються в базі даних на нові.

2.5 Висновки

У було розроблено модель роботи програмного продукту за допомогою UML діаграм, а також блок-схеми розроблюваних алгоритмів зберігання Paillier-ключів та гомоморфного віднімання. Окрім цього було розроблено і описано основний алгоритм роботи програмного застосунку і алгоритм здійснення переказу коштів.

3 РОЗРОБКА ПРОГРАМНИХ МОДУЛІВ РЕАЛІЗАЦІЇ СИСТЕМИ ЗБЕРІГАННЯ ТА ОБРОБКИ ДАНИХ ЗА ДОПОМОГОЮ ГОМОМОРФНОГО ШИФРУВАННЯ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення

Для розробки прикладного програмного забезпечення, що демонструє використання гомоморфного шифрування на прикладі криптосистеми Paillier у банківських операціях, було обрано низку технологій, які забезпечують ефективну розробку, підтримку наочності інтерфейсу та гнучкість у реалізації криптографічних алгоритмів.

Мова програмування Java була обрана як основна технологія реалізації програмного продукту. Java [17] – це об’єктно-орієнтована мова, яка має потужну стандартну бібліотеку та підтримує високий рівень безпеки, що є критичним у розробці систем, пов’язаних із обробкою конфіденційних даних. Завдяки своїй архітектурній незалежності та підтримці багатьох сучасних інструментів, Java дозволяє створювати надійні та масштабовані застосунки.

У якості середовища розробки було обрано IntelliJ IDEA [18], яке забезпечує зручну підтримку для Java-проєктів, вбудовану інтеграцію з системами керування версіями, інтелектуальну підсвітку синтаксису, автоматичне доповнення коду та зручне налагодження. Завдяки цим можливостям IntelliJ IDEA сприяє підвищенню продуктивності розробника та спрощує роботу з великими проєктами.

Для побудови графічного інтерфейсу користувача застосовано JavaFX [19], що є сучасною технологією для створення багатофункціональних GUI [20] у Java-додатках. JavaFX забезпечує підтримку роботи з .fxml-файлами [21], що дозволяє розділити логіку програми та її інтерфейс, що позитивно впливає на масштабованість і зручність підтримки проєкту. Крім того, JavaFX підтримує адаптивну верстку, стилізацію через CSS [22] та інтеграцію з мультимедійними елементами, що дозволяє створити зручний і сучасний інтерфейс.

Для візуального проєктування вікон було використано інструмент Scene Builder [23], який дозволяє швидко і без написання коду створювати графічні інтерфейси у форматі .fxml. Це особливо корисно на етапі макетування вікон користувача, оскільки забезпечує можливість побачити попередній вигляд елементів інтерфейсу та налаштувати їх властивості безпосередньо через графічний інтерфейс.

База даних була створена у MS Access [24]. Цей інструмент було обрано завдяки його простоті в освоєнні, швидкому створенню таблиць та зручності роботи з невеликими обсягами даних. Для розробки прототипу системи MS Access надає достатній функціонал і дозволяє швидко налагодити взаємодію з Java-додатком через відповідні драйвери.

Криптосистема Paillier була реалізована вручну без використання сторонніх бібліотек. Таке рішення було прийняте з метою глибшого розуміння принципів її роботи, включаючи генерацію ключів, шифрування, дешифрування та гомоморфні властивості. Реалізація з «нуля» дозволила детально дослідити математичні основи криптосистеми та продемонструвати її використання у практичному застосунку.

Через потребу у порівнянні введених користувачем значень із зашифрованими та збереженими в базі даних і відсутність безпечного способу порівнянь даних у Paillier, було реалізовано просту схему гешування на основі алгоритму SHA-256. Це дозволяє забезпечити базову перевірку конфіденційної інформації без зберігання її у відкритому вигляді. Використання гешування додає ще один рівень захисту при роботі з персональними даними користувачів.

Крім того, для зберігання даних користувача після автентифікації та забезпечення моментального доступу до системи було використано технологію JWT. У ньому зберігається інформація про користувача в зашифрованому вигляді, що дозволяє уникати повторної перевірки даних.

Таким чином, вибір зазначених технологій був зумовлений потребою у створенні зручного, безпечного та наочного програмного рішення. Поєднання Java, IntelliJ IDEA, JavaFX, Scene Builder та MS Access дозволяє ефективно

реалізувати програму з графічним інтерфейсом, а ручна реалізація криптосистеми Paillier дає змогу краще зрозуміти принцип її роботи та демонструє можливість використання гомоморфного шифрування в сучасних IT-системах.

3.2 Розробка модуля авторизації користувачів

При розробці модуля авторизації користувачів було створено декілька допоміжних модулів. Першим з них був модуль для логування (див. рисунок 3.1). Його було реалізовано для фіксації важливих подій під час роботи програми, налагодження, а також збереження діагностичної інформації.

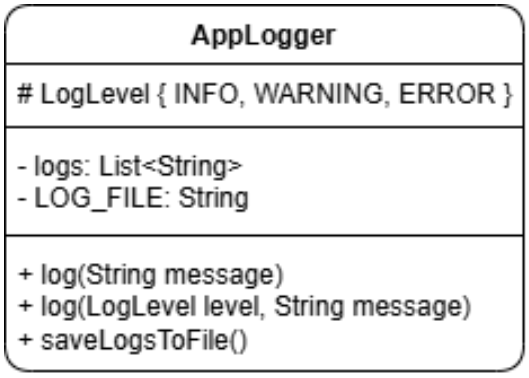

Рисунок 3.1 – Діаграма класу «AppLogger»

Даний клас містить перерахування рівнів важливості повідомлень (INFO, WARNING, ERROR), список логів, шлях до файлу зі збереженими логами і функції для реалізації роботи класу.

Перша функція потрібна для логування текстового повідомлення з рівнем «INFO» за замовчуванням. Вона викликає перевантажену другу функцію, передаючи їй рівень «INFO».

Друга функція логує текстове повідомлення із заданим рівнем важливості. Вона отримує поточну дату та час, формує мітку часу і виводить форматоване повідомлення в консоль та додає його до списку логів для подальшого збереження.

Третя функція зберігає всі накопичені лог-повідомлення у текстовий файл «log.txt». Дана функція відкриває або створює файл для запису і записує по порядку логи зі списку. Ця функція викликається автоматично при закритті програми.

Далі було розроблено короткий модуль для створення з’єднання з базою даних (див. рисунок 3.2).

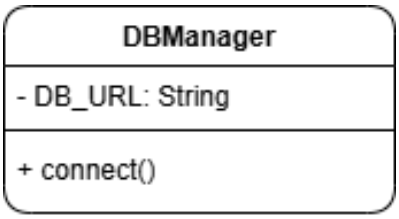

Рисунок 3.2 – Діаграма класу «DBManager»

Даний клас відповідає за встановлення з’єднання з базою даних, що розміщена у файлі database.accdb, використовуючи драйвер UCanAccess [25] для взаємодії з базами даних MS Access. Клас містить одну константу – шлях до бази даних, і одну функцію, що реалізує з’єднання.

Функція connect() намагається встановити з’єднання з базою даних за вказаною адресою. У разі успіху відбувається логування повідомлення про успішне підключення до бази даних та повернення об’єкту типу «Connection». Якщо під час підключення виникає помилка, то виводиться повідомлення про помилку з рівнем «ERROR», і функція повертає null.

Наступним допоміжним модулем був модуль «DialogUtil» для виведення діалогових вікон (див. рисунок 3.3).

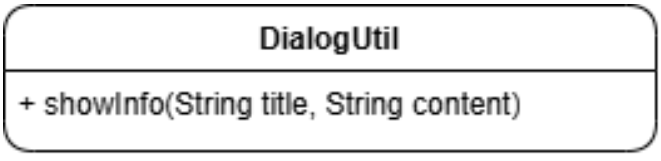

Рисунок 3.3 – Діаграма класу «DialogUtil»

Даний клас реалізує утилітну функцію для відображення довідкових вікон типу INFORMATION у графічному інтерфейсі користувача.

Функція showInfo() створює інформаційне діалогове вікно з переданими параметрами заголовка та тексту повідомлення. Перед відкриттям вікна виконується логування повідомлення з інформацією про його відкриття за допомогою «AppLogger». Далі створюється об’єкт «Alert», налаштовується його заголовок, відключається графічне зображення та заголовок вмісту, і застосовується стилізація з файлу «style.css», який підключається до «DialogPane». Після цього вікно відображається користувачу.

Після цього було зроблено допоміжний «JwtUtil» (див. рисунок 3.4). Його було створено для реалізації механізму збереження інформації про автентифікацію користувача, а також забезпечення можливості автоматичного входу в систему без повторного введення облікових даних.

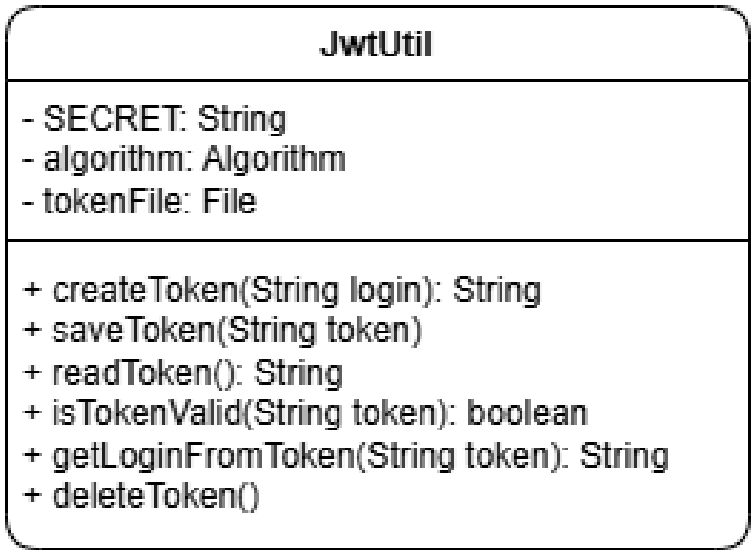

Рисунок 3.4 – Діаграма класу «JwtUtil»

Клас використовує симетричний алгоритм HMAC256 [26] з наперед визначеним секретним ключем для створення і перевірки токенів. Він містить набір функцій, які дозволяють створювати, зберігати, читати, перевіряти та видаляти токени, а також отримувати з них логін користувача.

Перша функція створює токен для заданого логіну. У токен вбудовується інформація про логін користувача, час створення та термін дії (одна година). Після створення відбувається запис до логів про створення токена.

Друга функція зберігає створений токен у файл «user.jwt». Вона відкриває або створює файл і записує в нього токен, після чого фіксує подію в логах.

Третя функція читає і повертає токен із файлу, якщо він існує. Якщо файл не знайдено, функція повертає null.

Наступна функція перевіряє дійсність токена. Вона розшифровує токен, перевіряє його підпис і дату закінчення дії. У разі успішної перевірки повертає true, інакше – false.

Далі йде функція, що розшифровує токен і витягує з нього логін користувача. Крім цього, вона фіксує подію автоматичного входу в логах.

Остання функція відповідає за видалення файлу з токеном. Якщо видалення не вдалося, генерується повідомлення про помилку. У разі успішного видалення відповідна подія також фіксується в логах.

Ще одним допоміжним модулем був «HashFunc» (див. рисунок 3.5). Даний клас реалізує утилітні функції для гешування текстових даних з використанням криптографічного алгоритму SHA-256 та порівняння гешованих значень.

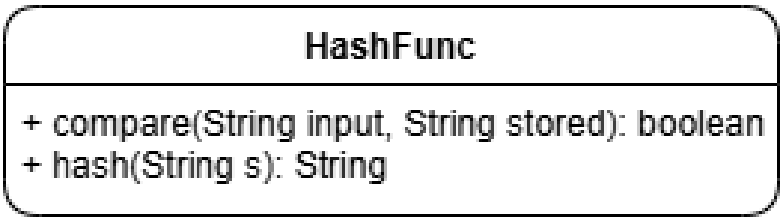

Рисунок 3.5 – Діаграма класу «HashFunc»

Клас забезпечує безпечну обробку чутливої інформації, такої як пін-коди та номери рахунків, у вигляді гешованих рядків.

Перша функція призначена для порівняння вхідного значення з уже збереженим гешем. Вона спочатку гешує вхідний рядок за допомогою наступної

функції, після чого порівнює отримане значення з переданим контрольним гешем. Повертає результат логічного порівняння.

Друга функція виконує гешування переданого рядка за допомогою алгоритму SHA-256. Вона використовує стандартний клас «MessageDigest» для створення байтового масиву гешу, а потім перетворює кожен байт у шістнадцяткове представлення та об’єднує їх у підсумковий рядок. У разі успіху здійснюється логування події. Якщо алгоритм SHA-256 не підтримується у середовищі виконання, виводиться повідомлення про помилку.

Після цього було розроблено сам модуль для авторизації користувача або ж клас «Login» (див. рисунок 3.6).

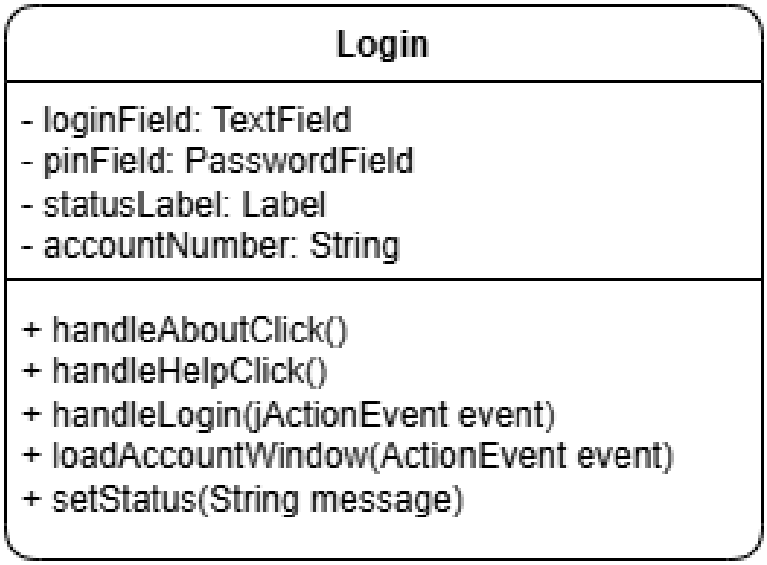

Рисунок 3.6 – Діаграма класу «Login»

Даний клас реалізує логіку авторизації користувача у застосунку та обробку відповідних подій. Він містить поля для введення логіна та пін-коду, мітку статусу, а також функції для перевірки введених даних, виклику інформаційних вікон і переходу до головного вікна акаунта.

Перша функція відкриває інформаційне вікно «Про застосунок», де відображається короткий опис призначення програми та ім’я розробника. Для цього використовується утиліта «DialogUtil».

Друга функція відкриває інформаційне вікно «Допомога», в якому міститься інструкція користування застосунком. Використовується той самий механізм, що й у попередній функції.

Третя функція обробляє спробу входу. Вона зчитує введені логін і пін-код, встановлює з’єднання з базою даних, виконує пошук користувача за логіном і перевіряє правильність пін-коду за допомогою функції гешування. У разі успішної перевірки зберігається JWT-токен і викликається функція для переходу до вікна акаунта.

Четверта функція завантажує головне вікно користувача. Вона встановлює нову сцену, передає номер рахунку відповідному контролеру і застосовує стилі оформлення.

П’ята функція виводить повідомлення про стан авторизації у відповідне текстове поле та логує повідомлення з рівнем «WARNING».

3.3 Розробка модуля керування банківським рахунком

При розробці даного модуля також було створено допоміжний модуль «ExRateLoader» (див. рисунок 3.8).

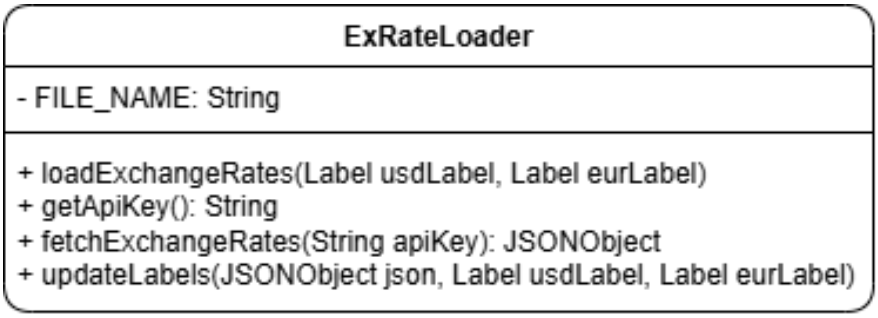

Рисунок 3.9 – Діаграма класу «ExRateLoader»

Даний клас відповідає за завантаження та відображення актуальних валютних курсів (долара США та євро) щодо української гривні. Його основна функція – автоматичне оновлення і виведення даних про курс валют у графічному інтерфейсі користувача.

Перша функція використовує усі наступні функції класу. Вона запускає окремий потік, у якому спершу зчитується API-ключ з локального файлу. Далі надсилається запит до онлайн-сервісу ExchangeRateAPI [27] для отримання курсу валют. Після отримання відповіді, значення курсу виводяться на мітках у графічному інтерфейсі. Якщо сталася помилка або ключ відсутній, на мітках виводиться повідомлення «Н/д».

Друга футнкція перевіряє наявність і читає API-ключ із локального конфігураційного файлу. У разі відсутності ключа або файлу виводиться повідомлення про помилку через логування.

Третя функція виконує HTTP-запит до API сервісу та повертає JSON-відповідь із курсами валют щодо гривні.

Четверта функція обробляє отриману відповідь та оновлює текст міток, відображаючи зворотні курси (тобто 1 / курс USD або EUR до UAH), що дозволяє користувачеві бачити, скільки гривень потрібно для купівлі 1 одиниці іноземної валюти.

Далі було розроблено власне модуль керування банківським рахунком – «Account» (див. рисунок 3.9).

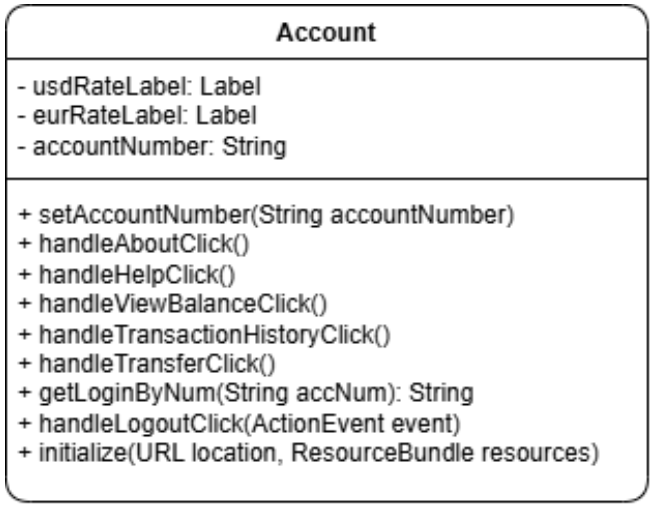

Рисунок 3.9 – Діаграма класу «Account»

Даний клас реалізує функціонал головного вікна користувача після входу в систему. Він відповідає за відображення актуальних курсів валют, обробку

натискань на кнопки та виклик відповідних вікон, таких як перегляд балансу, переказ коштів або історія транзакцій. Клас також забезпечує можливість виходу з облікового запису та виклик інформаційних діалогів про застосунок.

Даний модуль як і модуль входу в систему містить поле для збереження номера рахунку користувача і функції виклику діалогових вікон довідки. Окрім цього він має два графічні елементи для відображення курсів долара та євро, які оновлюються під час ініціалізації через метод initialize().

Четверта функція відкриває нове вікно перегляду балансу, де баланс користувача дешифрується і виводяться у зручному вигляді. Номер рахунку передається до відповідного підмодуля «ViewBalance» (див. рисунок 3.10), який відповідає за відображення цього вікна.

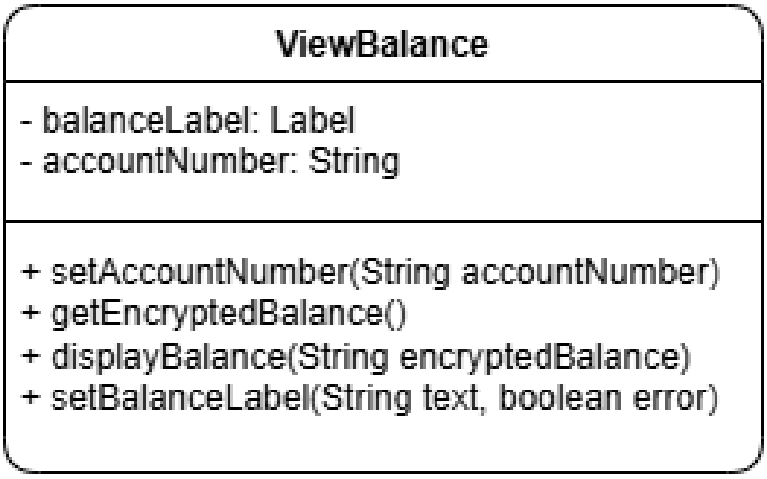

Рисунок 3.10 – Діаграма класу «ViewBalance»

Даний підмодуль містить мітку для виведення балансу та змінну для збереження номера рахунку, який передається з «Account».

Перша функція використовується для встановлення номера рахунку, за яким буде здійснюватися запит до бази даних.

Друга функція підключається до бази даних, виконує SQL-запит для пошуку користувача за номером рахунку, отримує зашифрований баланс і перевіряє, чи знайдено запис та чи не є баланс порожнім. У разі успіху

зашифроване значення передається на подальшу обробку, інакше виводиться повідомлення про помилку.

Третя функція виконує дешифрування балансу за допомогою модуля «Paillier» (див. рисунок 3.11), який реалізує однойменну криптосистему. для керування ключами цієї криптосистеми. Після цього обчислюється справжнє значення у гривнях, ділячи результат на 100. Отримане значення форматовано виводиться на екран.

Четверта функція встановлює текст у мітці балансу. Якщо сталася помилка, то вона логується як повідомлення рівня «ERROR».

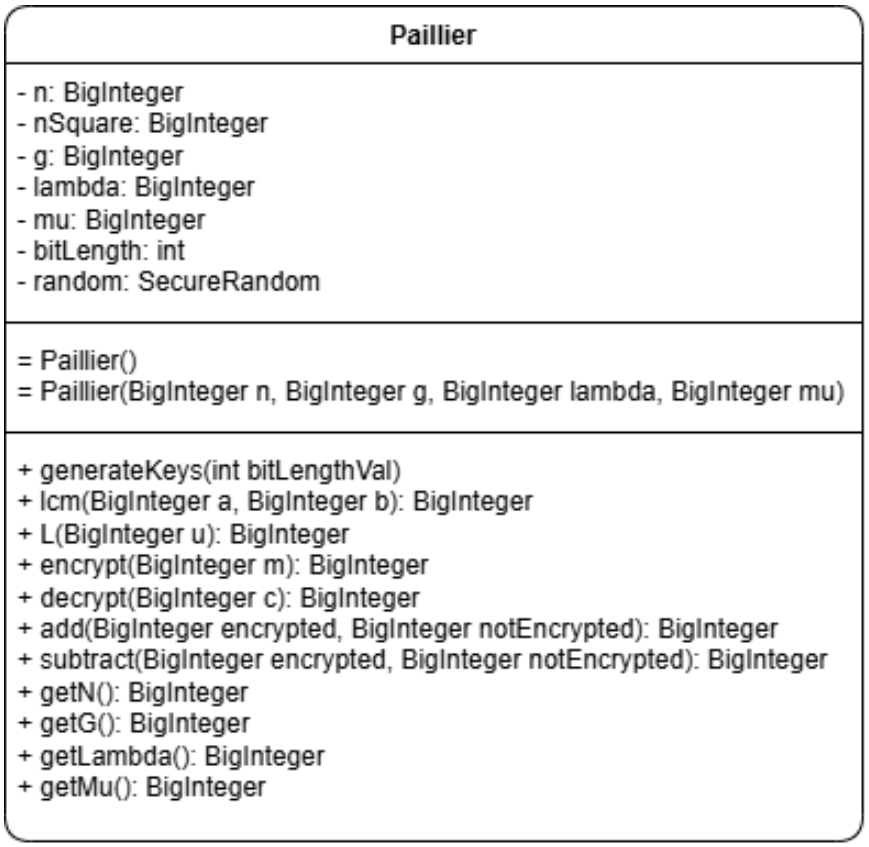

Рисунок 3.11 – Діаграма класу «Paillier»

Клас «Paillier» використовується для шифрування, дешифрування та виконання операцій додавання і віднімання без розшифрування проміжних результатів. Клас містить основні параметри алгоритму, генератор випадкових чисел, а також довжину ключа, яка визначається під час генерації.

Конструктори класу забезпечують два режими роботи: перший використовується для створення нових ключів за допомогою генератора випадкових чисел, другий – для ініціалізації з уже збереженими ключами.

Перша функція генерує пари ключів на основі двох великих простих чисел p і q. Далі обчислюються необхідні параметри криптосистеми: число n, функція λ як найменше спільне кратне або НСК(p-1, q-1), а також μ – обернене значення допоміжної функції L. В якості числа g використовується простий варіант: n + 1.

Четверта функція реалізує шифрування повідомлення m згідно з формулою криптосистеми. Для цього генерується випадкове число r, взаємно просте з n, і обчислюється результат. Перед виконанням логуються відомості про операцію шифрування.

П’ята функція виконує дешифрування отриманого шифротексту c за допомогою функцій L та ключів λ, μ. Результат – це початкове повідомлення у вигляді числа. Процес також супроводжується логуванням.

Шоста і сьома функції реалізують гомоморфне додавання та віднімання відповідно. При додаванні перемножується шифротекст з шифрованим доданком, а при відніманні – шифротекст з оберненим зашифрованим від’ємником.

Останні функції також називаються гетерами. Вони надають доступ до відкритих та закритих параметрів ключів і необхідні для безпечної взаємодії з модулем «KeyManager» (див. рисунок 3.12).

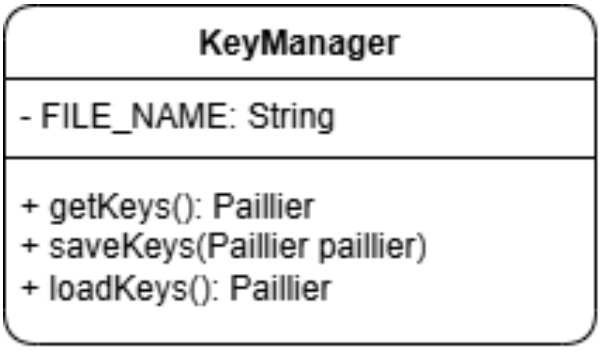

Рисунок 3.12 – Діаграма класу «KeyManager»

Даний клас відповідає за керування ключами криптосистеми, включаючи їху. генерацію, збереження у файл та завантаження з нього. Основна його мета – забезпечити зручний і безпечний доступ до криптографічних ключів, необхідних для виконання операцій шифрування, дешифрування та гомоморфних обчислень у застосунку.

Перша функція є головною, вона перевіряє наявність збережених ключів у файлі. Якщо ключі вже існують, вони завантажуються третьою функцією і використовуються надалі. Якщо ж файл відсутній або ключі не збережено, створюється новий екземпляр класу «Paillier», генеруються нові ключі, і вони зберігаються у файл за допомогою другої функції. Усі дії супроводжуються логуванням.

Друга функція відповідає за збереження ключів у файл paillier-keys.properties. Параметри відкритого ключа (n, g) зберігаються у відкритому вигляді, тоді як елементи закритого ключа (λ, μ) шифруються за допомогою класу «EncryptorAES» (див. рисунок 3.13).

Третя функція перевіряє, чи існує файл ключів, і якщо так – завантажує з нього усі необхідні параметри. Після відновлення всіх значень створюється об'єкт «Paillier» із завантаженими параметрами. Якщо файл відсутній, метод повертає null, що сигналізує про потребу в генерації нових ключів.

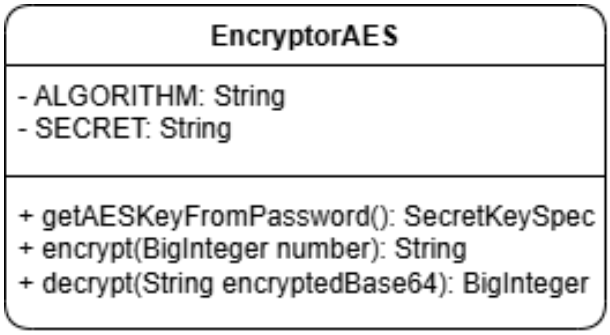

Рисунок 3.13 – Діаграма класу «EncryptorAES»

Даний клас реалізує шифрування та дешифрування чисел типу «BigInteger» з використанням симетричного алгоритму AES. Його основне

призначення – захист конфіденційних даних, таких як компоненти приватного ключа криптосистеми Paillier, перед збереженням у файл. У класі використовується фіксований пароль, з якого генерується AES-ключ довжиною 128 біт.

Перша функція виконує генерацію ключа AES на основі символьного пароля, гешованого алгоритмом SHA-256. Отримані байти обрізаються до 16 байтів для створення ключа, сумісного з AES-128. Метод забезпечує сталість ключа при кожному виклику, що дозволяє шифрувати й дешифрувати дані однією і тією ж функцією.

Друга функція приймає на вхід число, перетворює його у байтовий масив та шифрує за допомогою AES. Результат шифрування кодується у форматі Base64 [28], що дозволяє зручно зберігати його у текстових файлах.

Третя функція виконує зворотну операцію: вона приймає закодований у Base64 зашифрований текст, декодує його у байти, розшифровує з використанням того ж AES-ключа і повертає відновлене число.

Повертаючись до програмного модуля «Account», його п’ята функція відкриває вікно перегляду історії транзакцій. Для цього було створено підмодуль «TransactionHistory» (див. рисунок 3.14).

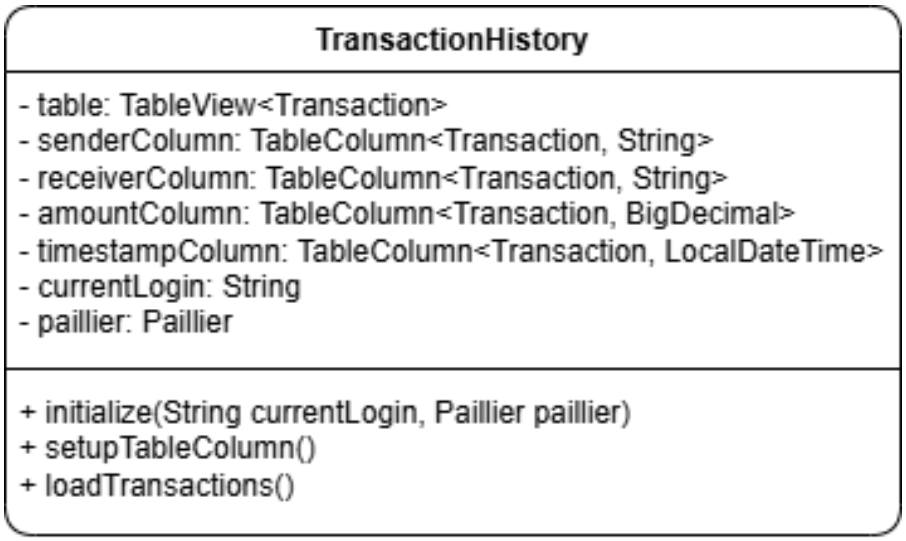

Рисунок 3.14 – Діаграма класу «TransactionHistory»

Даний клас відповідає за відображення історії транзакцій поточного користувача у вигляді таблиці. Він інтегрується з таблицею, у яку передаються об’єкти типу «Transaction» (див. рисунок 3.15), що містять інформацію про відправника, отримувача, суму переказу та дату з часом здійснення операції.

Перша функція викликається під час завантаження вікна та приймає логін поточного користувача і об’єкт «Paillier». Вона ініціалізує внутрішні поля та викликає дві допоміжні функції: для налаштування стовпців таблиці та для завантаження транзакцій з бази даних.

Друга функція встановлює відповідність між стовпцями таблиці JavaFX і властивостями об’єкта «Transaction». Завдяки цьому кожен стовпець автоматично відображає відповідне значення для кожної транзакції у таблиці.

Третя функція здійснює підключення до бази даних і виконує SQL-запит для вибірки всіх транзакцій, де поточний користувач є відправником або отримувачем. Сума транзакції, що зберігається в зашифрованому вигляді, розшифровується, після чого сума масштабується до формату з двома десятковими знаками. Якщо хоча б одне з критичних значень відсутнє, у лог записується повідомлення про помилку.

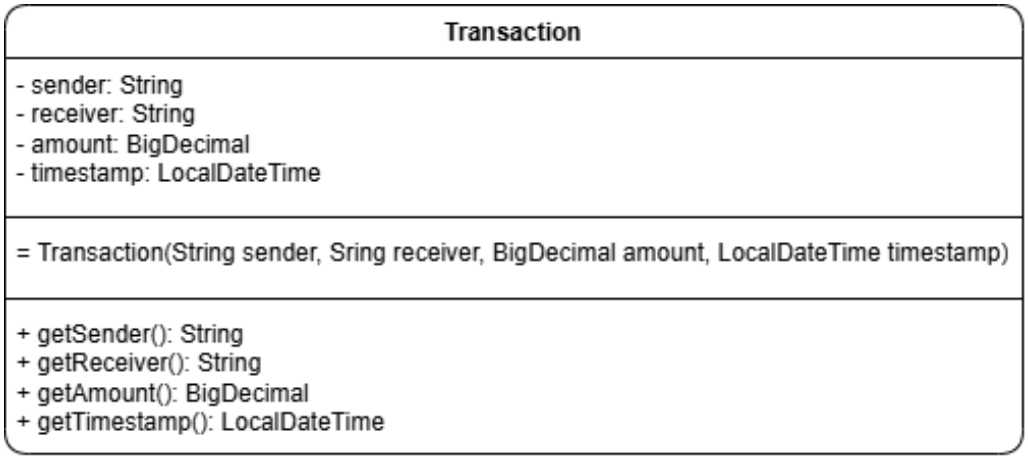

Рисунок 3.15 – Діаграма класу «Transaction»

Клас Transaction є простою моделлю даних, що використовується для зберігання інформації про одну банківську транзакцію. Його основне

призначення – представляти окремий запис переказу коштів між двома користувачами у зручному вигляді для подальшого відображення в інтерфейсі користувача або обробки.

Конструктор класу приймає логін відправника, логін отримувача, суму транзакції, а також дату і час здійснення переказу.

Гетери полів класу потрібні для правильної і захищеної передачі значень у колонки таблиці з транзакціями.

Знову повертаючись до класу «Account», варто зазначити, що його останні функції відкривають нове вікно для здійснення переказу коштів та повертають користувача до вікна авторизації відповідно.

3.4 Розробка модуля здійснення переказу коштів

Даний модуль (див. рисунок 3.16) відповідає за обробку грошових переказів між користувачами. Його основне завдання – забезпечити безпечний переказ коштів шляхом перевірки введених даних, взаємодії з базою даних, шифруванням та дешифруванням сум за допомогою класу криптосистеми, а також оновленням балансу відправника й отримувача.

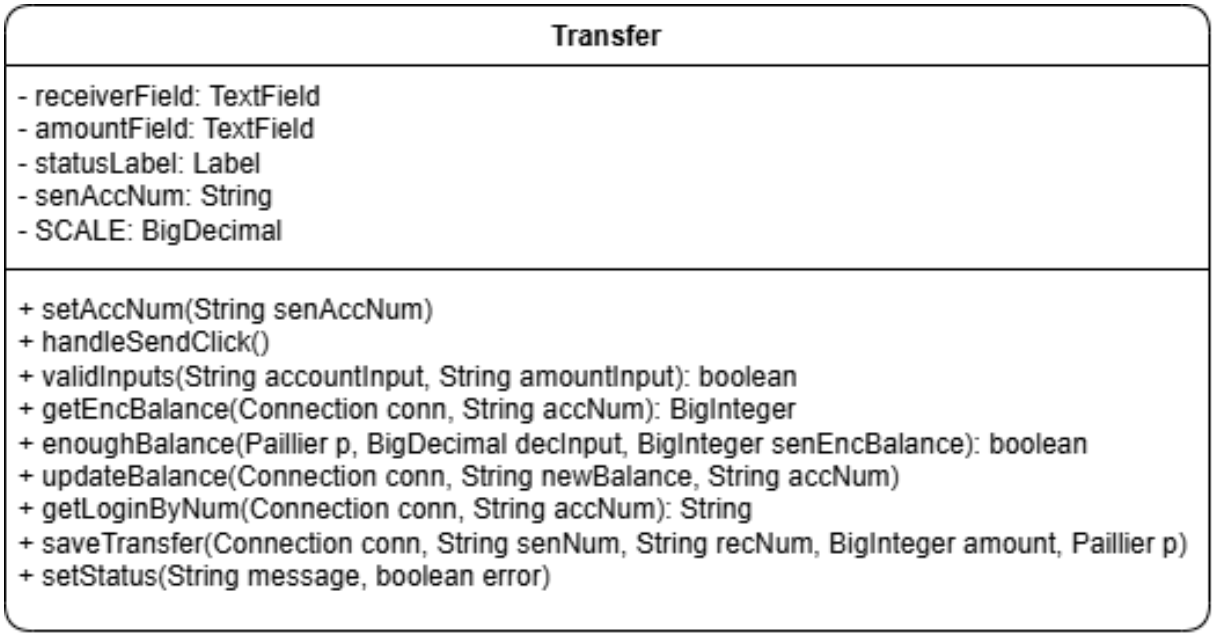

Рисунок 3.16 – Діаграма класу «Transfer»

Перша функція зберігає номер рахунку поточного авторизованого користувача, який надалі використовується для ідентифікації та перевірки балансу.

Друга функція є основною логікою переказу. Вона зчитує введені дані, виконує їхню перевірку через третю функцію і переводить суму у формат з масштабом 100 для коригування. Після цього функція завантажує ключі Paillier, під'єднується до бази і отримує зашифровані баланси відправника та отримувача. Далі проводиться перевірка на наявність достатньої кількості коштів, виконується обчислення нових зашифрованих балансів, оновлення їх у базі даних і запис переказу в таблиці транзакцій.

Третя функція виконує перевірку введених користувачем даних. Вона переконується, що поля не порожні, що номер рахунку отримувача не збігається з номером рахунку відправника, та що введена сума є додатнім числом з не більше ніж двома десятковими знаками.

Четверта функція виконує запит до бази даних, щоб отримати зашифрований баланс користувача за номером рахунку. Якщо дані відсутні, метод повертає null.

П'ята функція використовується для перевірки, чи має відправник достатньо коштів для переказу. Вона розшифровує баланс і порівнює його з введеною сумою.

Шоста функція оновлює зашифрований баланс у таблиці Клієнти для вказаного номера рахунку. Нове значення балансу передається як рядок.

Сьома функція отримує логін користувача на основі номера його рахунку. Це потрібно для коректного збереження інформації про відправника та отримувача в таблиці транзакцій.

Восьма функція створює новий запис у таблиці з транзакціями, шифрує суму переказу, встановлює поточну дату й час, і зберігає повну інформацію про транзакцію між користувачами.

Дев'ята функція відповідає за відображення повідомлення про результат операції у графічному інтерфейсі. У випадку помилки повідомлення

відображається червоним кольором, у разі успіху – зеленим. Також метод здійснює логування подій.

3.5 Розробка інтерфейсу програмного застосунку

При розробці інтерфейсу програмного забезпечення було приділено увагу його зрозумілості та зручності для користувача. Було обрано синій колір як основний зі світлішим і темнішим відтінками, а також білий і червоний як допоміжні кольори.

При відкритті програмного застосунку, користувача зустрічає вікно авторизації (див. рисунок 3.17). У даному вікні присутні поля для введення логіну та пін-коду і кнопку «Увійти» для підтвердження введених даних.

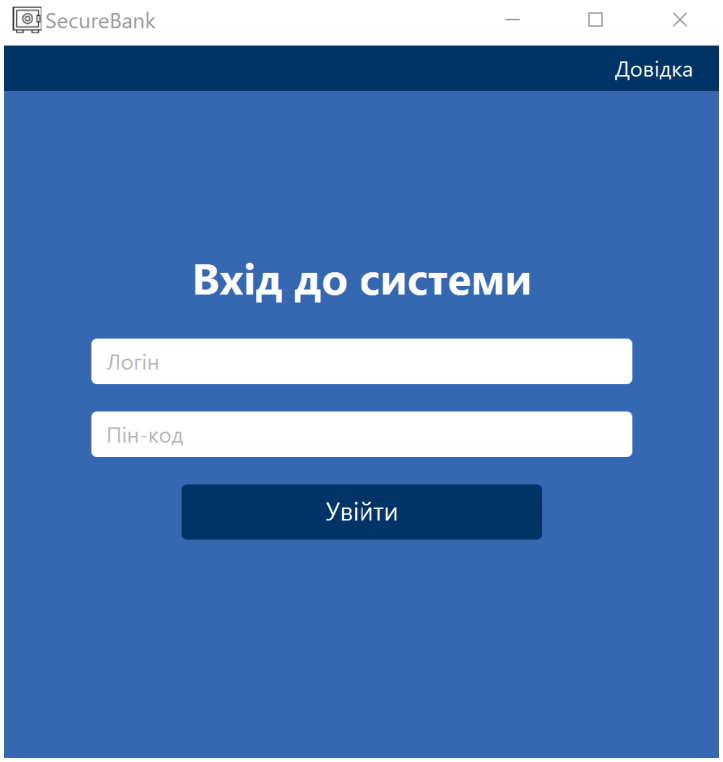

Рисунок 3.17 – Вікно авторизації

Окрім цього, вікно має меню з пунктом «Довідка», яке містить підпункти «Допомога» та «Про застосунок», які відкривають відповідні їм діалогові вікна (див. рисунок 3.18-3.19).

Вікно «Допомога» містить коротку інструкцію для користувача про те, як використовувати основні функції програми.

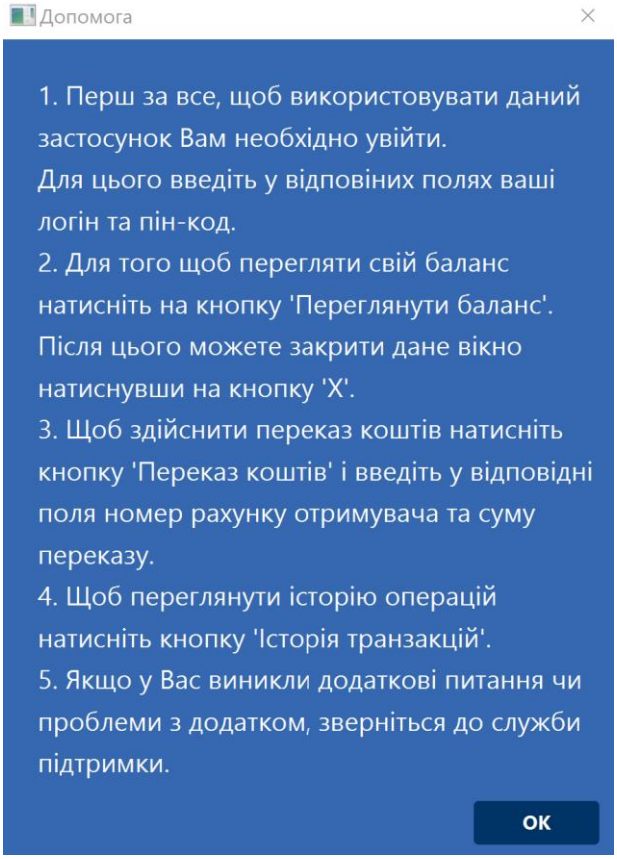

Рисунок 3.18 – Діалогове вікно «Допомога»

Вікно «Про застосунок» має короткий опис програмного застосунку та інформацію про розробника програми.

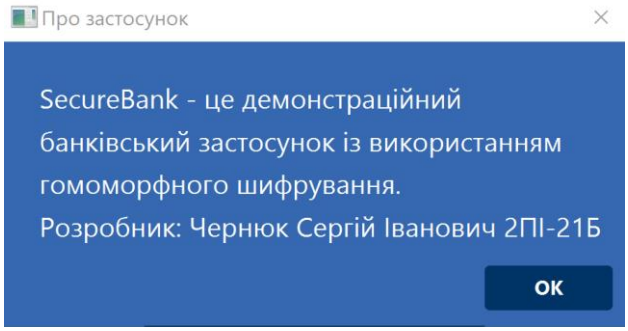

Рисунок 3.19 – Діалогове вікно «Про застосунок»

Після входу в систему, вікно авторизації замінюється на вікно кабінету користувача (див. рисунок 3.20). Воно також містить меню з пунктом «Довідка» на випадок, якщо у користувача виникли якісь проблеми з додатком після входу в систему. Також дане вікно містить курси валют і чотири кнопки: «Переглянути баланс», «Переказ коштів», «Історія транзакцій» та «Вихід».

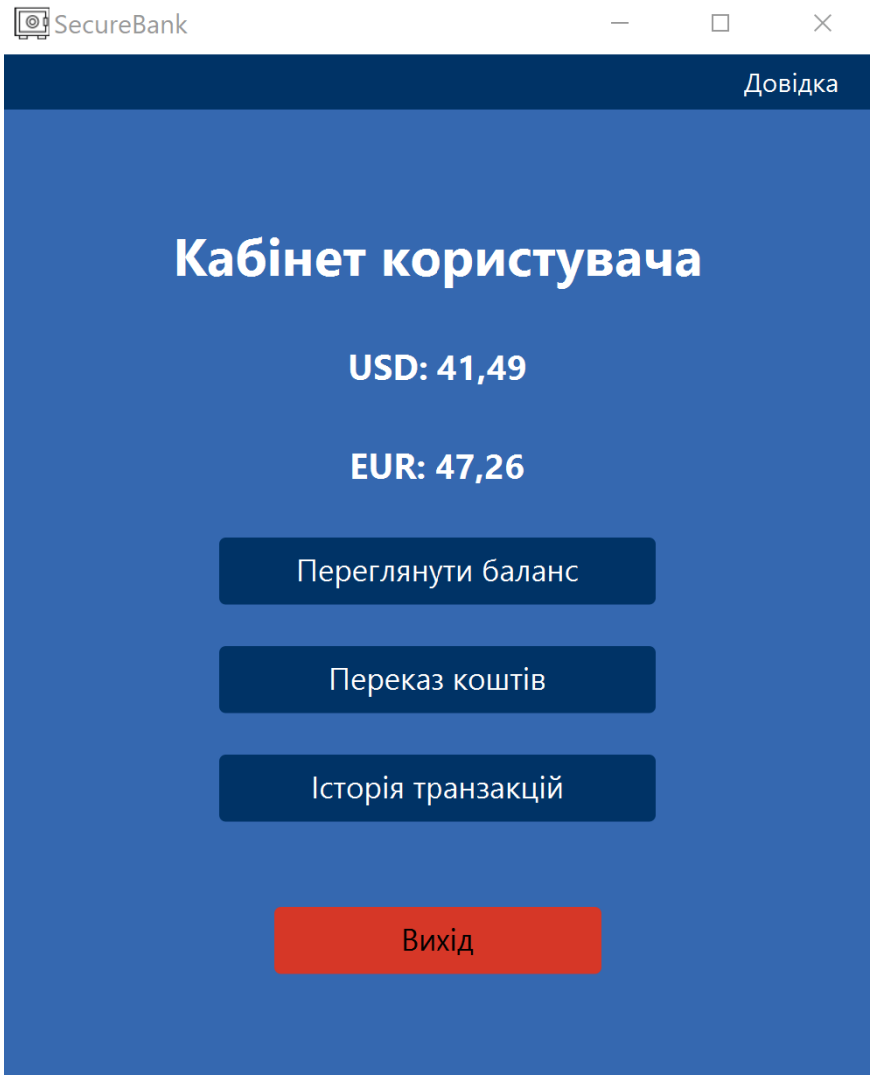

Рисунок 3.20 – Вікно кабінету користувача

Кнопка «Переглянути баланс» відкриває власне вікно перегляду балансу (див. рисунок 3.21). Це вікно виводить баланс на рахунку користувача. Воно було розроблене для того, щоб баланс не розшифровувався одразу після авторизації. Після перегляду, дане вікно можна просто закрити.

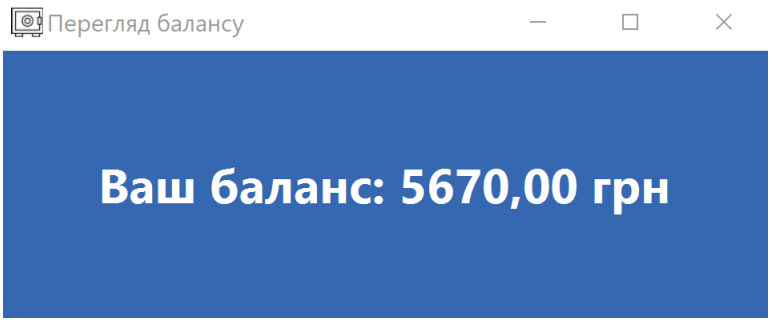

Рисунок 3.21 – Вікно перегляду балансу

При натисканні на кнопку «Переказ коштів» відкривається додаткове вікно для здійснення переказу (див. рисунок 3.22). Воно містить поля для введення номеру рахунку отримувача і суми переказу. Після того, як користувач переконався в правильності введених даних, він може натиснути на кнопку «Надіслати», і якщо все вірно, то дане вікно можна також спокійно закривати.

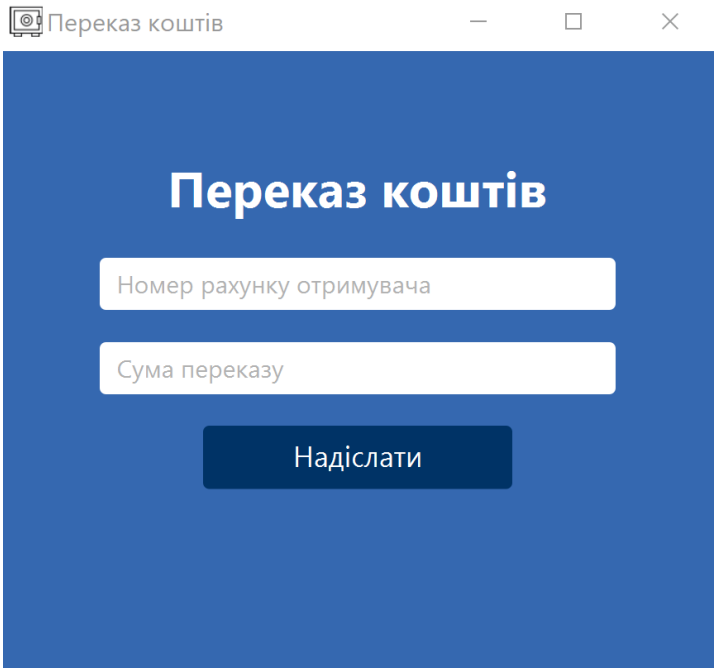

Рисунок 3.22 – Вікно здійснення переказу коштів

Після натискання на кнопку «Історія транзакцій» також відкривається додаткове вікно (див. рисунок 3.23). У даному вікні знаходиться таблиця з

інформацією про відправника, отримувача, суму, а також дату і час останніх транзакцій.

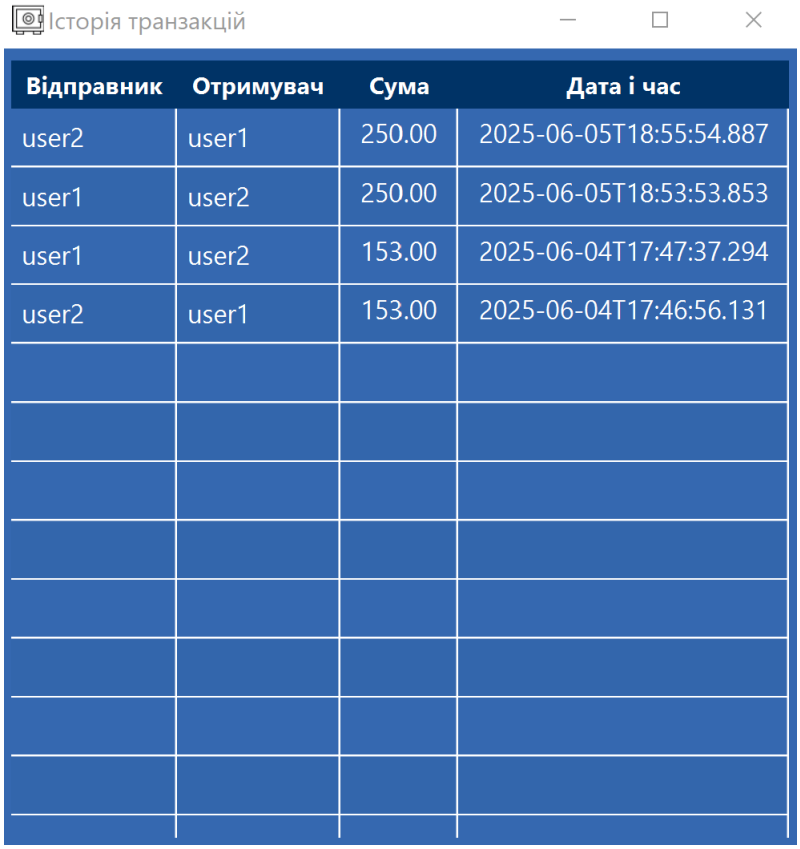

Рисунок 3.23 – Вікно перегляду історії транзакцій

Остання кнопка в кабінеті користувача повертає його до вікна входу, аби навіть не вимикаючи програму можна було не перейматися про безпеку рахунку.

3.6 Висновки

У третьому розділі було обґрунтовано вибір технологій та інструментів, використаних для розробки прикладного програмного забезпечення, що демонструє використання гомоморфного шифрування. З урахуванням вимог до безпеки, зручності та наочності інтерфейсу було обрано мову програмування Java та фреймворк JavaFX для створення графічного інтерфейсу. Середовищем розробки стала IntelliJ IDEA, а для візуального макетування інтерфейсу – Scene

Builder. Базу даних реалізовано у MS Access через її простоту та швидкість налаштування. Криптосистему Paillier було реалізовано вручну, що забезпечило глибше розуміння її принципів, а для перевірки конфіденційних даних розроблено просту схему гешування. Сукупність обраних засобів забезпечила реалізацію наочного, безпечного та ефективного програмного рішення. У розділі також було описано розробку основних програмних модулів та графічного інтерфейсу користувача, що забезпечують базову функціональність застосунку для демонстрації банківських операцій із використанням гомоморфного шифрування.

4 ТЕСТУВАННЯ ПРОГРАМИ

4.1 Тестування системи

Тестування програмного забезпечення є невід'ємною частиною процесу розробки та впровадження інформаційних систем, оскільки воно дозволяє виявити помилки, дефекти та невідповідності вимогам ще до етапу реального використання програми [29]. Основною метою тестування є забезпечення коректної та безпечної роботи програмного продукту, особливо у випадках, коли йдеться про обробку конфіденційних даних.

У межах цього проєкту тестування проводилося на локальній тестовій версії програми, встановленій на комп’ютері з операційною системою Windows. Перед початком тестування було підготовлено базу тестових даних у СКБД MS Access, а також створено вхідні сценарії для перевірки основних функціональних можливостей, зокрема шифрування, дешифрування, перегляду балансу та здійснення переказу коштів.

Процес тестування передбачає виконання кількох етапів. Першим таким етапом є функціональне тестування. Сюди входять перевірки правильності обробки введених і виведених даних, а також реакції програми на некоректні вхідні значення. Другим етапом є тестування на стійкість до помилок. Виконується оцінка програмного забезпечення щодо його спроможності відновлювати коректну роботу після виникнення збоїв або помилок.

Також може бути перевірка швидкодії та сумісності. У процесі цього тестування здійснюється перевірка працездатності програми в різних конфігураціях та середовищах, а також виявлення можливих конфліктів чи проблем під час взаємодії з іншими програмними компонентами.

Під час першого етапу функціонального тестування було перевірено стійкість програми до введення некоректних даних користувачем у вікні авторизації. У разі введення неправильного логіна, що не збігається з даними, збереженими в базі, програма виводить повідомлення: «Користувача не знайдено.» (див. рисунок 4.1).

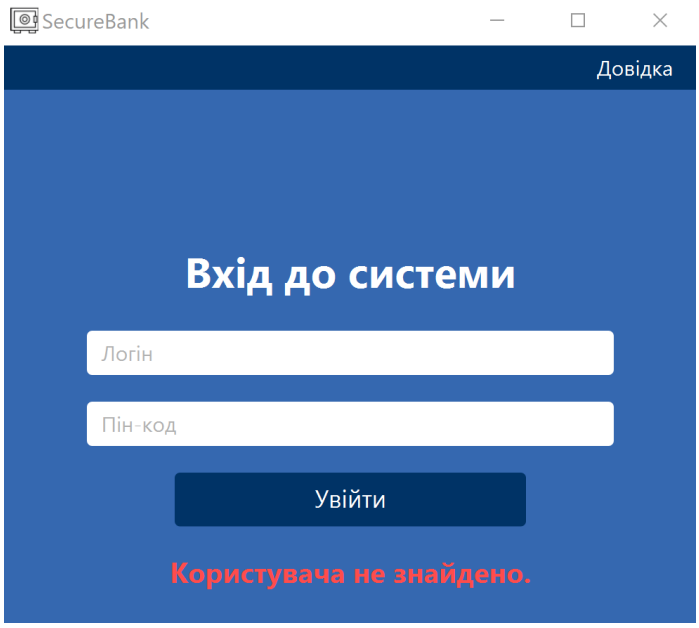

Рисунок 4.1 – Введено неправильний логін користувача

Аналогічно, якщо логін вказано правильно, але ПІН-код є некоректним, програма повідомляє: «Неправильний ПІН-код.» (див. рисунок 4.2). Ці перевірки дозволяють захистити доступ до системи та уникнути несанкціонованого входу, водночас надаючи користувачу чітке пояснення помилки.

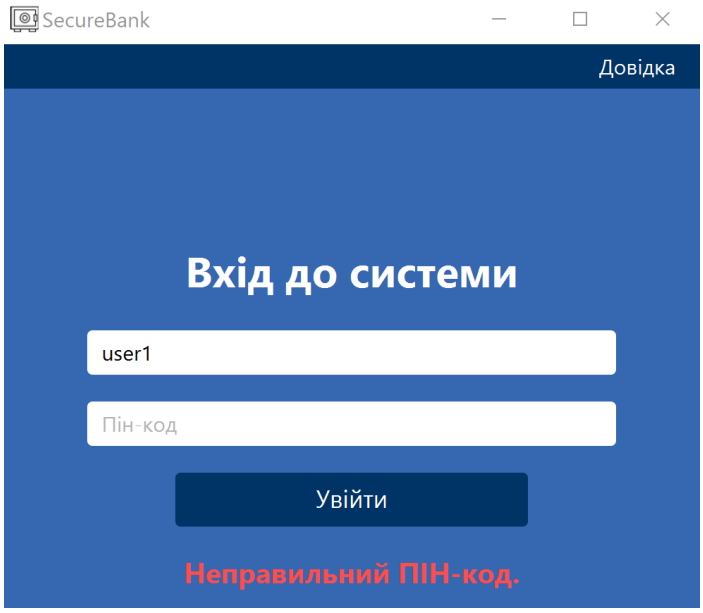

Рисунок 4.2 – Введено неправильний пін-код

Наступним етапом було функціональне тестування вікна переказу коштів. Якщо користувач натискає кнопку «Надіслати», не ввівши номер рахунку отримувача, програма виводить користувачу повідомлення : «Будь ласка, введіть номер рахунку.» (див. рисунок 4.3).

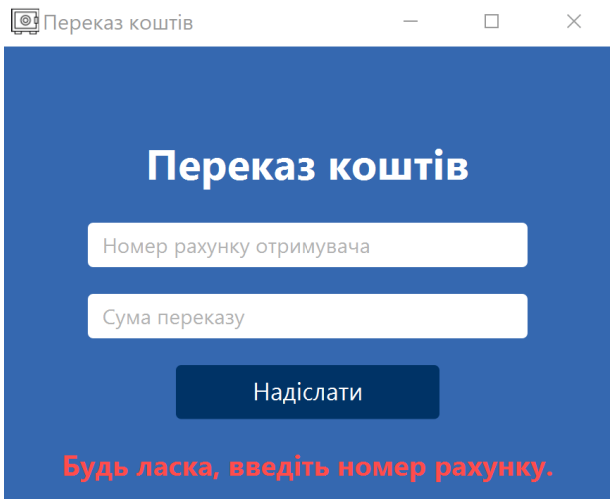

Рисунок 4.3 – Поле для номеру рахунку отримувача порожнє

У разі, коли номер рахунку отримувача збігається з власним номером рахунку користувача, система виводить попередження: «Номер рахунку отримувача співпадає з вашим.» (див. рисунок 4.4).

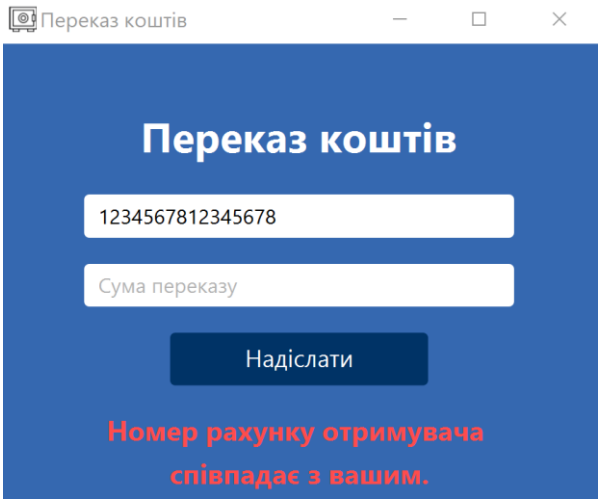

Рисунок 4.4 – Номер рахунку відправника і отримувача збігаються

Також аналогічним чином передбачено перевірку введення суми: якщо поле порожнє – «Будь ласка, введіть суму для переказу.» (див. рисунок 4.5).

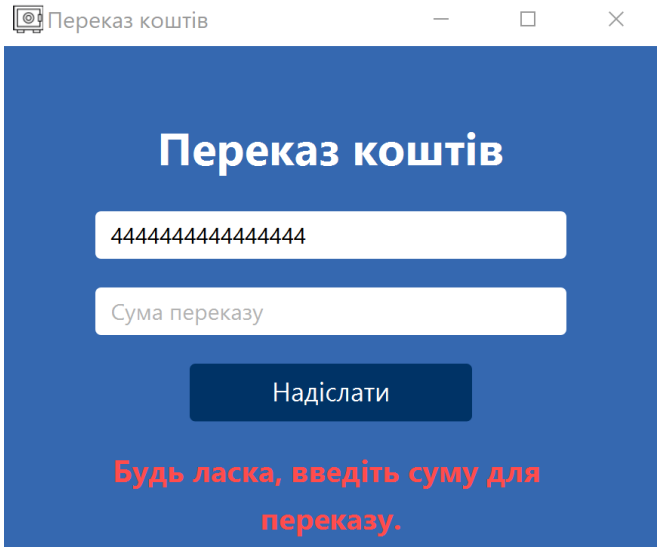

Рисунок 4.5 – Поле для суми переказу порожнє

Якщо введено недопустимий формат, то програма виведе на екран повідомлення: «Сума має бути цілим або дробовим числом (до 2 знаків після крапки).» (див. рисунок 4.6).

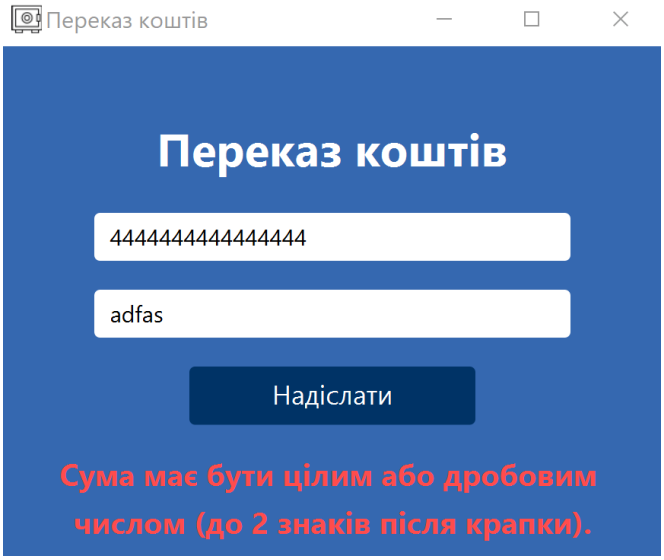

Рисунок 4.6 – Неправильно введено суму переказу

У випадку перевищення доступного балансу, з’явиться повідомлення «На рахунку недостатньо коштів для переказу.» (див. рисунок 4.7). Це дає змогу уникнути помилок в подальших обрахунках.

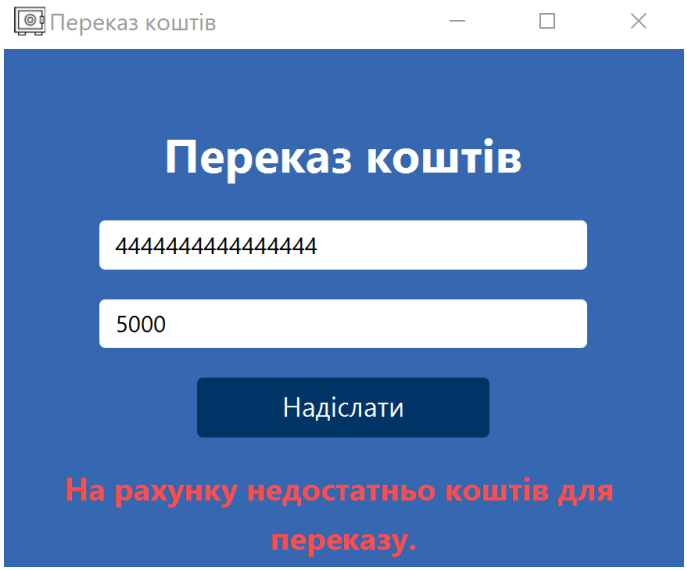

Рисунок 4.7 – Недостатньо коштів для здійснення переказу

Якщо ж вказано неіснуючий номер рахунку, то після наступної перевірки програма повідомляє, що «Отримувача не знайдено.» (див. рисунок 4.8).

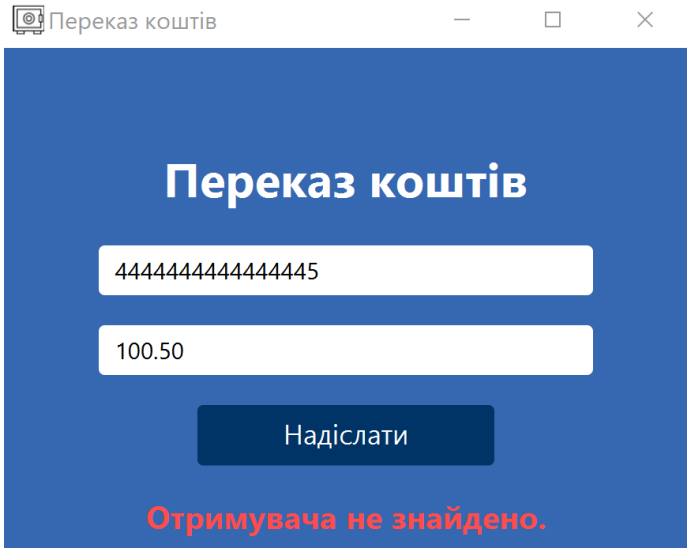

Рисунок 4.8 – Номер рахунку отримувача не знайдено

Окремо було протестовано ситуації, що перевіряють стійкість до помилок. У разі, якщо під час авторизації в базі відсутні дані про логін або ПІН-код, програма виводить ті самі повідомлення, що і при їх некоректному введенні, забезпечуючи послідовність у повідомленнях про помилки. Також було протестовано кабінет користувача: було видалено файл з API-ключем курсів валют, через що програма вивела на екран написи «Н/д» (див. рисунок 4.9).

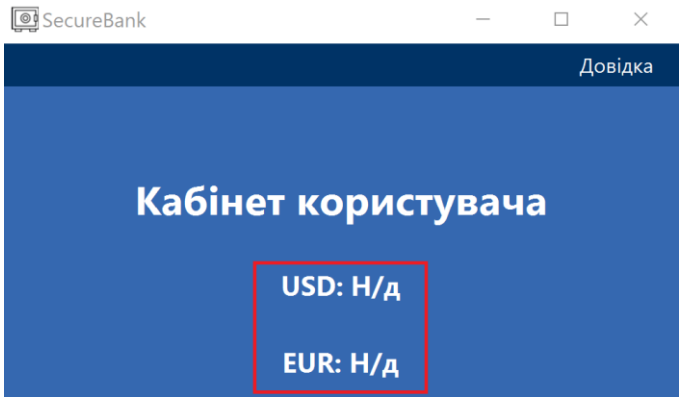

Рисунок 4.9 – Відсутній файл з API-ключем курсів валют

Додатково було протестовано вікно перегляду балансу. У ситуації, коли в базі відсутня інформація про баланс програма відповідно повідомляє: «Баланс не знайдено.» (див. рисунок 4.10).

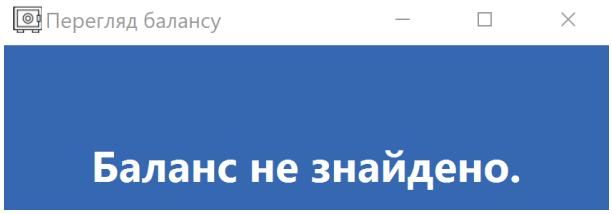

Рисунок 4.10 – Баланс на рахунку відсутній в базі даних

Аналогічним чином, якщо в базі немає номера рахунку користувача, то замість балансу програма пише «Рахунок не знайдено.» (див. рисунок 4.11), не допускаючи подальших дій з некоректними або неповними даними.

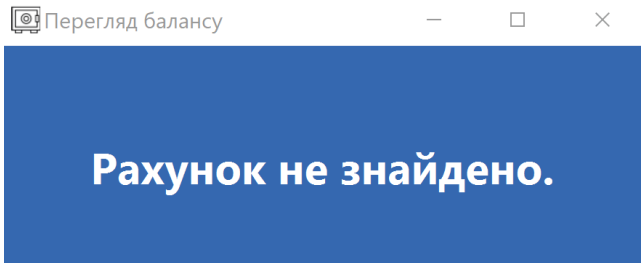

Рисунок 4.11 – Номер рахунку відсутній в базі даних

У вікні переказу коштів також перевірялися випадки, коли у базі відсутні необхідні дані про поточного користувача. Якщо бракує інформації про баланс або номер рахунку, з'являється повідомлення про те, що «Відправника не знайдено.» (див. рисунок 4.12). Це дозволяє уникнути збоїв при формуванні переказу.

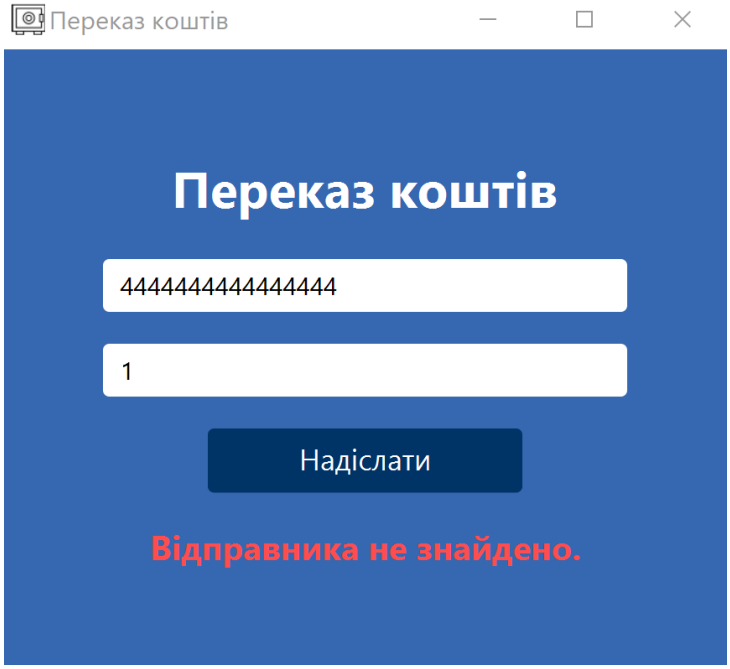

Рисунок 4.12 – Переказ коштів при відсутньому рахунку відправника

Також перевірялися випадки, коли у базі відсутні необхідні дані про транзакції. Так якщо у базі даних немає інформації хоча б про одну з транзакцій, то у вікні історії транзакцій буде повідомлення про те, що у таблиці немає контенту (див. рисунок 4.13).

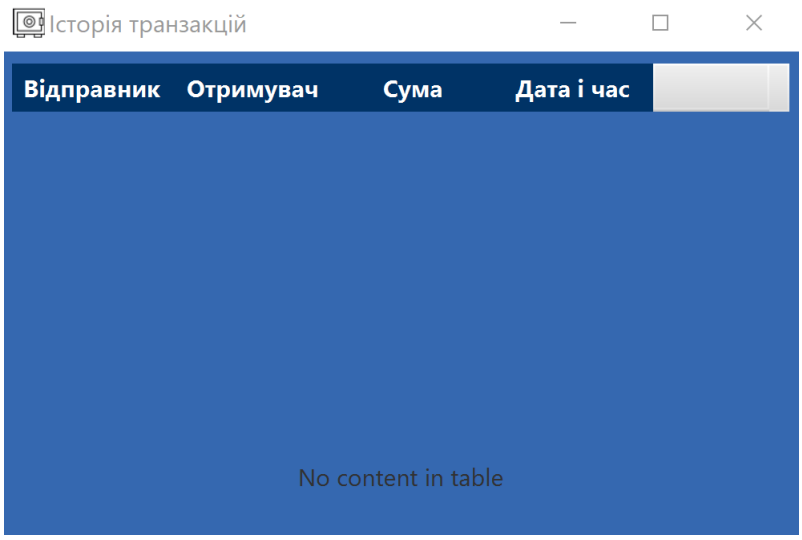

Рисунок 4.13 – Таблиця у вікні історії транзакцій з відсутніми даними

Окрім перевірки обробки помилкових ситуацій, важливо також протестувати основну функціональність системи – переказ коштів між користувачами. Це дає змогу переконатися, що програмне забезпечення коректно виконує головну бізнес-операцію згідно з вимогами. Як показано на рисунку 4.14, у разі успішного завершення переказу система відображає повідомлення: "Переказ успішно виконано.", що свідчить про правильну реалізацію логіки переказу та взаємодію з базою даних.

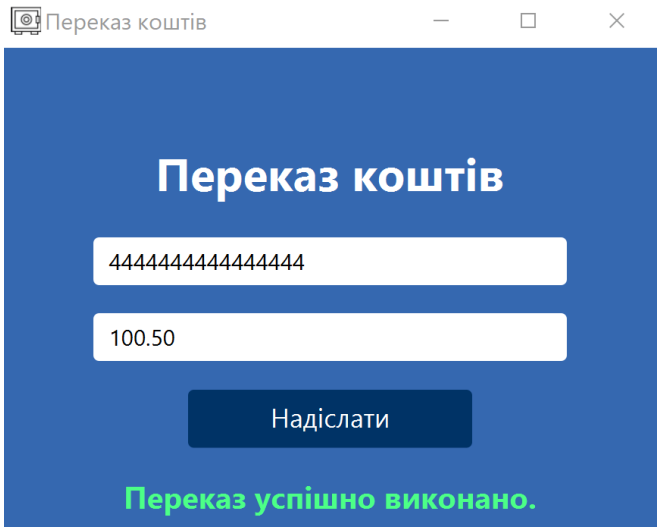

Рисунок 4.14 – Успішний переказ коштів

Також було проведено перевірку швидкодії програмного забезпечення з метою оцінки зручності його використання для кінцевого користувача. За результатами тестування встановлено, що запуск програми триває до 2 секунд, авторизація користувача – до 1 секунди, а виконання операції переказу коштів – до 2 секунд. Інші дії, пов’язані з навігацією по інтерфейсу або переглядом балансу, виконуються практично миттєво. Це свідчить про достатньо високу швидкість реакції системи, що забезпечує комфортну взаємодію користувача з програмою.

Усі перелічені перевірки було записано і оформлено у вигляді таблиці тест-кейсів (див. таблицю 4.1).

Таблиця 4.1 – Тест-кейси

№	**Назва тест-кейсу**	**Вхідні дані**	**Очікуваний результат**	**Результат**
1	Авторизація: неправильний логін	Логін: *пусто*, ПІН: *пусто*.	Повідомлення: «Користувача не знайдено.»	Пройдено
2	Авторизація: неправильний ПІН-код	Логін: «user1», ПІН: *пусто*.	Повідомлення: «Неправильний ПІН-код.»	Пройдено
3	Переказ: порожнє поле номера рахунку	Рахунок: *пусто* Сума: *пусто*.	Повідомлення: «Будь ласка, введіть номер рахунку.»	Пройдено
4	Переказ самому собі	Рахунок: *рахунок відправника*, Сума: *пусто*.	Повідомлення: «Номер рахунку отримувача співпадає з вашим.»	Пройдено

Продовження таблиці 4.1

№	Назва тест-кейсу	Вхідні дані	Очікуваний результат	Результат
5	Переказ: порожнє поле суми	Рахунок: 4444 4444 4444 4444, Сума: *пусто*.	Повідомлення: «Будь ласка, введіть суму для переказу.»	Пройдено
6	Переказ: неправильний формат суми	Рахунок: 4444 4444 4444 4444, Сума: «adfas».	Повідомлення: «Сума має бути цілим...»	Пройдено
7	Переказ: недостатньо коштів	Рахунок: 4444 4444 4444 4444, Сума: 5000.	Повідомлення: «На рахунку недостатньо коштів...»	Пройдено
8	Переказ: отримувача не знайдено	Рахунок: 4444 4444 4444 4445, Сума: 5000.	Повідомлення: «Отримувача не знайдено.»	Пройдено
9	Перевірка роботи без API-ключа курсів валют	Папка проєкту: відсутній файл з API-ключем	Повідомлення: «USD: Н/д» і « EUR: Н/д»	Пройдено
10	«Дірява» БД (авторизація)	БД: порожній Логін/ПІН	Як і при неправильних даних	Пройдено
11	«Дірява» БД (перегляд балансу)	БД: порожній рахунок або баланс	Повідомлення: «Баланс/Рахунок не знайдено.»	Пройдено
12	«Дірява» БД (переказ)	БД: порожній рахунок або баланс	Повідомлення: «Відправника не знайдено.»	Пройдено
13	«Дірява» БД (історія транзакцій)	БД: порожні будь-які дані про транзакцію	Пуста таблиця транзакцій	Пройдено

Продовження таблиці 4.1

№	Назва тест-кейсу	Вхідні дані	Очікуваний результат	Результат
14	Успішний переказ коштів	Рахунок: 4444 4444 4444 4444, Сума: 100.50.	Повідомлення: «Переказ успішно виконано.»	Пройдено
15	Продуктивність: запуск програми	-	Час запуску менше 2 секунд	Пройдено
16	Продуктивність: авторизація	-	Час відповіді менше 1 секунди	Пройдено
17	Продуктивність: переказ коштів	-	Час відповіді менше 2 секунд	Пройдено
18	Продуктивність: інші дії	-	Миттєва реакція інтерфейсу	Пройдено

Окрім даної таблиці, завдяки автоматичному логуванню дій системи, для зручного представлення результатів тестування, було зроблено і збережено в окремий файл записи використання додатку користувачем (див. рисунок 4.15).

log.txt: Блокнот

Файл Редагування Формат Вигляд Довідка

```
[2025-06-08T19:38:09.468573500] [INFO] JWT недійсний або не знайдений. Завантажено вікно входу.
[2025-06-08T19:38:14.086690500] [INFO] Відкрито довідкове вікно Допомога.
[2025-06-08T19:38:19.777519500] [INFO] З'єднання з базою даних успішно встановлено.
[2025-06-08T19:38:19.805444200] [INFO] Успішно здійснено гешування даних.
[2025-06-08T19:38:19.806442500] [WARNING] Неправильний ПІН-код.
[2025-06-08T19:38:22.891565800] [INFO] З'єднання з базою даних успішно встановлено.
[2025-06-08T19:38:22.895554600] [INFO] Успішно здійснено гешування даних.
[2025-06-08T19:38:22.895554600] [INFO] JWT створено для користувача: user1.
[2025-06-08T19:38:23.071617] [INFO] JWT збережено.
[2025-06-08T19:38:23.114431800] [INFO] Завантажено API-ключ курсу валют.
[2025-06-08T19:38:23.154324500] [INFO] Здійснено вхід у систему.
[2025-06-08T19:38:23.685723700] [INFO] Отримано курс валют.
[2025-06-08T19:38:25.365679500] [INFO] З'єднання з базою даних успішно встановлено.
[2025-06-08T19:38:25.443437500] [INFO] Ключі завантажено з файлу.
[2025-06-08T19:38:25.444434900] [INFO] Відбулося розшифрування даних.
[2025-06-08T19:38:27.765043900] [INFO] JWT токен видалено.
[2025-06-08T19:38:27.793968100] [INFO] Здійснено вихід із системи.
```

Рисунок 4.15 – Збережені у файлі логи дій системи

4.2 Розробка інструкції користувача

Інструкція користувача містить опис основних вимог до апаратного забезпечення, порядку встановлення та використання розроблюваного програмного продукту [30]. У таблиці 4.2 наведено мінімальні та рекомендовані технічні характеристики, необхідні для коректної роботи програмного забезпечення.

Таблиця 4.2 – Вимоги до апаратного забезпечення

Вимоги до конфігурування	**Рекомендовано**	**Мінімально**
Процесор	Intel Core i5 або AMD Ryzen 5	Intel Core i3 або AMD Ryzen 3
Оперативна пам’ять	8 ГБ RAM	4 ГБ RAM
Вільний простір на диску	500 МБ	200 МБ
Відеокарта	Вбудована, сумісна з JavaFX	Вбудована, підтримка JavaFX
Операційна система	Windows 10 / 11, macOS 11 або новіші	Windows 7 / macOS 10.13
Монітор	Роздільна здатність 1920×1080, діагональ 17”	Роздільна здатність 1280×720, діагональ 14”

Після встановлення та запуску програмного забезпечення, користувач потрапляє на вікно авторизації. Для входу необхідно ввести Логін та ПІН-код, які попередньо зберігаються в базі даних. У разі успішної авторизації відкривається головне вікно або ж кабінет користувача, де доступні функції перегляду балансу, здійснення переказу коштів, перегляду історії транзакцій і виходу з системи, а також представлено поточний курс валют. При наступних

запусках, за умови що користувач не вийшов із системи, буде одразу завантажуватися вікно кабінету користувача.

Для перегляду балансу необхідно натиснути відповідну кнопку, після чого програма автоматично виконує пошук зашифрованого балансу користувача, розшифровує його та відображає на інформацію екрані. У разі виникнення помилки, користувач побачить відповідне повідомлення.

Функція переказу коштів дозволяє здійснити переказ іншому користувачу, ввівши його номер рахунку та суму. У разі успішної операції на екрані з’являється повідомлення: «Переказ успішно виконано». Якщо виникають помилки (некоректні дані, недостатній баланс, відсутній користувач у базі), програма повідомляє про це відповідним текстом.

Щоб переглянути історію транзакцій потрібно натиснути на відповідну кнопку, після чого відкриється додаткове вікно і відбувається завантаження таблиці з транзакціями. У дані й таблиці відображаються дані про логіни учасників транзакцій, суму переказу та дату і час транзакції.

Користувач також може звернутися до пунктів «Довідка» або «Про програму», щоб ознайомитися з інформацією про розробника або отримати коротку інструкцію з використання. Вихід із системи виконується натисканням на кнопку «Вийти».

4.3 Висновки

У четвертому розділі було проведено тестування програмних модулів програмного застосунку. Перевірено реакцію системи на помилкові дії користувача, некоректні або неповні вхідні дані, а також на нестандартні ситуації, пов’язані з обробкою збереженої інформації. Здійснено аналіз швидкодії програмного забезпечення, результати якого підтверджують ефективну обробку основних операцій у межах прийнятного часу. Окрім того, було розроблено інструкцію користувача, що містить опис технічних вимог, послідовність дій при роботі з програмою та пояснення ключових функцій інтерфейсу.

ВИСНОВКИ

У рамках бакалаврської кваліфікаційної роботи було розроблено програмний застосунок для банківської системи з використанням гомоморфного шифрування, що забезпечує конфіденційність даних клієнтів навіть під час обробки. Для розробки програмного продукту використовувалося середовище програмування IntelliJ IDEA, мова Java, а також технології JavaFX і СКБД MS Access. Реалізація бакалаврської роботи відповідає методичним вказівкам [31].

У процесі виконання роботи було здійснено аналіз сучасних методів захисту даних, зокрема криптографічних систем, та обґрунтовано вибір криптосистеми Paillier для реалізації гомоморфного шифрування. Проведено огляд аналогічних рішень і виявлено їхні недоліки, що дало змогу сформулювати технічне завдання даного проєкту.

У межах бакалаврської роботи було виконано такі завдання:

- аналіз основних видів гомоморфного шифрування та вибір криптосистеми для подальшої розробки;
- розробка програмних модулів для шифрування, зберігання і обробки даних користувача;
- реалізація програмного продукту з практичним використанням розроблених модулів;
- розробка зручного та адаптивного графічного інтерфейсу програмного забезпечення для роботи з модулями;
- проведення тестування розробленого функціоналу.

Було розроблено модель системи з використанням UML діаграм, а також блок-схеми алгоритмів зберігання Paillier-ключів та гомоморфного віднімання. Окрім цього було розроблено і описано основний алгоритм роботи програмного застосунку і алгоритм здійснення переказу коштів.

Результати тестування підтвердили працездатність ПЗ, його стійкість до помилкових дій користувача, а також ефективність використання гомоморфного шифрування для забезпечення конфіденційності даних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ENISA. Cybersecurity for SMEs – Challenges and Recommendations. European Union Agency for Cybersecurity, 2021. – 62 p.

2. Чернюк С.І. Можливості та обмеження криптографічних методів шифрування в сучасних іт-системах / С.І. Чернюк, О.Я. Стахов // Молодь в науці: дослідження, проблеми, перспективи (МН-2025). [Електронний ресурс]. – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/view/24962 (дата звернення: 25.03.25).

3. Gorantala S., Springer R., Gipson B. Unlocking the Potential of Fully Homomorphic Encryption. Communications of the ACM. 2023. Vol. 66, No. 5 P. 75-81.

4. Microsoft SEAL is an easy-to-use and powerful homomorphic encryption library. [Електронний ресурс]. – Режим доступу: https://github.com/microsoft/SEAL (дата звернення: 26.03.25).

5. Types of Homomorphic Encryption - IEEE Digital Privacy. [Електронний ресурс]. – Режим доступу: https://digitalprivacy.ieee.org/publications/topics/types-of-homomorphic-encryption (дата звернення: 27.03.25).

6. Paillier Homomorphic Encryption: A Comprehensive Guide. [Електронний ресурс]. – Режим доступу: https://medium.com/@aannkkiittaa/paillier-homomorphic-encryption-a-comprehensive-guide-ce7fe2c245bd (дата звернення: 28.03.25).

7. Building a Secure Voting System with Homomorphic Encryption using C++ & HElib. [Електронний ресурс]. – Режим доступу: https://medium.com/@jamie.brian.gilchrist/building-a-secure-voting-system-with-homomorphic-encryption-using-c-helib-cd8c27d79dfe (дата звернення: 29.03.25).

8. razi-rais/homomorphic-encryption: This repo contains code/learning resources related to Homomorphic Encryption. [Електронний ресурс]. – Режим

доступу: https://github.com/razi-rais/homomorphic-encryption (дата звернення: 29.03.25).

9. narger-ef/LowMemoryFHEResNet20. [Електронний ресурс]. – Режим доступу: https://github.com/narger-ef/LowMemoryFHEResNet20?tab=readme-ov-file (дата звернення: 30.03.25).

10. UML діаграми, їхні основні типи та процес розроблення. [Електронний ресурс]. – Режим доступу: https://foxminded.ua/uml-diagramy/ (дата звернення: 31.03.25).

11. JSON Web Token Introduction - jwt.io. [Електронний ресурс]. – Режим доступу: https://jwt.io/introduction (дата звернення: 04.04.25).

12. draw.io. [Електронний ресурс]. – Режим доступу: https://www.drawio.com/ (дата звернення: 05.04.25).

13. SHA-256 Algorithm: Characteristics, Steps, and Applications. [Електронний ресурс]. – Режим доступу: https://www.simplilearn.com/tutorials/cyber-security-tutorial/sha-256-algorithm (дата звернення: 06.04.25).

14. What Is AES? How Does It Work? [Електронний ресурс]. – Режим доступу: https://www.encryptionconsulting.com/education-center/what-is-aes/ (дата звернення: 07.04.25).

15. Paillier Subtraction with Kryptology. [Електронний ресурс]. – Режим доступу: https://asecuritysite.com/golang/pail02 (дата звернення: 08.04.25).

16. Lagrange's theorem (group theory). [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Lagrange%27s_theorem_(group_theory) (дата звернення: 09.04.25).

17. Java Programming Language. [Електронний ресурс]. – Режим доступу: https://docs.oracle.com/javase/8/docs/technotes/guides/language/ (дата звернення: 11.04.25).

18. IntelliJ IDEA – the IDE for Pro Java and Kotlin Development. [Електронний ресурс]. – Режим доступу: https://www.jetbrains.com/idea/ (дата звернення: 12.04.25).

19. JavaFX. [Електронний ресурс]. – Режим доступу: https://openjfx.io/ (дата звернення: 13.04.25).

20. What Is GUI? Graphical User Interfaces, Explained. [Електронний ресурс]. – Режим доступу: https://blog.hubspot.com/website/what-is-gui (дата звернення: 14.04.25).

21. JavaFX FXML. [Електронний ресурс]. – Режим доступу: https://jenkov.com/tutorials/javafx/fxml.html (дата звернення: 15.04.25).

22. CSS: Cascading Style Sheets - MDN Web Docs. [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/CSS (дата звернення: 16.04.25).

23. JavaFX Scene Builder Information. [Електронний ресурс]. – Режим доступу: https://www.oracle.com/java/technologies/javase/javafxscenebuilder-info.html (дата звернення: 17.04.25).

24. Microsoft Access. [Електронний ресурс]. – Режим доступу: https://www.microsoft.com/uk-ua/microsoft-365/access (дата звернення: 18.04.25).

25. UCanAccess-A Pure Java JDBC Driver for Access. [Електронний ресурс]. – Режим доступу: https://ucanaccess.sourceforge.net/site.html (дата звернення: 18.04.25).

26. What is Hash-based Message Authentication Code (HMAC)? [Електронний ресурс]. – Режим доступу: https://www.techtarget.com/searchsecurity/definition/Hash-based-Message-Authentication-Code-HMAC (дата звернення: 19.04.25).

27. ExchangeRate-API - Free & Pro Currency Converter API. [Електронний ресурс]. – Режим доступу: https://www.exchangerate-api.com/ (дата звернення: 20.04.25).

28. Base64 Encoding: What Is It? How Does It Work? [Електронний ресурс]. – Режим доступу: https://builtin.com/software-engineering-perspectives/base64-encoding (дата звернення: 20.04.25).

29. Samaroo A., Thompson G., Morgan P., Williams P., Hambling B. Software Testing: An ISTQB-BCS Certified Tester Foundation guide. 4th Edition. BCS, The

Chartered Institute for IT, 2019 – 296p.

30. Що таке інструкція користувача? – Вказівки професіоналів. [Електронний ресурс]. – Режим доступу: https://smih.oda.v.ua/vzaiemodiya/shho-take-instrukciya-koristuvacha.html (дата звернення: 21.04.25).

31. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів студентів спеціальності 121 "Інженерія програмного забезпечення" / Уклад. О. Н. Романюк, Г. О. Черноволик – Вінниця: ВНТУ, 2022. – 51с.

ДОДАТКИ

ДОДАТОК А

Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ

д.т.н., проф. О. Н. Романюк

"__25__" ____березня____ 2025 р.

Технічне завдання
на бакалаврську кваліфікаційну роботу
«Програмне забезпечення для зберігання та обробки даних за допомогою гомоморфного шифрування»
за спеціальністю 121 – Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:

_____PhD, ст. вик. каф. ПЗ Стахов О.Я.

"__24__" ____березня____ 2025 р.

Виконав:

_____студент гр. 2ПІ-21б Чернюк І.М.

"__24__" березня 2025 р.

Вінниця – 2025 року

1. Найменування та галузь застосування.

Бакалаврська кваліфікаційна робота: «Програмне забезпечення для зберігання та обробки даних за допомогою гомоморфного шифрування».

Галузь застосування – захист інформації.

2. Підстава для розробки.

Підставою для виконання бакалаврської кваліфікаційної роботи (БКР) є індивідуальне завдання на БКР та наказ №97 від 20 березня 2025 р. ректора ВНТУ про закріплення тем БКР.

3. Мета та призначення розробки.

Метою роботи є підвищення конфіденційності даних користувачів завдяки використанню гомоморфного шифрування, що дозволяє здійснювати обчислення над зашифрованими даними без їхнього розшифрування.

Призначення роботи – створення програмних компонентів, які реалізують повноцінну систему для зберігання та обробки зашифрованих даних користувача.

4. Вихідні дані для проведення НДР.

Перелік основних джерел, на основі яких буде виконуватись БКР:

1. Gorantala S., Springer R., Gipson B. Unlocking the Potential of Fully Homomorphic Encryption. Communications of the ACM. 2023. Vol.66, No.5 P.75-81.

2. Paillier Homomorphic Encryption: A Comprehensive Guide. [Електронний ресурс]. – Режим доступу: https://medium.com/@aannkkiittaa/paillier-homomorphic-encryption-a-comprehensive-guide-ce7fe2c245bd.

3. Paillier Subtraction with Kryptology. [Електронний ресурс]. – Режим доступу: https://asecuritysite.com/golang/pail02.

5. Технічні вимоги.

Тип програми – десктопний кросплатформний застосунок; модель

розробки – ітеративна; засоби організації даних програми – Maven, JavaFX, SQL; вхідні дані: логін користувача, пін-код, номер рахунку, сума переказу; вихідні дані: оновлений зашифрований баланс, історія транзакцій; середовище розробки – Intellij IDEA; мова програмування – Java.

6. Конструктивні вимоги.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

– пояснювальна записка до БКР;

– технічне завдання;

– лістинг програми.

8. Вимоги до рівня уніфікації та стандартизації.

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

Ч.ч.	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз предметної області та постановка задач розробки	25.03.25 – 30.03.25
2	Розробка моделі та алгоритмів роботи програмного застосунку	31.03.25 – 10.04.25
3	Розробка програмних модулів реалізації системи зберігання та обробки даних за допомогою гомоморфного шифрування	11.04.25 – 20.04.24
4	Тестування програмного застосунку	21.04.25 – 27.04.25
5	Оформлення матеріалів до захисту БКР	28.04.25 – 30.05.25

10. Порядок контролю та прийняття.

Виконання етапів бакалаврської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи. Прийняття бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

ДОДАТОК Б
ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: «Програмне забезпечення для зберігання та обробки даних за допомогою гомоморфного шифрування»

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра програмного забезпечення, ФІТКІ
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 5,8 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

- ■ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту
- ☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.
- ☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

_____ _____
(прізвище, ініціали, посада) (підпис)

_____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Черноволик Г. О.

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Стахов О.Я.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Чернюк С.І.
(підпис) (прізвище, ініціали)

ДОДАТОК В

Лістинг програми

Модуль «Main»:

```
package org.example.fxproject;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.stage.Stage;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Objects;

import org.example.fxproject.controllers.Account;
import org.example.fxproject.util.AppLogger;
import org.example.fxproject.util.DBManager;
import org.example.fxproject.util.JwtUtil;

public class Main extends Application {
    @Override
    public void start(Stage stage) throws Exception {
        Parent root = getRoot();
```

```
        Scene scene = new Scene(root);

scene.getStylesheets().add(Objects.requireNonNull(getClass().getResource("/style.css")).toExternal
Form());

        stage.setTitle("SecureBank");

        stage.getIcons().add(new
Image(Objects.requireNonNull(getClass().getResourceAsStream("/icon.png"))));

        stage.setScene(scene);

        stage.show();

    }


    private Parent getRoot() throws Exception {

        String token = JwtUtil.readToken();


        // Якщо токен відсутній або недійсний — завантажується вікно входу

        if (token == null || !JwtUtil.isTokenValid(token)) {

            AppLogger.log("JWT недійсний або не знайдений. Завантажено вікно входу.");

            return
FXMLLoader.load(Objects.requireNonNull(getClass().getResource("/org/example/fxproject/login.f
xml")));

        }


        // Отримуємо номер рахунку

        String accNum = getAccNum(token);

        if (accNum == null) {

            return
FXMLLoader.load(Objects.requireNonNull(getClass().getResource("/org/example/fxproject/login.f
xml")));

        }


        FXMLLoader loader = new
FXMLLoader(getClass().getResource("/org/example/fxproject/account.fxml"));
```

```
        Parent root = loader.load();
        Account controller = loader.getController();

        controller.setAccountNumber(accNum);
        return root;
    }

    private String getAccNum(String token) throws SQLException {
        // Отримуємо логін із токена
        String login = JwtUtil.getLoginFromToken(token);

        // Підключення до бази даних
        Connection conn = DBManager.connect();
        if (conn == null) {
            return null;
        }

        PreparedStatement ps = conn.prepareStatement("SELECT * FROM Клієнти WHERE [Логін]
= ?");
        ps.setString(1, login);
        ResultSet rs = ps.executeQuery();

        if (!rs.next()) {
            AppLogger.log(AppLogger.LogLevel.ERROR, "Користувача не знайдено.");
            return null;
        }

        return rs.getString("Номер рахунку");
```

```
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

Модуль «Login»:

```
package org.example.fxproject.controllers;

import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Node;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.stage.Stage;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.Objects;

import org.example.fxproject.util.DBManager;
import org.example.fxproject.util.DialogUtil;
import org.example.fxproject.crypto.HashFunc;
import org.example.fxproject.util.JwtUtil;

import static org.example.fxproject.util.AppLogger.*;

public class Login {
    @FXML private TextField loginField;
```

```
    @FXML private PasswordField pinField;
    @FXML private Label statusLabel;

    private String accountNumber;

    @FXML
    private void handleAboutClick() {
        DialogUtil.showInfo("Про застосунок", """
                SecureBank - це демонстраційний
                банківський застосунок із використанням
                гомоморфного шифрування.
                Розробник: Чернюк Сергій Іванович 2ПІ-21Б""");
    }

    @FXML
    private void handleHelpClick() {
        DialogUtil.showInfo("Допомога", """
                1. Перш за все, щоб використовувати даний застосунок Вам необхідно увійти.
                Для цього введіть у відповіних полях ваші логін та пін-код.
                2. Для того щоб перегляти свій баланс натисніть на кнопку 'Переглянути баланс'.
                Після цього можете закрити дане вікно натиснувши на кнопку 'X'.
                3. Щоб здійснити переказ коштів натисніть кнопку 'Переказ коштів' і введіть у \
                відповідні поля номер рахунку отримувача та суму переказу.
                4. Щоб переглянути історію операцій натисніть кнопку 'Історія транзакцій'.
                5. Якщо у Вас виникли додаткові питання чи проблеми з додатком, зверніться до служби підтримки.""");
    }

    @FXML
    private void handleLogin(javafx.event.ActionEvent event) throws Exception {
        String login = loginField.getText();
        String pin = pinField.getText();

        Connection conn = DBManager.connect();
```

```
        if (conn == null) {
            return;
        }

        PreparedStatement ps = conn.prepareStatement("SELECT * FROM Клієнти WHERE [Логін]
= ?");
        ps.setString(1, login);

        ResultSet rs = ps.executeQuery();
        if (!rs.next()) {
            setStatus("Користувача не знайдено.");
            return;
        }

        String storedPin = rs.getString("Пін-код");
        if (!HashFunc.compare(pin, storedPin)) {
            setStatus("Неправильний ПІН-код.");
            return;
        }

        accountNumber = rs.getString("Номер рахунку");

        // Зберігаємо JWT
        String token = JwtUtil.createToken(login);
        JwtUtil.saveToken(token);

        loadAccountWindow(event);
    }

    private void loadAccountWindow(javafx.event.ActionEvent event) throws Exception {
        FXMLLoader loader = new
FXMLLoader(getClass().getResource("/org/example/fxproject/account.fxml"));
        Parent root = loader.load();
```

```
        // Отримуємо контролер і передаємо дані
        Account aCon = loader.getController();
        aCon.setAccountNumber(accountNumber);

        // Встановлюємо нову сцену
        Stage stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
        Scene scene = new Scene(root);

scene.getStylesheets().add(Objects.requireNonNull(getClass().getResource("/style.css")).toExternal
Form());
        stage.setScene(scene);
        stage.show();
        log("Здійснено вхід у систему.");
    }

    private void setStatus(String message) {
        statusLabel.setText(message);
        log(LogLevel.WARNING, message);
    }
}
```

Модуль «Account»:

```
package org.example.fxproject.controllers;

import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.scene.Node;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.image.Image;
import javafx.stage.Stage;
```

```
import java.io.IOException;
import java.net.URL;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Objects;
import java.util.ResourceBundle;

import org.example.fxproject.crypto.KeyManager;
import org.example.fxproject.util.*;

public class Account implements Initializable {
    @FXML
    private Label usdRateLabel;
    @FXML
    private Label eurRateLabel;

    private String accountNumber;

    public void setAccountNumber(String accountNumber) {
        this.accountNumber = accountNumber;
    }

    @FXML
    private void handleAboutClick() {
        DialogUtil.showInfo("Про застосунок", """
                SecureBank - це демонстраційний
                банківський застосунок із використанням
                гомоморфного шифрування.
                Розробник: Чернюк Сергій Іванович 2ПІ-21Б""");
    }
```

```
    @FXML
    private void handleHelpClick() {
        DialogUtil.showInfo("Допомога", """
                1. Перш за все, щоб використовувати даний застосунок Вам необхідно увійти.
                Для цього введіть у відповіних полях ваші логін та пін-код.
                2. Для того щоб перегляти свій баланс натисніть на кнопку 'Переглянути баланс'.
                Після цього можете закрити дане вікно натиснувши на кнопку 'X'.
                3. Щоб здійснити переказ коштів натисніть кнопку 'Переказ коштів' і введіть у \
                відповідні поля номер рахунку отримувача та суму переказу.
                4. Щоб переглянути історію операцій натисніть кнопку 'Історія транзакцій'.
            5. Якщо у Вас виникли додаткові питання чи проблеми з додатком, зверніться до служби підтримки.""");
    }

    @FXML
    private void handleViewBalanceClick() throws Exception {
        FXMLLoader loader = new FXMLLoader(getClass().getResource("/org/example/fxproject/viewBalance.fxml"));
        Parent root = loader.load();

        // Отримуємо контролер ViewBalanceController
        ViewBalance vbCon = loader.getController();
        vbCon.setAccountNumber(accountNumber);
        vbCon.getEncryptedBalance();

        Stage stage = new Stage();
        stage.setTitle("Перегляд балансу");
        stage.getIcons().add(new Image(Objects.requireNonNull(getClass().getResourceAsStream("/icon.png"))));
        Scene scene = new Scene(root);

        scene.getStylesheets().add(Objects.requireNonNull(getClass().getResource("/style.css")).toExternalForm());
        stage.setScene(scene);
        stage.show();
```

```
    }

    @FXML
    private void handleTransferClick() throws IOException {
        FXMLLoader loader = new
FXMLLoader(getClass().getResource("/org/example/fxproject/transfer.fxml"));
        Parent root = loader.load();

        // Отримуємо контролер
        Transfer tCon = loader.getController();
        tCon.setSenAccNum(accountNumber);

        Stage stage = new Stage();
        stage.setTitle("Переказ коштів");
        stage.getIcons().add(new
Image(Objects.requireNonNull(getClass().getResourceAsStream("/icon.png"))));
        Scene scene = new Scene(root);

scene.getStylesheets().add(Objects.requireNonNull(getClass().getResource("/style.css")).toExternal
Form());
        stage.setScene(scene);
        stage.show();
    }

    private String getLoginByNum(String accNum) throws SQLException {
        Connection conn = DBManager.connect();
        if (conn == null) {
            return null;
        }

        String sql = "SELECT Логін FROM Клієнти WHERE [Номер рахунку] = ?";
        PreparedStatement ps = conn.prepareStatement(sql);
        ps.setString(1, accNum);
        ResultSet rs = ps.executeQuery();
        if (rs.next()) {
```

```
            return rs.getString("Логін");
        }

        return null;
    }

    @FXML
    private void handleTransactionHistoryClick() throws Exception {
        FXMLLoader loader = new
FXMLLoader(getClass().getResource("/org/example/fxproject/transactionHistory.fxml"));
        Parent root = loader.load();

        // Отримуємо контролер і передаємо дані
        TransactionHistory thCon = loader.getController();
        thCon.initialize(getLoginByNum(accountNumber), KeyManager.getKeys());

        Stage stage = new Stage();
        stage.setTitle("Історія транзакцій");
        stage.getIcons().add(new
Image(Objects.requireNonNull(getClass().getResourceAsStream("/icon.png"))));
        Scene scene = new Scene(root);

scene.getStylesheets().add(Objects.requireNonNull(getClass().getResource("/style.css")).toExternal
Form());
        stage.setScene(scene);
        stage.show();
    }

    @FXML
    private void handleLogoutClick(javafx.event.ActionEvent event) throws IOException {
        JwtUtil.deleteToken();

        // Завантажуємо FXML вікна входу
        FXMLLoader loader = new
FXMLLoader(getClass().getResource("/org/example/fxproject/login.fxml"));
```

```
        Parent root = loader.load();

        // Встановлюємо нову сцену
        Stage stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
        Scene scene = new Scene(root);

scene.getStylesheets().add(Objects.requireNonNull(getClass().getResource("/style.css")).toExternal
Form());
        stage.setScene(scene);
        stage.show();
        AppLogger.log("Здійснено вихід із системи.");
    }

    @Override
    public void initialize(URL location, ResourceBundle resources) {
        ExRateLoader.loadExchangeRates(usdRateLabel, eurRateLabel);
    }
}
```

Модуль «ViewBalance»:

```
package org.example.fxproject.controllers;

import javafx.fxml.FXML;
import javafx.scene.control.Label;
import org.example.fxproject.crypto.Paillier;

import java.math.BigDecimal;
import java.math.BigInteger;
import java.math.RoundingMode;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

import org.example.fxproject.util.DBManager;
```

```
import org.example.fxproject.crypto.KeyManager;
import static org.example.fxproject.util.AppLogger.*;

public class ViewBalance {
    @FXML
    private Label balanceLabel;

    private String accountNumber;

    public void setAccountNumber(String accountNumber) {
        this.accountNumber = accountNumber;
    }

    public void getEncryptedBalance() throws Exception {
        Connection conn = DBManager.connect();
        if (conn == null) {
            return;
        }

        PreparedStatement ps = conn.prepareStatement("SELECT * FROM Клієнти WHERE [Номер рахунку] = ?");
        ps.setString(1, accountNumber);

        ResultSet rs = ps.executeQuery();

        // Перевірка чи є рахунок користувача
        if (!rs.next()) {
            setBalanceLabel("Рахунок не знайдено.", true);
            return;
        }

        String encBalance = rs.getString("Баланс");

        // Перевірка чи є баланс користувача
```

```
        if (encBalance == null) {
            setBalanceLabel("Баланс не знайдено.", true);
            return;
        }

        displayBalance(encBalance);
    }

    private void displayBalance(String encryptedBalance) throws Exception {
        BigDecimal SCALE = new BigDecimal(100);
        Paillier paillier = KeyManager.getKeys();

        BigInteger decryptedBalance = paillier.decrypt(new BigInteger(encryptedBalance));
        BigDecimal realBalance = new BigDecimal(decryptedBalance).divide(SCALE, 2,
RoundingMode.HALF_UP);

        setBalanceLabel(String.format("Ваш баланс: %.2f грн", realBalance), false);
    }

    private void setBalanceLabel(String text, boolean error) {
        balanceLabel.setText(text);
        if (error) {
            log(LogLevel.ERROR, text);
        }
    }
}
```

Модуль «Transfer»:

```
package org.example.fxproject.controllers;

import javafx.fxml.FXML;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
```

```
import java.math.BigDecimal;
import java.math.BigInteger;
import java.math.RoundingMode;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Timestamp;
import java.time.LocalDateTime;

import org.example.fxproject.crypto.Paillier;
import org.example.fxproject.util.DBManager;
import org.example.fxproject.crypto.HashFunc;
import org.example.fxproject.crypto.KeyManager;
import static org.example.fxproject.util.AppLogger.*;

public class Transfer {
    @FXML private TextField receiverField;
    @FXML private TextField amountField;
    @FXML private Label statusLabel;

    private String senAccNum;
    private final BigDecimal SCALE = new BigDecimal(100);

    public void setSenAccNum(String senAccNum) {
        this.senAccNum = senAccNum;
    }

    @FXML
    private void handleSendClick() throws Exception {
        String accountInput = receiverField.getText();
        String amountInput = amountField.getText();
```

```
if (!validInputs(accountInput, amountInput)) return;

BigDecimal decimalInput = new BigDecimal(amountInput);
BigInteger amount = decimalInput.multiply(SCALE).toBigIntegerExact();

Paillier paillier = KeyManager.getKeys();
Connection conn = DBManager.connect();
if (conn == null) {
    return;
}

// Отримуємо зашифрований баланс відправника
BigInteger senderEncBalance = getEncBalance(conn, senAccNum);

if (senderEncBalance == null) {
    setStatus("Відправника не знайдено.", true);
    return;
}

// Перевіряємо чи достатньо коштів
if (!enoughBalance(paillier, decimalInput, senderEncBalance)) {
    setStatus("На рахунку недостатньо коштів для переказу.", true);
    return;
}

// Отримуємо зашифрований баланс отримувача
String recAccNum = HashFunc.hash(accountInput);
BigInteger receiverEncBalance = getEncBalance(conn, recAccNum);

if (receiverEncBalance == null) {
    setStatus("Отримувача не знайдено.", true);
    return;
}
```

```
        BigInteger newSenderEncBalance = paillier.subtract(senderEncBalance, amount);
        BigInteger newReceiverEncBalance = paillier.add(receiverEncBalance, amount);

        updateBalance(conn, newSenderEncBalance.toString(), senAccNum);
        updateBalance(conn, newReceiverEncBalance.toString(), recAccNum);

        setStatus("Переказ успішно виконано.", false);
        saveTransfer(conn, senAccNum, recAccNum, amount, paillier);
    }

    private boolean validInputs(String accountInput, String amountInput) {
        if (accountInput == null || accountInput.isEmpty()) {
            setStatus("Будь ласка, введіть номер рахунку.", true);
            return false;
        }

        if (HashFunc.compare(accountInput, senAccNum)) {
            setStatus("Номер рахунку отримувача співпадає з вашим.", true);
            return false;
        }

        if (amountInput == null || amountInput.isEmpty()) {
            setStatus("Будь ласка, введіть суму для переказу.", true);
            return false;
        }

        if (!amountInput.matches("\\d+(\\.\\d{1,2})?") || amountInput.equals("0")) {
            setStatus("Сума має бути цілим або дробовим числом (до 2 знаків після крапки).",
true);
            return false;
        }

        return true;
```

```
    }

    private BigInteger getEncBalance(Connection conn, String accNum) throws Exception {
        String query = "SELECT Баланс FROM Клієнти WHERE [Номер рахунку] = ?";

        PreparedStatement ps = conn.prepareStatement(query);
        ps.setString(1, accNum);
        ResultSet rs = ps.executeQuery();

        if (!rs.next()) {
            return null;
        }

        String balanceSTR = rs.getString("Баланс");

        if (balanceSTR == null) {
            return null;
        }

        return new BigInteger(balanceSTR);
    }

    private boolean enoughBalance(Paillier paillier, BigDecimal decInput, BigInteger
senEncBalance) {
        BigInteger senderDecBalance = paillier.decrypt(senEncBalance);
        BigDecimal senderRealBalance = new BigDecimal(senderDecBalance).divide(SCALE, 2,
RoundingMode.HALF_UP);

        return decInput.compareTo(senderRealBalance) < 0;
    }

    private void updateBalance(Connection conn, String newBalance, String accNum) throws
Exception {
        String query = "UPDATE Клієнти SET Баланс = ? WHERE [Номер рахунку] = ?";
```

```
        PreparedStatement psUpdate = conn.prepareStatement(query);

        psUpdate.setString(1, newBalance);
        psUpdate.setString(2, accNum);

        psUpdate.executeUpdate();
        log("Баланси користувачів оновлено.");
    }

    private String getLoginByNum(Connection conn, String accNum) throws SQLException {
        String sql = "SELECT Логін FROM Клієнти WHERE [Номер рахунку] = ?";
        PreparedStatement ps = conn.prepareStatement(sql);
        ps.setString(1, accNum);
        ResultSet rs = ps.executeQuery();
        if (rs.next()) {
            return rs.getString("Логін");
        }

        return null;
    }

    private void saveTransfer(Connection conn, String senNum, String recNum, BigInteger amount,
Paillier paillier) throws SQLException {
        BigInteger encAmount = paillier.encrypt(amount);

        String sql = "INSERT INTO Транзакції (Відправник, Отримувач, Сума, [Дата і час])
VALUES (?, ?, ?, ?)";
        PreparedStatement psSave = conn.prepareStatement(sql);

        psSave.setString(1, getLoginByNum(conn, senNum));
        psSave.setString(2, getLoginByNum(conn, recNum));
        psSave.setString(3, encAmount.toString());
        psSave.setTimestamp(4, Timestamp.valueOf(LocalDateTime.now()));
```

```
            psSave.executeUpdate();
            log("Транзакцію збережено.");
        }

        private void setStatus(String message, boolean error) {
            if (error) {
                statusLabel.setStyle("-fx-text-fill: #ff4d4d;");
                log(LogLevel.WARNING, message);
            } else {
                statusLabel.setStyle("-fx-text-fill: #4dff88;");
                log(message);
            }

            statusLabel.setText(message);
        }
    }
```

Модуль «TransactionHistory»:

```
package org.example.fxproject.controllers;

import javafx.fxml.FXML;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.cell.PropertyValueFactory;
import org.example.fxproject.crypto.Paillier;
import org.example.fxproject.Transaction;
import org.example.fxproject.util.DBManager;

import java.math.BigDecimal;
import java.math.BigInteger;
import java.math.RoundingMode;
import java.sql.Connection;
import java.sql.PreparedStatement;
```

```
import java.sql.ResultSet;
import java.sql.SQLException;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;

import static org.example.fxproject.util.AppLogger.*;

public class TransactionHistory {
    @FXML private TableView<Transaction> table;
    @FXML private TableColumn<Transaction, String> senderColumn;
    @FXML private TableColumn<Transaction, String> receiverColumn;
    @FXML private TableColumn<Transaction, BigDecimal> amountColumn;
    @FXML private TableColumn<Transaction, LocalDateTime> timestampColumn;

    private String currentLogin;
    private Paillier paillier;

    public void initialize(String currentLogin, Paillier paillier) throws SQLException {
        this.currentLogin = currentLogin;
        this.paillier = paillier;

        setupTableColumns();
        loadTransactions();
    }

    private void setupTableColumns() {
        senderColumn.setCellValueFactory(new PropertyValueFactory<>("sender"));
        receiverColumn.setCellValueFactory(new PropertyValueFactory<>("receiver"));
        amountColumn.setCellValueFactory(new PropertyValueFactory<>("amount"));
        timestampColumn.setCellValueFactory(new PropertyValueFactory<>("timestamp"));
    }
```

```
    private void loadTransactions() throws SQLException {
        List<Transaction> transactions = new ArrayList<>();
        BigDecimal SCALE = new BigDecimal(100);

        Connection conn = DBManager.connect();
        if (conn == null) {
            return;
        }

        String sql = "SELECT * FROM Транзакції WHERE Відправник = ? OR Отримувач = ? ORDER BY [Дата і час] DESC";
        PreparedStatement ps = conn.prepareStatement(sql);

        ps.setString(1, currentLogin);
        ps.setString(2, currentLogin);
        ResultSet rs = ps.executeQuery();

        while (rs.next()) {
            String sender = rs.getString("Відправник");
            String receiver = rs.getString("Отримувач");

            if (rs.getString("Сума") == null || rs.getTimestamp("Дата і час") == null) {
                log(LogLevel.ERROR, "Відсутня інформація про транзації.");
                return;
            }

            BigInteger encAmount = new BigInteger(rs.getString("Сума"));
            BigInteger decAmount = paillier.decrypt(encAmount);
            BigDecimal realAmount = new BigDecimal(decAmount).divide(SCALE, 2, RoundingMode.HALF_UP);
            LocalDateTime timestamp = rs.getTimestamp("Дата і час").toLocalDateTime();

            transactions.add(new Transaction(sender, receiver, realAmount, timestamp));
        }
```

```
        table.getItems().setAll(transactions);
        log("Історію транзакцій завантажено.");
    }
}
```

Модуль «Transaction»:

```
package org.example.fxproject;

import java.math.BigDecimal;
import java.time.LocalDateTime;

public class Transaction {
    private final String sender;
    private final String receiver;
    private final BigDecimal amount;
    private final LocalDateTime timestamp;

    public Transaction(String sender, String receiver, BigDecimal amount, LocalDateTime
timestamp) {
        this.sender = sender;
        this.receiver = receiver;
        this.amount = amount;
        this.timestamp = timestamp;
    }

    public String getSender() {
        return sender;
    }

    public String getReceiver() {
        return receiver;
    }
```

```
    public BigDecimal getAmount() {
        return amount;
    }

    public LocalDateTime getTimestamp() {
        return timestamp;
    }
}
```

Модуль «AppLogger»:

```
package org.example.fxproject.util;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;

public class AppLogger {
    public enum LogLevel {
        INFO, WARNING, ERROR
    }

    private static final List<String> logs = new ArrayList<>();
    private static final String LOG_FILE = "log.txt";

    static {
        // Автоматичне збереження логів при завершенні програми
        Runtime.getRuntime().addShutdownHook(new Thread(AppLogger::saveLogsToFile));
    }

    public static void log(String message) {
        log(LogLevel.INFO, message);
```

```
    }

    public static void log(LogLevel level, String message) {
        String timestamp = LocalDateTime.now().toString();
        String formatted = String.format("[%s] [%s] %s", timestamp, level.name(), message);
        System.out.println(formatted);
        logs.add(formatted);
    }

    private static void saveLogsToFile() {
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(LOG_FILE, false))) {
            for (String log : logs) {
                writer.write(log);
                writer.newLine();
            }
        } catch (IOException e) {
            System.err.println("Не вдалося записати логи у файл: " + e.getMessage());
        }
    }
}
```

Модуль «DBManager»:

```
package org.example.fxproject.util;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

import static org.example.fxproject.util.AppLogger.*;

public class DBManager {
    private static final String DB_URL = "jdbc:ucanaccess://src/main/resources/database.accdb";

    public static Connection connect() {
```

```
        try {
            Connection conn = DriverManager.getConnection(DB_URL);
            log("З'єднання з базою даних успішно встановлено.");
            return conn;
        } catch (SQLException e) {
            log(LogLevel.ERROR, "Помилка з'єднання з базою даних: " + e.getMessage());
            return null;
        }
    }
}
```

Модуль «DialogUtil»:

```
package org.example.fxproject.util;

import javafx.scene.control.Alert;
import javafx.scene.control.DialogPane;

import java.util.Objects;

public class DialogUtil {

    public static void showInfo(String title, String content) {
        AppLogger.log("Відкрито довідкове вікно " + title + ".");
        Alert alert = new Alert(Alert.AlertType.INFORMATION);
        alert.setTitle(title);

        alert.setHeaderText(null);
        alert.setGraphic(null);

        // Стилізація
        DialogPane dialogPane = alert.getDialogPane();
        dialogPane.getStylesheets().add(
                Objects.requireNonNull(DialogUtil.class.getResource("/style.css")).toExternalForm()
        );
        dialogPane.getStyleClass().add("custom-dialog");
```

```
        alert.setContentText(content);
        alert.showAndWait();
    }
}
```

Модуль «ExRateLoader»:

```
package org.example.fxproject.util;

import javafx.application.Platform;
import javafx.scene.control.Label;
import org.json.JSONObject;

import java.io.*;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.Properties;

import static org.example.fxproject.util.AppLogger.*;

public class ExRateLoader {
    private static final String FILE_NAME = "exchange-key.properties";

    public static void loadExchangeRates(Label usdLabel, Label eurLabel) {
        new Thread(() -> {
            try {
                String apiKey = getApiKey();
                if (apiKey == null) {
                    Platform.runLater(() -> {
                        usdLabel.setText("USD: Н/д");
                        eurLabel.setText("EUR: Н/д");
                    });
                    return;
                }
```

```
            JSONObject response = fetchExchangeRates(apiKey);
            updateLabels(response, usdLabel, eurLabel);
        } catch (Exception e) {
            e.printStackTrace();
            Platform.runLater(() -> {
                usdLabel.setText("USD: Н/д");
                eurLabel.setText("EUR: Н/д");
            });
        }
    }).start();
}

private static String getApiKey() throws IOException {
    File file = new File(FILE_NAME);
    if (!file.exists()) {
        log(LogLevel.ERROR,"Файл " + FILE_NAME + " не знайдено.");
        return null;
    }

    Properties props = new Properties();
    try (FileInputStream fis = new FileInputStream(file)) {
        props.load(fis);
    }

    String key = props.getProperty("exchange_api_key");
    if (key == null || key.isBlank()) {
        log(LogLevel.ERROR, "API-ключ відсутній у файлі " + FILE_NAME + ".");
        return null;
    }

    log("Завантажено API-ключ курсу валют.");
    return key;
}

private static JSONObject fetchExchangeRates(String apiKey) throws IOException {
```

```
        URL url = new URL("https://v6.exchangerate-api.com/v6/" + apiKey + "/latest/UAH");
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
        conn.setRequestMethod("GET");

        try (BufferedReader reader = new BufferedReader(new
InputStreamReader(conn.getInputStream()))) {
            StringBuilder sb = new StringBuilder();
            String line;
            while ((line = reader.readLine()) != null) sb.append(line);
            log("Отримано курс валют.");
            return new JSONObject(sb.toString());
        }
    }

    private static void updateLabels(JSONObject json, Label usdLabel, Label eurLabel) {
        JSONObject rates = json.getJSONObject("conversion_rates");
        double usd = rates.getDouble("USD");
        double eur = rates.getDouble("EUR");

        Platform.runLater(() -> {
            usdLabel.setText(String.format("USD: %.2f", 1 / usd));
            eurLabel.setText(String.format("EUR: %.2f", 1 / eur));
        });
    }
}
```

Модуль «JwtUtil»:

```
package org.example.fxproject.util;

import com.auth0.jwt.JWT;
import com.auth0.jwt.algorithms.Algorithm;
import com.auth0.jwt.interfaces.DecodedJWT;
import com.auth0.jwt.interfaces.JWTVerifier;

import java.io.File;
```

```
import java.io.FileWriter;
import java.nio.file.Files;
import java.time.Instant;
import java.util.Date;

import static org.example.fxproject.util.AppLogger.*;

public class JwtUtil {
    private static final String SECRET = "a3B9e@91Ld!zXc!eF9j2L$kwpYvH7gPl";
    private static final Algorithm algorithm = Algorithm.HMAC256(SECRET);
    private static final File tokenFile = new File("user.jwt");

    public static String createToken(String login) {
        log("JWT створено для користувача: " + login + ".");
        return JWT.create()
                .withSubject(login)
                .withIssuedAt(new Date())
                .withExpiresAt(Date.from(Instant.now().plusSeconds(3600))) // 1 година
                .sign(algorithm);
    }

    public static void saveToken(String token) throws Exception {
        try (FileWriter writer = new FileWriter(tokenFile)) {
            writer.write(token);
            log("JWT збережено.");
        }
    }

    public static String readToken() throws Exception {
        if (!tokenFile.exists()) return null;
        return Files.readString(tokenFile.toPath());
    }

    public static boolean isTokenValid(String token) {
        try {
```

```
            JWTVerifier verifier = JWT.require(algorithm).build();
            DecodedJWT jwt = verifier.verify(token);
            return jwt.getExpiresAt().after(new Date());
        } catch (Exception e) {
            return false;
        }
    }

    public static String getLoginFromToken(String token) {
        String login = JWT.decode(token).getSubject();
        log("Автоматичний вхід для користувача: " + login + ".");
        return login;
    }

    public static void deleteToken() {
        if (tokenFile.exists()) {
            boolean deleted = tokenFile.delete();

            if (!deleted) {
                log(LogLevel.ERROR, "Не вдалося видалити JWT токен.");
                return;
            }

            log("JWT токен видалено.");
        }
    }
}
```

Модуль «EncryptorAES»:

```
package org.example.fxproject.crypto;

import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;
import java.math.BigInteger;
import java.nio.charset.StandardCharsets;
```

```
import java.security.MessageDigest;
import java.util.Base64;

public class EncryptorAES {
    private static final String ALGORITHM = "AES";
    private static final String SECRET = "DemonstrationPassword";

    private static SecretKeySpec getAESKeyFromPassword() throws Exception {
        MessageDigest sha = MessageDigest.getInstance("SHA-256");
        byte[] keyBytes = sha.digest(SECRET.getBytes(StandardCharsets.UTF_8));
        byte[] key128 = new byte[16]; // AES-128
        System.arraycopy(keyBytes, 0, key128, 0, 16);
        return new SecretKeySpec(key128, ALGORITHM);
    }

    public static String encrypt(BigInteger number) throws Exception {
        byte[] inputBytes = number.toByteArray();
        Cipher cipher = Cipher.getInstance(ALGORITHM);
        cipher.init(Cipher.ENCRYPT_MODE, getAESKeyFromPassword());
        byte[] encryptedBytes = cipher.doFinal(inputBytes);
        return Base64.getEncoder().encodeToString(encryptedBytes);
    }

    public static BigInteger decrypt(String encryptedBase64) throws Exception {
        byte[] encryptedBytes = Base64.getDecoder().decode(encryptedBase64);
        Cipher cipher = Cipher.getInstance(ALGORITHM);
        cipher.init(Cipher.DECRYPT_MODE, getAESKeyFromPassword());
        byte[] decryptedBytes = cipher.doFinal(encryptedBytes);
        return new BigInteger(decryptedBytes);
    }
}
```

Модуль «HashFunc»:

```
package org.example.fxproject.crypto;
```

```
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import static org.example.fxproject.util.AppLogger.*;

public class HashFunc {
    public static boolean compare(String input, String stored) {
        String hashedInput = hash(input);
        return hashedInput.equals(stored);
    }

    public static String hash(String s) {
        try {
            MessageDigest digest = MessageDigest.getInstance("SHA-256");

            byte[] hashBytes = digest.digest(s.getBytes(StandardCharsets.UTF_8));

            StringBuilder hexString = new StringBuilder();

            for (byte b : hashBytes) {
                String hex = Integer.toHexString(0xff & b);
                if (hex.length() == 1) hexString.append('0');
                hexString.append(hex);
            }

            log("Успішно здійснено гешування даних.");
            return hexString.toString();

        } catch (NoSuchAlgorithmException e) {
            log(LogLevel.ERROR, "Помилка: SHA-256 алгоритм не знайдено — " + e.getMessage());
            throw new RuntimeException("SHA-256 алгоритм не знайдено", e);
        }
    }
}
```

Модуль «KeyManager»:

```
package org.example.fxproject.crypto;

import java.io.*;
import java.math.BigInteger;
import java.util.Properties;

import org.example.fxproject.util.AppLogger;

public class KeyManager {
    private static final String FILE_NAME = "paillier-keys.properties";

    public static Paillier getKeys() throws Exception {
        Paillier paillier;
        Paillier loaded = KeyManager.loadKeys();

        if (loaded != null) {
            paillier = loaded;
            AppLogger.log("Ключі завантажено з файлу.");
        } else {
            paillier = new Paillier();
            paillier.generateKeys(512);
            KeyManager.saveKeys(paillier);
            AppLogger.log("Ключі згенеровано і збережено.");
        }

        return paillier;
    }

    private static void saveKeys(Paillier paillier) throws Exception {
        Properties props = new Properties();
        props.setProperty("n", paillier.getN().toString());
        props.setProperty("g", paillier.getG().toString());
        props.setProperty("lambda", EncryptorAES.encrypt(paillier.getLambda()));
```

```
        props.setProperty("mu", EncryptorAES.encrypt(paillier.getMu()));

        try (FileOutputStream fos = new FileOutputStream(FILE_NAME)) {
            props.store(fos, "Paillier Keys");
        }
    }

    private static Paillier loadKeys() throws Exception {
        File file = new File(FILE_NAME);
        if (!file.exists()) {
            return null; // Ключі ще не збережено
        }

        Properties props = new Properties();
        try (FileInputStream fis = new FileInputStream(file)) {
            props.load(fis);
        }

        BigInteger n = new BigInteger(props.getProperty("n"));
        BigInteger g = new BigInteger(props.getProperty("g"));
        BigInteger lambda = EncryptorAES.decrypt(props.getProperty("lambda"));
        BigInteger mu = EncryptorAES.decrypt(props.getProperty("mu"));

        return new Paillier(n, g, lambda, mu);
    }
}
```

Модуль «Paillier»:

```
package org.example.fxproject.crypto;

import org.example.fxproject.util.AppLogger;

import java.math.BigInteger;
import java.security.SecureRandom;

public class Paillier {
```

```
public BigInteger n, nSquare, g, lambda, mu;
private int bitLength = 512;
private final SecureRandom random;

// Конструктор для генерації ключів
public Paillier() {
    random = new SecureRandom();
}

// Конструктор для завантаження збережених ключів
public Paillier(BigInteger n, BigInteger g, BigInteger lambda, BigInteger mu) {
    this.n = n;
    this.g = g;
    this.lambda = lambda;
    this.mu = mu;
    this.nSquare = n.multiply(n);
    this.random = new SecureRandom();
}

public void generateKeys(int bitLengthVal) {
    bitLength = bitLengthVal;
    BigInteger p = BigInteger.probablePrime(bitLength / 2, random);
    BigInteger q = BigInteger.probablePrime(bitLength / 2, random);
    n = p.multiply(q);
    nSquare = n.multiply(n);
    g = n.add(BigInteger.ONE); // проста форма
    lambda = lcm(p.subtract(BigInteger.ONE), q.subtract(BigInteger.ONE));
    mu = L(g.modPow(lambda, nSquare)).modInverse(n);
}

private BigInteger lcm(BigInteger a, BigInteger b) {
    return a.multiply(b).divide(a.gcd(b));
}

private BigInteger L(BigInteger u) {
```

```
    return u.subtract(BigInteger.ONE).divide(n);
}

public BigInteger encrypt(BigInteger m) {
    BigInteger r;
    do {
        r = new BigInteger(bitLength, random);
    } while (r.compareTo(n) >= 0 || r.gcd(n).intValue() != 1);
    AppLogger.log("Відбулося шифрування даних.");

    return g.modPow(m, nSquare).multiply(r.modPow(n, nSquare)).mod(nSquare);
}

public BigInteger decrypt(BigInteger c) {
    AppLogger.log("Відбулося розшифрування даних.");
    return L(c.modPow(lambda, nSquare)).multiply(mu).mod(n);
}

// Метод для гомоморфного додавання
public BigInteger add(BigInteger encrypted, BigInteger notEncrypted) {
    AppLogger.log("Здійснюється гомоморфне додавання.");
    return encrypted.multiply(encrypt(notEncrypted)).mod(nSquare);
}

// Метод для гомоморфного віднімання
public BigInteger subtract(BigInteger encrypted, BigInteger notEncrypted) {
    AppLogger.log("Здійснюється гомоморфне віднімання.");
    return encrypted.multiply(encrypt(notEncrypted.negate())).mod(nSquare);
}

// n, g - public key
public BigInteger getN() {
    return n;
}
```

```
    public BigInteger getG() {
        return g;
    }

    // lambda, mu - private key
    public BigInteger getLambda() {
        return lambda;
    }

    public BigInteger getMu() {
        return mu;
    }
}
```

ДОДАТОК Г

Ілюстративна частина

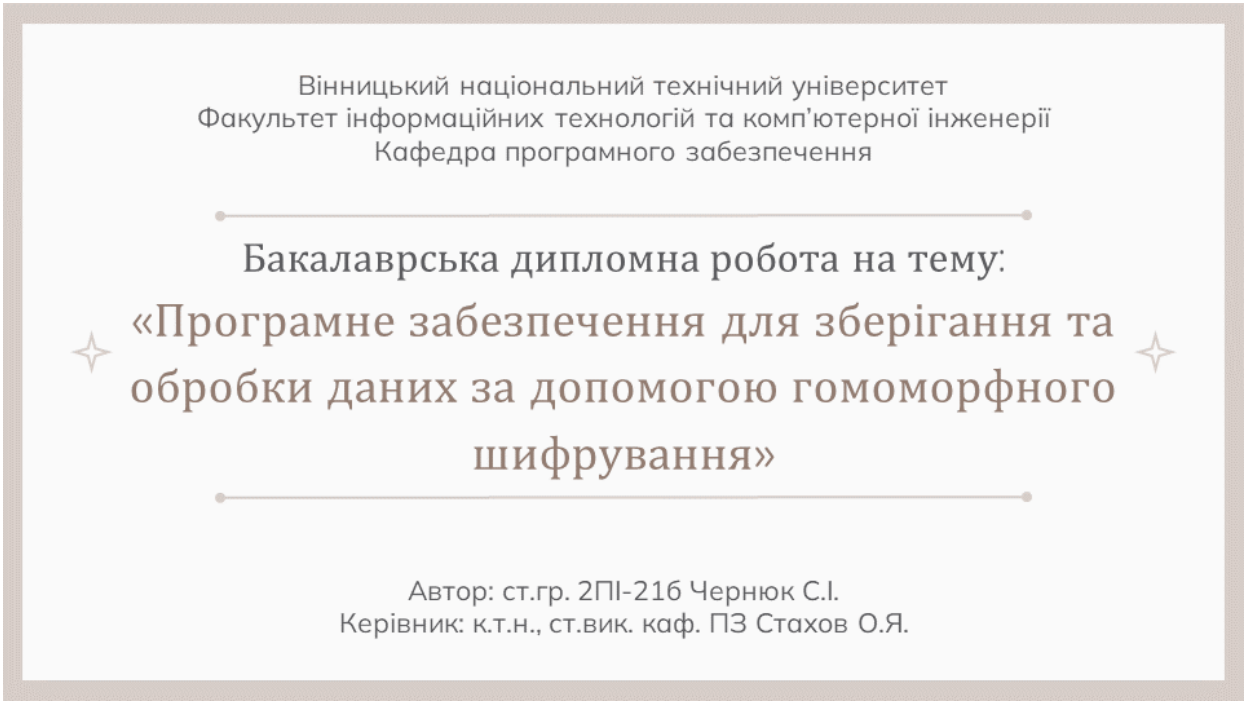

Рисунок Г.1 – Титульний слайд

Рисунок Г.2 – Що таке гомоморфне шифрування

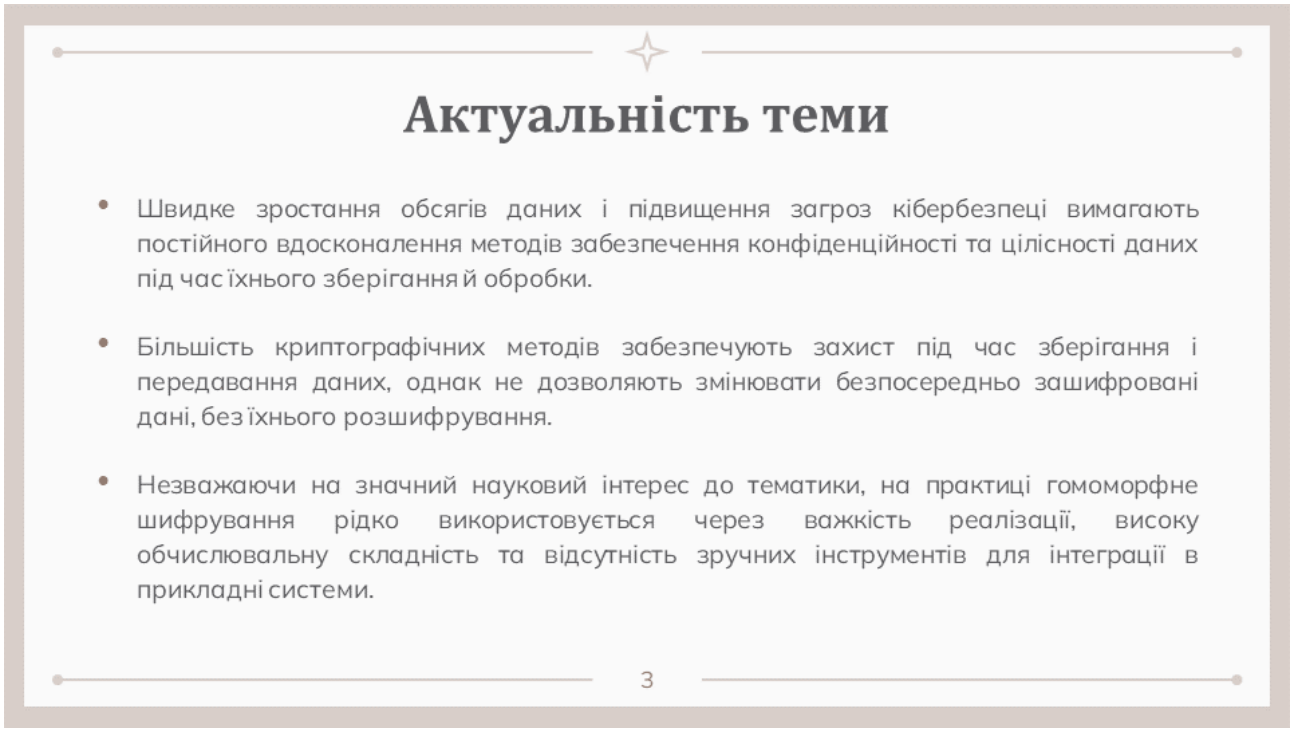

Рисунок Г.3 – Актуальність теми

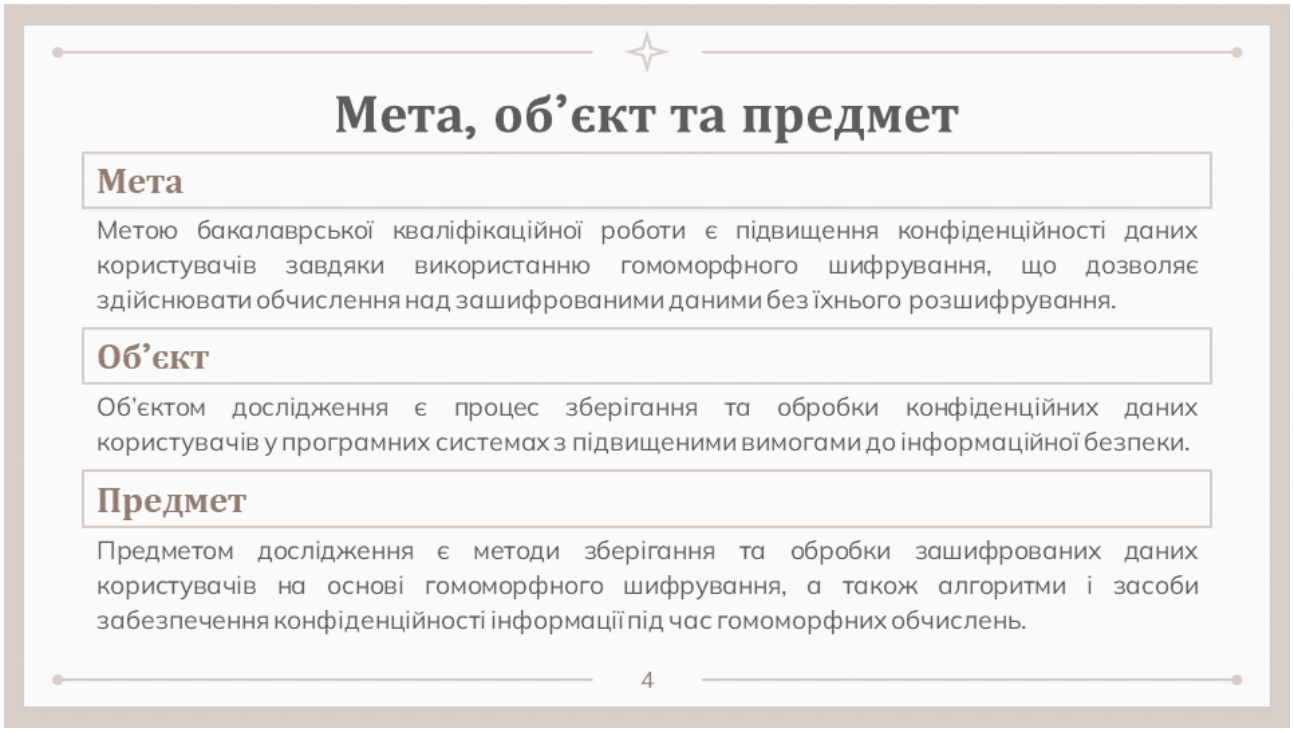

Рисунок Г.4 – Мета, об’єкт та предмет

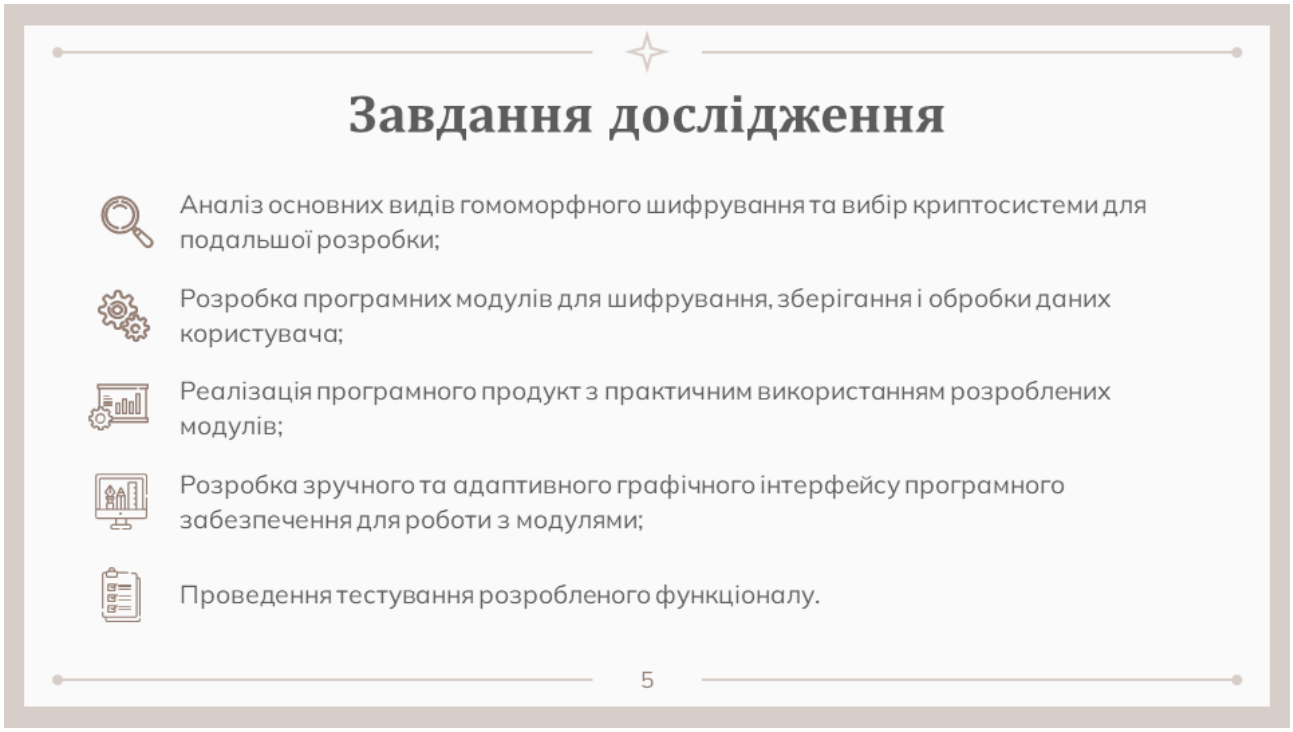

Рисунок Г.5 – Завдання дослідження

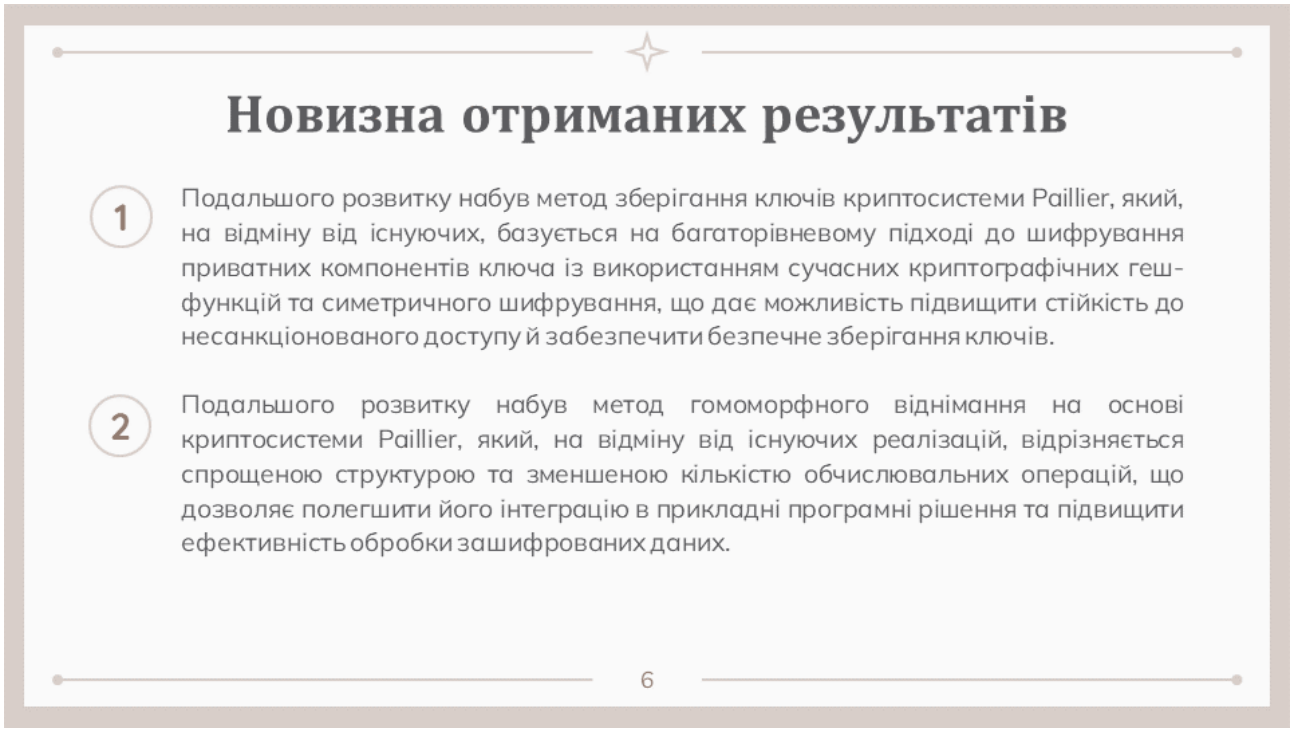

Рисунок Г.6 – Новизна отриманих результатів

Практичне значення результатів

Практичне значення отриманих результатів полягає в тому, що на основі отриманих в бакалаврській кваліфікаційній роботі теоретичних положень запропоновано алгоритми та розроблено програмні засоби захисту даних користувачів із використанням гомоморфного шифрування.

7

Рисунок Г.7 – Практичне значення результатів

Порівняльний аналіз аналогів

	Власна розробка	Система електронного голосування	Обчислювач рівня ризику діабету	Класифікатор зашифрованих зображень
Реалізація FHE	-	+	-	+
Наявність GUI/WebUI	+	-	+	-
Доступність коду	+/-	+	+	+
Використання додаткових методів захисту даних	+	-	-	-
Можливість розширення	+	-	+	+

8

Рисунок Г.8 – Порівняльний аналіз аналогів

Рисунок Г.9 – Модель роботи програми

Рисунок Г.10 – Алгоритм роботи програми

Рисунок Г.11 – Алгоритм зберігання ключів криптосистеми Paillier

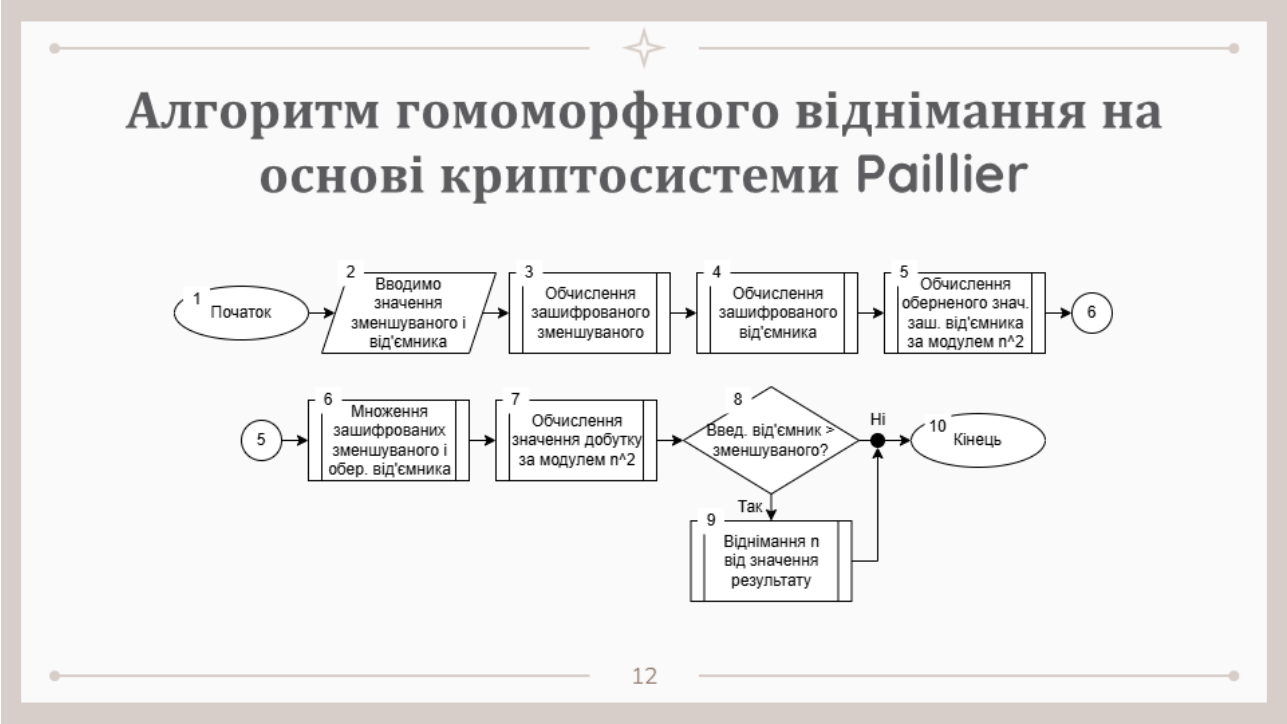

Рисунок Г.12 – Алгоритм гомоморфного віднімання на основі криптосистеми Paillier (1)

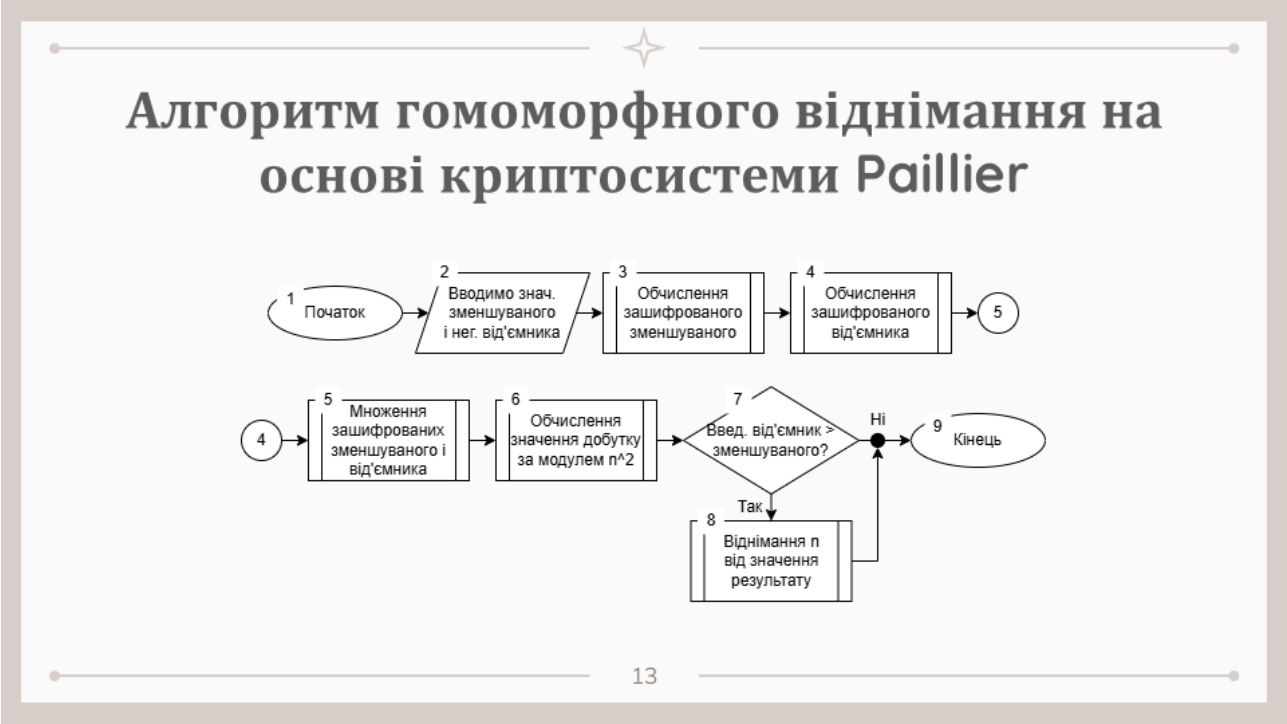

Рисунок Г.13 – Алгоритм гомоморфного віднімання на основі криптосистеми Paillier (2)

Рисунок Г.14 – Алгоритм здійснення переказу коштів

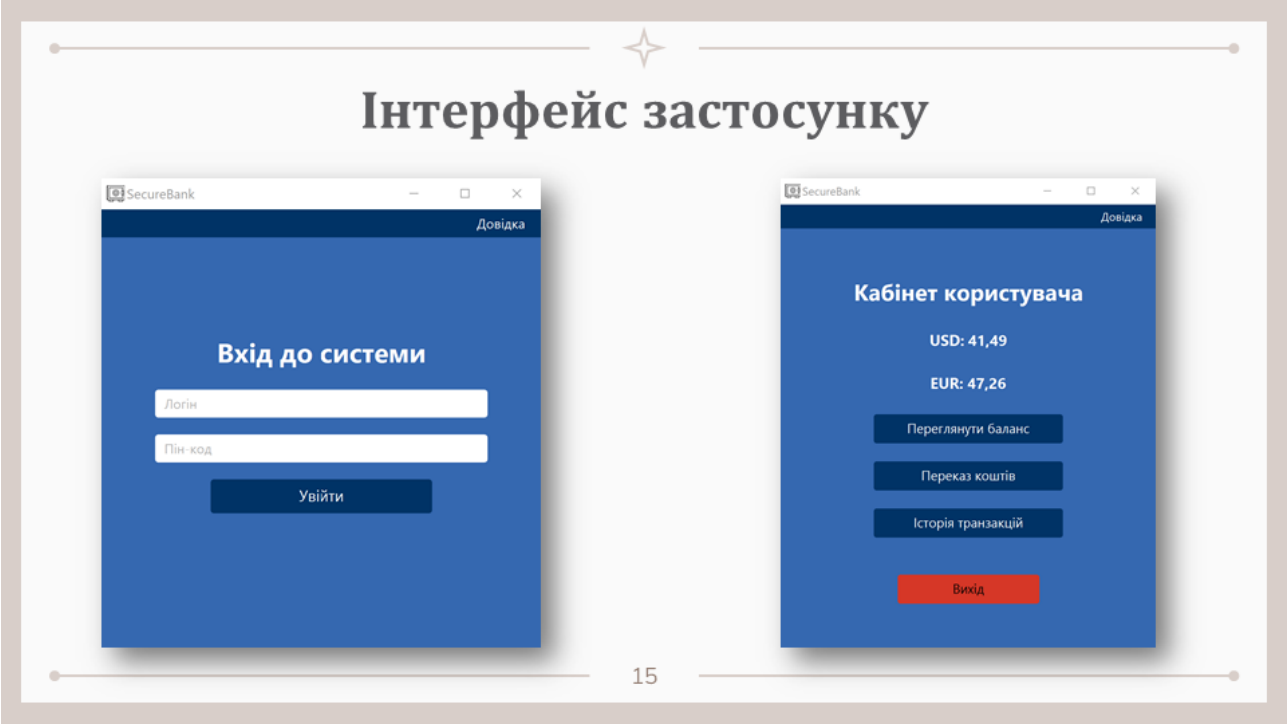

Рисунок Г.15 – Інтерфейс застосунку (1)

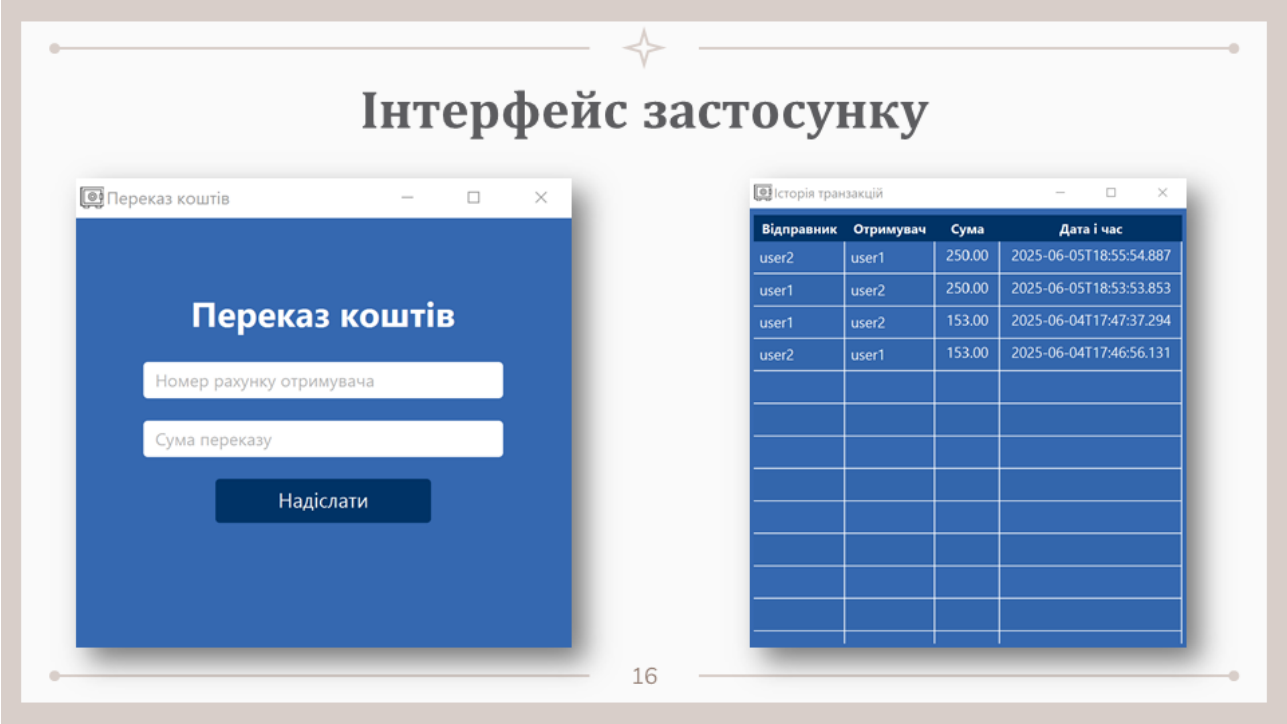

Рисунок Г.16 – Інтерфейс застосунку (2)

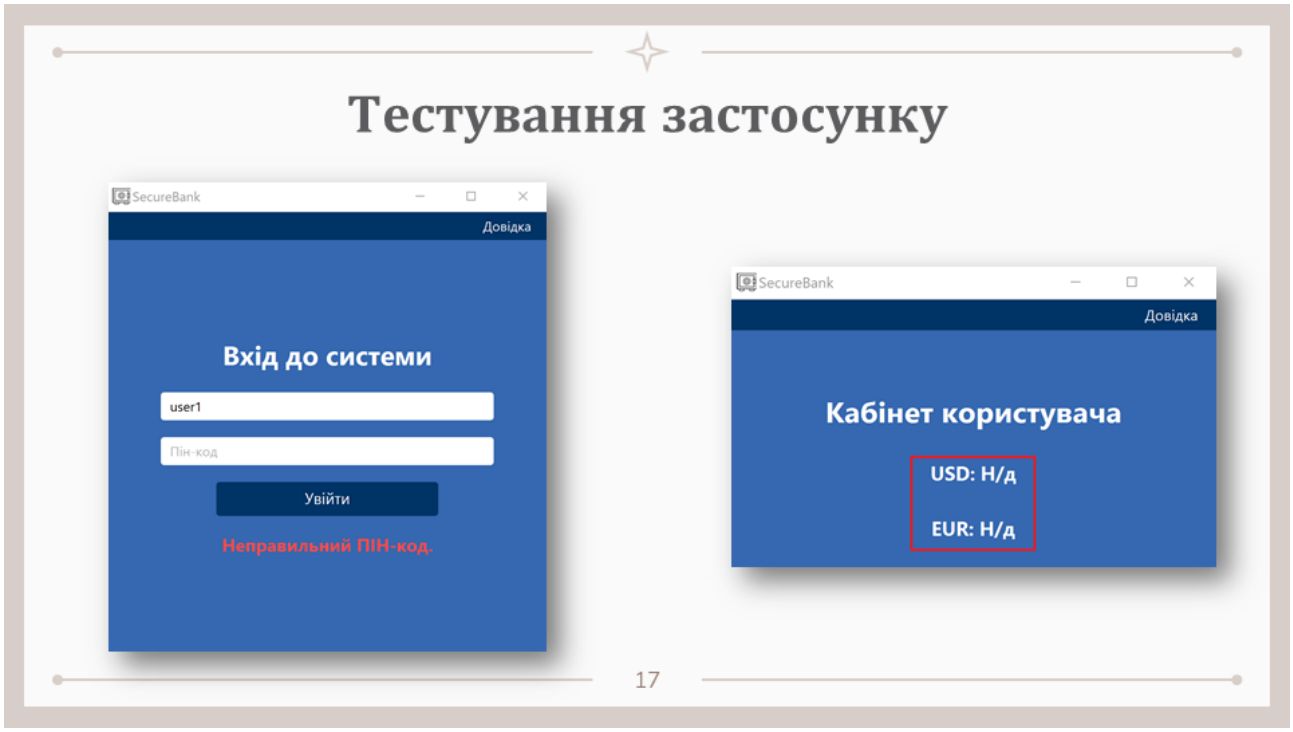

Рисунок Г.17 – Тестування застосунку (1)

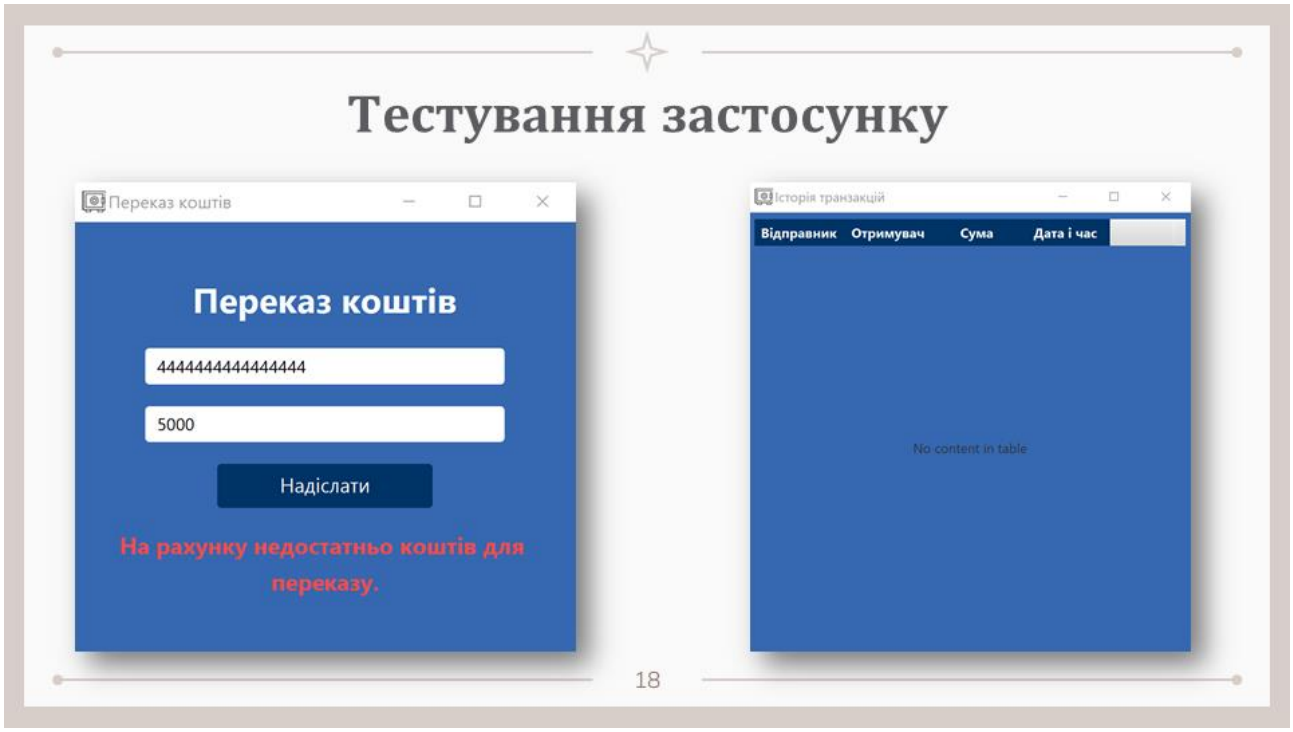

Рисунок Г.18 – Тестування застосунку (1)

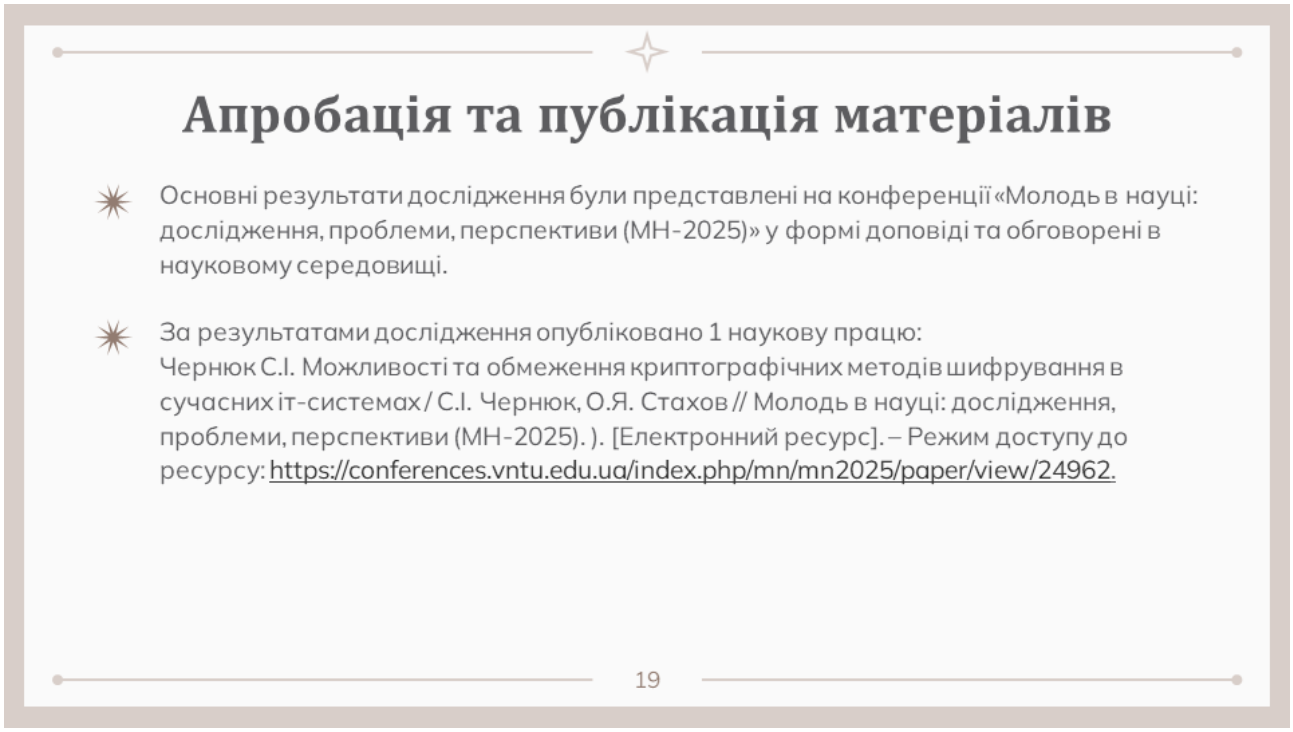

Рисунок Г.19 – Апробація та публікація матеріалів

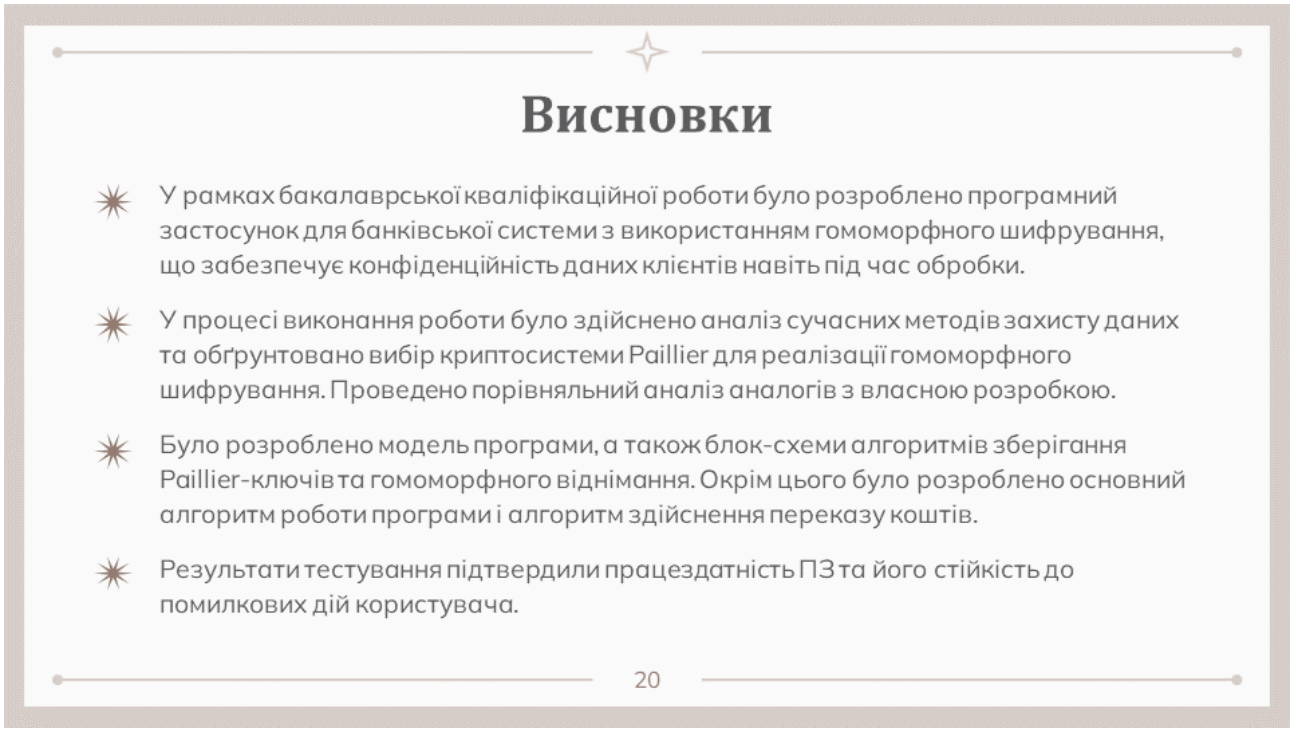

Рисунок Г.20 – Висновки

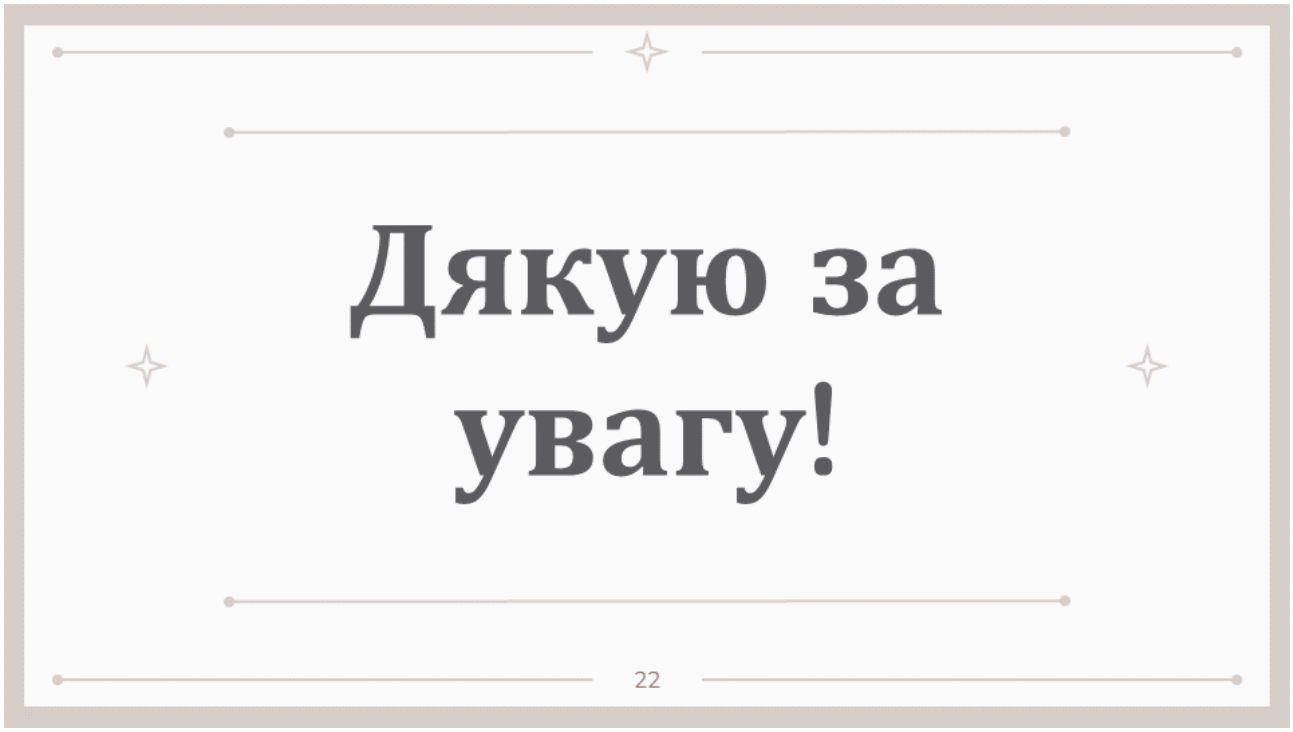

Рисунок Г.21 – Фінальний слайд