

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ**

Факультет інформаційних технологій

**ПОГОДЖЕНО
Декан факультету**

_____ **Ігор БОЛБОТ**
(підпис)

«__» _____ 2026 р.

**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ
Завідувач кафедри інформаційних
систем і технологій**

_____ **Михайло ШВИДЕНКО**
(підпис)

«__» _____ 2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Інформаційна система для готівкових P2P обмінів»

Спеціальність «126 Інформаційні системи та технології»

Освітньо-професійна програма «Інформаційно управляючі системи та технології»

Гарант освітньої програми

д.е.н., професор

**Керівник бакалаврської
кваліфікаційної роботи**

д.ф. з к.н.

Виконав

_____ **Максим МОКРІЄВ**
(підпис)

_____ **Роман ЗОЛОТУХА**
(підпис)

_____ **Євгеній ІЩЕНКО**
(підпис)

Київ-2026

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ**

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри інформаційних систем і
технологій

к.е.н., доцент _____ Михайло ШВИДЕНКО
(підпис)

«___» _____ 2026 р.

ЗАВДАННЯ
ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
ЗДОБУВАЧУ

Іщенко Євгеній Олександрович

Спеціальність 126 – «Інформаційні системи та технології»

ОП «Інформаційно управляючі системи та технології»

Тема бакалаврської кваліфікаційної роботи «Інформаційна система для готівкових P2P обмінів» затверджена наказом від «19» грудень 2025 р. № 3205 «С2»

Термін подання завершеної роботи на кафедру «25» травня 2026 р.

Вихідні дані до бакалаврської кваліфікаційної роботи (проєкту): матеріали щодо організації P2P-обміну готівкових валют між користувачами; вимоги до створення заявок, пошуку пропозицій, узгодження умов угоди, обміну повідомленнями та фіксації результатів операцій; дані користувачів, заявок, повідомлень, валютних курсів і статусів угод. Перелік питань, що підлягають дослідженню:

- Аналіз предметної області P2P-обміну готівкових валют між користувачами
- Визначення вимог до інформаційної системи та моделювання її основних процесів

- Проектування структури бази даних, архітектури та програмних модулів застосунку
- Реалізація функцій створення заявок, пошуку пропозицій, повідомлень і підтвердження угод
- Тестування, розгортання та оцінювання результатів роботи системи

Дата видачі завдання «06» січня 2026 р.

**Керівник бакалаврської
кваліфікаційної роботи**

(підпис)

Роман ЗОЛОТУХА

**Завдання взяв до
виконання**

(підпис)

Євгеній ІЩЕНКО

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ	8
ВСТУП	10
1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ	14
1.1 Опис предметної області	14
1.2 Аналіз існуючих програмних рішень.....	17
1.3 Моделювання предметної області.....	23
1.4 Аналіз вимог до інформаційної системи Р2Р-обміну готівки нестандартних валют.....	27
1.5 Постановка завдання.....	30
Висновки до розділу 1	32
2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	34
2.1 Логічна модель даних інформаційної системи Р2Р-обміну готівки нестандартних валют	34
2.2 Діаграма класів і кооперації системи.....	38
2.3 Діаграма компонентів	43
2.4 Діаграма пакетів	47
2.5 Висновки до другого розділу	50
3 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ....	52
3.1 Вибір технологій та інструментальних засобів реалізації	52
3.2 Архітектура системи.....	55
3.3 Фізична модель даних локального застосунку Р2Р-обміну готівки нестандартних валют	57
3.4 Алгоритмізація програмних модулів системи	60
3.5 Висновки до третього розділу.....	64
4 ТЕСТУВАННЯ, РОЗГОРТАННЯ ТА ВПРОВАДЖЕННЯ СИСТЕМИ.....	66
4.1 Тестування програмних модулів застосунку « “Фінансовий помічник” для організації готівкових Р2Р-обмінів	66

4.2 Тестування системи	70
4.3 Розгортання системи, склад інсталяційного пакету	77
4.4 Висновки до четвертого розділу.....	81
ВИСНОВКИ.....	83
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	86
ДОДАТОК А.....	90
ДОДАТОК Б	93
ДОДАТОК В.....	101

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

1. P2P - peer-to-peer, модель взаємодії між користувачами без безпосереднього посередника.
2. БД - база даних.
3. СУБД - система управління базами даних.
4. SQLite - вбудована реляційна база даних, що використовується для локального зберігання даних застосунку.
5. PyQt6 - програмний фреймворк для створення графічного інтерфейсу користувача мовою Python.
6. Python - мова програмування, використана для реалізації логіки застосунку.
7. GUI - graphical user interface, графічний інтерфейс користувача.
8. QSS - Qt Style Sheets, засіб стилізації елементів інтерфейсу PyQt6.
9. SQL - structured query language, мова структурованих запитів до бази даних.
10. UML - unified modeling language, уніфікована мова моделювання програмних систем.
11. DFD - data flow diagram, діаграма потоків даних.
12. BPMN - business process model and notation, нотація моделювання бізнес-процесів.
13. ID - унікальний ідентифікатор запису або об'єкта в системі.
14. PK - primary key, первинний ключ таблиці бази даних.
15. FK - foreign key, зовнішній ключ таблиці бази даних.
16. UAH - українська гривня.
17. GEL - грузинський ларі.
18. AZN - азербайджанський манат.
19. TRY - турецька ліра.
20. KZT - казахстанський тенге.

21. MDL - молдовський лей.
22. BGN - болгарський лев.
23. CSV - comma-separated values, формат експорту табличних даних.
24. API - application programming interface, програмний інтерфейс взаємодії з зовнішніми сервісами.
25. JSON - JavaScript Object Notation, текстовий формат зберігання та передавання структурованих даних.

ВСТУП

Актуальність теми зумовлена активним розвитком цифрових фінансових сервісів та зростаючою потребою користувачів у швидких, зручних і доступних способах обміну валют. У сучасних умовах глобалізації, міжнародної мобільності населення, розвитку електронної комерції та розширення транскордонних фінансових операцій значна кількість користувачів стикається з необхідністю обміну грошових одиниць, які не завжди доступні в банківських установах або офіційних обмінних пунктах. Особливо це стосується нестандартних, локальних, регіональних або малопоширених валют, для яких відсутня розвинена інфраструктура обміну.

Традиційні банківські сервіси та офіційні обмінні пункти здебільшого орієнтовані на роботу з поширеними валютами, що мають стабільний попит і високу ліквідність. У випадку необхідності обміну нестандартних валют користувачі часто змушені витратити значний час на пошук відповідного обмінника, погоджувати не вигідні умови або звертатися до неформальних каналів взаємодії. Це створює додаткові часові та фінансові витрати, а також ускладнює контроль домовленостей між учасниками обміну.

Одним із перспективних напрямів вирішення цієї проблеми є використання цифрових платформ типу peer-to-peer (P2P), які дають змогу організовувати прямий обмін між користувачами без обов'язкового залучення фінансових посередників. Такі системи забезпечують гнучкішу взаємодію між учасниками, дають можливість самостійно створювати пропозиції, знаходити відповідні заявки, узгоджувати умови операції та фіксувати результати домовленостей. P2P-підхід є особливо доцільним для роботи з нестандартними валютами, оскільки дозволяє поєднувати користувачів, які мають взаємну потребу в обміні, навіть за відсутності централізованої інфраструктури.

Разом з тим ефективна реалізація таких платформ потребує створення спеціалізованого програмного забезпечення, яке забезпечує не лише розміщення

заявок, а й підтримку повного циклу взаємодії користувачів. Інформаційна система має надавати можливість реєстрації користувачів, створення заявок на обмін, пошуку відповідних пропозицій, фіксації домовленостей, контролю статусів угоди, збереження історії операцій і формування звітної інформації. Відсутність комплексних локальних рішень, орієнтованих саме на організацію готівкового P2P-обміну нестандартних валют, зумовлює доцільність розроблення застосунку, який поєднає ці функції в єдиному програмному середовищі.

Метою кваліфікаційної роботи є розроблення програмного забезпечення застосунку “Фінансовий помічник”, призначеного для підтримки процесу готівкового P2P-обміну нестандартних валют між користувачами. Розроблювана система має забезпечувати створення та оброблення заявок на обмін, пошук відповідних пропозицій, взаємодію між учасниками угоди та фіксацію результатів проведених операцій, що сприятиме підвищенню зручності й ефективності здійснення фінансових обмінів між користувачами.

Для досягнення поставленої мети у роботі необхідно вирішити такі основні **завдання:**

1. здійснити аналіз предметної області та особливостей функціонування систем P2P-обміну валют;
2. визначити функціональні та нефункціональні вимоги до інформаційної системи обміну готівки між користувачами;
3. побудувати функціональні та інформаційні моделі системи з використанням методів структурного та об’єктно-орієнтованого моделювання;
4. розробити архітектуру програмного забезпечення та модель даних системи;
5. реалізувати основні програмні модулі застосунку, що забезпечують створення заявок, пошук партнерів для обміну та взаємодію користувачів;
6. провести тестування програмного забезпечення та оцінити ефективність функціонування розробленої системи.

Об'єктом дослідження є процеси обміну валют між користувачами в умовах використання цифрових P2P-платформ.

Предметом дослідження є методи, моделі та програмні засоби організації й підтримки процесу готівкового P2P-обміну нестандартних валют у межах інформаційної системи.

У процесі виконання роботи використано методи системного аналізу для дослідження предметної області та визначення вимог до системи, методи структурного й об'єктно-орієнтованого моделювання для проєктування архітектури програмного забезпечення, методи аналізу інформаційних потоків для побудови функціональних моделей системи, методи проєктування баз даних для розроблення інформаційної моделі, а також методи програмної інженерії для реалізації, тестування та оцінювання програмного застосунку.

Науково-практична новизна роботи полягає у розробленні моделі локальної інформаційної системи для організації готівкових P2P-обмінів нестандартних валют, яка поєднує механізми створення та оброблення заявок, пошуку відповідних пропозицій, фіксації домовленостей між користувачами, контролю статусів угоди та формування звітної інформації. Запропонований підхід дає змогу впорядкувати процес прямої взаємодії між учасниками обміну, зменшити часові витрати на пошук партнера для операції та підвищити прозорість виконання домовленостей у межах єдиного програмного середовища.

Практичне значення одержаних результатів полягає в можливості використання розробленого застосунку для організації процесу готівкового обміну нестандартних валют між користувачами. Система може бути застосована як локальне програмне рішення для створення заявок, пошуку відповідних пропозицій, супроводу домовленостей між учасниками та отримання звітної інформації щодо виконаних операцій. Результати роботи також можуть бути використані як основа для подальшого розширення функціоналу системи, зокрема додавання механізмів рейтингу користувачів, перевірки надійності учасників, розширеної аналітики та інтеграції з зовнішніми фінансовими сервісами.

Пояснювальна записка складається зі вступу, чотирьох розділів, висновків, списку використаних джерел і додатків. У першому розділі розглянуто предметну область Р2Р-обміну валют, проаналізовано існуючі рішення та визначено вимоги до системи. У другому розділі виконано проєктування інформаційного та програмного забезпечення, розроблено модель даних, UML-діаграми й архітектуру системи. У третьому розділі описано реалізацію основних програмних модулів застосунку. У четвертому розділі наведено результати тестування програмного забезпечення та оцінювання ефективності його роботи.

Пояснювальна записка містить *77 сторінок, 31 рисунок, 17 таблиць, в 32 джерела, 3 додатки.*

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Предметна область дослідження охоплює процеси реєстрації користувачів, формування заявок на обмін, пошуку відповідних пропозицій, узгодження умов угоди та підтвердження факту передачі готівки в межах P2P-застосунку для обміну нестандартних валют. На відміну від класичних валютних сервісів, у такій системі основна взаємодія відбувається безпосередньо між користувачами, а програмне забезпечення виконує функції координації, обробки даних, збереження стану заявок і підтримки інформаційного супроводу угоди.

Ключовими об'єктами предметної області є користувач, профіль користувача, заявка на обмін, повідомлення, угода, статус угоди та довідкова інформація про валютний курс. Функціонування системи передбачає, що один користувач створює заявку із заданими параметрами обміну, а інший користувач виконує пошук доступних пропозицій, обирає відповідну заявку та ініціює запит на обмін. Після цього система забезпечує передачу повідомлень між сторонами, фіксацію підтверджень та оновлення стану угоди. Додатково передбачається участь адміністратора, який здійснює керування користувачами та розгляд спірних ситуацій, а також зовнішнього сервісу курсів валют, що надає довідкові значення для оцінки еквіваленту обміну.

На рисунку 1.1 наведено DFD-діаграму першого рівня інформаційної системи P2P-обміну готівки нестандартних валют.

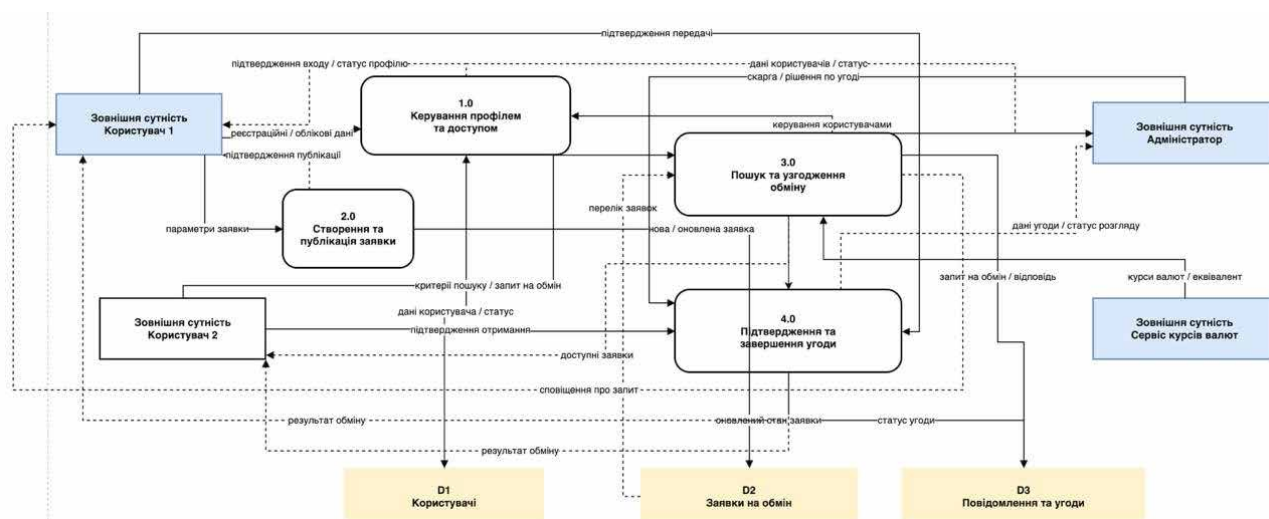


Рисунок 1.1 – DFD-діаграма першого рівня інформаційної системи P2P-обміну готівки нестандартних валют

DFD-модель відображає чотири основні процеси системи: 1.0 Керування профілем та доступом, 2.0 Створення та публікація заявки, 3.0 Пошук та узгодження обміну, 4.0 Підтвердження та завершення угоди. Зовнішніми сутностями є Користувач 1, Користувач 2, Адміністратор та Сервіс курсів валют. Для збереження інформації використовуються три сховища даних: D1 Користувачі, D2 Заявки на обмін, D3 Повідомлення та угоди.

Процес 1.0 Керування профілем та доступом приймає реєстраційні та облікові дані користувача, перевіряє їх, формує статус профілю та взаємодіє зі сховищем D1 Користувачі. Через цей же процес адміністратор отримує можливість керування користувачами. Процес 2.0 Створення та публікація заявки приймає від користувача параметри заявки, після чого записує нову або оновлену заявку до сховища D2 Заявки на обмін. Процес 3.0 Пошук та узгодження обміну обробляє критерії пошуку, надає перелік заявок, взаємодіє зі сховищем D2, формує запит на обмін і, за потреби, використовує дані сервісу курсів валют. Процес 4.0 Підтвердження та завершення угоди фіксує підтвердження сторін, оновлює статус угоди, зберігає інформацію у D3 Повідомлення та угоди та оновлює стан заявки у D2. Таким чином, DFD-діаграма описує інформаційні потоки між користувачами, адміністративним модулем, зовнішнім сервісом і внутрішніми сховищами даних.

На рисунку 1.2 наведено BPMN-діаграму бізнес-процесу обміну валют між користувачами.

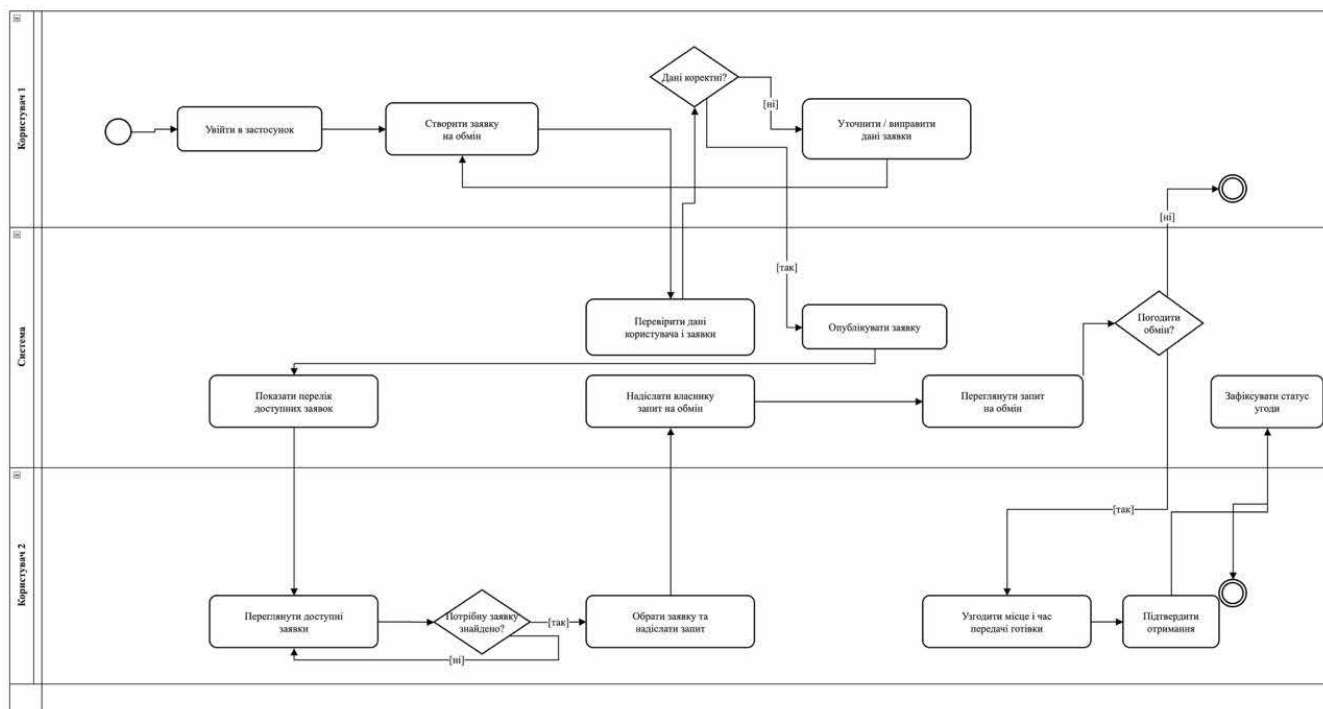


Рисунок 1.2 – BPMN-діаграма бізнес-процесу P2P-обміну готівки нестандартних валют

BPMN-модель поділена на три доріжки: Користувач 1, Система, Користувач 2, що дає змогу чітко розмежувати відповідальність учасників процесу. Процес починається у доріжці Користувач 1 з входу в застосунок і створення заявки на обмін. Далі керування переходить до доріжки Система, де виконується перевірка даних користувача і заявки. У точці розгалуження «Дані коректні?» реалізовано два сценарії: при негативному результаті користувач уточнює або виправляє дані, при позитивному — система публікує заявку.

Після публікації система відображає перелік доступних заявок для другого користувача. У доріжці Користувач 2 виконується перегляд доступних заявок та перевірка умови «Потрібну заявку знайдено?». Якщо потрібної заявки не знайдено, процес повертається до повторного перегляду. Якщо заявку знайдено, користувач обирає її та надсилає запит на обмін. Система передає цей запит власнику заявки, після чого у доріжці Користувач 1 виконується перегляд запиту та перевірка умови «Погодити обмін?». Якщо обмін не погоджено, процес

завершується без виконання угоди. Якщо погоджено, у доріжці Користувач 2 виконується узгодження місця і часу передачі готівки та підтвердження отримання коштів. Завершальна системна дія полягає у фіксації статусу угоди.

Для формалізації учасників предметної області доцільно виділити такі зовнішні сутності.

Таблиця 1.1 – Основні зовнішні сутності системи та їх функції

Позначення	Сутність	Функціональна роль
E1	Користувач 1	Реєстрація, створення заявки, погодження запиту
E2	Користувач 2	Пошук заявки, надсилання запиту, підтвердження отримання
E3	Адміністратор	Керування користувачами, розгляд спірних ситуацій
E4	Сервіс курсів валют	Надання довідкових курсів та валютного еквіваленту

Отже, предметна область системи являє собою набір взаємопов'язаних процесів обробки профілів, заявок, запитів та угод, у яких програмний застосунок виступає координуючим середовищем між двома користувачами, адміністратором і зовнішнім сервісом курсів валют. DFD-діаграма описує структуру даних і потоків, а BPMN-діаграма - послідовність виконання бізнес-процесу обміну.

1.2 Аналіз існуючих програмних рішень

У сфері цифрових фінансових сервісів існує значна кількість програмних платформ, що реалізують функції електронних платежів, міжнародних грошових переказів та P2P-взаємодії між користувачами. Такі системи дозволяють здійснювати обмін валют або переказ коштів між користувачами через мобільні застосунки або веб-платформи. Однак більшість з них орієнтовані на банківські транзакції або електронні платежі та не підтримують безпосередній обмін готівки між користувачами. До найбільш поширених сервісів, що частково

реалізують механізми P2P-фінансових операцій, належать Revolut, PayPal, CurrencyFair, Paxful, Deriv P2P.

Платформа Revolut є цифровим банківським сервісом, що забезпечує мультивалютні рахунки, міжнародні перекази та конвертацію валют у межах мобільного застосунку. Користувачі можуть здійснювати обмін валют за поточним курсом та надсилати кошти іншим користувачам системи. На рисунку 1.3 наведено приклад інтерфейсу веб-платформи Revolut.

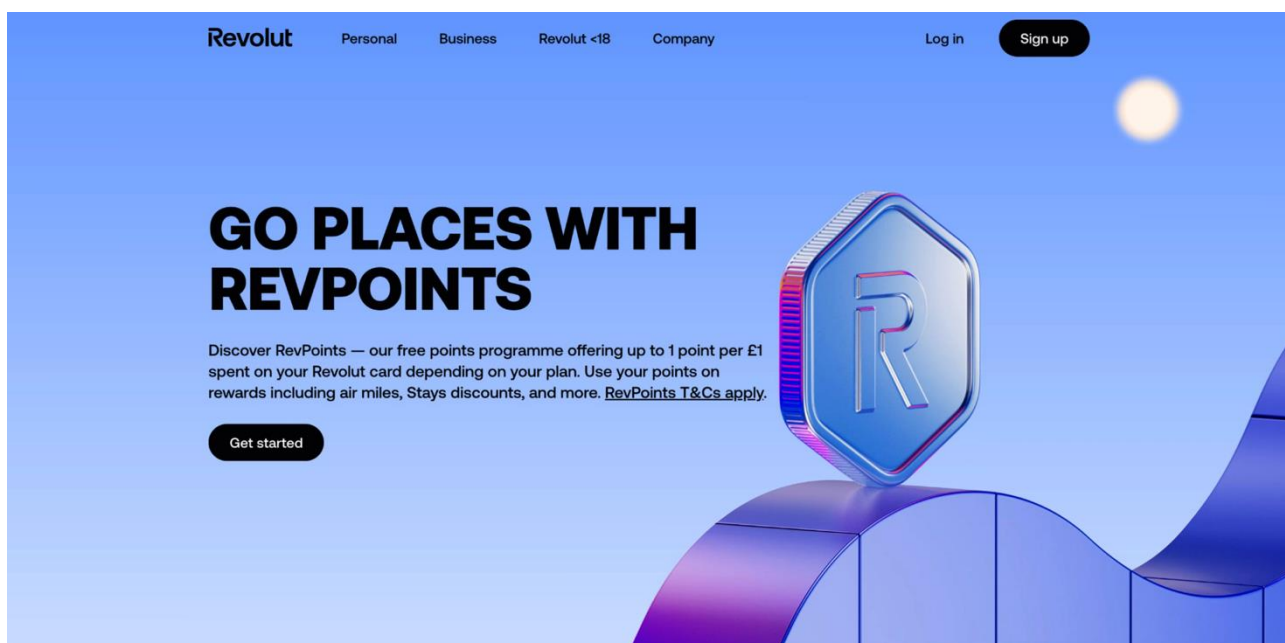


Рисунок 1.3 – Інтерфейс цифрового фінансового сервісу Revolut

Основною перевагою Revolut є підтримка великої кількості валют та швидкі транзакції між користувачами. Водночас сервіс функціонує як фінансова установа, а всі операції виконуються через банківські рахунки або електронні кошти, що не дозволяє організувати безпосередній обмін готівки між користувачами.

Іншим поширеним сервісом є PayPal, який забезпечує міжнародні електронні платежі та P2P-перекази між користувачами. Платформа підтримує інтеграцію з банківськими картками та електронними рахунками, дозволяючи виконувати швидкі перекази коштів у різних валютах. На рисунку 1.4 представлено приклад інтерфейсу мобільного застосунку PayPal.

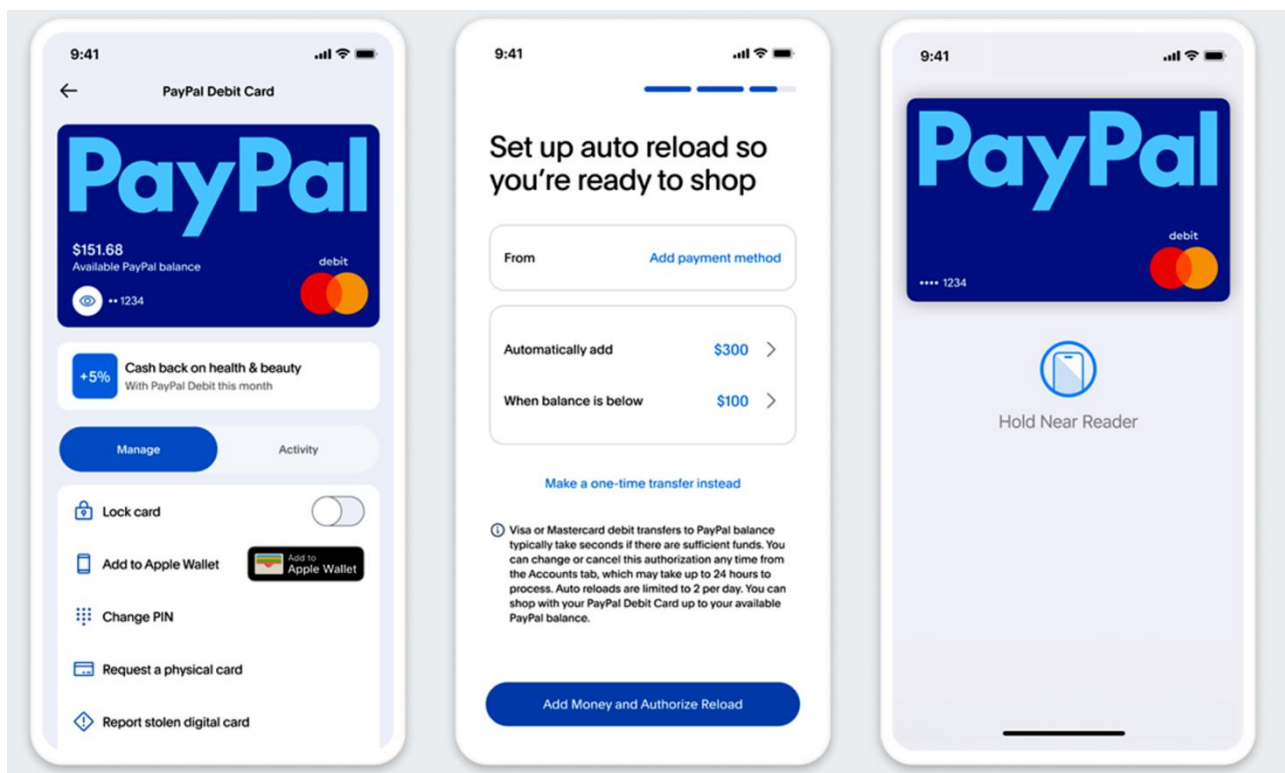


Рисунок 1.4 – Інтерфейс мобільного фінансового застосунку PayPal

Система PayPal забезпечує високий рівень безпеки транзакцій та зручний користувацький інтерфейс, однак усі операції здійснюються через електронний баланс або банківські інструменти. Таким чином, сервіс не підтримує механізмів узгодження фізичного обміну готівкових коштів між користувачами.

Платформа CurrencyFair реалізує модель P2P-обміну валют, у якій користувачі можуть здійснювати обмін безпосередньо між собою. Система виступає координатором процесу, забезпечуючи обробку заявок та зіставлення пропозицій користувачів. На рисунку 1.5 наведено приклад інтерфейсу системи CurrencyFair.

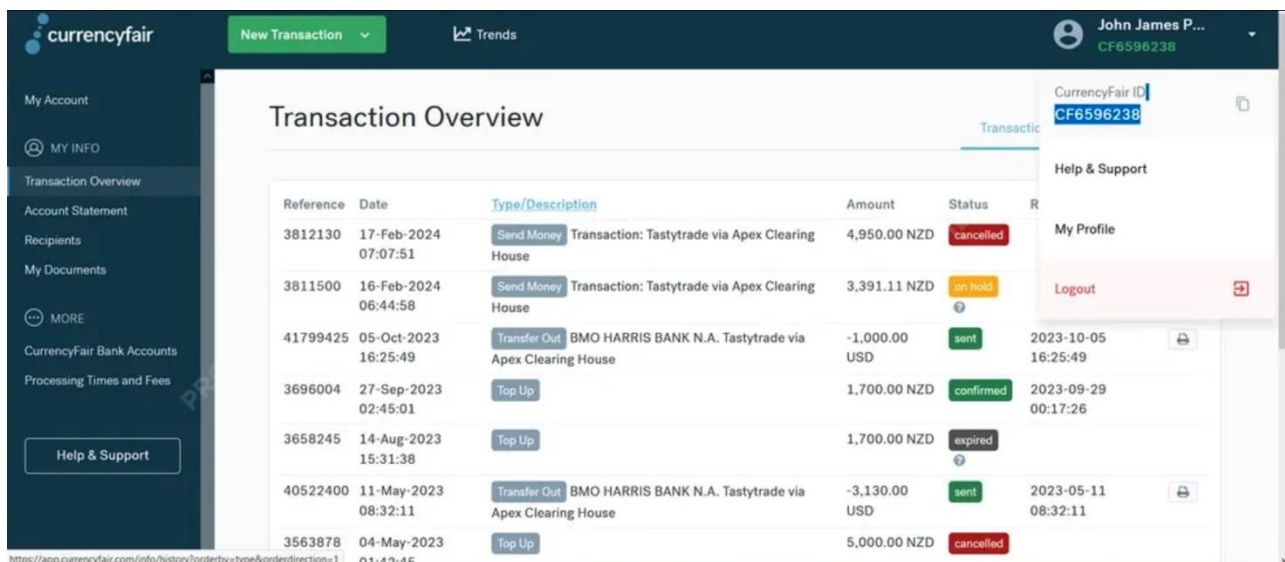


Рисунок 1.5 – Інтерфейс платформи P2P-обміну валют CurrencyFair

CurrencyFair дозволяє користувачам визначати власні курси обміну та знаходити відповідні пропозиції інших користувачів. Проте обмін коштів здійснюється через банківські рахунки, а не шляхом безпосередньої передачі готівки.

Ще одним прикладом P2P-платформи є Paxful, яка дозволяє користувачам здійснювати фінансові операції між собою з використанням різних способів оплати. Основною функцією системи є створення пропозицій обміну та пошук відповідних партнерів для виконання угоди. На рисунку 1.6 показано приклад веб-інтерфейсу сервісу Paxful.

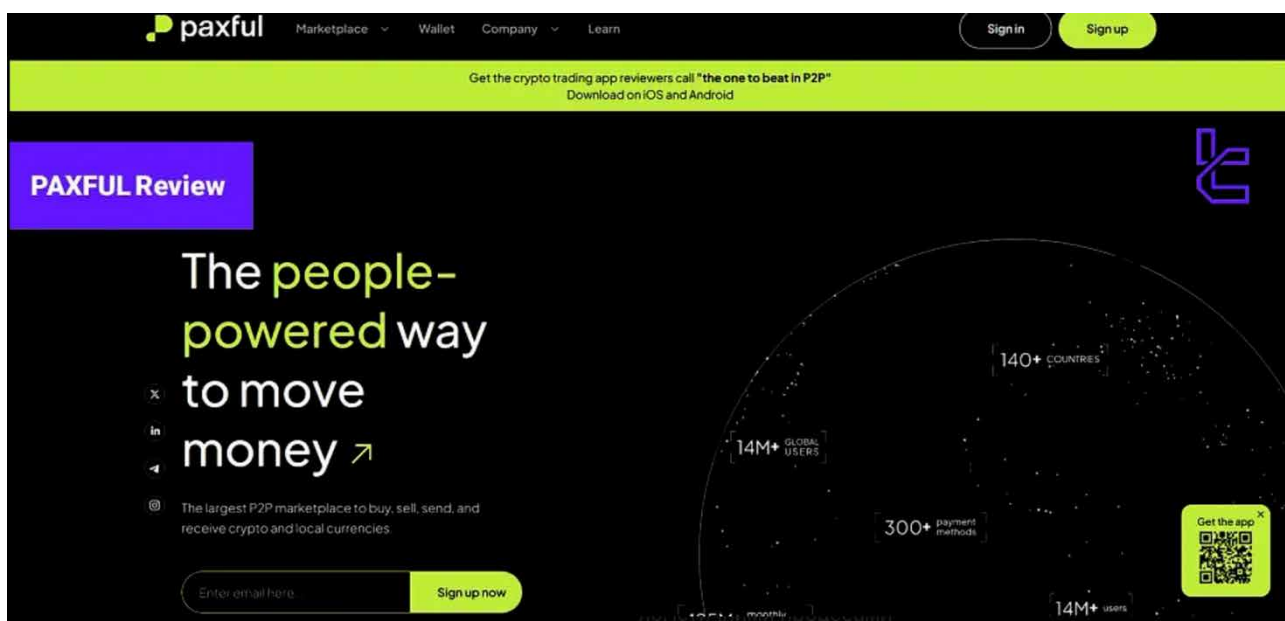


Рисунок 1.6 – Інтерфейс P2P-платформи Paxful

Raxful забезпечує гнучкі умови обміну та підтримує велику кількість платіжних методів, однак основна функціональність сервісу орієнтована на обмін криптовалюта та електронних активів, а не готівкових валют.

Сервіс Deriv P2P також реалізує модель взаємодії між користувачами, що дозволяє здійснювати фінансові операції без прямого банківського посередництва. Користувачі можуть створювати пропозиції купівлі або продажу валют та узгоджувати умови угоди. На рисунку 1.7 представлено інтерфейс користувацького профілю системи Deriv P2P.

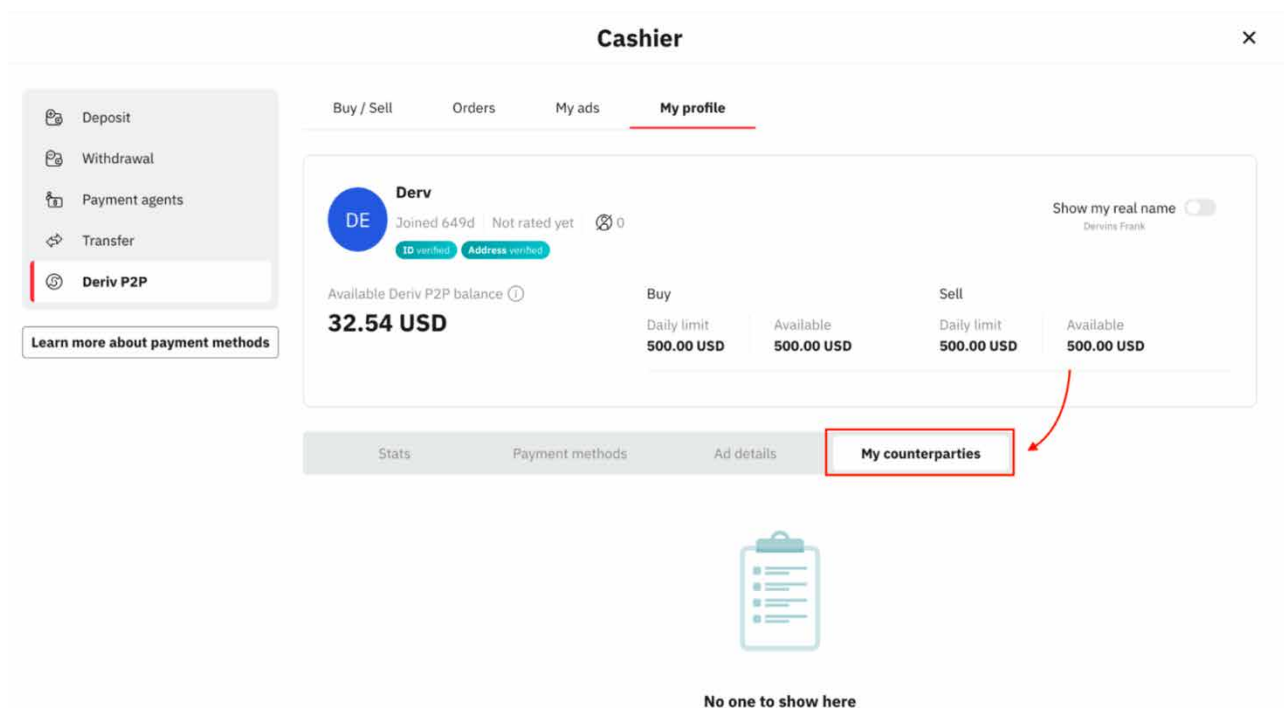


Рисунок 1.7 – Інтерфейс платформи P2P-обміну Deriv P2P

Deriv P2P дозволяє організувати взаємодію між користувачами та виконувати фінансові операції через різні платіжні інструменти. Однак система орієнтована переважно на електронні платежі та не передбачає повноцінного механізму організації обміну готівкових коштів між користувачами.

Для узагальнення функціональних можливостей розглянутих систем у таблиці 1.2 наведено порівняльний аналіз основних характеристик існуючих програмних рішень та розроблюваної системи.

Таблиця 1.2 – Порівняльний аналіз існуючих сервісів P2P-обміну

Характеристика	Revolut	PayPal	CurrencyFair	Paxful	Deriv P2P	Розроблювана система
P2P-перекази між користувачами	+	+	+	+	+	+
Підтримка багатьох валют	+	+	+	+	+	+
Створення заявок на обмін	—	—	+	+	+	+
Узгодження умов угоди	—	—	+	+	+	+
Обмін готівки між користувачами	—	—	—	—	—	+
Інтеграція сервісу курсів валют	+	+	+	+	+	+
Керування статусом угоди	Обмежено	Обмежено	Частково	Частково	Частково	Повністю

Проведений аналіз показує, що більшість сучасних фінансових сервісів реалізують лише окремі аспекти P2P-взаємодії, зосереджуючись переважно на електронних платежах та банківських операціях. Натомість вони не забезпечують повноцінної підтримки сценаріїв обміну нестандартних валют у форматі готівкових P2P-угод між користувачами.

Запропонована в роботі інформаційна система орієнтована на створення спеціалізованого середовища для організації такого обміну. Вона поєднує механізми створення заявок, пошуку відповідних пропозицій, узгодження умов

угоди та фіксації її результатів у межах єдиного програмного середовища, що дозволяє підвищити зручність і ефективність взаємодії між користувачами.

1.3 Моделювання предметної області

Моделювання предметної області інформаційної системи «Фінансовий помічник для P2P-обміну готівки нестандартних валют» виконано з метою формалізації основних функціональних процесів системи, визначення ролей користувачів та опису сценаріїв взаємодії між учасниками платформи. Для цього використано UML-моделі, які дозволяють структуровано представити функціональні можливості системи, послідовність виконання операцій та логіку бізнес-процесів, пов'язаних з організацією обміну валют між користувачами.

Розроблювана система забезпечує можливість створення заявок на обмін готівкових валют, пошуку відповідних пропозицій, узгодження умов угоди між користувачами та фіксації результатів обміну. Крім того, система інтегрується із сервісом курсів валют для отримання актуальних значень валютних еквівалентів.

На рис. 1.8 наведено діаграму прецедентів, яка відображає функціональні можливості системи та взаємодію акторів із програмною платформою. Основними акторами системи є користувач, інший користувач, адміністратор та сервіс курсів валют.

Користувач може виконувати реєстрацію та вхід у систему, керувати власним профілем, створювати заявки на обмін валют, переглядати доступні заявки інших користувачів, узгоджувати умови угоди, обмінюватися повідомленнями та підтверджувати передачу готівки. Інший користувач має аналогічні можливості щодо перегляду заявок та ініціювання обміну.

Адміністратор виконує функції контролю платформи, зокрема керує заявками та скаргами користувачів, а також здійснює модерацію користувачів. Зовнішній сервіс курсів валют забезпечує отримання актуальних значень курсів, які використовуються системою для визначення валютного еквіваленту під час створення заявок.

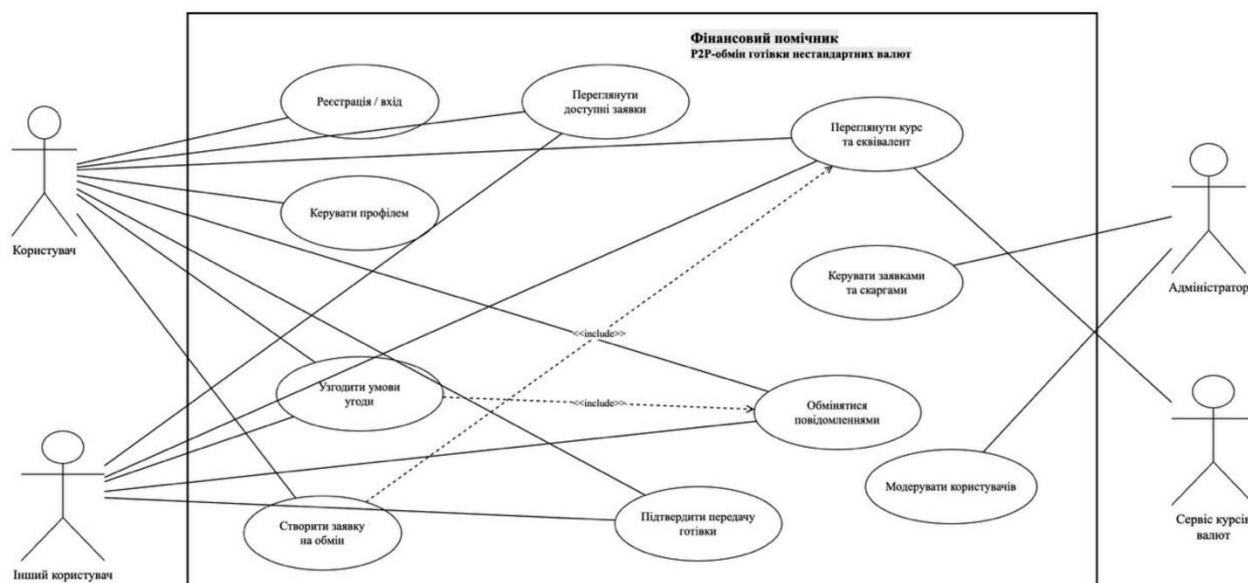


Рисунок 1.8 – Діаграма прецедентів системи P2P-обміну готівки нестандартних валют

На рис. 1.9 представлено діаграму послідовності, яка відображає сценарій взаємодії між користувачами під час виконання обміну валют. Процес починається з входу користувача до системи та створення заявки на обмін. Після цього система виконує перевірку введених даних, зберігає заявку та публікує її у списку доступних пропозицій.

Інший користувач переглядає доступні заявки, обирає відповідну пропозицію та надсилає запит на обмін. Платформа передає сповіщення власнику заявки, після чого користувачі можуть узгодити умови угоди. Якщо сторони погоджують умови, вони підтверджують передачу готівки, а система фіксує результат обміну. У разі відмови або непогодження умов запит може бути скасований, а заявка залишається активною.

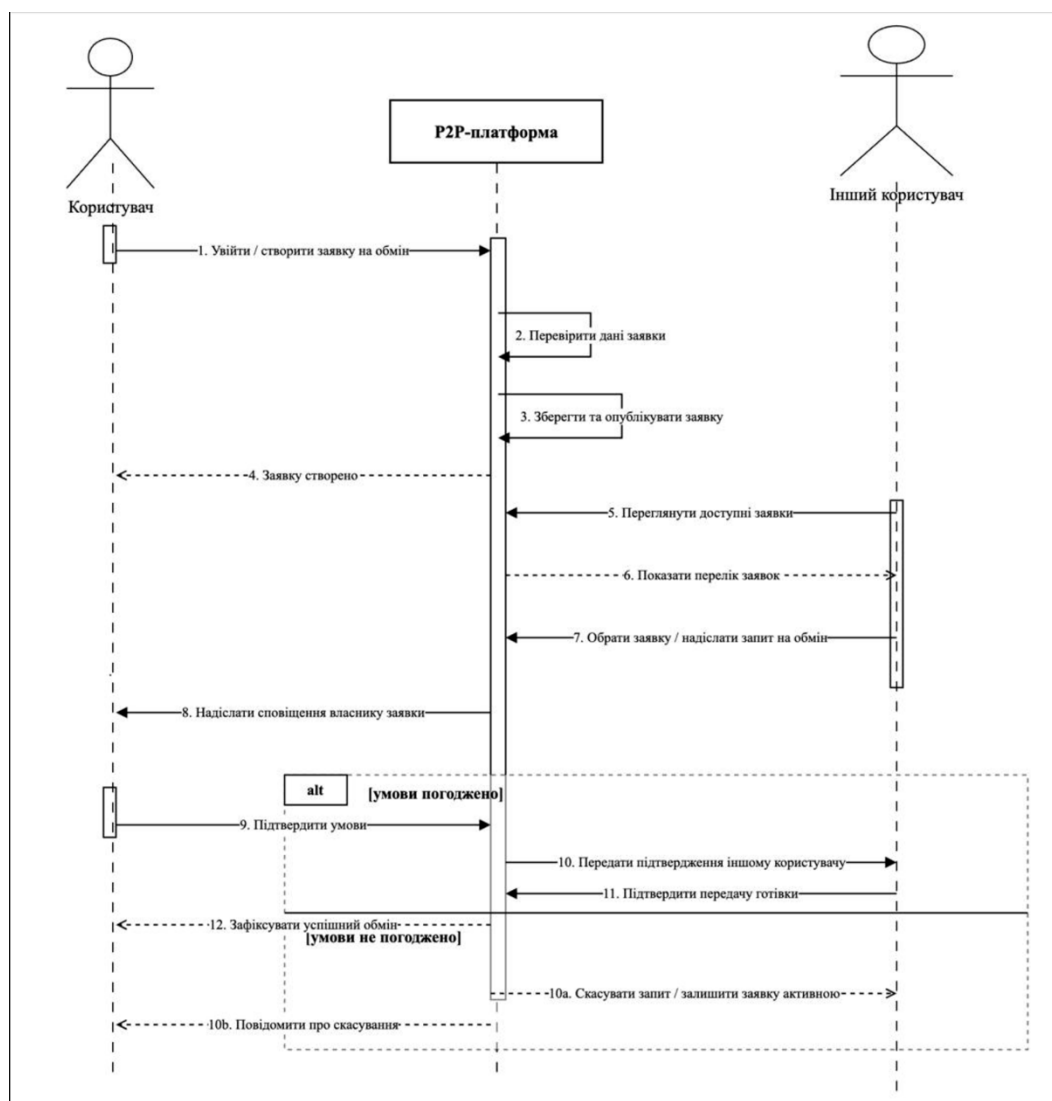


Рисунок 1.9 – Діаграма послідовності виконання P2P-обміну валют між користувачами

На рис. 1.10 подано діаграму активності, яка відображає бізнес-процес виконання обміну валют у системі. Діаграма демонструє послідовність дій користувачів та системи починаючи з входу до застосунку та створення заявки на обмін.

Після введення даних система виконує перевірку коректності інформації. Якщо дані введено правильно, заявка публікується у системі та стає доступною для інших користувачів. Інший користувач може переглянути список заявок та знайти відповідну пропозицію. У разі вибору заявки надсилається запит на обмін.

Після цього користувачі узгоджують місце та час передачі готівки. Після фактичної передачі коштів сторони підтверджують виконання угоди, а система фіксує успішний обмін. У випадку відмови або скасування угоди заявка може залишитися активною або бути видалена.

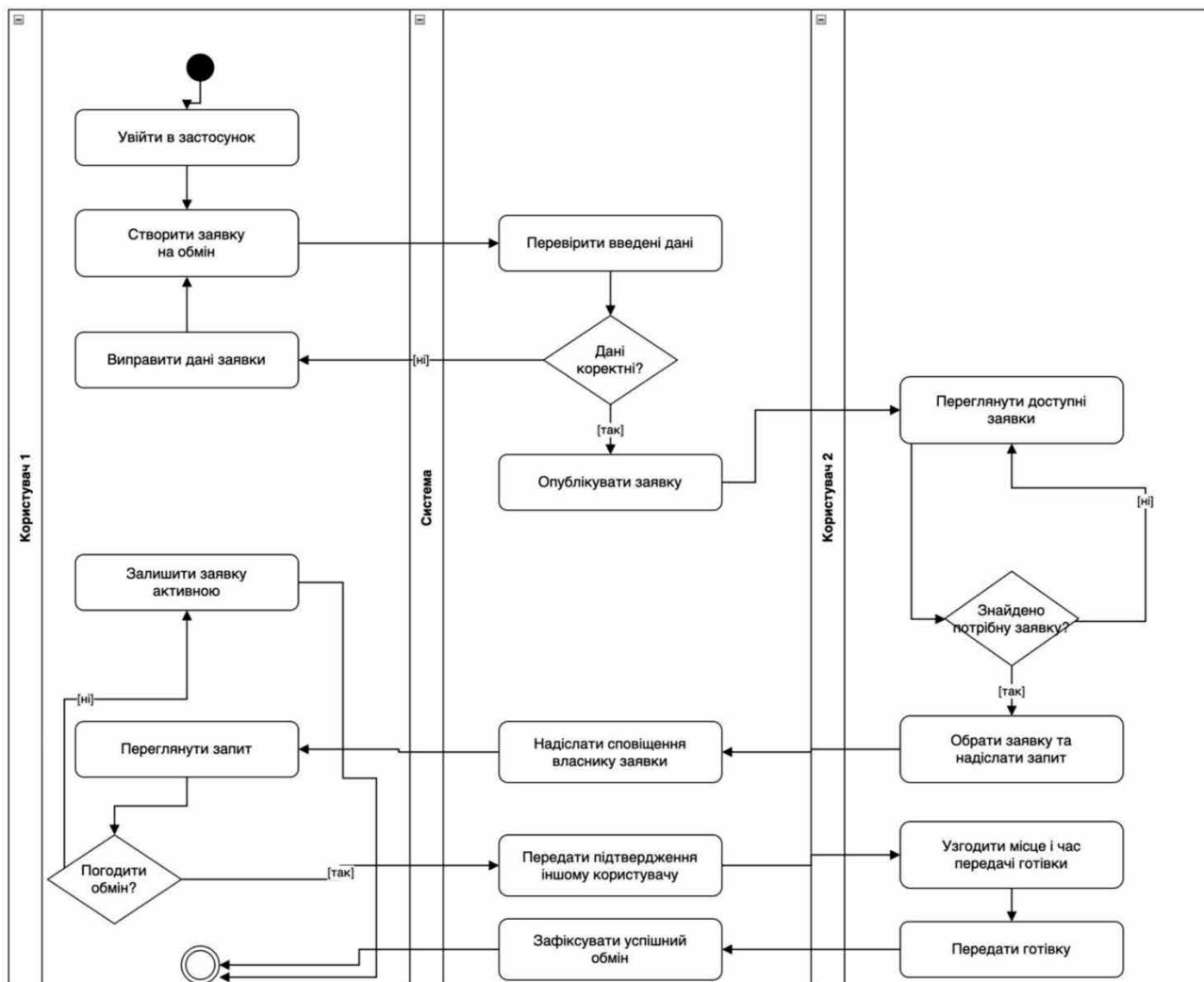


Рисунок 1.10 – Діаграма активності процесу P2P-обміну валют

Таким чином, побудовані UML-діаграми дозволяють формалізувати логіку роботи системи та відобразити основні сценарії взаємодії між користувачами платформи. Отримані моделі є основою для подальшого проєктування архітектури програмного забезпечення та реалізації інформаційної системи для P2P-обміну готівкових валют.

1.4 Аналіз вимог до інформаційної системи P2P-обміну готівки нестандартних валют

Аналіз вимог до інформаційної системи «Фінансовий помічник для P2P-обміну готівки нестандартних валют» спрямований на формалізацію функціональних можливостей програмного забезпечення, визначення експлуатаційних характеристик системи, а також встановлення вимог до інформаційної безпеки з урахуванням особливостей фінансових операцій між користувачами. Формування вимог здійснено на основі проведеного аналізу предметної області, побудованих UML-моделей та дослідження існуючих програмних рішень у сфері P2P-платежів та обміну валют.

Основним призначенням системи є забезпечення зручної платформи для взаємодії користувачів, які бажають здійснити обмін готівкових валют між собою. Система повинна підтримувати створення та пошук заявок на обмін, узгодження умов угоди між користувачами, обмін повідомленнями, а також фіксацію результатів виконаної операції.

До функціональних вимог системи належать можливості керування користувацькими обліковими записами, створення та публікації заявок на обмін валют, перегляду доступних пропозицій інших користувачів, узгодження умов обміну, підтвердження передачі готівки та отримання актуальних курсів валют. Сукупність функціональних вимог наведено у табл. 1.3, де визначено основні функції системи та їх призначення.

Таблиця 1.3 – Функціональні вимоги інформаційної системи P2P-обміну валют

№	Назва вимоги	Опис
F1	Реєстрація та вхід	Створення облікового запису користувача та авторизація в системі
F2	Керування профілем	Редагування персональних даних користувача
F3	Створення заявки	Формування та публікація заявки на обмін валют
F4	Перегляд заявок	Перегляд доступних пропозицій інших користувачів
F5	Пошук обміну	Вибір відповідної заявки для обміну

Продовження таблиці 1.3

F6	Узгодження умов	Узгодження місця, часу та умов обміну між користувачами
F7	Обмін повідомленнями	Передавання повідомлень між учасниками угоди
F8	Підтвердження обміну	Фіксація факту передачі готівки між користувачами
F9	Курс валют	Отримання актуальних курсів валют із зовнішнього сервісу
F10	Модерація	Керування заявками, скаргами та користувачами адміністратором

Нефункціональні вимоги визначають технічні та експлуатаційні характеристики системи, які забезпечують її стабільну роботу, масштабованість та зручність використання. Система повинна забезпечувати швидкий доступ до інформації, підтримувати одночасну роботу багатьох користувачів, а також мати інтуїтивно зрозумілий інтерфейс для роботи з платформою. Узагальнений перелік нефункціональних вимог наведено у табл. 1.4.

Таблиця 1.4 – Нефункціональні вимоги системи

№	Категорія	Вимога
NF1	Продуктивність	Час оброблення заявки або запиту — не більше 2–3 с
NF2	Масштабованість	Можливість подальшого масштабування багатокористувацької архітектури
NF3	Надійність	Забезпечення збереження даних при системних збоях
NF5	Зручність	Простий та інтуїтивно зрозумілий інтерфейс користувача
NF6	Сумісність	Локальна робота на операційних системах Windows, macOS, Linux
NF7	Модульність	Можливість розширення функціоналу системи

Оскільки система оперує фінансовою інформацією користувачів, особливу увагу приділено вимогам до інформаційної безпеки. Система повинна

забезпечувати захист персональних даних користувачів, контроль доступу до функціональних модулів, а також безпечну передачу інформації між компонентами програмного забезпечення.

Основні вимоги до безпеки наведено у табл. 1.5.

Таблиця 1.5 – Вимоги до інформаційної безпеки системи

№	Вимога	Опис
S1	Автентифікація	Перевірка особи користувача під час входу в систему
S2	Авторизація	Розмежування доступу до функцій відповідно до ролей
S3	Захищене з'єднання	Шифрування передачі даних через HTTPS
S4	Контроль доступу	HTTPS — для майбутнього клієнт-серверної версії
S5	Журналювання	Фіксація дій користувачів у системному журналі
S6	Захист повідомлень	Безпечний обмін повідомленнями між користувачами
S7	Захист від зловживань	Запобігання несанкціонованим операціям та шахрайству

Узагальнення результатів аналізу вимог показує, що сформований набір функціональних, нефункціональних та безпекових вимог забезпечує комплексну реалізацію інформаційної системи для організації P2P-обміну готівкових валют між користувачами. Реалізація цих вимог дозволить створити надійну платформу для взаємодії користувачів, підвищити зручність пошуку партнерів для обміну та забезпечити безпечне виконання фінансових операцій.

1.5 Постановка завдання

На підставі проведеного аналізу предметної області, дослідження існуючих програмних рішень, побудованих UML-моделей та сформованих функціональних і нефункціональних вимог (пункти 1.1–1.4) у даній кваліфікаційній роботі ставиться завдання розроблення інформаційної системи «Фінансовий помічник для P2P-обміну готівки нестандартних валют», яка забезпечує організацію взаємодії користувачів для здійснення обміну валют у форматі peer-to-peer.

Основною метою розроблюваної системи є створення програмної платформи, що дозволяє користувачам знаходити партнерів для обміну валют, створювати та публікувати заявки на обмін, узгоджувати умови угоди та фіксувати результат виконаної операції. Система повинна забезпечувати централізоване управління заявками на обмін, підтримку комунікації між користувачами, інтеграцію із зовнішнім сервісом курсів валют та контроль виконання угод.

Для досягнення поставленої мети у межах роботи необхідно розв'язати такі основні завдання:

- розробити модель інформаційної системи для організації P2P-обміну готівкових валют між користувачами;
- реалізувати механізм реєстрації та автентифікації користувачів у системі;
- забезпечити можливість створення, редагування та публікації заявок на обмін валют;
- реалізувати функції перегляду та пошуку доступних заявок інших користувачів;
- забезпечити механізм узгодження умов угоди між учасниками обміну;
- реалізувати систему обміну повідомленнями між користувачами платформи;

- забезпечити інтеграцію із зовнішнім сервісом курсів валют для визначення валютного еквіваленту;
- реалізувати механізм підтвердження передачі готівки та фіксації результатів обміну;
- передбачити функції адміністративного керування заявками, користувачами та скаргами.

Вхідні та вихідні дані системи визначають інформаційні потоки, що формуються під час взаємодії користувачів із платформою та зовнішніми сервісами. Узагальнений перелік основних типів даних, які використовуються у процесі функціонування системи, наведено в табл. 1.6.

Таблиця 1.6 – Вхідні та вихідні дані інформаційної системи P2P-обміну валют

№	Категорія даних	Опис
Вхідні дані		
I1	Дані користувачів	Реєстраційні дані, контактна інформація, параметри профілю
I2	Дані заявки	Валюта обміну, сума, курс, місце та умови передачі
I3	Дані запиту	Інформація про запит іншого користувача на обмін
I4	Дані повідомлень	Повідомлення між користувачами під час узгодження умов
I5	Дані курсів валют	Інформація, отримана із зовнішнього сервісу курсів валют
Вихідні дані		
O1	Список доступних заявок	Перелік активних пропозицій обміну
O2	Статус заявки	Поточний стан заявки (активна, погоджена, завершена)
O3	Підтвердження угоди	Фіксація факту передачі готівки між користувачами
O4	Повідомлення	Системні та користувацькі повідомлення
O5	Інформація про курс	Актуальний валютний еквівалент для користувача

У результаті виконання кваліфікаційної роботи має бути створена програмна система, яка забезпечить зручну та безпечну платформу для організації P2P-обміну готівкових валют між користувачами. Реалізована система повинна забезпечувати швидкий пошук партнерів для обміну, ефективну комунікацію між учасниками угоди, контроль виконання операцій та підтримку актуальної інформації про валютні курси.

Отримані результати мають практичну цінність для створення цифрових сервісів у сфері P2P-фінансів, оскільки розроблювана система дозволяє підвищити зручність організації валютного обміну між користувачами, спростити процес пошуку відповідних пропозицій та забезпечити прозорість виконання фінансових операцій.

Висновки до розділу 1

У першому розділі кваліфікаційної роботи проведено аналіз предметної області створення інформаційної системи P2P-обміну готівки нестандартних валют, що дозволило визначити основні особливості організації процесу обміну валют між користувачами та сформулювати концептуальну основу для розроблення програмного забезпечення.

У межах дослідження предметної області встановлено, що сучасні фінансові сервіси здебільшого орієнтовані на електронні платежі та банківські транзакції, тоді як процеси безпосереднього обміну готівкових валют між користувачами залишаються недостатньо автоматизованими. Визначено основні учасники процесу обміну, інформаційні потоки між ними та ключові операції, які повинна підтримувати інформаційна система, зокрема створення заявок на обмін, пошук відповідних пропозицій, узгодження умов угоди та підтвердження передачі коштів.

У результаті аналізу існуючих програмних рішень, таких як Revolut, PayPal, CurrencyFair, Paxful та Deriv P2P, встановлено, що більшість сучасних платформ реалізують лише окремі функції P2P-взаємодії. Основна частина таких

систем орієнтована на електронні платежі або обмін цифрових активів і не підтримує повноцінну організацію обміну готівкових валют між користувачами. Це підтверджує актуальність розроблення спеціалізованої інформаційної системи, що забезпечує підтримку таких сценаріїв взаємодії.

У процесі моделювання предметної області побудовано UML-діаграми, зокрема діаграму прецедентів, діаграму послідовності та діаграму активності, які відображають функціональні можливості системи, сценарії взаємодії користувачів та логіку виконання основних бізнес-процесів. Побудовані моделі дозволили формалізувати структуру системи, визначити ролі користувачів і встановити основні етапи виконання операцій обміну валют.

На основі проведеного аналізу сформовано функціональні, нефункціональні та безпекові вимоги до програмної системи. Визначено ключові функції системи, серед яких реєстрація користувачів, керування профілем, створення та пошук заявок на обмін, узгодження умов угоди, обмін повідомленнями між користувачами, інтеграція із сервісом курсів валют та підтвердження виконання обміну. Також визначено вимоги до продуктивності, надійності, масштабованості та інформаційної безпеки системи.

На основі отриманих результатів сформульовано постановку завдання кваліфікаційної роботи, яка полягає у розробленні інформаційної системи «Фінансовий помічник для P2P-обміну готівки нестандартних валют», що забезпечує організацію взаємодії користувачів, автоматизацію процесу створення та оброблення заявок на обмін, підтримку комунікації між учасниками угоди та фіксацію результатів виконання обмінних операцій.

Отримані результати першого розділу створюють теоретичну та методичну основу для подальшого проєктування архітектури системи, розроблення програмних компонентів і реалізації інформаційної платформи для організації P2P-обміну валют між користувачами.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних інформаційної системи Р2Р-обміну готівки нестандартних валют

Логічна модель даних визначає структуру інформаційних об'єктів інформаційної системи «Фінансовий помічник для Р2Р-обміну готівки нестандартних валют», а також зв'язки між ними. Побудова логічної моделі є важливим етапом проєктування програмного забезпечення, оскільки дає змогу формалізувати основні сутності предметної області, визначити їх ключові атрибути та встановити правила взаємодії між даними у межах системи.

Логічна модель даних сформована з урахуванням процесів реєстрації користувачів, створення заявок на обмін, вибору валют, узгодження умов угоди, обміну повідомленнями та фіксації результату передачі готівки. У моделі використано первинні та зовнішні ключі, що забезпечують цілісність даних і дозволяють однозначно ідентифікувати користувачів, заявки, угоди, повідомлення, валюти та курси валют. Структура моделі побудована таким чином, щоб уникнути дублювання інформації та забезпечити подальшу реалізацію реляційної бази даних.

На рисунку 2.1 представлено логічну модель даних інформаційної системи Р2Р-обміну готівки нестандартних валют. Діаграма відображає основні інформаційні сутності системи, їх атрибути та зв'язки, які формують основу інформаційного забезпечення програмного застосунку.

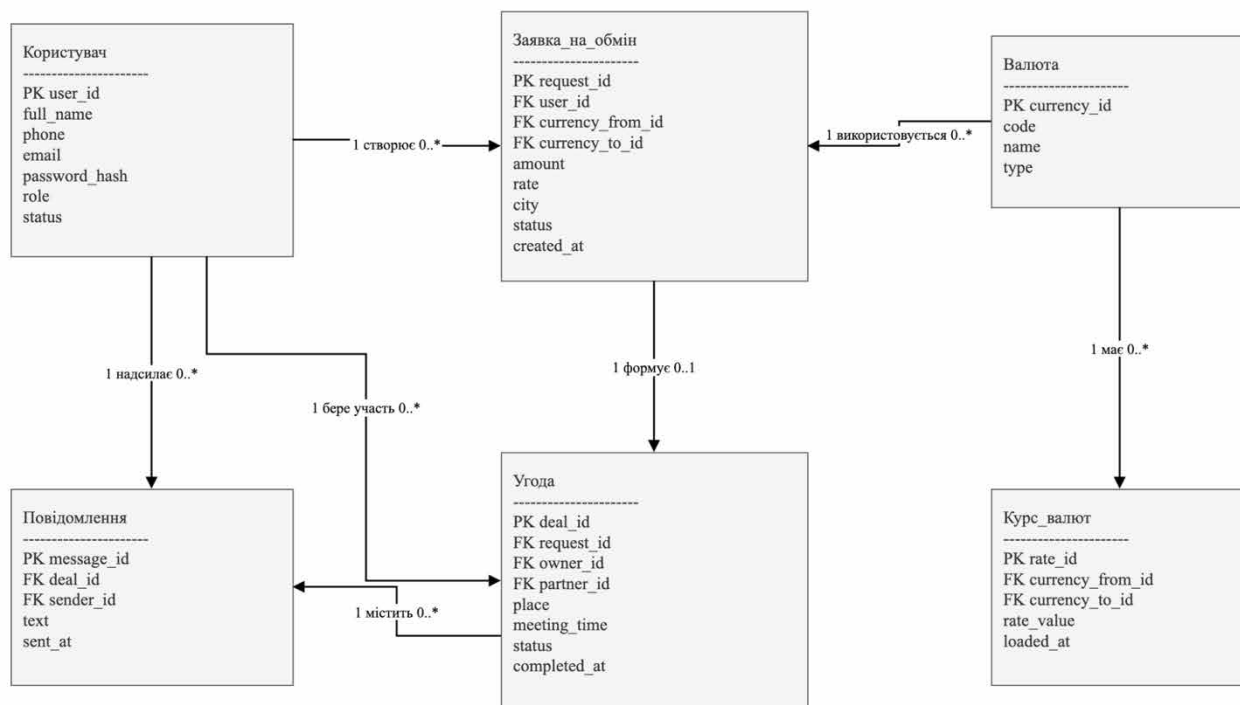


Рисунок 2.1- Логічна модель даних інформаційної системи P2P-обміну готівки нестандартних валют

Під час побудови логічної моделі враховано вимоги нормалізації бази даних. На рівні першої нормальної форми всі атрибути сутностей подано в атомарному вигляді, тобто кожне поле зберігає одне значення й не містить повторюваних груп даних. Наприклад, дані користувача, параметри заявки, повідомлення, валюта обміну та статус угоди зберігаються в окремих атрибутах. Це спрощує пошук, фільтрацію та оновлення інформації в базі даних.

Вимоги другої нормальної форми забезпечено тим, що кожна сутність має власний первинний ключ, а неключові атрибути залежать від нього повністю. Наприклад, у сутності «Заявка_на_обмін» такі атрибути, як сума, запропонований курс, місто, статус і дата створення, описують саме конкретну заявку, а не користувача або валюту загалом. Аналогічно у сутності «Повідомлення» текст повідомлення та час надсилання належать до конкретного повідомлення, що визначається його ідентифікатором.

Вимоги третьої нормальної форми реалізовано шляхом винесення довідкових і повторюваних даних в окремі сутності. Зокрема, інформація про валюти зберігається в окремій сутності «Валюта», а курси валют — у сутності

«Курс_валют». Це дозволяє уникнути дублювання назв і кодів валют у заявках та угодах. Дані про ролі користувачів і статуси також доцільно зберігати як окремі довідкові значення або обмежені переліки, що зменшує кількість транзитивних залежностей і підвищує цілісність даних. Таким чином, структура бази даних відповідає основним вимогам 1НФ, 2НФ і 3НФ, що забезпечує логічну впорядкованість моделі та зменшує ризик появи аномалій вставлення, оновлення й видалення даних.

Центральним елементом моделі є сутність «Заявка_на_обмін», яка описує пропозицію користувача щодо обміну певної суми однієї валюти на іншу. Вона містить ідентифікатор заявки, посилання на користувача, валюти обміну, суму, запропонований курс, місто, статус і дату створення. Кожна заявка створюється одним користувачем, проте один користувач може створювати багато заявок. Такий зв'язок відображає реальний сценарій роботи системи, коли учасник платформи має можливість одночасно розміщувати декілька пропозицій обміну.

Сутність «Користувач» зберігає основні облікові дані учасників платформи: ідентифікатор, повне ім'я, телефон, електронну пошту, хеш пароля, роль і статус. Користувач може створювати заявки, брати участь в угодах і надсилати повідомлення іншим учасникам. Завдяки наявності ролі в моделі можна розмежувати звичайних користувачів і адміністративних учасників системи, які виконують функції контролю та модерації.

Сутність «Угода» описує процес фактичного узгодження та виконання обміну між двома користувачами. Вона пов'язана із заявкою на обмін і містить дані про власника заявки, партнера обміну, місце зустрічі, час зустрічі, статус угоди та дату завершення. Зв'язок між заявкою та угодою має обмеження «один до нуля або одного», оскільки не кожна заявка обов'язково переходить в активну або завершену угоду. Якщо користувачі погодили умови обміну, на основі заявки формується відповідна угода.

Сутність «Повідомлення» використовується для збереження комунікації між учасниками обміну. Вона містить ідентифікатор повідомлення, посилання на угоду, відправника, текст повідомлення та час надсилання. Повідомлення

пов'язуються з конкретною угодою, що дозволяє зберігати історію обговорення умов обміну, місця зустрічі та підтвердження домовленостей.

Окрему роль у моделі виконують сутності «Валюта» та «Курс_валют». Сутність «Валюта» містить довідкову інформацію про грошові одиниці, зокрема код, назву та тип валюти. Сутність «Курс_валют» зберігає значення курсу між двома валютами, дату завантаження та посилання на відповідні валютні одиниці. Така структура дає змогу використовувати актуальні довідкові курси для орієнтовного розрахунку еквіваленту під час створення заявки на обмін, не дублюючи валютні дані в інших таблицях

Основні інформаційні сутності логічної моделі та їх призначення узагальнено у таблиці 2.1.

Таблиця 2.1 - Основні сутності логічної моделі даних системи

Сутність	Призначенн
Користувач	Зберігання облікових і контактних даних користувачів, їх ролей та поточного статусу в системі
Заявка_на_обмін	Опис пропозиції користувача щодо обміну готівки із зазначенням валют, суми, курсу, міста та статусу заявки
Угода	Фіксація погодженого обміну між двома користувачами, місця та часу зустрічі, статусу виконання і дати завершення
Повідомлення	Збереження повідомлень між учасниками угоди під час узгодження умов обміну
Валюта	Зберігання довідкової інформації про валюти, що використовуються у заявках та курсах
Курс_валют	Зберігання довідкових курсів між валютами для розрахунку орієнтовного еквіваленту обміну

Запропонована логічна модель даних забезпечує чітке розмежування основних інформаційних об'єктів системи та відображає їх роль у процесі P2P-обміну готівки нестандартних валют. Використання зв'язків типу «один-до-багатьох» між користувачами, заявками, повідомленнями та курсами валют дозволяє коректно моделювати реальні сценарії роботи платформи. Зв'язок між заявкою та угодою забезпечує фіксацію переходу від опублікованої пропозиції до погодженого обміну між двома користувачами.

Отже, сформована логічна модель даних є основою для подальшого проєктування фізичної структури бази даних, визначення таблиць, ключів, обмежень цілісності та реалізації програмних модулів інформаційної системи. Вона забезпечує узгоджене зберігання даних про користувачів, заявки, угоди, повідомлення, валюти та курси валют, що є необхідним для стабільної роботи застосунку « “Фінансовий помічник” для організації готівкових P2P-обмінів» у межах організації P2P-обміну готівки нестандартних валют.

2.2 Діаграма класів і кооперації системи

Проектування програмної структури інформаційної системи «Фінансовий помічник для P2P-обміну готівки нестандартних валют» передбачає визначення основних програмних класів, їх атрибутів, операцій та взаємозв'язків. Для формалізації статичної структури системи використано UML-діаграму класів, яка дає змогу відобразити основні об'єкти предметної області та логіку їх взаємодії під час створення заявок, пошуку пропозицій, узгодження умов обміну, передавання повідомлень і завершення угоди.

Діаграма класів системи представлена на рисунку 2.2. Вона відображає програмну модель застосунку, у якій основними класами є User, UserProfile, ExchangeOrder, ExchangeDeal, Message, Complaint, Currency та CurrencyRate. Така структура відповідає функціональним вимогам системи, оскільки охоплює облікові записи користувачів, профільні дані, заявки на обмін, угоди, повідомлення, скарги, довідник валют і курси валют.

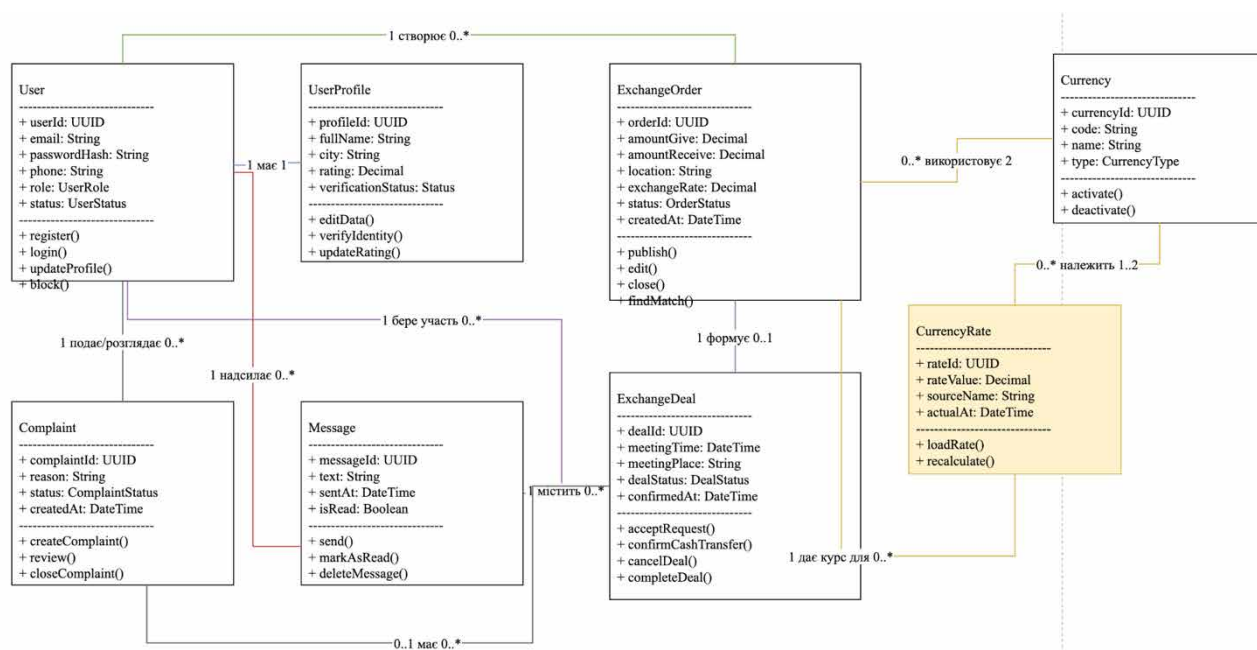


Рисунок 2.2 – UML-діаграма класів інформаційної системи P2P-обміну готівки нестандартних валют

Клас `User` є базовим класом для представлення користувача системи. Він містить ідентифікатор, електронну пошту, хеш пароля, номер телефону, роль і статус користувача. Через операції `register()`, `login()`, `updateProfile()` та `block()` реалізуються основні дії, пов'язані з реєстрацією, входом у систему, оновленням даних і блокуванням облікового запису. Один користувач має один профіль, може створювати багато заявок на обмін, брати участь у багатьох угодах, надсилати повідомлення та подавати або розглядати скарги.

Клас `UserProfile` деталізує профіль користувача та зберігає його повне ім'я, місто, рейтинг і статус верифікації. Наявність окремого класу профілю дозволяє відокремити облікові дані для входу від розширеної інформації, яка використовується під час вибору партнера для обміну. Операції `editData()`, `verifyIdentity()` та `updateRating()` забезпечують редагування профілю, перевірку користувача та оновлення рейтингу після завершення угод.

Клас `ExchangeOrder` описує заявку на обмін готівки. Він містить ідентифікатор заявки, суму валюти, яку користувач віддає, суму валюти, яку очікує отримати, місце обміну, запропонований курс, статус заявки та дату створення. Основними операціями цього класу є `publish()`, `edit()`, `close()` та

findMatch(). Вони забезпечують публікацію заявки, її редагування, закриття після завершення обміну та пошук відповідних пропозицій. Кожна заявка створюється одним користувачем і використовує дві валюти: валюту, яку користувач віддає, та валюту, яку бажає отримати.

Клас Currency виконує роль довідника валют, що використовуються у заявках та курсах. Він містить код валюти, її назву та тип. Операції activate() і deactivate() призначені для керування доступністю валюти в системі. Із класом Currency пов'язаний клас CurrencyRate, який містить значення курсу, джерело отримання даних і час актуальності курсу. Це дозволяє використовувати довідкові курси валют під час створення заявки та орієнтовного розрахунку еквіваленту обміну.

Клас ExchangeDeal відображає погоджену угоду між двома користувачами. Він містить ідентифікатор угоди, місце зустрічі, час зустрічі, статус угоди та дату підтвердження. Операції acceptRequest(), confirmCashTransfer(), cancelDeal() і completeDeal() описують основні етапи життєвого циклу угоди: прийняття запиту, підтвердження передачі готівки, скасування та завершення. Заявка може сформувати не більше однієї активної угоди, що відповідає логіці P2P-обміну: після погодження умов заявка переходить у стан зарезервованої або завершеної.

Клас Message використовується для організації комунікації між учасниками угоди. Він містить ідентифікатор повідомлення, текст, час надсилання та ознаку прочитання. Операції send(), markAsRead() і deleteMessage() забезпечують надсилання, перегляд і видалення повідомлень. Повідомлення пов'язуються з конкретною угодою, що дає змогу зберігати історію узгодження місця, часу та умов обміну.

Клас Complaint призначений для фіксації скарг користувачів у спірних ситуаціях. Він містить ідентифікатор скарги, причину, статус і дату створення. Операції createComplaint(), review() і closeComplaint() забезпечують створення, розгляд і закриття скарги. Такий клас необхідний для підтримки адміністративного контролю та підвищення довіри користувачів до платформи.

Для деталізації поведінкової частини програмної моделі побудовано три діаграми кооперації, які відображають основні сценарії взаємодії об'єктів системи. Перша кооперація, наведена на рисунку 2.3, описує процес створення та публікації заявки на обмін.

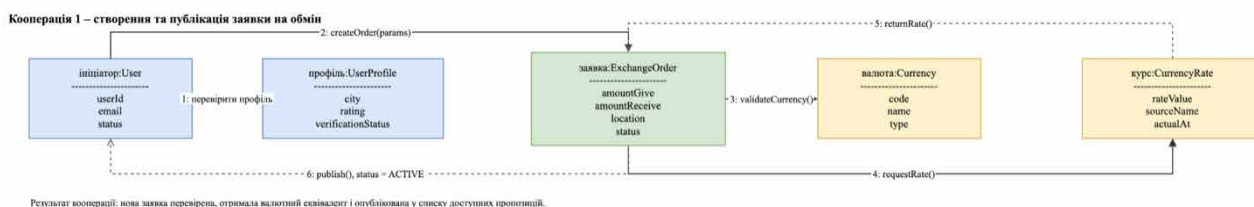


Рисунок 2.3 – Діаграма кооперації створення та публікації заявки на обмін

У цьому сценарії ініціатор обміну проходить перевірку профілю, після чого створює заявку із зазначенням параметрів обміну. Об'єкт ExchangeOrder перевіряє вибрані валюти через Currency, запитує актуальний курс у CurrencyRate та після отримання довідкового значення публікує заявку зі статусом ACTIVE. У результаті нова заявка стає доступною іншим користувачам для перегляду та подальшого вибору.

Друга діаграма кооперації представлена на рисунку 2.4 і відображає процес пошуку заявки та узгодження умов угоди між користувачами.

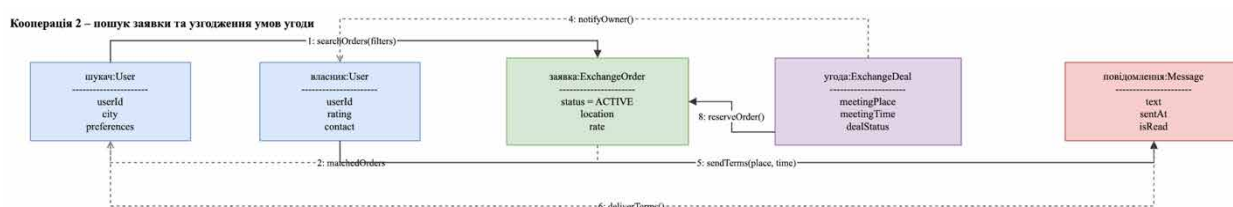


Рисунок 2.4 – Діаграма кооперації пошуку заявки та узгодження умов угоди

У межах цього сценарію користувач виконує пошук доступних заявок за встановленими фільтрами. Об'єкт ExchangeOrder повертає список відповідних пропозицій, після чого користувач обирає заявку та ініціює створення угоди. Об'єкт ExchangeDeal повідомляє власника заявки про запит, а через об'єкт Message сторони уточнюють місце, час і додаткові умови передачі готівки. Після погодження умов угода отримує відповідний статус, а заявка резервується для конкретного обміну.

Третя діаграма кооперації, наведена на рисунку 2.5, описує процес підтвердження передачі готівки та завершення угоди.

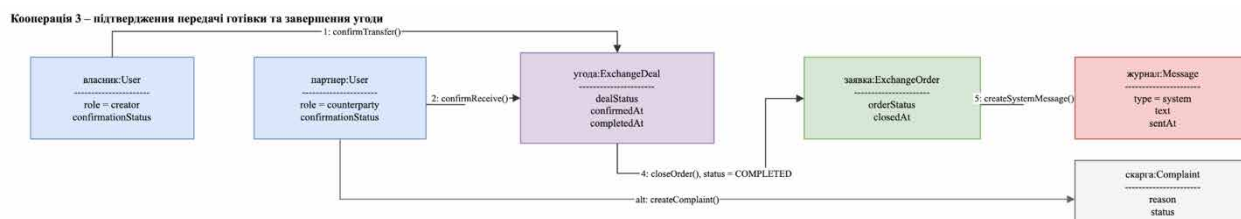


Рисунок 2.5 – Діаграма кооперації підтвердження передачі готівки та завершення угоди

У цьому сценарії власник заявки та партнер обміну підтверджують виконання домовленостей після фактичної передачі готівки. Об'єкт ExchangeDeal перевіряє наявність підтвердження з обох сторін, після чого ініціює закриття заявки через ExchangeOrder. Далі створюється системне повідомлення в Message, яке фіксує результат завершення обміну. Якщо одна зі сторін не підтверджує виконання умов або виникає спірна ситуація, може бути створена скарга через об'єкт Complaint для подальшого адміністративного розгляду.

Основні програмні класи системи та їх призначення узагальнено у таблиці 2.2.

Таблиця 2.2 - Основні класи програмної моделі системи

Клас	Призначення
User	Представлення користувача системи, зберігання облікових даних, ролі та статусу
UserProfile	Зберігання профільної інформації користувача, рейтингу та статусу верифікації
ExchangeOrder	Опис заявки на обмін готівки із зазначенням валют, суми, курсу, місця та статусу
ExchangeDeal	Представлення погодженої угоди між двома користувачами та контроль її виконання
Message	Організація обміну повідомленнями між учасниками угоди
Complaint	Фіксація скарг користувачів і підтримка розгляду спірних ситуацій
Currency	Зберігання довідкової інформації про валюти, доступні для обміну
CurrencyRate	Зберігання довідкових курсів валют і часу їх актуальності

Отже, побудована UML-діаграма класів формалізує статичну структуру інформаційної системи P2P-обміну готівки нестандартних валют, а діаграми кооперації деталізують основні сценарії взаємодії між об'єктами. Запропонована програмна модель відображає логіку створення заявок, пошуку пропозицій, узгодження умов угоди, обміну повідомленнями, підтвердження передачі готівки та обробки спірних ситуацій. Це створює основу для подальшого проєктування компонентної архітектури, фізичної моделі даних і реалізації програмного забезпечення застосунку “Фінансовий помічник” для організації готівкових P2P-обмінів

2.3 Діаграма компонентів

Діаграма компонентів інформаційної системи наведена на рисунку 2.6. Вона відображає локальну компонентну архітектуру desktop-застосунку “Фінансовий помічник”, реалізованого з використанням Python, PyQt6 та SQLite. На відміну від клієнт-серверної моделі, у цій архітектурі не передбачено окремого серверного рівня, API застосунку або передавання даних між клієнтом і сервером через HTTPS. Основні функції системи виконуються безпосередньо в межах локального застосунку, а збереження даних здійснюється у локальній базі даних SQLite.

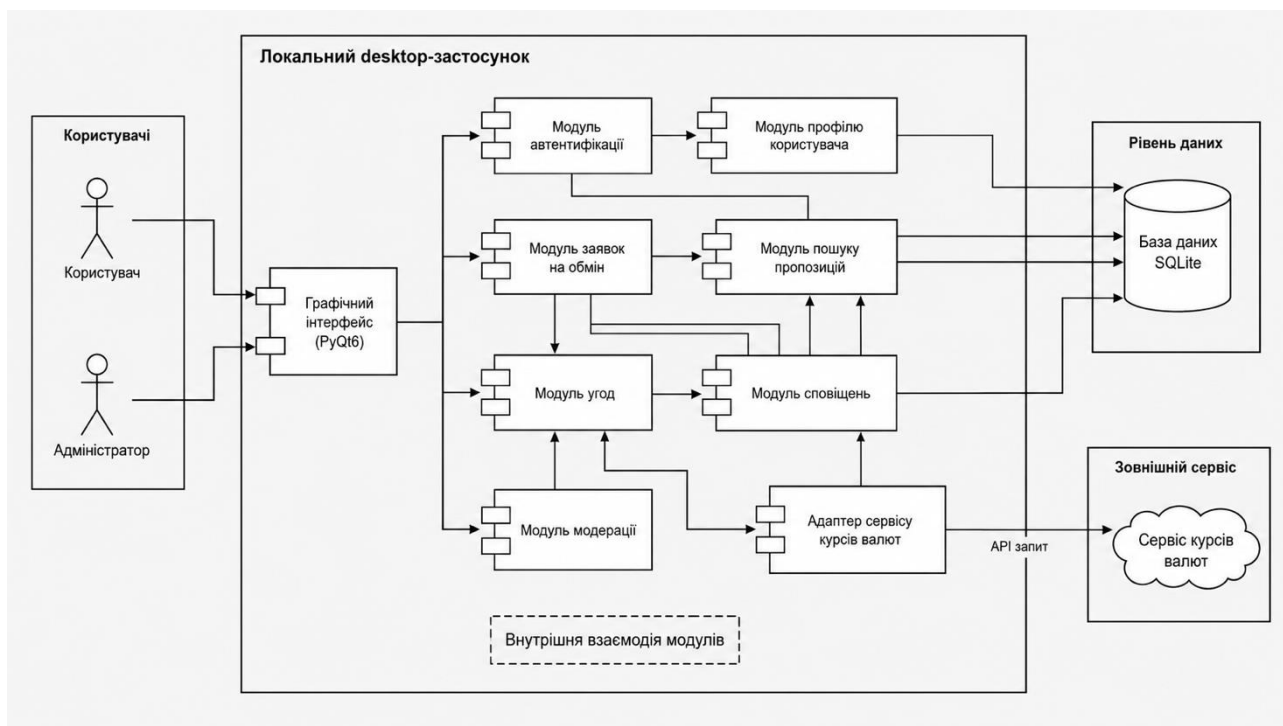


Рисунок 2.6 – UML-діаграма компонентів локального desktop-застосунку P2P-обміну готівки нестандартних валют

Застосунок використовується двома основними категоріями користувачів: користувачем та адміністратором. Користувач через графічний інтерфейс виконує реєстрацію, вхід у систему, переглядає доступні пропозиції, створює заявки на обмін, погоджує умови угоди та отримує системні сповіщення. Адміністратор використовує той самий інтерфейс для перегляду заявок, контролю користувачів, перевірки проблемних операцій і виконання модераційних дій.

Графічний інтерфейс реалізується засобами PyQt6 і є основною точкою взаємодії користувача з програмою. Через нього користувач передає дані до внутрішніх модулів застосунку, а система відображає результати обробки: список заявок, знайдені пропозиції, статуси угод, повідомлення та довідкову інформацію про курси валют. Усі функціональні модулі працюють у межах одного локального програмного середовища, тому взаємодія між ними виконується як внутрішній обмін даними між компонентами desktop-застосунку.

Модуль автентифікації відповідає за перевірку облікових даних користувача, визначення ролі та надання доступу до відповідних функцій

системи. Після успішного входу користувач отримує доступ до модуля профілю, заявок, пошуку пропозицій та угод. Для адміністратора відкриваються додаткові можливості модерації користувачів і заявок.

Модуль профілю користувача забезпечує збереження та редагування персональних даних, контактної інформації, налаштувань облікового запису та історії взаємодії із системою. Дані профілю зберігаються у локальній базі SQLite та використовуються іншими модулями під час створення заявок, формування угод і відображення інформації про учасників обміну.

Модуль заявок на обмін реалізує створення, редагування, публікацію та закриття заявок. У заявці користувач зазначає валюту, суму, бажані умови обміну, місто або місце зустрічі, коментар та інші параметри. Цей модуль взаємодіє з модулем пошуку пропозицій, модулем угод, модулем сповіщень і базою даних SQLite.

Модуль пошуку пропозицій виконує відбір доступних заявок за заданими параметрами: валютою, сумою, напрямом обміну, містом, курсом або статусом. Його робота ґрунтується на даних, що зберігаються у локальній базі, а також може використовувати довідкові курси валют, отримані через адаптер зовнішнього сервісу. Результатом роботи модуля є список пропозицій, які відповідають умовам користувача.

Модуль угод відповідає за формування домовленості між учасниками обміну. Він фіксує обрану заявку, сторони угоди, погоджені умови, поточний статус і результат виконання. Для контролю процесу використовуються статуси, наприклад: створено, очікує підтвердження, погоджено, виконано або скасовано. Зміна статусу угоди зберігається у базі даних і може супроводжуватися системним сповіщенням.

Модуль сповіщень забезпечує інформування користувачів про важливі події в системі: створення нової заявки, зміну статусу угоди, підтвердження домовленості, скасування операції або адміністративне повідомлення. У межах локального desktop-застосунку сповіщення можуть відображатися у вікні програми або зберігатися в історії повідомлень.

Модуль модерації призначений для адміністративного контролю роботи системи. Він дає змогу переглядати користувачів, перевіряти заявки, аналізувати проблемні угоди, обробляти скарги та за потреби змінювати статус окремих записів. Наявність цього модуля підвищує керованість системи та дає змогу підтримувати порядок у процесі взаємодії між учасниками обміну.

Адаптер сервісу курсів валют забезпечує отримання довідкових курсів із зовнішнього сервісу. На відміну від основних модулів, цей компонент може звертатися до зовнішнього джерела через API-запит. Отримані значення використовуються лише як допоміжна інформація для орієнтовного розрахунку еквіваленту обміну. Остаточні умови угоди визначаються безпосередньо користувачами.

Рівень даних представлений локальною базою даних SQLite. У ній зберігаються облікові записи користувачів, профілі, заявки на обмін, угоди, сповіщення, дані модерації, довідник валют, отримані курси валют і журнал дій. Використання SQLite є доцільним для локального desktop-застосунку, оскільки така база даних не потребує окремого серверного середовища, легко розгортається разом із програмою та забезпечує достатню швидкість для роботи з локальними даними.

Основні компоненти архітектури локального застосунку та їх призначення наведено в таблиці 2.3.

Таблиця 2.3 - Основні компоненти архітектури локального desktop-застосунку

Компонент	Призначення
Графічний інтерфейс (PyQt6)	Забезпечує взаємодію користувача й адміністратора із застосунком, відображає форми, таблиці, статуси угод і системні повідомлення
Модуль автентифікації	Виконує перевірку облікових даних, визначення ролі користувача та контроль доступу до функцій системи
Модуль профілю користувача	Забезпечує створення, перегляд і редагування профільної інформації користувача
Модуль заявок на обмін	Реалізує створення, редагування, публікацію та закриття заявок на обмін нестандартних валют
Модуль пошуку пропозицій	Виконує пошук доступних заявок за валютою, сумою, містом, курсом, статусом та іншими параметрами

Продовження таблиці 2.3

Модуль угод	Формує угоду між користувачами, контролює її статус і фіксує результат виконання
Модуль сповіщень	Забезпечує відображення системних повідомлень про зміну статусів заявок, угод і дій користувачів
Модуль модерації	Підтримує адміністративний контроль користувачів, заявок, скарг і проблемних угод
Адаптер сервісу курсів валют	Отримує довідкові курси валют із зовнішнього сервісу через API-запит
База даних SQLite	Зберігає користувачів, профілі, заявки, угоди, сповіщення, дані модерації, валюти, курси та журнал дій

Запропонована компонентна архітектура забезпечує чіткий розподіл функцій між окремими модулями локального застосунку. Графічний інтерфейс відповідає за взаємодію з користувачем, функціональні модулі реалізують основну логіку роботи системи, база даних SQLite забезпечує збереження інформації, а зовнішній сервіс курсів валют використовується лише як допоміжне джерело довідкових даних.

Отже, побудована UML-діаграма компонентів дозволяє формалізувати архітектуру локального desktop-застосунку для організації P2P-обміну готівки нестандартних валют. Така структура відповідає обраній технологічній реалізації на Python, PyQt6 і SQLite, не потребує окремого серверного середовища та є зручною для розгортання на персональному комп'ютері користувача. Архітектура залишається модульною та придатною для подальшого розширення, зокрема додавання рейтингу користувачів, розширеної історії операцій, додаткових фільтрів пошуку та нових джерел валютних курсів.

2.4 Діаграма пакетів

Для структуризації програмного забезпечення інформаційної системи «Фінансовий помічник для P2P-обміну готівки нестандартних валют» побудовано UML-діаграму пакетів. Вона дає змогу подати програмну архітектуру у вигляді логічно відокремлених частин, кожна з яких відповідає за окремий напрям роботи системи: взаємодію з користувачем, оброблення заявок,

формування угод, обмін повідомленнями, модерацію, роботу з валютними курсами та збереження даних.

Діаграма пакетів інформаційної системи наведена на рисунку 2.7. Вона демонструє пакетну структуру локального Python-застосунку, розділяючи функціональність на логічні компоненти. Такий підхід забезпечує зрозумілу організацію коду, спрощує супровід системи та створює основу для подальшого розширення функціональності застосунку.

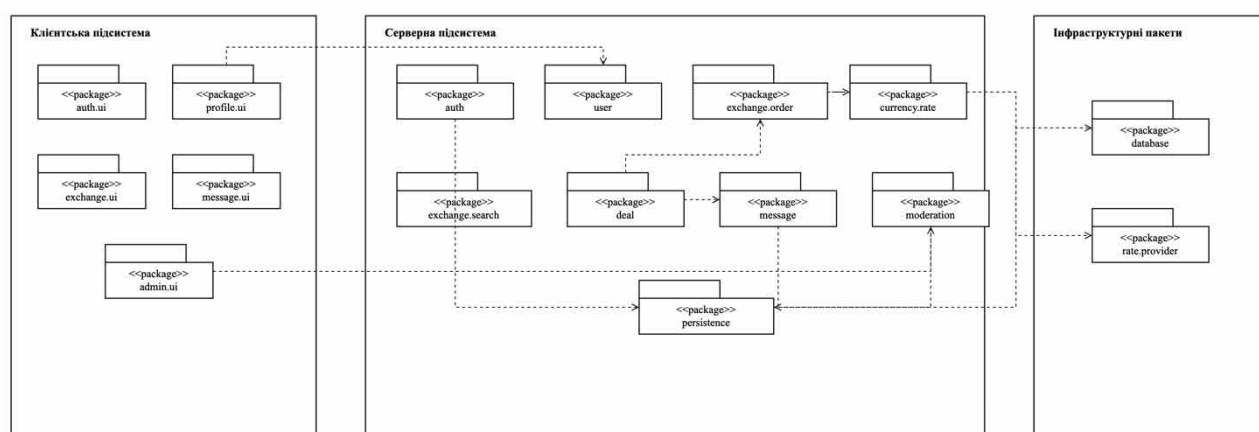


Рисунок 2.7 – UML-діаграма пакетів інформаційної системи P2P-обміну готівки нестандартних валют

Пакети інтерфейсу користувача (UI) включають auth.ui, profile.ui, exchange.ui, message.ui та admin.ui. Вони відповідають за представлення реєстрації й входу, роботу з профілем користувача, створення та перегляд заявок, обмін повідомленнями між учасниками угоди, а також адміністративне керування скаргами та користувачами.

Серверна підсистема містить пакети auth, user, exchange.order, exchange.search, deal, message, moderation та currency.rate. Вони реалізують основну бізнес-логіку системи: автентифікацію користувачів, керування профільними даними, створення заявок на обмін, пошук доступних пропозицій, погодження угод, оброблення повідомлень, модерацію спірних ситуацій і отримання довідкових курсів валют.

Пакети даних та ресурсів представлені database та rate.provider. Пакет database забезпечує локальне збереження інформації про користувачів, заявки,

угоди, повідомлення, скарги та курси валют через SQLite, а rate.provider використовується для взаємодії із зовнішнім сервісом валютних курсів через API.

Основні пакети програмної архітектури системи та їх призначення наведено в таблиці 2.4.

Таблиця 2.4 - Основні пакети програмної архітектури системи

Пакет	Призначення
auth.ui	Інтерфейс реєстрації, входу та відновлення доступу користувача
profile.ui	Відображення та редагування профільної інформації користувача
exchange.ui	Інтерфейс створення, перегляду та вибору заявок на обмін
message.ui	Інтерфейс повідомлень між учасниками угоди
admin.ui	Інтерфейс адміністратора для модерації користувачів, заявок і скарг
auth	Автентифікація користувачів, перевірка доступу та контроль ролей
user	Оброблення даних користувачів, профілів, статусів і рейтингу
exchange.order	Створення, редагування, публікація та закриття заявок на обмін
exchange.search	Пошук активних заявок за валютою, містом, курсом і сумою
deal	Формування угод, зміна їх статусів і підтвердження обміну
message	Оброблення користувацьких і системних повідомлень
moderation	Розгляд скарг, блокування порушників і контроль спірних ситуацій
currency.rate	Отримання та використання довідкових курсів валют
persistence	Організація доступу до бази даних і збереження об'єктів системи
database	Зберігання основних даних системи

Запропонована пакетна структура забезпечує розділення системи на незалежні функціональні частини. Це спрощує реалізацію програмного забезпечення, полегшує тестування окремих модулів і дає змогу надалі розширювати застосунок без суттєвого порушення його загальної архітектури.

Отже, UML-діаграма пакетів формалізує логічну організацію програмного забезпечення інформаційної системи Р2Р-обміну готівки нестандартних валют. Вона показує основні групи програмних модулів і залежності між ними, що створює основу для подальшої реалізації серверної логіки, клієнтського інтерфейсу та інфраструктури збереження даних.

2.5 Висновки до другого розділу

Розділ охоплює побудову логічної моделі даних, UML-діаграми класів, діаграм кооперації, діаграми компонентів і діаграми пакетів, що разом формують основу для подальшої програмної реалізації системи.

У межах проєктування логічної моделі даних визначено основні інформаційні сутності системи: користувач, заявка на обмін, угода, повідомлення, валюта та курс валют. Така структура відображає ключові процеси роботи платформи, зокрема створення заявок, пошук пропозицій, узгодження умов обміну, фіксацію повідомлень і використання довідкових курсів валют.

За допомогою UML-діаграми класів формалізовано основні програмні об'єкти системи, їх атрибути, операції та взаємозв'язки. Додатково побудовано діаграми кооперації, які відображають типові сценарії взаємодії об'єктів під час створення заявки, пошуку пропозиції, погодження умов угоди та підтвердження передачі готівки.

Діаграма компонентів дала змогу визначити основні модулі програмної архітектури: клієнтський інтерфейс, API застосунку, модулі автентифікації, профілю користувача, заявок, пошуку, угод, повідомлень, модерації, адаптер курсів валют і базу даних. Такий поділ забезпечує модульність системи та спрощує подальшу реалізацію окремих функціональних частин.

Діаграма пакетів дозволила структурувати програмне забезпечення на клієнтські, серверні та інфраструктурні пакети. Запропонована організація забезпечує чітке розмежування відповідальності між частинами системи, полегшує супровід програмного коду та створює можливість для подальшого розширення функціональності застосунку.

Отже, у другому розділі сформовано цілісну проєктну основу інформаційної системи Р2Р-обміну готівки нестандартних валют. Розроблені моделі забезпечують перехід від аналізу предметної області до практичної реалізації програмного забезпечення та можуть бути використані для створення

бази даних, серверної логіки, клієнтського інтерфейсу й механізмів взаємодії користувачів у межах застосунку.

3 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Вибір технологій та інструментальних засобів реалізації

Розроблення програмного застосунку «Фінансовий помічник для P2P-обміну готівки нестандартних валют» передбачає вибір технологій, які забезпечують зручну роботу користувача з локальним інтерфейсом, надійне зберігання даних, швидке виконання операцій із заявками та простоту подальшого супроводу програмного забезпечення. Оскільки система орієнтована на роботу з профілями користувачів, заявками на обмін, угодами, повідомленнями, валютами та курсами валют, основними критеріями вибору технологій стали стабільність, простота розгортання, підтримка локальної бази даних і можливість створення зручного графічного інтерфейсу.

Для реалізації застосунку обрано мову програмування Python, графічний фреймворк PyQt6 та локальну реляційну базу даних SQLite. Такий технологічний стек є доцільним для створення настільного програмного забезпечення, оскільки не потребує окремого серверного середовища, складного налаштування веб-сервера або розгортання зовнішньої СУБД. Уся основна логіка системи виконується в межах локального застосунку, а дані зберігаються у файлі бази даних SQLite.

Основні технології та інструментальні засоби, використані під час реалізації програмного забезпечення, наведено в таблиці 3.1.

Таблиця 3.1 - Вибір технологій та інструментальних засобів реалізації

№	Компонент системи	Технологія / інструмент	Призначення
1	Мова програмування	Python	Реалізація логіки застосунку, оброблення даних, робота із заявками, угодами та повідомленнями

Продовження таблиці 3.1

2	Графічний інтерфейс	PyQt6	Створення вікон, форм, таблиць, кнопок і діалогових елементів користувацького інтерфейсу
3	Стилізація інтерфейсу	QSS	Оформлення зовнішнього вигляду елементів PyQt6
4	База даних	SQLite	Локальне зберігання користувачів, заявок, угод, повідомлень, валют і курсів
5	Модуль роботи з БД	sqlite3	Виконання SQL-запитів до локальної бази даних
6	Формат службового обміну	JSON	Зберігання або передавання допоміжних параметрів, налаштувань і відповідей сервісів
7	Середовище розроблення	PyCharm	Написання, запуск і налагодження програмного коду
8	Контроль версій	Git	Збереження історії змін програмного коду
9	Пакування застосунку	PyInstaller	Формування виконуваного файлу для запуску застосунку без середовища розроблення
10	Захист даних	Хешування паролів, контроль ролей	Безпечне зберігання облікових даних і розмежування доступу

Мова програмування Python обрана як основний інструмент реалізації програмної логіки завдяки простому синтаксису, широкій бібліотечній підтримці та зручності роботи з локальними базами даних. У межах застосунку Python використовується для оброблення дій користувача, перевірки введених даних, створення заявок на обмін, пошуку пропозицій, формування угод, збереження повідомлень і роботи зі статусами операцій.

Для створення графічного інтерфейсу використано PyQt6. Цей фреймворк дозволяє реалізувати повноцінний настільний застосунок із вікнами авторизації, формами створення заявок, таблицями перегляду пропозицій, вікнами угод, повідомлень і адміністративного керування. Використання PyQt6 є доцільним для цієї роботи, оскільки система не потребує обов'язкового веб-розгортання та може працювати як локальний застосунок на персональному комп'ютері.

Для оформлення інтерфейсу застосовується QSS, що за принципом роботи подібний до CSS, але використовується для стилізації елементів PyQt6. Завдяки QSS можна задати єдиний візуальний стиль кнопок, таблиць, полів введення, вкладок і діалогових вікон, що підвищує зручність використання програми.

Як систему керування базою даних обрано SQLite. Вона є вбудованою реляційною СУБД, яка зберігає всі дані в одному локальному файлі та не потребує встановлення окремого сервера бази даних. Для розроблюваного застосунку це є оптимальним рішенням, оскільки обсяг даних є помірним, а структура інформації має чіткі зв'язки між користувачами, заявками, угодами, повідомленнями, валютами та курсами валют.

Робота з базою даних виконується за допомогою модуля sqlite3. Він забезпечує створення таблиць, виконання SQL-запитів, додавання, оновлення, видалення та вибірку даних. У базі даних передбачено таблиці для зберігання користувачів, профілів, валют, курсів валют, заявок на обмін, угод, повідомлень і скарг.

Середовище PyCharm використовується для написання та налагодження програмного коду. Воно забезпечує зручну роботу з Python-проєктом, запуск окремих модулів, перевірку помилок і організацію файлової структури застосунку. Для контролю версій програмного коду використовується Git, що дозволяє фіксувати зміни та підтримувати актуальний стан проєкту.

Для формування виконуваного файлу може використовуватися PyInstaller. Це дає змогу підготувати інсталяційний або автономний варіант застосунку, який користувач зможе запускати без відкриття середовища розроблення. До складу такого пакету входить виконуваний файл програми, файл бази даних SQLite, службові ресурси інтерфейсу та конфігураційні файли.

Отже, обраний технологічний стек Python, PyQt6 і SQLite відповідає вимогам до розроблення локального настільного застосунку “Фінансовий помічник” для організації готівкових P2P-обмінів. Він забезпечує простоту запуску, зручну реалізацію графічного інтерфейсу, надійне локальне зберігання даних і можливість подальшого розширення функціональності системи.

Локальна база даних SQLite є основним сховищем інформації системи. У ній зберігаються користувачі, профілі, валюти, курси валют, заявки, запити на обмін, угоди, повідомлення, скарги та журнал дій. Використання SQLite спрощує розгортання застосунку, оскільки для його роботи не потрібен окремий сервер бази даних.

Основні елементи архітектури програмного застосунку та їх призначення наведено у таблиці 3.2.

Таблиця 3.2 - Основні елементи архітектури програмного застосунку

Елемент архітектури	Призначення
PyQt6-інтерфейс	Забезпечує взаємодію користувача із застосунком через графічні вікна, форми й таблиці
Вікно авторизації	Реалізує вхід користувача та перевірку облікових даних
Вікно заявок	Відображає доступні заявки та забезпечує пошук пропозицій
Форма створення заявки	Дозволяє користувачу сформулювати нову пропозицію обміну
Вікно угоди	Відображає дані погодженої угоди та її поточний статус
Модуль автентифікації	Реалізує реєстрацію, вхід і контроль ролей користувачів
Модуль профілю	Обробляє профільні дані користувача
Модуль заявок	Забезпечує створення, редагування, публікацію та закриття заявок
Модуль пошуку	Виконує відбір активних заявок за заданими параметрами
Модуль угод	Формує угоди, контролює статус і підтвердження сторін
Модуль повідомлень	Забезпечує збереження та перегляд повідомлень між учасниками угоди
Модуль модерації	Підтримує адміністративний контроль скарг, користувачів і заявок
Модуль доступу до БД	Виконує SQL-запити та передає дані між застосунком і SQLite
SQLite	Зберігає основні дані інформаційної системи
Модуль курсів валют	Забезпечує збереження або оновлення довідкових курсів валют

Запропонована архітектура забезпечує чіткий розподіл функцій між інтерфейсом, прикладною логікою та рівнем зберігання даних. PyQt6 відповідає за відображення інформації та взаємодію з користувачем, Python-модулі

реалізують бізнес-логіку системи, а SQLite забезпечує локальне зберігання даних.

Отже, архітектура програмного застосунку “Фінансовий помічник” для організації готівкових P2P-обмінів відповідає вимогам до локальної інформаційної системи для організації P2P-обміну готівки нестандартних валют. Вона забезпечує роботу з користувачами, заявками, угодами, повідомленнями, валютами, курсами та скаргами без потреби у зовнішньому серверному середовищі, що спрощує встановлення, запуск і практичне використання програми.

3.3 Фізична модель даних локального застосунку P2P-обміну готівки нестандартних валют

Фізична модель даних визначає практичну структуру зберігання інформації у програмному застосунку «Фінансовий помічник для P2P-обміну готівки нестандартних валют». Оскільки система реалізується як локальний desktop-застосунок на Python із використанням PyQt6, для зберігання даних обрано SQLite. Такий підхід є доцільним для даного програмного забезпечення, оскільки не потребує окремого сервера бази даних, забезпечує швидкий доступ до інформації та дозволяє зберігати всі основні дані системи в одному локальному файлі.

Фізична модель даних наведена на рисунку 3.2. Вона відображає таблиці, первинні та зовнішні ключі, а також зв'язки між сутностями, які забезпечують роботу основних функцій застосунку: реєстрацію користувачів, ведення профілів, створення заявок на обмін, формування запитів, укладання угод, обмін повідомленнями, подання скарг і збереження курсів валют.

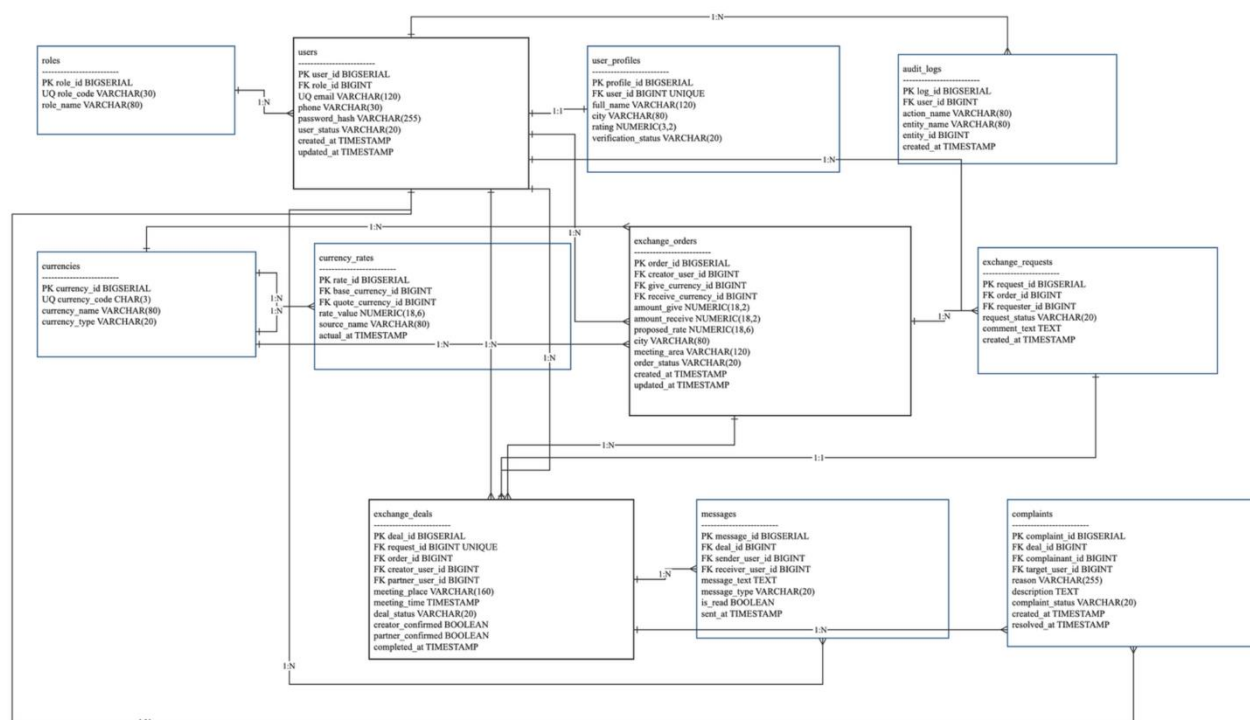


Рисунок 3.2 – Фізична модель даних локального застосунку P2P-обміну готівки нестандартних валют

У базі даних передбачено таблиці `roles`, `users`, `user_profiles`, `currencies`, `currency_rates`, `exchange_orders`, `exchange_requests`, `exchange_deals`, `messages`, `complaints` та `audit_logs`. Вони формують цілісну структуру зберігання даних і відповідають основним об'єктам предметної області, визначеним у попередніх розділах роботи. Для SQLite ідентифікатори записів реалізуються через тип `INTEGER PRIMARY KEY AUTOINCREMENT`, текстові поля через `TEXT`, числові значення сум і курсів через `REAL` або `NUMERIC`, а логічні ознаки зберігаються як `INTEGER` зі значеннями 0 або 1.

Основною таблицею є `users`, у якій зберігаються облікові дані користувачів: електронна пошта, телефон, хеш пароля, роль, статус і часові позначки створення та оновлення запису. Таблиця `user_profiles` деталізує профіль користувача та містить ім'я, місто, рейтинг і статус верифікації. Такий поділ дозволяє відокремити дані авторизації від профільної інформації.

Для роботи з валютами використовуються таблиці `currencies` та `currency_rates`. Перша містить довідник валют, а друга зберігає курси між валютними парами, джерело отримання курсу та час його актуальності. Ці дані

використовуються під час створення заявки та розрахунку орієнтовного еквіваленту обміну.

Таблиця `exchange_orders` призначена для зберігання заявок на обмін. У ній фіксуються користувач, валюта, яку він віддає, валюта, яку хоче отримати, сума, курс, місто, місце зустрічі, статус заявки та дата створення. Запити інших користувачів на виконання обміну зберігаються у таблиці `exchange_requests`, а погоджені домовленості між сторонами фіксуються у таблиці `exchange_deals`.

Для комунікації між учасниками угоди використовується таблиця `messages`. Вона зберігає текст повідомлення, відправника, отримувача, тип повідомлення, ознаку прочитання та час надсилання. У разі виникнення спірної ситуації дані зберігаються у таблиці `complaints`, яка містить інформацію про скаржника, користувача, на якого подано скаргу, причину, опис і статус розгляду. Для контролю важливих дій користувачів передбачено таблицю `audit_logs`.

Основні таблиці фізичної моделі даних та їх призначення наведено в таблиці 3.3.

Таблиця 3.3 - Основні таблиці фізичної моделі даних

Таблиця	Призначення
<code>roles</code>	Зберігання ролей користувачів системи
<code>users</code>	Зберігання облікових даних користувачів
<code>user_profiles</code>	Зберігання профільної інформації користувачів
<code>currencies</code>	Довідник валют, доступних у системі
<code>currency_rates</code>	Зберігання курсів валют і часу їх актуальності
<code>exchange_orders</code>	Зберігання заявок на обмін готівки
<code>exchange_requests</code>	Зберігання запитів користувачів на обмін
<code>exchange_deals</code>	Фіксація погоджених угод між користувачами
<code>messages</code>	Зберігання повідомлень між учасниками угоди
<code>complaints</code>	Зберігання скарг і спірних ситуацій
<code>audit_logs</code>	Журналювання важливих дій у системі

Запропонована фізична модель даних забезпечує цілісне зберігання інформації, необхідної для роботи локального застосунку. Використання зовнішніх ключів дозволяє пов'язати користувачів із профілями, заявками,

угодами, повідомленнями та скаргами, а індекси можуть застосовуватися для пришвидшення пошуку активних заявок за містом, статусом і валютною парою.

Отже, фізична модель даних є основою для реалізації локальної бази SQLite у застосунку на PyQt6. Вона забезпечує збереження всіх ключових об'єктів системи, підтримує зв'язки між ними та дає змогу реалізувати основні функції програмного забезпечення: створення заявок, пошук пропозицій, узгодження угод, обмін повідомленнями, облік курсів валют і розгляд скарг користувачів.

3.4 Алгоритмізація програмних модулів системи

У межах роботи розроблено алгоритми для ключових процесів: реєстрації та входу користувача, створення заявки на обмін, а також пошуку заявки, узгодження і завершення угоди.

Перший алгоритм описує процес реєстрації та входу користувача в систему. Він починається з відкриття форми авторизації, після чого користувач вводить облікові дані. Система перевіряє наявність облікового запису в локальній базі даних SQLite, виконує перевірку логіна і пароля та визначає можливість доступу до основного меню застосунку. У разі некоректних даних користувачу виводиться повідомлення про помилку, після чого він може повторити введення. Такий алгоритм забезпечує базовий контроль доступу та розмежування користувачів за ролями.

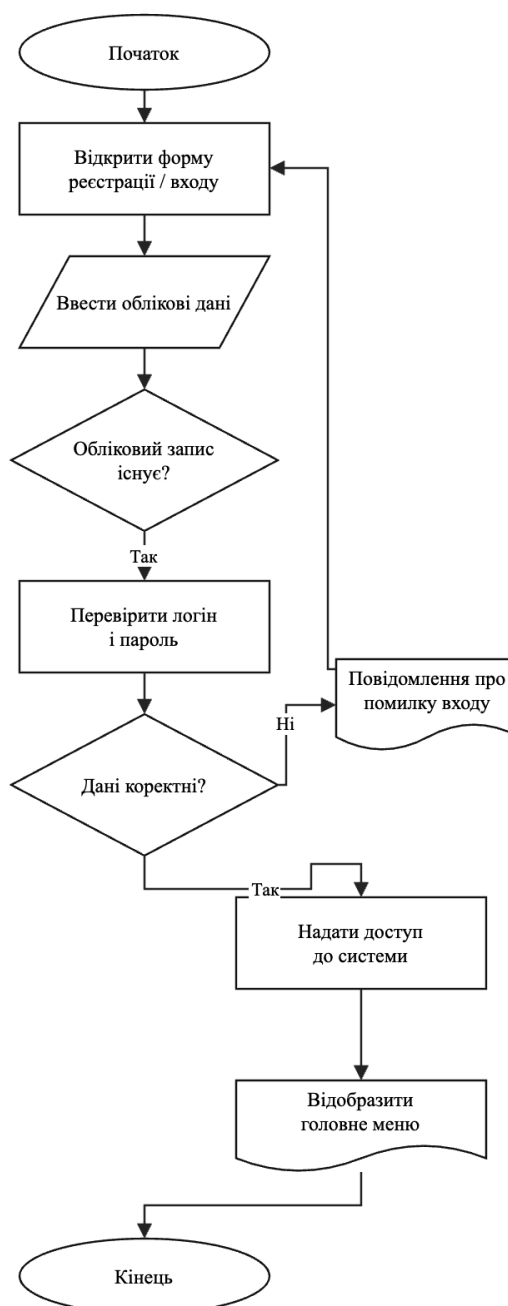


Рисунок 3.3 – Блок-схема алгоритму реєстрації та входу користувача

Другий алгоритм відображає процес створення заявки на обмін готівки. Користувач відкриває відповідний модуль, вводить параметри заявки: валюту, суму, курс, місто та умови передачі коштів. Після цього система перевіряє повноту заповнення форми та коректність введених значень. Якщо дані містять помилки або заповнені не повністю, користувачу відображається відповідне повідомлення. У разі успішної перевірки система отримує довідковий курс

валюти, зберігає заявку у базі даних SQLite і переводить її у статус ACTIVE, після чого вона стає доступною для перегляду іншими користувачами.

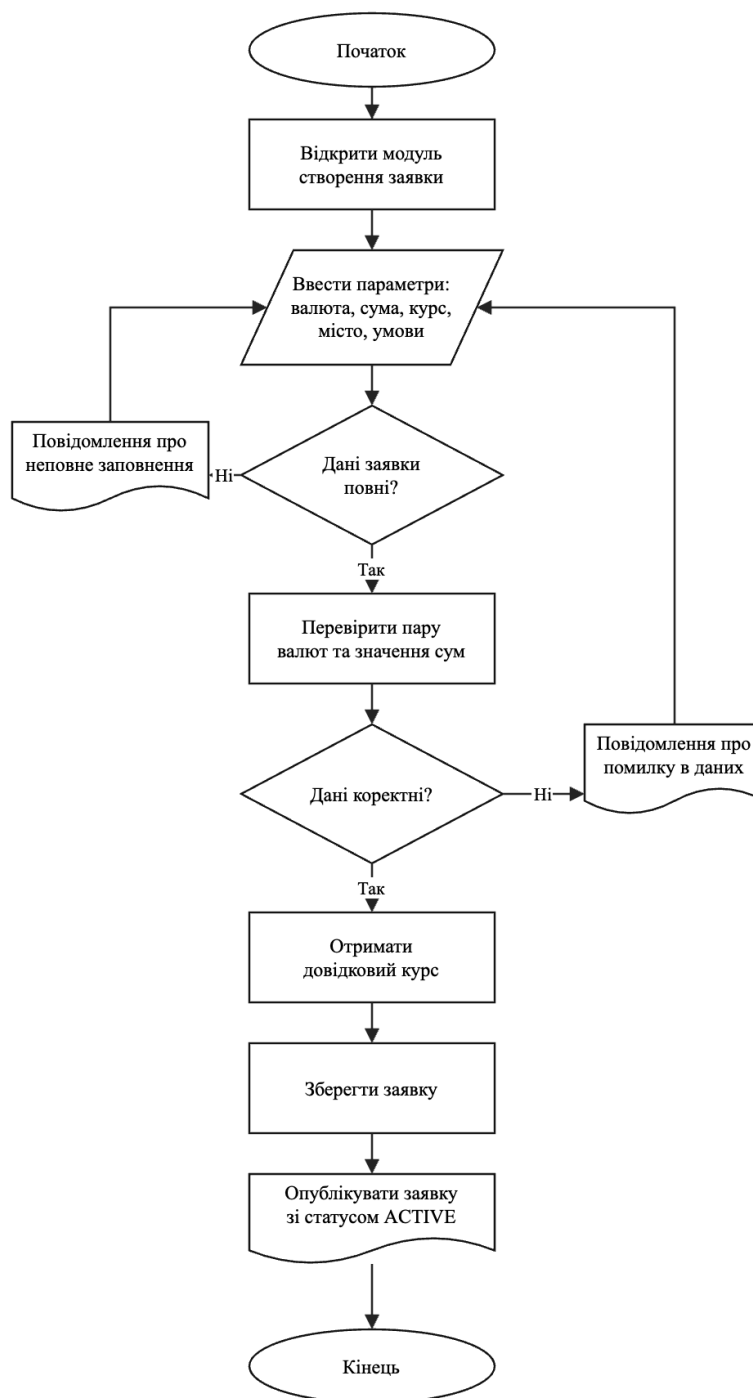


Рисунок 3.4 – Блок-схема алгоритму створення та публікації заявки на обмін

Третій алгоритм описує процес пошуку заявки, узгодження умов обміну та завершення угоди. Користувач переглядає доступні заявки, обирає потрібну пропозицію та надсилає запит на обмін. Якщо власник заявки погоджує запит,

сторони узгоджують місце і час зустрічі, після чого здійснюється фактична передача готівки. Завершення угоди відбувається лише після підтвердження з обох сторін. Якщо підтвердження не отримано або виникає спірна ситуація, система очікує повторного підтвердження або дає можливість створити скаргу для подальшого розгляду адміністратором.

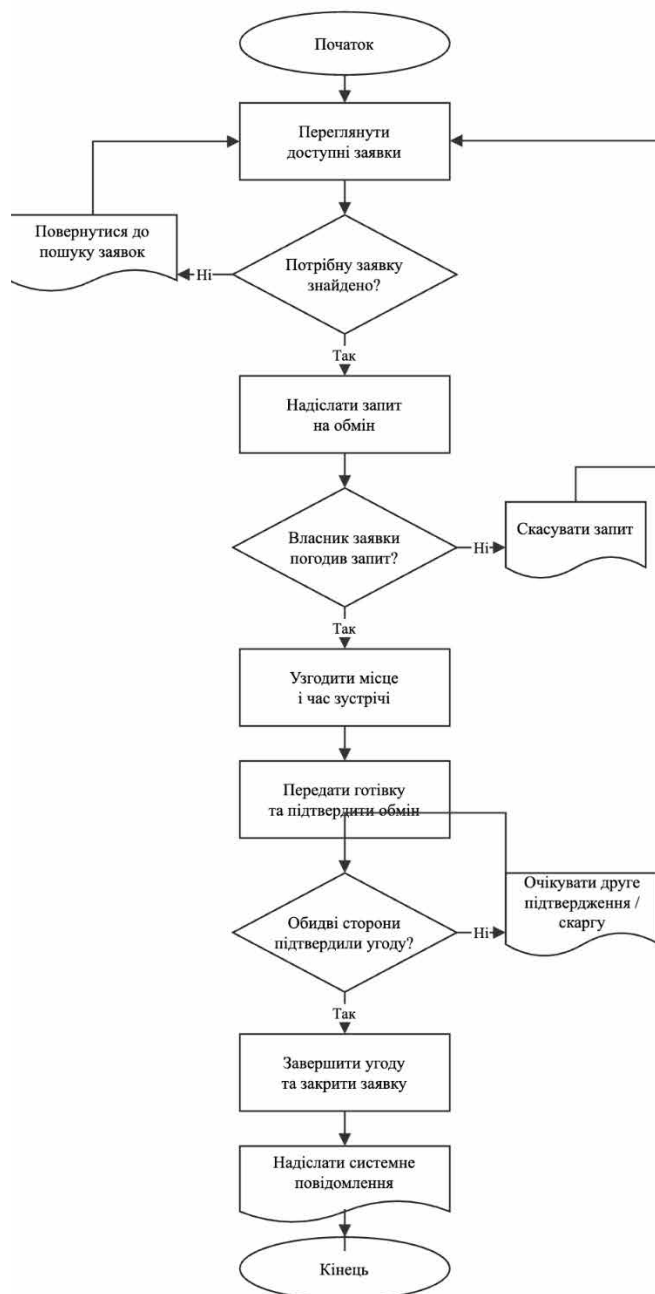


Рисунок 3.5 – Блок-схема алгоритму узгодження та завершення угоди

Розроблені алгоритми охоплюють основні сценарії роботи програмного застосунку та забезпечують логічну послідовність виконання операцій. Вони враховують перевірку введених даних, роботу зі статусами заявок і угод,

збереження інформації в SQLite та відображення результатів у графічному інтерфейсі PyQt6. Такий підхід дозволяє зменшити кількість помилок під час реалізації програмного забезпечення та забезпечити зрозумілу поведінку системи для користувача.

Отже, алгоритмізація програмних модулів створює основу для практичної реалізації застосунку “Фінансовий помічник” для організації готівкових P2P-обмінів. Запропоновані блок-схеми формалізують логіку основних процесів системи: авторизації користувача, створення заявки, пошуку пропозицій, узгодження умов обміну, підтвердження передачі готівки та завершення угоди. Це забезпечує узгоджену роботу інтерфейсу, програмної логіки та локальної бази даних у межах єдиного desktop-застосунку.

3.5 Висновки до третього розділу

У третьому розділі кваліфікаційної роботи виконано проєктування та опис реалізації програмного забезпечення застосунку «Фінансовий помічник для P2P-обміну готівки нестандартних валют». Основну увагу приділено вибору технологій, формуванню архітектури системи, побудові фізичної моделі даних та алгоритмізації ключових програмних модулів.

Для реалізації застосунку обґрунтовано використання мови програмування Python, фреймворку PyQt6 для створення графічного інтерфейсу та локальної бази даних SQLite для зберігання інформації. Такий технологічний стек є доцільним для розроблення desktop-застосунку, оскільки забезпечує простоту запуску, незалежність від зовнішнього серверного середовища, швидку роботу з даними та можливість подальшого розширення функціональності системи.

У межах архітектурного проєктування визначено основні складові програмного забезпечення: графічний інтерфейс користувача, модулі автентифікації, профілю, заявок, пошуку, угод, повідомлень, модерації, роботи з курсами валют та доступу до бази даних. Запропонована архітектура забезпечує чіткий розподіл функцій між інтерфейсною частиною, прикладною логікою та

рівнем зберігання даних. Це підвищує зрозумілість структури програмного коду та спрощує подальше тестування і супровід застосунку.

Розроблена фізична модель даних відображає структуру локальної бази SQLite, що містить таблиці для зберігання користувачів, профілів, ролей, валют, курсів валют, заявок на обмін, запитів, угод, повідомлень, скарг і журналу дій. Використання первинних і зовнішніх ключів забезпечує логічну цілісність даних та дозволяє коректно відстежувати повний життєвий цикл обмінної операції: від створення заявки до завершення угоди або розгляду спірної ситуації.

У процесі алгоритмізації програмних модулів розроблено блок-схеми основних процесів системи: реєстрації та входу користувача, створення й публікації заявки на обмін, а також пошуку заявки, узгодження умов і завершення угоди. Запропоновані алгоритми враховують перевірку введених даних, роботу зі статусами заявок та угод, збереження інформації у базі SQLite і виведення результатів через інтерфейс PyQt6.

Отже, у третьому розділі сформовано технічну основу реалізації програмного застосунку “Фінансовий помічник” для організації готівкових P2P-обмінів.

Обрані технології, архітектурна структура, фізична модель даних та алгоритми роботи модулів забезпечують можливість практичної реалізації локальної інформаційної системи для організації P2P-обміну готівки нестандартних валют між користувачами.

4 ТЕСТУВАННЯ, РОЗГОРТАННЯ ТА ВПРОВАДЖЕННЯ СИСТЕМИ

4.1 Тестування програмних модулів застосунку “Фінансовий помічник” для організації готівкових P2P-обмінів

Тестування програмних модулів застосунку «Фінансовий помічник для P2P-обміну готівки нестандартних валют» спрямоване на перевірку коректності роботи основних функціональних частин системи, стабільності графічного інтерфейсу PyQt6, правильності виконання операцій із локальною базою даних SQLite та надійності оброблення заявок, угод, повідомлень, валют і скарг користувачів. На відміну від веб-платформ, розроблена система функціонує як локальний desktop-застосунок, тому основну увагу під час тестування приділено взаємодії між вікнами інтерфейсу, модулями прикладної логіки та файлом бази даних SQLite.

Метою тестування є підтвердження того, що застосунок забезпечує коректну реєстрацію та вхід користувача, створення і публікацію заявок на обмін, пошук доступних пропозицій, формування угод між користувачами, обмін повідомленнями, підтвердження передачі готівки та оброблення спірних ситуацій. Додатково перевіряється цілісність збережених даних, коректність зміни статусів заявок і угод, а також стійкість програми до некоректного введення інформації.

Загальний план тестування програмних модулів застосунку наведено у таблиці 4.1.

Таблиця 4.1 - План тестування основних програмних модулів застосунку

№	Модуль / компонент	Тип тестування	Тестові дії	Очікуваний результат
1	Модуль автентифікації	Модульне	Реєстрація користувача, вхід у систему, перевірка неправильного пароля	Користувач отримує доступ лише після введення коректних облікових даних

Продовження таблиці 4.1

2	Модуль профілю користувача	Функціональне	Перегляд і редагування профільних даних	Дані профілю коректно оновлюються та зберігаються у SQLite
3	Модуль заявок на обмін	Функціональне	Створення, редагування, публікація та закриття заявки	Заявка зберігається в базі даних і отримує правильний статус
4	Модуль пошуку пропозицій	Функціональне	Пошук активних заявок за валютою, містом, сумою та курсом	У таблиці відображаються лише релевантні активні заявки
5	Модуль угод	Інтеграційне	Надсилення запиту, погодження умов, підтвердження обміну	Угода створюється, змінює статус і пов'язується з відповідною заявкою
6	Модуль повідомлень	Функціональне	Надсилення і перегляд повідомлень між учасниками угоди	Повідомлення зберігаються в базі даних і відображаються у вікні угоди
7	Модуль валют і курсів	Функціональне	Додавання валют, перегляд довідкового курсу, використання курсу в заявці	Валютні дані коректно використовуються під час створення заявки
8	Модуль модерації	Функціональне	Перегляд скарг, зміна статусу скарги, блокування користувача	Адміністратор може опрацювати спірну ситуацію
9	Модуль доступу до бази даних	Інтеграційне	Запис, читання, оновлення та видалення даних у SQLite	SQL-операції виконуються без втрати або дублювання інформації
10	Графічний інтерфейс PyQt6	Інтерфейсне	Перехід між вікнами, натискання кнопок, заповнення форм	Інтерфейс реагує коректно, дані передаються до відповідних модулів
11	Валідація введених даних	Модульне	Введення порожніх полів, неправильних сум, некоректних валют	Система блокує помилкові дії та виводить повідомлення користувачу

Методика тестування базується на порівнянні фактичної поведінки застосунку з очікуваними результатами для кожного програмного модуля. Для перевірки використовуються типові сценарії роботи користувача: створення облікового запису, вхід у систему, формування заявки, пошук пропозиції, погодження обміну, надсилання повідомлень і завершення угоди. Окремо перевіряються помилкові сценарії, зокрема введення неповних даних, неправильного пароля, некоректної суми або спроба підтвердити угоду без погоджених умов.

Під час тестування модуля автентифікації перевіряється створення нового користувача, пошук облікового запису в базі SQLite, порівняння хешу пароля та відкриття головного меню після успішного входу. У разі помилки система не надає доступ до основних функцій і відображає повідомлення про некоректні облікові дані. Це забезпечує базовий рівень захисту застосунку та розмежування доступу між користувачем і адміністратором.

Функціональне тестування модуля заявок охоплює перевірку введення параметрів обміну, збереження заявки в таблиці `exchange_orders`, присвоєння статусу `active` та подальше відображення заявки у списку доступних пропозицій. Також перевіряється реакція системи на неповне заповнення форми, від'ємні суми, однакові валюти в полях обміну та відсутність міста або умов передачі готівки.

Модуль пошуку пропозицій тестується шляхом створення декількох заявок із різними валютами, сумами, містами та статусами. Після цього виконується фільтрація активних заявок за заданими параметрами. Очікуваним результатом є відображення лише тих пропозицій, які відповідають умовам пошуку та мають актуальний статус для обміну.

Інтеграційне тестування модуля угод передбачає перевірку переходу від заявки до погодженої операції між двома користувачами. У процесі тестування контролюється створення запису в таблиці `exchange_deals`, зв'язок із відповідною заявкою, збереження учасників угоди, місця та часу зустрічі, а

також зміна статусу після підтвердження передачі готівки. Угода вважається завершеною лише після підтвердження з обох сторін.

Модуль повідомлень перевіряється через надсилання текстових повідомлень між учасниками угоди. Контролюється правильність збереження відправника, отримувача, тексту повідомлення, часу надсилання та ознаки прочитання. Успішним результатом є коректне відображення історії повідомлень у межах конкретної угоди без змішування даних між різними користувачами.

Окремо тестується модуль модерації, який використовується адміністратором для розгляду скарг і контролю спірних ситуацій. Перевіряється створення скарги користувачем, її відображення в адміністративному вікні, зміна статусу розгляду та фіксація результату. Це дозволяє перевірити, чи забезпечує система мінімальний контроль за коректністю взаємодії учасників P2P-обміну.

Під час тестування роботи з базою SQLite перевіряється виконання основних SQL-операцій: додавання, читання, оновлення та видалення записів. Особлива увага приділяється зв'язкам між таблицями `users`, `user_profiles`, `currencies`, `currency_rates`, `exchange_orders`, `exchange_requests`, `exchange_deals`, `messages` і `complaints`. Коректність цих зв'язків є важливою умовою стабільної роботи застосунку.

Тестування графічного інтерфейсу PyQt6 виконується через перевірку переходів між вікнами, натискання кнопок, заповнення форм, оновлення таблиць і відображення повідомлень про помилки. Інтерфейс повинен забезпечувати зрозумілу послідовність дій користувача та не допускати виконання операцій із некоректними або неповними даними.

У процесі навантажувального тестування перевіряється стабільність локального застосунку під час роботи з великою кількістю тестових записів. Для цього створюються декілька користувачів, заявок, угод і повідомлень, після чого виконується пошук, фільтрація та оновлення статусів. Очікуваним результатом є збереження прийнятної швидкодії інтерфейсу та відсутність помилок під час звернення до SQLite.

Отже, тестування програмних модулів підтверджує працездатність основних функцій застосунку. Перевірені сценарії охоплюють повний цикл P2P-обміну готівки нестандартних валют: від входу користувача та створення заявки до погодження угоди, обміну повідомленнями, підтвердження передачі коштів і розгляду можливих скарг. Запропонований план тестування створює основу для подальшого оцінювання ефективності та надійності розробленої системи.

4.2 Тестування системи

Тестування системи «Фінансовий помічник для P2P-обміну готівки нестандартних валют» виконано з метою перевірки працездатності розробленого desktop-застосунку, реалізованого засобами Python, PyQt6 та SQLite. Основна увага приділялася перевірці інтерфейсних модулів, коректності збереження даних у локальній базі, правильності виконання операцій із заявками, угодами, повідомленнями, валютами та формуванням аналітичної інформації.

Під час тестування перевірялися типові сценарії роботи користувача: авторизація, реєстрація нового облікового запису, перегляд загальної статистики, робота з каталогом валют, перегляд заявок на обмін, створення нової заявки, керування угодами та формування звітів. Окремо контролювалася реакція системи на некоректні або неповні дані, а також правильність оновлення інформації після виконання користувацьких дій.

На рисунку 4.1 наведено вікно авторизації користувача. У межах цього тесту перевірено введення електронної пошти та пароля, роботу кнопки входу, перемикання між режимами авторизації та реєстрації, а також відкриття основного інтерфейсу після успішного входу.

Вхід до системи

Темна професійна оболонка, SQLite-БД, ролі користувача та адміністратора

Авторизація Реєстрація

Електронна пошта dmin@example.com

Пароль

Увійти

Рисунок 4.1 – Вікно авторизації користувача

На рисунку 4.2 представлено форму реєстрації нового користувача. Тестування цього модуля передбачало введення ПІБ, електронної пошти, телефону, пароля та повтору пароля. Система перевіряє заповнення обов'язкових полів, збіг паролів і коректність створення нового запису користувача у базі SQLite.

Вхід до системи

Темна професійна оболонка, SQLite-БД, ролі користувача та адміністратора

Авторизація Реєстрація

ПІБ |

Email

Телефон

Пароль

Повтор пароля

Зареєструватися

Рисунок 4.2 – Вікно реєстрації користувача

Після входу до системи користувачу відкривається головна панель огляду, наведена на рисунку 4.3. Вона містить основні показники роботи застосунку: кількість відкритих заявок, активних угод, завершених операцій та загальний обсяг завершених обмінів. Під час тестування перевірено, що ці показники формуються на основі фактичних даних із локальної бази та змінюються після створення або завершення угод.

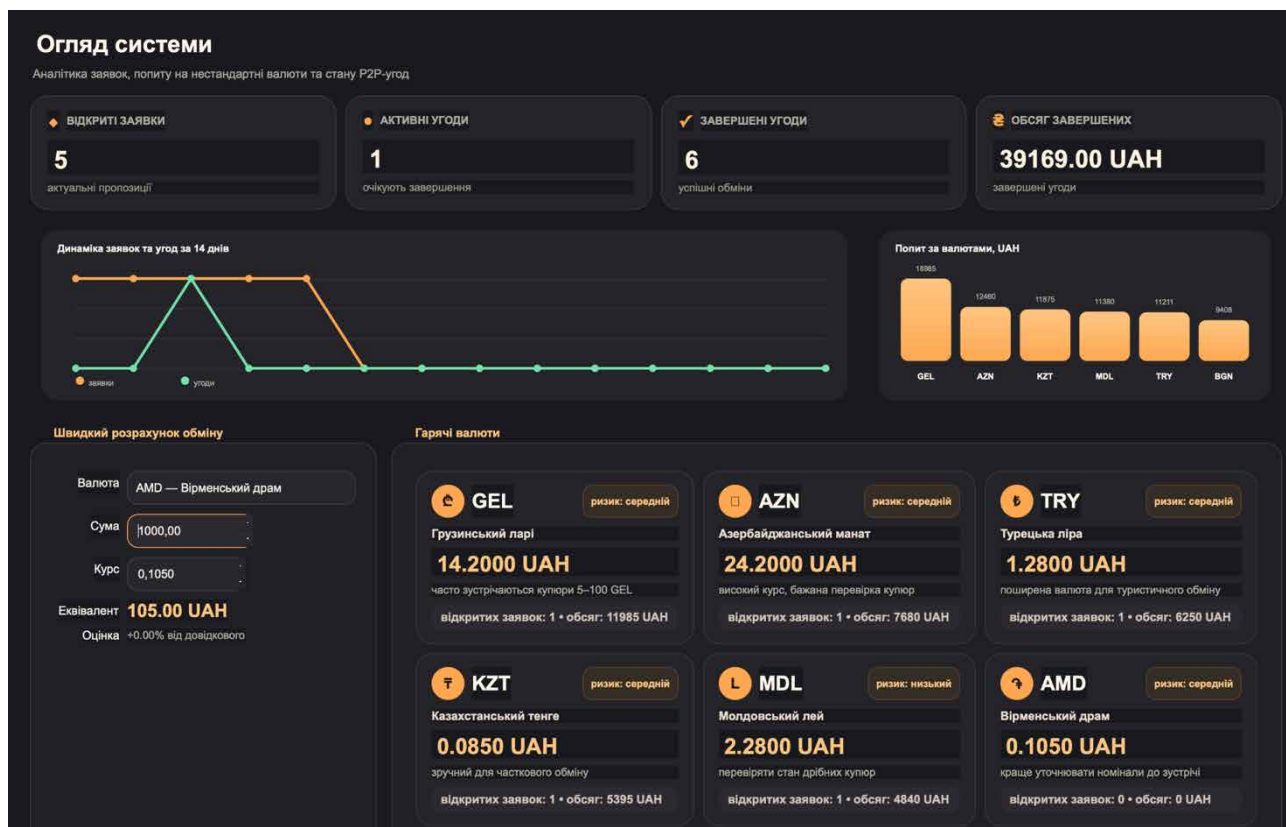


Рисунок 4.3 – Головна панель огляду системи

На рисунку 4.4 показано модуль «Валюти та ринкова активність». Він використовується для перегляду нестандартних валют, довідкових курсів, рівня ризику та обсягу відкритих заявок. У процесі тестування перевірено оновлення довідкового курсу, відображення карток валют і правильність розрахунку обсягу попиту за відкритими заявками.

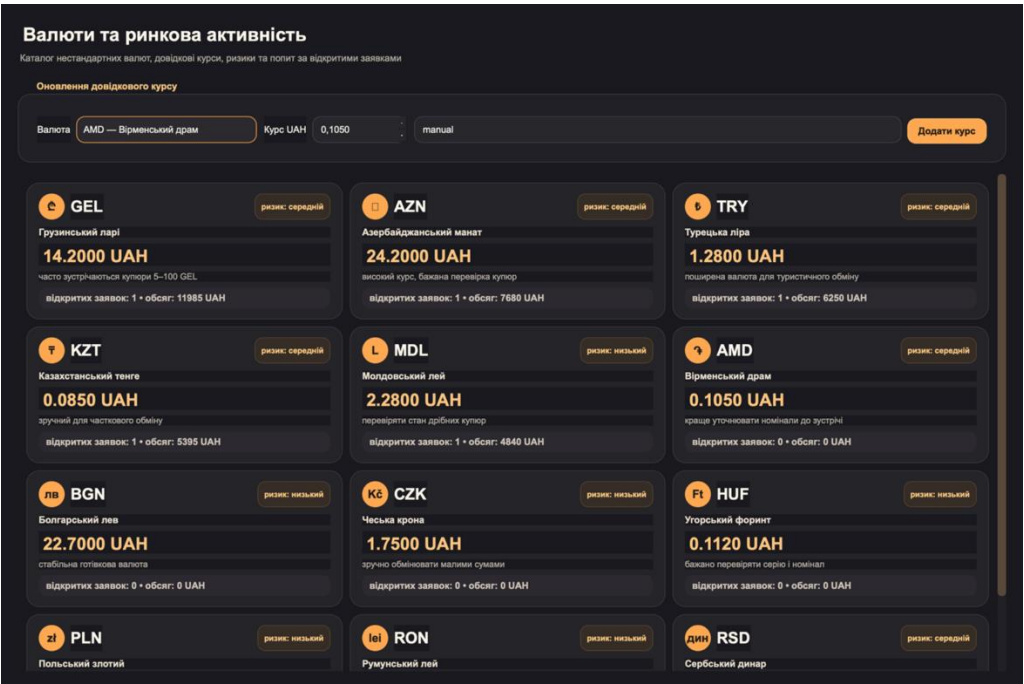


Рисунок 4.4 – Модуль валют та ринкової активності

Модуль заявок на обмін наведено на рисунку 4.5. У ньому перевірялися фільтрація за валютою, статусом, містом і сумою, вибір рядка таблиці, копіювання деталей, створення угоди на основі заявки та закриття заявки. Результати тестування показали, що застосунок коректно відображає активні заявки та оновлює їхній стан після виконання операцій.

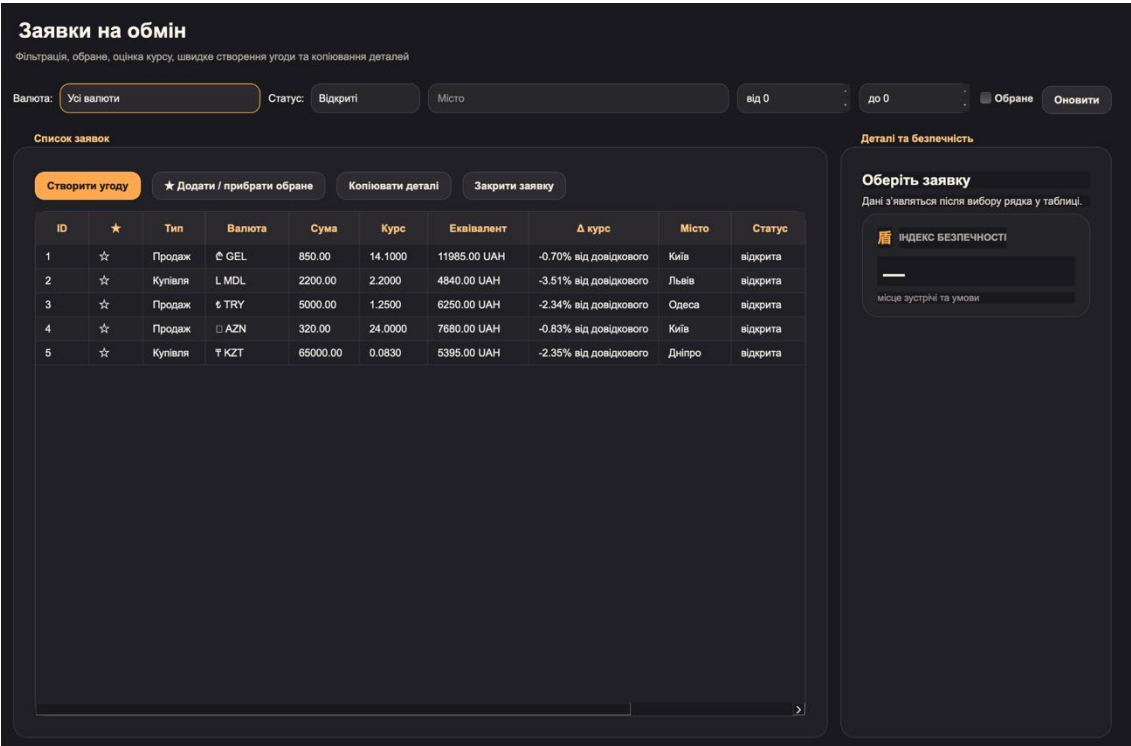


Рисунок 4.5 – Вікно перегляду заявок на обмін

На рисунку 4.6 наведено форму створення заявки. Тестування цього модуля включало введення типу операції, вибір валюти, зазначення суми, курсу, міста, місця зустрічі, терміну дії заявки та коментаря. Система автоматично розраховує еквівалент у гривні, визначає відхилення від довідкового курсу та формує попередню оцінку безпечності зустрічі. Після натискання кнопки «Опублікувати заявку» дані зберігаються у базі SQLite та стають доступними в загальному списку заявок.

Створення заявки
Публікація пропозиції з автоматичним розрахунком еквівалента, відхилення курсу та індексу безпечності

Параметри обміну

Тип операції: Продати нестандартну валюту

Валюта: AMD — Вірменський драм

Базова валюта: UAH

Сума: 1000.00

Курс: 0.1050

Взяти довідковий курс

Місто: Наприклад: Київ

Шаблон місця: Обрати шаблон безпечного місця

Місце зустрічі: Люднє безпечнє ...

Термін дії, днів: 14

Шаблон коментаря: Додати типовий коментар

Коментар: Додаткові умови: номінали купюр, частковий обмін, час зустрічі тощо

Опублікувати заявку

Попередній перегляд

ЕКВІВАЛЕНТ
105.00 UAH
сума 1000.00 × курс 0.1050

% ВІДХИЛЕННЯ КУРСУ
+0.00% від довідкового
довідковий курс: 0.1050

БЕЗПЕЧНІСТЬ ЗУСТРІЧІ
45/100
потрібна обережність

Продаж 1000.00 AMD за курсом 0.1050. Місто: —. Місце: —.

Рисунок 4.6 – Форма створення заявки на обмін

На рисунку 4.7 представлено модуль «Угоди та повідомлення». Під час тестування перевірено перегляд створених і завершених угод, зміну статусу, підтвердження, завершення або скасування угоди, а також роботу чату між учасниками. Дані про угоди та повідомлення коректно зберігаються у відповідних таблицях бази даних і відображаються після оновлення інтерфейсу.

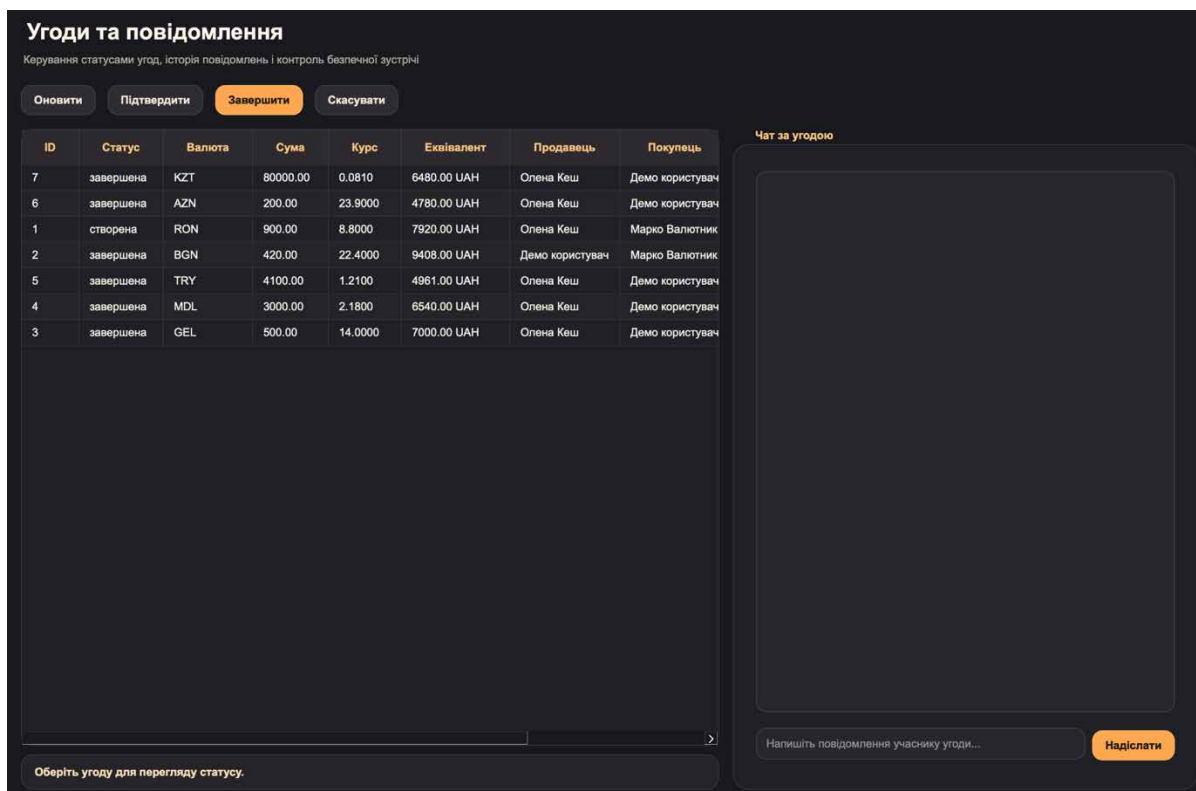


Рисунок 4.7 – Вікно керування угодами та повідомленнями

На рисунку 4.8 наведено модуль звітів та аналітики. Він забезпечує формування фінансового звіту за вибраний період, підрахунок кількості угод, завершеного обсягу, середнього курсу, розподілу за валютами та статусами. Тестування підтвердило, що аналітичні показники формуються на основі завершених угод і можуть використовуватися для оцінювання активності P2P-обміну.

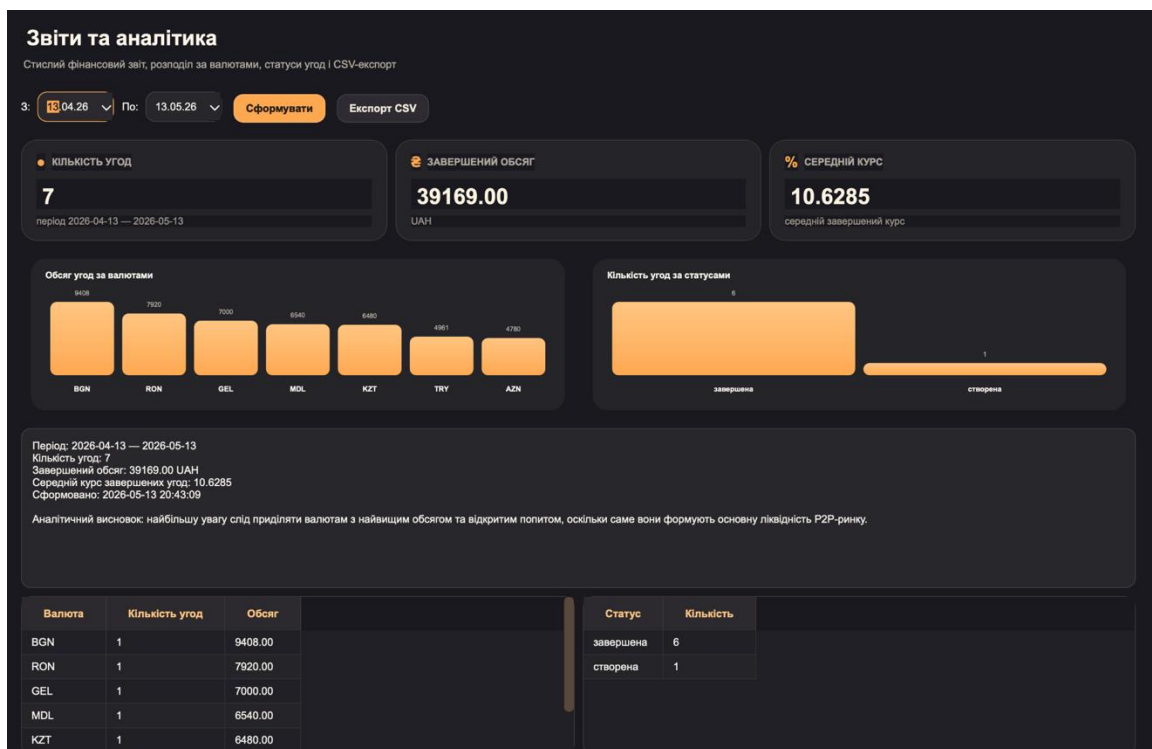


Рисунок 4.8 – Вікно звітів та аналітики системи

Результати тестування системи організовано в два блоки: позитивні сценарії (таблиця 4.2), що перевіряють коректну роботу функцій, та негативні сценарії (таблиця 4.3), що перевіряють відповідь системи на помилкові дані. Позитивні тест-кейси перевіряють типові сценарії: реєстрацію, вхід, створення заявок, пошук, обмін повідомленнями та формування звітів. Негативні тест-кейси перевіряють обробку помилок: неправильні формати даних, відсутність ресурсів, недійсні операції та контроль доступу.

Таблиця 4.2 - Результати тестування системи

№	Модуль системи	Перевірені дії	Результат
1	Авторизація	Вхід користувача за email і паролем	Успішно, доступ надається після коректного введення даних
2	Реєстрація	Створення нового облікового запису	Успішно, дані користувача зберігаються у SQLite
3	Головна панель	Перегляд загальних показників системи	Успішно, статистика формується на основі даних БД
4	Валюти	Перегляд валют і оновлення довідкового курсу	Успішно, курс і параметри валют відображаються коректно
5	Заявки	Фільтрація, перегляд і вибір заявок	Успішно, таблиця відображає актуальні записи

6	Створення заявки	Введення параметрів і публікація заявки	Успішно, заявка отримує статус відкритої
7	Угоди	Підтвердження, завершення та скасування угод	Успішно, статуси змінюються коректно
8	Повідомлення	Надсилення повідомлення в межах угоди	Успішно, повідомлення зберігається та відображається

Проведене тестування підтвердило, що розроблений застосунок забезпечує виконання основних функціональних сценаріїв. Інтерфейс PyQt6 коректно реагує на дії користувача, модулі прикладної логіки правильно обробляють введені дані, а база SQLite забезпечує збереження користувачів, валют, заявок, угод, повідомлень і звітної інформації.

Отже, результати тестування свідчать про працездатність системи та відповідність реалізованих модулів поставленим вимогам. Застосунок забезпечує повний цикл P2P-обміну готівки нестандартних валют: від реєстрації користувача та створення заявки до формування угоди, обміну повідомленнями, завершення операції й аналізу результатів.

4.3 Розгортання системи, склад інсталяційного пакету

Розгортання програмного застосунку «Фінансовий помічник для P2P-обміну готівки нестандартних валют» передбачає підготовку локального середовища для запуску desktop-програми, реалізованої на Python із використанням PyQt6 та бази даних SQLite. На відміну від веб-систем, застосунок не потребує окремого сервера, веб-хостингу або зовнішньої СУБД, оскільки основна логіка, графічний інтерфейс і база даних працюють у межах одного локального програмного середовища.

Основним способом розгортання є встановлення або запуск виконуваного файлу застосунку на персональному комп'ютері користувача. Після запуску програми відкривається графічний інтерфейс, через який користувач може пройти авторизацію, створити заявку на обмін, переглянути доступні пропозиції, сформувати угоду, обмінюватися повідомленнями та переглядати звіти. Уся

інформація зберігається у локальному файлі бази даних SQLite, що спрощує встановлення та забезпечує автономну роботу системи.

Розгортання застосунку виконується у декілька етапів. Спочатку готується робочий каталог програми, до якого входять виконуваний файл, база даних, службові ресурси інтерфейсу та конфігураційні файли. Далі перевіряється наявність файлу бази даних. Якщо база відсутня, застосунок створює її структуру автоматично або використовує підготовлений шаблон із таблицями користувачів, валют, заявок, угод, повідомлень, скарг і журналу дій. Після цього запускається головний модуль програми, який ініціалізує інтерфейс PyQt6 та підключення до SQLite.

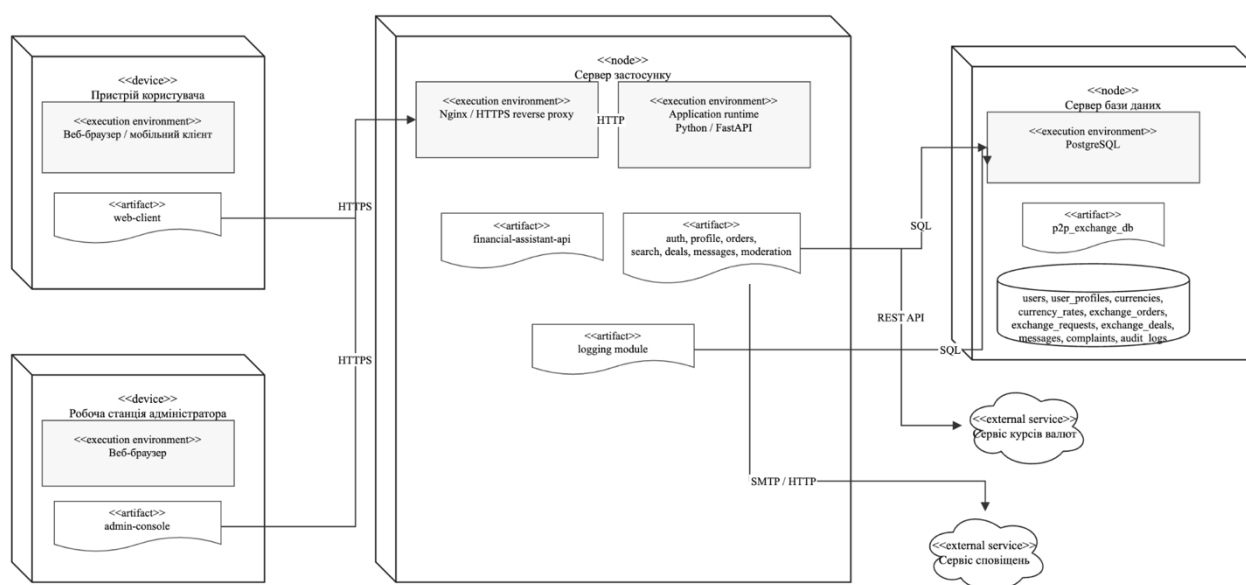


Рисунок 4.9 – Діаграма розгортання програмного застосунку «Фінансовий помічник»

У межах розгортання важливо забезпечити коректне розміщення всіх файлів, необхідних для роботи програми. Виконуваний файл відповідає за запуск застосунку, файл бази даних містить основні записи системи, а ресурси інтерфейсу забезпечують відображення стилів, іконок і допоміжних елементів. Окремо можуть використовуватися файли налаштувань, у яких зберігаються параметри теми інтерфейсу, шлях до бази даних і службові параметри запуску.

Склад інсталяційного пакету наведено в таблиці 4.3.

Таблиця 4.3 - Склад інсталяційного пакету застосунку

Компонент пакету	Призначення
financial_assistant.exe	Основний виконуваний файл для запуску застосунку
main.py	Головний програмний модуль у версії для запуску з середовища Python
ui/	Каталог із файлами графічного інтерфейсу PyQt6
styles/	Каталог із QSS-стилями для оформлення інтерфейсу
resources/	Каталог із іконками, службовими зображеннями та іншими ресурсами
database/p2p_exchange.db	Локальна база даних SQLite
database/schema.sql	SQL-скрипт створення структури бази даних
database/seed.sql	Початкові довідкові дані для ролей, валют і тестових записів
modules/auth.py	Модуль авторизації та реєстрації користувачів
modules/orders.py	Модуль створення, пошуку й оброблення заявок
modules/deals.py	Модуль формування та завершення угод
modules/messages.py	Модуль повідомлень між учасниками угоди
modules/reports.py	Модуль формування звітів та аналітики
config.json	Файл базових налаштувань застосунку
README.txt	Інструкція із запуску та використання програми

Для користувача процес встановлення зводиться до розпакування інсталяційного пакету в окремий каталог і запуску виконуваного файлу. Якщо застосунок поширюється у вигляді exe-файлу, на комп'ютері користувача не потрібно окремо встановлювати Python, PyQt6 або додаткові бібліотеки, оскільки вони включаються до зібраного пакету через PyInstaller. У разі запуску з вихідного коду необхідно встановити Python, залежності проекту та запустити головний файл main.py.

Для стабільної роботи застосунку потрібно забезпечити права на читання та запис у каталозі, де розміщена база даних SQLite. Це необхідно для збереження нових користувачів, заявок, угод, повідомлень і звітної інформації. У разі переміщення програми до іншого каталогу файл бази даних має залишатися доступним для основного модуля застосунку.

Рекомендовані мінімальні вимоги до середовища запуску наведено у таблиці 4.4.

Таблиця 4.4 - Мінімальні вимоги до запуску застосунку

Параметр	Мінімальне значення
Операційна система	Windows 10 / Windows 11
Процесор	Двоядерний процесор із частотою від 2,0 ГГц
Оперативна пам'ять	4 ГБ
Вільне місце на диску	200 МБ
База даних	SQLite, локальний файл .db
Інтерфейс	Desktop GUI на PyQt6
Режим роботи	Локальний запуск без окремого сервера

Після розгортання виконується контрольний запуск застосунку. Перевіряється відкриття вікна авторизації, доступність реєстрації, коректне підключення до SQLite, відображення списку валют, створення тестової заявки та збереження її у базі даних. Якщо всі перевірки виконуються без помилок, застосунок вважається готовим до використання.

До складу інсталяційного пакету також доцільно включити початкові довідкові дані. До них належать ролі користувача й адміністратора, перелік нестандартних валют, тестові курси валют і демонстраційні записи для перевірки роботи системи. Це дає змогу швидко перевірити функціональність програми після встановлення без ручного наповнення бази даних.

Отже, розгортання застосунку є простим і не потребує складної серверної інфраструктури. Використання PyQt6 та SQLite дозволяє реалізувати автономну desktop-систему, яка запускається локально, зберігає дані у файлі бази даних і забезпечує повний цикл роботи з P2P-обміном готівки нестандартних валют. Склад інсталяційного пакету забезпечує готовність програми до запуску, тестування та подальшого практичного використання.

4.4 Висновки до четвертого розділу

У четвертому розділі кваліфікаційної роботи виконано тестування та оцінювання працездатності програмного застосунку «Фінансовий помічник для P2P-обміну готівки нестандартних валют». Основну увагу приділено перевірці функціональних модулів, графічного інтерфейсу PyQt6, коректності роботи з локальною базою даних SQLite та стабільності виконання основних сценаріїв обміну.

У процесі тестування перевірено модулі авторизації, реєстрації, профілю користувача, перегляду валют, створення заявок, пошуку пропозицій, формування угод, обміну повідомленнями, звітності та аналітики. Результати перевірки показали, що система коректно обробляє введені дані, зберігає інформацію у базі SQLite, оновлює статуси заявок і угод та забезпечує послідовну взаємодію користувача з основними функціями застосунку.

Окремо перевірено роботу інтерфейсних модулів. Вікна авторизації, реєстрації, огляду системи, валютної активності, заявок, створення нової заявки, угод і звітів забезпечують виконання відповідних операцій без критичних помилок. Інтерфейс реагує на дії користувача, відображає актуальні дані з бази та надає повідомлення у випадку некоректного або неповного введення інформації.

Під час тестування роботи з базою даних підтверджено правильність збереження користувачів, валют, курсів, заявок, угод, повідомлень і звітних даних. Використання SQLite забезпечило швидкий локальний доступ до інформації та спростило розгортання застосунку без потреби в окремому сервері бази даних.

У межах розділу також описано розгортання системи та склад інсталяційного пакету. Визначено, що застосунок може працювати як автономна desktop-програма, до складу якої входять виконуваний файл, модулі Python, ресурси інтерфейсу, QSS-стилі, файл бази даних SQLite, SQL-скрипти створення

структури БД та службові конфігураційні файли. Такий склад забезпечує простий запуск і можливість подальшого супроводу програмного забезпечення.

Отже, проведене тестування підтвердило працездатність розробленої системи та її відповідність основним функціональним вимогам. Застосунок забезпечує повний цикл P2P-обміну готівки нестандартних валют: від реєстрації користувача і створення заявки до узгодження угоди, обміну повідомленнями, завершення операції та формування аналітичного звіту.

ВИСНОВКИ

У кваліфікаційній роботі вирішено практичне завдання розроблення програмного забезпечення застосунку “Фінансовий помічник” для підтримки процесу готівкового P2P-обміну нестандартних валют між користувачами. У результаті виконання роботи створено локальний desktop-застосунок, який забезпечує роботу з користувачами, заявками на обмін, пошуком пропозицій, угодами, повідомленнями, валютними довідниками та звітною інформацією.

Перше завдання роботи виконано шляхом аналізу предметної області P2P-обміну валют. Було розглянуто особливості прямої взаємодії між користувачами, проблеми обміну нестандартних і малопоширених валют, а також обмеження традиційних банківських сервісів та обмінних пунктів. Установлено, що для таких операцій доцільним є використання цифрових систем, які дають змогу створювати заявки, знаходити відповідні пропозиції та фіксувати домовленості між учасниками обміну.

Друге завдання виконано через формування функціональних і нефункціональних вимог до системи. До основних функціональних вимог віднесено реєстрацію та авторизацію користувачів, створення заявок на обмін, пошук пропозицій, формування угод, надсилання повідомлень, використання довідника валют, отримання курсів валют і формування звітів. Нефункціональні вимоги охоплюють зручність інтерфейсу, надійність збереження даних, коректність обробки введеної інформації, захист паролів шляхом хешування та можливість локального розгортання застосунку.

Третє завдання виконано завдяки побудові функціональних та інформаційних моделей системи. У роботі розроблено UML-діаграми, які відображають основні сценарії використання, взаємодію користувачів із системою, структуру компонентів і логіку роботи програмних модулів. Також сформовано логічну модель бази даних, у якій визначено основні сутності: користувач, заявка на обмін, угода, повідомлення, валюта та курс валют.

Четверте завдання виконано шляхом розроблення архітектури програмного забезпечення та моделі даних. Для реалізації системи обрано локальну desktop-архітектуру на основі Python, PyQt6 і SQLite. Такий підхід дав змогу створити застосунок без окремого серверного середовища, із локальним збереженням даних у базі SQLite. Було спроектовано фізичну структуру бази даних, визначено таблиці, первинні та зовнішні ключі, зв'язки між сутностями й обмеження цілісності. Структура бази даних приведена до основних вимог нормалізації, що зменшує дублювання інформації та підвищує узгодженість даних.

П'яте завдання виконано через реалізацію основних програмних модулів застосунку. Було реалізовано модуль автентифікації з хешуванням пароля, модуль створення заявок на обмін, модуль пошуку доступних пропозицій, модуль формування угод, модуль повідомлень і модуль формування звітної інформації. Окремо передбачено SQL-структуру бази даних, що дає змогу створити необхідні таблиці, довідники, індекси та зв'язки для роботи системи.

Шосте завдання виконано шляхом проведення тестування програмного забезпечення. Перевірено роботу основних функцій застосунку: авторизації користувача, створення заявки, пошуку пропозицій, формування угоди, надсилення повідомлення, зміни статусів і формування звітів. За результатами тестування встановлено, що розроблені модулі коректно взаємодіють із базою даних, забезпечують виконання основних сценаріїв роботи та відповідають поставленим вимогам.

Практичне значення роботи полягає у створенні програмного прототипу системи, який може використовуватися для моделювання процесу готівкового P2P-обміну нестандартних валют. Застосунок дозволяє впорядкувати роботу із заявками, фіксувати домовленості між користувачами, зберігати історію операцій і формувати звітну інформацію. Розроблена система може бути використана як основа для подальшого створення повноцінної платформи обміну валют.

Водночас розроблена версія має певні обмеження. Система реалізована як локальний desktop-застосунок і не забезпечує повноцінну багатокористувацьку мережеву взаємодію в реальному часі. Використання SQLite є доцільним для локального прототипу, однак для повноцінної P2P-платформи необхідне серверне середовище, централізована база даних, захищений API, синхронізація дій між користувачами та механізми реального обміну повідомленнями. Тому поточну версію доцільно розглядати як функціональний прототип або імітаційну модель P2P-системи.

Напрямами подальшого розвитку є перехід до клієнт-серверної архітектури, реалізація мережевої взаємодії між користувачами, підключення централізованої бази даних, створення веб- або мобільного інтерфейсу, додавання рейтингу користувачів, механізмів верифікації, системи скарг, розширеного журналювання подій і підвищених засобів захисту даних. Це дозволить перетворити локальний прототип на повноцінну багатокористувацьку платформу для організації P2P-обміну готівки нестандартних валют.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ДСТУ ISO/IEC/IEEE 12207:2018. Інженерія систем і програмних засобів. Процеси життєвого циклу програмних засобів. Київ : ДП «УкрНДНЦ», 2018.
2. ДСТУ ISO/IEC 25010:2016. Інженерія систем і програмних засобів. Вимоги до якості систем і програмних засобів та її оцінювання (SQuaRE). Моделі якості системи та програмних засобів. Київ : ДП «УкрНДНЦ», 2016.
3. ISO/IEC 25010:2011. Systems and software engineering. Systems and software Quality Requirements and Evaluation (SQuaRE). System and software quality models. Geneva : ISO, 2011.
4. ISO/IEC/IEEE 12207:2017. Systems and software engineering. Software life cycle processes. Geneva : ISO, 2017.
5. OMG Unified Modeling Language. Version 2.5.1 [Electronic resource]. Object Management Group, 2017. URL: <https://www.omg.org/spec/UML/2.5.1> (date of access: 22.05.2026).
6. Sommerville I. Software Engineering. 10th ed. Boston : Pearson, 2016. 816 p.
7. Pressman R. S., Maxim B. R. Software Engineering: A Practitioner's Approach. 9th ed. New York : McGraw-Hill Education, 2019. 880 p.
8. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. 3rd ed. Boston : Addison-Wesley, 2003. 208 p.
9. Larman C. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. 3rd ed. Upper Saddle River : Prentice Hall, 2004. 736 p.
10. Elmasri R., Navathe S. B. Fundamentals of Database Systems. 7th ed. Boston : Pearson, 2016. 1280 p.
11. Silberschatz A., Korth H. F., Sudarshan S. Database System Concepts. 7th ed. New York : McGraw-Hill Education, 2019. 1344 p.

12. Python 3 Documentation [Electronic resource] / Python Software Foundation. URL: <https://docs.python.org/3/> (date of access: 22.05.2026).
13. sqlite3 — DB-API 2.0 interface for SQLite databases [Electronic resource] / Python Software Foundation. URL: <https://docs.python.org/3/library/sqlite3.html> (date of access: 22.05.2026).
14. PyQt6 [Electronic resource] / Python Package Index. URL: <https://pypi.org/project/PyQt6/> (date of access: 22.05.2026).
15. PyQt Components [Electronic resource] / Riverbank Computing. URL: <https://riverbankcomputing.com/software/pyqt/intro> (date of access: 22.05.2026).
16. Qt for Python Documentation [Electronic resource] / The Qt Company. URL: <https://doc.qt.io/qtforpython-6/> (date of access: 22.05.2026).
17. PySide6.QtWidgets [Electronic resource] / Qt for Python Documentation. The Qt Company. URL: <https://doc.qt.io/qtforpython-6/PySide6/QtWidgets/index.html> (date of access: 22.05.2026).
18. Qt Style Sheets Examples [Electronic resource] / Qt Documentation. The Qt Company. URL: <https://doc.qt.io/qt-6/stylesheet-examples.html> (date of access: 22.05.2026).
19. SQLite Documentation [Electronic resource] / SQLite Consortium. URL: <https://sqlite.org/docs.html> (date of access: 22.05.2026).
20. Datatypes In SQLite [Electronic resource] / SQLite Documentation. SQLite Consortium. URL: <https://sqlite.org/datatype3.html> (date of access: 22.05.2026).
21. SQLite Foreign Key Support [Electronic resource] / SQLite Documentation. SQLite Consortium. URL: <https://sqlite.org/foreignkeys.html> (date of access: 22.05.2026).
22. PyInstaller Manual [Electronic resource] / PyInstaller Development Team. URL: <https://pyinstaller.org/> (date of access: 22.05.2026).
23. OWASP Authentication Cheat Sheet [Electronic resource] / OWASP Cheat Sheet Series. Open Worldwide Application Security Project.

URL: https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html (date of access: 22.05.2026).

24. OWASP Password Storage Cheat Sheet [Electronic resource] / OWASP Cheat Sheet Series. Open Worldwide Application Security Project. URL: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html (date of access: 22.05.2026).

25. OWASP SQL Injection Prevention Cheat Sheet [Electronic resource] / OWASP Cheat Sheet Series. Open Worldwide Application Security Project. URL: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html (date of access: 22.05.2026).

26. Temoshok D. et al. NIST Special Publication 800-63B-4. Digital Identity Guidelines: Authentication and Authenticator Management [Electronic resource]. Gaithersburg : National Institute of Standards and Technology, 2025. URL: <https://csrc.nist.gov/pubs/sp/800/63/b/4/final> (date of access: 22.05.2026).

27. Revolut. Multi-Currency Bank Account [Electronic resource] / Revolut Ltd. URL: <https://www.revolut.com/multi-currency-account/> (date of access: 22.05.2026).

28. PayPal. Send Money, Pay Online or Set Up a Merchant Account [Electronic resource] / PayPal Holdings, Inc. URL: <https://www.paypal.com/> (date of access: 22.05.2026).

29. CurrencyFair. Send Money Abroad. International Transfers [Electronic resource] / CurrencyFair Ltd. URL: <https://www.currencyfair.com/> (date of access: 22.05.2026).

30. Paxful. Leading Peer-to-Peer Crypto Marketplace Worldwide [Electronic resource] / Paxful, Inc. URL: <https://withdraw.paxful.com/> (date of access: 22.05.2026).

31. Deriv P2P. Peer-to-Peer Payments Service [Electronic resource] / Deriv Group. URL: <https://deriv.com/p2p> (date of access: 22.05.2026).

32. Your guide to safe and secure Deriv P2P transactions [Electronic resource] / Deriv Trader's Academy. URL: <https://traders-academy.deriv.com/trading->

[guides/your-guide-to-safe-and-secure-deriv-p2p-transactions](#) (date of access: 22.05.2026).

ДОДАТОК А

SQL-структура бази даних локального застосунку

```
-- schema.sql
-- SQL-структура бази даних локального desktop-застосунку
-- "Фінансовий помічник" для P2P-обміну готівки нестандартних валют

PRAGMA foreign_keys = ON;

CREATE TABLE IF NOT EXISTS roles (
    role_id INTEGER PRIMARY KEY AUTOINCREMENT,
    role_name TEXT NOT NULL UNIQUE
);

CREATE TABLE IF NOT EXISTS users (
    user_id INTEGER PRIMARY KEY AUTOINCREMENT,
    full_name TEXT NOT NULL,
    phone TEXT,
    email TEXT NOT NULL UNIQUE,
    password_hash TEXT NOT NULL,
    password_salt TEXT NOT NULL,
    role_id INTEGER NOT NULL,
    status TEXT NOT NULL DEFAULT 'active',
    created_at TEXT NOT NULL DEFAULT CURRENT_TIMESTAMP,

    FOREIGN KEY (role_id) REFERENCES roles(role_id)
        ON UPDATE CASCADE
        ON DELETE RESTRICT,

    CHECK (status IN ('active', 'blocked', 'deleted'))
);

CREATE TABLE IF NOT EXISTS currencies (
    currency_id INTEGER PRIMARY KEY AUTOINCREMENT,
    currency_code TEXT NOT NULL UNIQUE,
    currency_name TEXT NOT NULL,
    currency_type TEXT NOT NULL DEFAULT 'standard',

    CHECK (currency_type IN ('standard', 'non_standard', 'regional', 'other'))
);

CREATE TABLE IF NOT EXISTS exchange_rates (
    rate_id INTEGER PRIMARY KEY AUTOINCREMENT,
    from_currency_id INTEGER NOT NULL,
    to_currency_id INTEGER NOT NULL,
    rate_value REAL NOT NULL,
    loaded_at TEXT NOT NULL DEFAULT CURRENT_TIMESTAMP,

    FOREIGN KEY (from_currency_id) REFERENCES currencies(currency_id)
        ON UPDATE CASCADE
        ON DELETE RESTRICT,

    FOREIGN KEY (to_currency_id) REFERENCES currencies(currency_id)
        ON UPDATE CASCADE
        ON DELETE RESTRICT,

    CHECK (rate_value > 0),
    CHECK (from_currency_id <> to_currency_id)
```

```

);

CREATE TABLE IF NOT EXISTS exchange_requests (
    request_id INTEGER PRIMARY KEY AUTOINCREMENT,
    user_id INTEGER NOT NULL,
    give_currency_id INTEGER NOT NULL,
    receive_currency_id INTEGER NOT NULL,
    amount REAL NOT NULL,
    proposed_rate REAL,
    city TEXT NOT NULL,
    comment TEXT,
    status TEXT NOT NULL DEFAULT 'active',
    created_at TEXT NOT NULL DEFAULT CURRENT_TIMESTAMP,
    closed_at TEXT,

    FOREIGN KEY (user_id) REFERENCES users(user_id)
        ON UPDATE CASCADE
        ON DELETE CASCADE,

    FOREIGN KEY (give_currency_id) REFERENCES currencies(currency_id)
        ON UPDATE CASCADE
        ON DELETE RESTRICT,

    FOREIGN KEY (receive_currency_id) REFERENCES currencies(currency_id)
        ON UPDATE CASCADE
        ON DELETE RESTRICT,

    CHECK (amount > 0),
    CHECK (proposed_rate IS NULL OR proposed_rate > 0),
    CHECK (give_currency_id <> receive_currency_id),
    CHECK (status IN ('active', 'reserved', 'closed', 'cancelled'))
);

CREATE TABLE IF NOT EXISTS deals (
    deal_id INTEGER PRIMARY KEY AUTOINCREMENT,
    request_id INTEGER NOT NULL UNIQUE,
    owner_user_id INTEGER NOT NULL,
    partner_user_id INTEGER NOT NULL,
    meeting_place TEXT,
    meeting_time TEXT,
    status TEXT NOT NULL DEFAULT 'created',
    created_at TEXT NOT NULL DEFAULT CURRENT_TIMESTAMP,
    completed_at TEXT,

    FOREIGN KEY (request_id) REFERENCES exchange_requests(request_id)
        ON UPDATE CASCADE
        ON DELETE CASCADE,

    FOREIGN KEY (owner_user_id) REFERENCES users(user_id)
        ON UPDATE CASCADE
        ON DELETE RESTRICT,

    FOREIGN KEY (partner_user_id) REFERENCES users(user_id)
        ON UPDATE CASCADE
        ON DELETE RESTRICT,

    CHECK (owner_user_id <> partner_user_id),
    CHECK (status IN ('created', 'confirmed', 'completed', 'cancelled',
'disputed'))
);

CREATE TABLE IF NOT EXISTS messages (
    message_id INTEGER PRIMARY KEY AUTOINCREMENT,

```

```

deal_id INTEGER NOT NULL,
sender_user_id INTEGER NOT NULL,
message_text TEXT NOT NULL,
sent_at TEXT NOT NULL DEFAULT CURRENT_TIMESTAMP,

FOREIGN KEY (deal_id) REFERENCES deals(deal_id)
    ON UPDATE CASCADE
    ON DELETE CASCADE,

FOREIGN KEY (sender_user_id) REFERENCES users(user_id)
    ON UPDATE CASCADE
    ON DELETE RESTRICT
);

CREATE TABLE IF NOT EXISTS action_logs (
    log_id INTEGER PRIMARY KEY AUTOINCREMENT,
    user_id INTEGER,
    action_type TEXT NOT NULL,
    description TEXT,
    created_at TEXT NOT NULL DEFAULT CURRENT_TIMESTAMP,

    FOREIGN KEY (user_id) REFERENCES users(user_id)
        ON UPDATE CASCADE
        ON DELETE SET NULL
);

CREATE INDEX IF NOT EXISTS idx_users_email
ON users(email);

CREATE INDEX IF NOT EXISTS idx_requests_user
ON exchange_requests(user_id);

CREATE INDEX IF NOT EXISTS idx_requests_status
ON exchange_requests(status);

CREATE INDEX IF NOT EXISTS idx_requests_city
ON exchange_requests(city);

CREATE INDEX IF NOT EXISTS idx_deals_status
ON deals(status);

CREATE INDEX IF NOT EXISTS idx_messages_deal
ON messages(deal_id);

INSERT OR IGNORE INTO roles (role_id, role_name)
VALUES
    (1, 'user'),
    (2, 'admin');

INSERT OR IGNORE INTO currencies (currency_code, currency_name, currency_type)
VALUES
    ('USD', 'Долар США', 'standard'),
    ('EUR', 'Євро', 'standard'),
    ('PLN', 'Польський злотий', 'standard'),
    ('TRY', 'Турецька ліра', 'standard'),
    ('GEL', 'Грузинський ларі', 'non_standard'),
    ('MDL', 'Молдовський лей', 'non_standard'),
    ('RON', 'Румунський лей', 'non_standard'),
    ('BGN', 'Болгарський лев', 'non_standard');

```

ДОДАТОК Б

Фрагменти програмного коду основних модулів застосунку

```
# app_core.py
# Основні функції локального desktop-застосунку
# "Фінансовий помічник" для P2P-обміну готівки нестандартних валют

import os
import sqlite3
import hashlib
import binascii
from datetime import datetime

DB_PATH = "financial_helper.db"

def get_connection():
    """Створення підключення до локальної бази даних SQLite."""
    connection = sqlite3.connect(DB_PATH)
    connection.row_factory = sqlite3.Row
    connection.execute("PRAGMA foreign_keys = ON;")
    return connection

def init_database(schema_path="schema.sql"):
    """Ініціалізація бази даних за SQL-файлом schema.sql."""
    with open(schema_path, "r", encoding="utf-8") as file:
        schema = file.read()

    with get_connection() as connection:
        connection.executescript(schema)
        connection.commit()

def hash_password(password: str) -> tuple[str, str]:
    """
    Формування хешу пароля з використанням випадкової солі.
    Для підвищення безпеки використовується алгоритм PBKDF2-HMAC-SHA256.
    """
    salt = os.urandom(16)
    password_hash = hashlib.pbkdf2_hmac(
        "sha256",
        password.encode("utf-8"),
        salt,
        100_000
    )

    return (
        binascii.hexlify(password_hash).decode("utf-8"),
        binascii.hexlify(salt).decode("utf-8")
    )

def verify_password(password: str, stored_hash: str, stored_salt: str) -> bool:
    """Перевірка введеного пароля шляхом повторного хешування."""
    salt = binascii.unhexlify(stored_salt.encode("utf-8"))
    password_hash = hashlib.pbkdf2_hmac(
```

```

        "sha256",
        password.encode("utf-8"),
        salt,
        100_000
    )

    calculated_hash = binascii.hexlify(password_hash).decode("utf-8")
    return calculated_hash == stored_hash

def register_user(full_name: str, phone: str, email: str, password: str) -> int:
    """Реєстрація нового користувача в системі."""
    password_hash, password_salt = hash_password(password)

    with get_connection() as connection:
        cursor = connection.cursor()

        cursor.execute(
            """
            INSERT INTO users (
                full_name, phone, email, password_hash, password_salt, role_id,
status
            )
            VALUES (?, ?, ?, ?, ?, 1, 'active');
            """
            ,
            (full_name, phone, email, password_hash, password_salt)
        )

        connection.commit()
        return cursor.lastrowid

def login_user(email: str, password: str):
    """Авторизація користувача за email і паролем."""
    with get_connection() as connection:
        cursor = connection.cursor()

        cursor.execute(
            """
            SELECT
                u.user_id,
                u.full_name,
                u.email,
                u.password_hash,
                u.password_salt,
                u.status,
                r.role_name
            FROM users u
            JOIN roles r ON r.role_id = u.role_id
            WHERE u.email = ?;
            """
            ,
            (email,)
        )

        user = cursor.fetchone()

        if user is None:
            return None

        if user["status"] != "active":
            return None

        if not verify_password(password, user["password_hash"],

```

```

user["password_salt"]):
    return None

    return {
        "user_id": user["user_id"],
        "full_name": user["full_name"],
        "email": user["email"],
        "role": user["role_name"]
    }

def get_currency_id(currency_code: str) -> int:
    """Отримання ідентифікатора валюти за її кодом."""
    with get_connection() as connection:
        cursor = connection.cursor()

        cursor.execute(
            """
            SELECT currency_id
            FROM currencies
            WHERE currency_code = ?;
            """,
            (currency_code.upper(),)
        )

        currency = cursor.fetchone()

        if currency is None:
            raise ValueError(f"Валюту {currency_code} не знайдено в довіднику.")

        return currency["currency_id"]

def create_exchange_request(
    user_id: int,
    give_currency_code: str,
    receive_currency_code: str,
    amount: float,
    proposed_rate: float,
    city: str,
    comment: str = ""
) -> int:
    """Створення заявки на обмін валюти."""
    give_currency_id = get_currency_id(give_currency_code)
    receive_currency_id = get_currency_id(receive_currency_code)

    if amount <= 0:
        raise ValueError("Сума обміну повинна бути більшою за нуль.")

    if proposed_rate is not None and proposed_rate <= 0:
        raise ValueError("Запропонований курс повинен бути більшим за нуль.")

    with get_connection() as connection:
        cursor = connection.cursor()

        cursor.execute(
            """
            INSERT INTO exchange_requests (
                user_id,
                give_currency_id,
                receive_currency_id,
                amount,
                proposed_rate,
            """

```

```

        city,
        comment,
        status
    )
VALUES (?, ?, ?, ?, ?, ?, ?, 'active');
"""
(
    user_id,
    give_currency_id,
    receive_currency_id,
    amount,
    proposed_rate,
    city,
    comment
)
)

request_id = cursor.lastrowid

cursor.execute(
    """
    INSERT INTO action_logs (user_id, action_type, description)
    VALUES (?, 'create_request', ?);
    """
    (user_id, f"Створено заявку на обмін №{request_id}")
)

connection.commit()
return request_id

def search_exchange_requests(
    give_currency_code: str | None = None,
    receive_currency_code: str | None = None,
    city: str | None = None,
    min_amount: float | None = None,
    max_amount: float | None = None
) -> list[dict]:
    """Пошук активних заявок за заданими параметрами."""
    query = """
    SELECT
        er.request_id,
        u.full_name AS owner_name,
        c1.currency_code AS give_currency,
        c2.currency_code AS receive_currency,
        er.amount,
        er.proposed_rate,
        er.city,
        er.comment,
        er.status,
        er.created_at
    FROM exchange_requests er
    JOIN users u ON u.user_id = er.user_id
    JOIN currencies c1 ON c1.currency_id = er.give_currency_id
    JOIN currencies c2 ON c2.currency_id = er.receive_currency_id
    WHERE er.status = 'active'
    """

    params = []

    if give_currency_code:
        query += " AND c1.currency_code = ?"
        params.append(give_currency_code.upper())

```

```

if receive_currency_code:
    query += " AND c2.currency_code = ?"
    params.append(receive_currency_code.upper())

if city:
    query += " AND LOWER(er.city) LIKE LOWER(?)"
    params.append(f"%{city}%")

if min_amount is not None:
    query += " AND er.amount >= ?"
    params.append(min_amount)

if max_amount is not None:
    query += " AND er.amount <= ?"
    params.append(max_amount)

query += " ORDER BY er.created_at DESC;"

with get_connection() as connection:
    cursor = connection.cursor()
    cursor.execute(query, params)

    return [dict(row) for row in cursor.fetchall()]

def create_deal(
    request_id: int,
    partner_user_id: int,
    meeting_place: str,
    meeting_time: str
) -> int:
    """Створення угоди на основі активної заявки."""
    with get_connection() as connection:
        cursor = connection.cursor()

        cursor.execute(
            """
            SELECT request_id, user_id, status
            FROM exchange_requests
            WHERE request_id = ?;
            """,
            (request_id,)
        )

        request = cursor.fetchone()

        if request is None:
            raise ValueError("Заявку не знайдено.")

        if request["status"] != "active":
            raise ValueError("На основі цієї заявки неможливо створити угоду.")

        owner_user_id = request["user_id"]

        if owner_user_id == partner_user_id:
            raise ValueError("Користувач не може створити угоду із власною
заявкою.")

        cursor.execute(
            """
            INSERT INTO deals (
                request_id,

```

```

        owner_user_id,
        partner_user_id,
        meeting_place,
        meeting_time,
        status
    )
VALUES (?, ?, ?, ?, ?, 'created');
"""
(
    request_id,
    owner_user_id,
    partner_user_id,
    meeting_place,
    meeting_time
)
)

deal_id = cursor.lastrowid

cursor.execute(
    """
    UPDATE exchange_requests
    SET status = 'reserved'
    WHERE request_id = ?;
    """,
    (request_id,)
)

cursor.execute(
    """
    INSERT INTO action_logs (user_id, action_type, description)
    VALUES (?, 'create_deal', ?);
    """,
    (partner_user_id, f"Створено угоду №{deal_id} на основі заявки
№{request_id}")
)

connection.commit()
return deal_id

def send_message(deal_id: int, sender_user_id: int, message_text: str) -> int:
    """Надсилення повідомлення в межах конкретної угоди."""
    if not message_text.strip():
        raise ValueError("Текст повідомлення не може бути порожнім.")

    with get_connection() as connection:
        cursor = connection.cursor()

        cursor.execute(
            """
            SELECT deal_id, owner_user_id, partner_user_id
            FROM deals
            WHERE deal_id = ?;
            """,
            (deal_id,)
        )

        deal = cursor.fetchone()

        if deal is None:
            raise ValueError("Угоду не знайдено.")

```

```

        if sender_user_id not in (deal["owner_user_id"],
deal["partner_user_id"]):
            raise PermissionError("Користувач не є учасником цієї угоди.")

        cursor.execute(
            """
            INSERT INTO messages (deal_id, sender_user_id, message_text)
            VALUES (?, ?, ?);
            """
            ,
            (deal_id, sender_user_id, message_text.strip())
        )

        message_id = cursor.lastrowid

        cursor.execute(
            """
            INSERT INTO action_logs (user_id, action_type, description)
            VALUES (?, 'send_message', ?);
            """
            ,
            (sender_user_id, f"Надіслано повідомлення в угоді №{deal_id}")
        )

        connection.commit()
        return message_id

def update_deal_status(deal_id: int, status: str, user_id: int) -> None:
    """Оновлення статусу угоди."""
    allowed_statuses = {"created", "confirmed", "completed", "cancelled",
"disputed"}

    if status not in allowed_statuses:
        raise ValueError("Некоректний статус угоди.")

    completed_at = datetime.now().isoformat(timespec="seconds") if status ==
"completed" else None

    with get_connection() as connection:
        cursor = connection.cursor()

        cursor.execute(
            """
            UPDATE deals
            SET status = ?, completed_at = COALESCE(?, completed_at)
            WHERE deal_id = ?;
            """
            ,
            (status, completed_at, deal_id)
        )

        if status == "completed":
            cursor.execute(
                """
                UPDATE exchange_requests
                SET status = 'closed', closed_at = CURRENT_TIMESTAMP
                WHERE request_id = (
                    SELECT request_id
                    FROM deals
                    WHERE deal_id = ?
                );
                """
                ,
                (deal_id,)
            )

```

```
cursor.execute(
    """
    INSERT INTO action_logs (user_id, action_type, description)
    VALUES (?, 'update_deal_status', ?);
    """
    ,
    (user_id, f"Обновлено статус угоды №{deal_id}: {status}")
)

connection.commit()
```

ДОДАТОК В

Програмний код формування звітної інформації

```
# reports.py
# Формування звітної інформації для локального застосунку
# "Фінансовий помічник"

import csv
from datetime import datetime
from app_core import get_connection

def get_general_report() -> dict:
    """
    Формування загального звіту щодо роботи системи:
    кількість користувачів, заявок, угод, повідомлень
    та завершених операцій обміну.
    """
    with get_connection() as connection:
        cursor = connection.cursor()

        cursor.execute("SELECT COUNT(*) AS total FROM users;")
        users_count = cursor.fetchone()["total"]

        cursor.execute("SELECT COUNT(*) AS total FROM exchange_requests;")
        requests_count = cursor.fetchone()["total"]

        cursor.execute(
            """
            SELECT COUNT(*) AS total
            FROM exchange_requests
            WHERE status = 'active';
            """
        )
        active_requests_count = cursor.fetchone()["total"]

        cursor.execute("SELECT COUNT(*) AS total FROM deals;")
        deals_count = cursor.fetchone()["total"]

        cursor.execute(
            """
            SELECT COUNT(*) AS total
            FROM deals
            WHERE status = 'completed';
            """
        )
        completed_deals_count = cursor.fetchone()["total"]

        cursor.execute("SELECT COUNT(*) AS total FROM messages;")
        messages_count = cursor.fetchone()["total"]

    return {
        "users_count": users_count,
        "requests_count": requests_count,
        "active_requests_count": active_requests_count,
        "deals_count": deals_count,
        "completed_deals_count": completed_deals_count,
        "messages_count": messages_count,
    }
```

```

        "created_at": datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    }

def get_deals_report() -> list[dict]:
    """
    Формування деталізованого звіту за угодами.
    У звіт включаються дані про заявку, учасників угоди,
    валюти, суму, статус і дату створення.
    """
    with get_connection() as connection:
        cursor = connection.cursor()

        cursor.execute(
            """
            SELECT
                d.deal_id,
                er.request_id,
                owner.full_name AS owner_name,
                partner.full_name AS partner_name,
                c1.currency_code AS give_currency,
                c2.currency_code AS receive_currency,
                er.amount,
                er.proposed_rate,
                er.city,
                d.meeting_place,
                d.meeting_time,
                d.status,
                d.created_at,
                d.completed_at
            FROM deals d
            JOIN exchange_requests er ON er.request_id = d.request_id
            JOIN users owner ON owner.user_id = d.owner_user_id
            JOIN users partner ON partner.user_id = d.partner_user_id
            JOIN currencies c1 ON c1.currency_id = er.give_currency_id
            JOIN currencies c2 ON c2.currency_id = er.receive_currency_id
            ORDER BY d.created_at DESC;
            """
        )

        return [dict(row) for row in cursor.fetchall()]

def export_deals_report_to_csv(file_path: str = "deals_report.csv") -> str:
    """
    Експорт звіту за угодами у CSV-файл.
    Такий файл може бути відкритий у табличному редакторі
    або доданий до матеріалів тестування системи.
    """
    report_rows = get_deals_report()

    fieldnames = [
        "deal_id",
        "request_id",
        "owner_name",
        "partner_name",
        "give_currency",
        "receive_currency",
        "amount",
        "proposed_rate",
        "city",
        "meeting_place",
        "meeting_time",
    ]

```

```

        "status",
        "created_at",
        "completed_at"
    ]

    with open(file_path, "w", encoding="utf-8-sig", newline="") as file:
        writer = csv.DictWriter(file, fieldnames=fieldnames, delimiter=";")
        writer.writeheader()

        for row in report_rows:
            writer.writerow(row)

    return file_path

def get_user_activity_report(user_id: int) -> dict:
    """
    Формування звіту активності конкретного користувача:
    кількість створених заявок, участь в угодах,
    кількість повідомлень і дій у журналі.
    """
    with get_connection() as connection:
        cursor = connection.cursor()

        cursor.execute(
            """
            SELECT full_name, email
            FROM users
            WHERE user_id = ?;
            """,
            (user_id,)
        )
        user = cursor.fetchone()

        if user is None:
            raise ValueError("Користувача не знайдено.")

        cursor.execute(
            """
            SELECT COUNT(*) AS total
            FROM exchange_requests
            WHERE user_id = ?;
            """,
            (user_id,)
        )
        created_requests = cursor.fetchone()["total"]

        cursor.execute(
            """
            SELECT COUNT(*) AS total
            FROM deals
            WHERE owner_user_id = ? OR partner_user_id = ?;
            """,
            (user_id, user_id)
        )
        user_deals = cursor.fetchone()["total"]

        cursor.execute(
            """
            SELECT COUNT(*) AS total
            FROM messages
            WHERE sender_user_id = ?;
            """,

```

```

        (user_id,)
    )
    sent_messages = cursor.fetchone()["total"]

    cursor.execute(
        """
        SELECT COUNT(*) AS total
        FROM action_logs
        WHERE user_id = ?;
        """,
        (user_id,)
    )
    logged_actions = cursor.fetchone()["total"]

    return {
        "user_id": user_id,
        "full_name": user["full_name"],
        "email": user["email"],
        "created_requests": created_requests,
        "user_deals": user_deals,
        "sent_messages": sent_messages,
        "logged_actions": logged_actions,
        "created_at": datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    }

def print_general_report() -> None:
    """
    Виведення загального звіту у консоль.
    У desktop-застосунку ці дані можуть відображатися
    у вікні адміністратора або на вкладці статистики.
    """
    report = get_general_report()

    print("ЗАГАЛЬНИЙ ЗВІТ СИСТЕМИ")
    print(f"Дата формування: {report['created_at']}")
    print(f"Кількість користувачів: {report['users_count']}")
    print(f"Кількість заявок: {report['requests_count']}")
    print(f"Активні заявки: {report['active_requests_count']}")
    print(f"Кількість угод: {report['deals_count']}")
    print(f"Завершені угоди: {report['completed_deals_count']}")
    print(f"Кількість повідомлень: {report['messages_count']}")

if __name__ == "__main__":
    print_general_report()
    export_path = export_deals_report_to_csv()
    print(f"Звіт за угодами експортовано у файл: {export_path}")

```

Таблиця 4.2 – Результати тестування системи (позитивні сценарії)

I D	Сцена рій	Вхідні дані	Очікува ний результат	Факти чний результат	Ст атус
--------	--------------	----------------	--------------------------	----------------------------	------------

T C1	Реєстрація користувача	Email: iveran@exampl e.com, Пароль: Pass1234	Користувач створюється у БД	Користувач успішно зареєстрований	✓ Пройдено
T C2	Вхід у систему	Коректні облікові дані	Доступ до головного меню	Вхід успішний, меню доступно	✓ Пройдено
T C3	Перегляд статистики	Користувач у системі	Статистика з БД відображена	Активні заявки: 3, Угоди: 2	✓ Пройдено
T C4	Створення заявки	GEL, 500, 28.5 грн/1	Заявка статус ACTIVE	Заявка появилась в каталозі	✓ Пройдено
T C5	Пошук за фільтром	Валюта =USD, Статус=ACTIVE	Список активних USD	Таблиця відфільтрована коректно	✓ Пройдено
T C6	Ініціювання обміну	Заявка ID=15 вибрана	Угода status=PENDING	Запис в "Моїх угодах"	✓ Пройдено
T C7	Узгодження умов	Місто, час заповнені	Статус AGREED	Угода оновлена для обох	✓ Пройдено
T C8	Завершення угоди	Обидва натиснули 'Завершити'	Статус COMPLETED	Угода закривається	✓ Пройдено
T C9	Обмін повідомленнями	Текст: 'Зустрітись о 15:00?'	Повідомлення в БД	Появляється в чаті	✓ Пройдено
T C10	Оновлення курсів	Натиснуто 'Оновити'	API повертає курси	Таблиця валют оновлена	✓ Пройдено

ДОДАТОК Г – ФРАГМЕНТ КОДУ ІНТЕРФЕЙСУ КОРИСТУВАЧА (PyQt6)

Основний клас вікна застосунку:

```
class MainApplicationWindow(QMainWindow):
    """Основне вікно застосунку для P2P-обміну готівки"""
    def __init__(self, user_id: int, user_email: str):
        super().__init__()
        self.user_id = user_id
        self.user_email = user_email
        self.setWindowTitle("Фінансовий помічник")
        self.setGeometry(100, 100, 1200, 700)
        self.init_ui()
```

Структура інтерфейсу:

- QMainWindow: основне вікно застосунку
- QTabWidget: вкладки для навігації
- QTableWidget: таблиці для заявок, угод
- QPushButton: кнопки для дій (Створити, Відправити)
- QLineEdit: поля введення для фільтрів і повідомлень

Основні методи:

```
init_ui(): ініціалізація всіх компонентів інтерфейсу
load_user_statistics(): завантаження даних з SQLite
create_orders_tab(): створення вкладки для заявок
send_message(): відправка повідомлення в угоді
generate_report(): формування звіту за угодами
```

Інтеграція з БД:

Клас використовує `sqlite3` для підключення до файлу `database.db`

Виконує SQL-запити для завантаження статистики користувача

Оновлює таблиці при створенні угод або публікації заявок

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І
ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ**

**Факультет інформаційних технологій Декларація про академічну
добročесність та використання ШІ**

Я, Іщенко Євгеній Олександрович, здобувач НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота на тему «Інформаційна система для готівкових P2P обмінів» є результатом моєї особистої праці.
2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.
3. Щодо використання інструментів ШІ зазначаю, що:
інструменти ШІ Gemini були використані для:
 - + перекладу іншомовних джерел;
 - + стилістичного редагування та перевірки граматики;
 - + допомоги у написанні/відлагодженні програмного коду;
 - + інше: пошуку додаткових інструментів у розробці програмного продукту.

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

«22» Травня 2026 р. _____

Євгеній ІЩЕНКО