НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ПОГОДЖЕНО **Декан факультету**	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ** **Завідувач кафедри комп'ютерних наук**
_____ Ігор БОЛБОТ (підпис)	**_____ Белла ГОЛУБ** (підпис)
«___» _____ 2026 р.	«___» _____ 2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Програмне забезпечення для обміну торгівельної діяльності трейдера»

Спеціальність «121 Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення»

Гарант освітньої програми _____ **Ганна ВАЙГАНГ**
к.т.н., доцент (підпис)

Керівник бакалаврської кваліфікаційної роботи _____ **Віктор КИРИЧЕНКО**
к.ф.-м.н., доцент (підпис)

Виконав _____ **Максим АВДІЄВСЬКИЙ**
(підпис)

Київ-2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук

к.т.н., доцент _____ Белла ГОЛУБ
(підпис)

«___» _____ 2025 р.

ЗАВДАННЯ
ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
ЗДОБУВАЧУ

Авдієвському Максиму Олександровичу
(прізвище, ім'я, по батькові)

Спеціальність «121 Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»
Тема бакалаврської кваліфікаційної роботи
«Програмне забезпечення для обміну торгівельної діяльності трейдера»

затверджена наказом від 19.12.2025 р. № 3204 «С»
Термін подання завершеної роботи на кафедру 22.05.2026 р.
Вихідні дані до бакалаврської кваліфікаційної роботи (проєкту):
предметна область обліку торгової діяльності трейдера; вимоги до мобільного застосунку для ведення журналу угод; дані про торгові операції, прибуток і збиток, параметри позицій, статистику торгівлі та API криптовалютної біржі.
Перелік питань, що підлягають дослідженню:

1. Аналіз предметної області обліку торгівельної діяльності трейдера.
2. Проєктування системи.
3. Розробка мобільного застосунку.
4. Тестування.
5. Впровадження програмного забезпечення.

Перелік графічних документів (за необхідності).

_____

Дата видачі завдання 19.12.2026 р.

Керівник бакалаврської кваліфікаційної роботи _____ (підпис) **Віктор КИРИЧЕНКО**

Завдання взяв до виконання _____ (підпис) **Максим АВДІЄВСЬКИЙ**

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface, програмний інтерфейс застосунку

API Key – відкритий ключ доступу до API біржі

API Secret – секретний ключ доступу до API біржі

APK – Android Package Kit, інсталяційний пакет Android-застосунку

БД – база даних

C# – мова програмування C Sharp

CRUD – Create, Read, Update, Delete; базові операції створення, читання, оновлення та видалення даних

ER – Entity-Relationship, модель «сутність-зв’язок»

Futures – ф’ючерсний ринок

HTTP – HyperText Transfer Protocol, протокол передавання гіпертексту

HTTPS – HyperText Transfer Protocol Secure, захищений протокол передавання гіпертексту

Long – напрямок торгової позиції на зростання ціни активу

.NET MAUI – .NET Multi-platform App UI, платформа для створення кросплатформених застосунків

NuGet – менеджер пакетів для платформи .NET

ORM – Object-Relational Mapping, об’єктно-реляційне відображення

OS – Operating System, операційна система

PnL – Profit and Loss, прибуток або збиток

REST – Representational State Transfer, архітектурний стиль взаємодії через web API

ROI – Return on Investment, показник рентабельності інвестицій

SDK – Software Development Kit, набір засобів розробки програмного забезпечення

SecureStorage – механізм захищеного локального збереження конфіденційних даних

Short – напрямок торгової позиції на зниження ціни активу

SQL – Structured Query Language, мова структурованих запитів

SQLite – вбудована реляційна система керування базами даних

UI – User Interface, інтерфейс користувача

UML – Unified Modeling Language, уніфікована мова моделювання

USD – United States Dollar, долар США

USDT – Tether, стейблкоїн, прив’язаний до долара США

XAML – Extensible Application Markup Language, мова розмітки для створення інтерфейсу

xUnit – фреймворк для модульного тестування програмного забезпечення платформи .NET

ВСТУП

Торгівля цифровими активами сьогодні є одним із найпоширеніших напрямів роботи на фінансових ринках. Багато користувачів працюють із криптовалютними біржами, відкривають торгові позиції, аналізують результати угод і намагаються покращувати власний підхід до торгівлі. Для цього важливо не лише виконувати самі торгові операції, а й зберігати інформацію про них, переглядати результати та бачити загальну картину своєї діяльності.

Під час торгівлі трейдер постійно отримує нові дані: які угоди були відкриті, який результат вони принесли, які комісії були сплачені, скільки часу тривала позиція, який напрямок угод був більш результативним. Якщо така інформація не зберігається в зручному вигляді, її складно аналізувати. У такому випадку трейдер бачить лише окремі результати, але не завжди може оцінити свою роботу за тиждень, місяць або інший період.

Для вирішення цієї задачі використовують торговий журнал. Його призначення полягає в тому, щоб зберігати історію угод і допомагати користувачу аналізувати свої результати. Завдяки журналу можна переглядати виконані операції, бачити загальний прибуток або збиток, оцінювати кількість успішних і невдалих угод, а також робити висновки щодо покращення своєї торгової стратегії.

На практиці частина трейдерів веде такий облік вручну. Для цього часто використовують електронні таблиці, нотатки або універсальні сервіси для зберігання інформації. Такий спосіб простий на початку, але з часом стає незручним. Якщо угод багато, їх потрібно постійно вносити вручну, перевіряти правильність даних, рахувати підсумки та оновлювати таблиці. Це займає час і може призводити до помилок.

Існують також готові сервіси для ведення торгових журналів. Вони можуть мати багато корисних функцій, але не завжди зручні для щоденного використання. Деякі рішення мають складний інтерфейс, частина з них

орієнтована переважно на web-версії або desktop-застосунки, а окремі сервіси мають обмеження у безкоштовному використанні. Через це виникає потреба у простому мобільному застосунку, який дає змогу швидко переглядати угоди, додавати нові записи та бачити основну статистику без зайвого функціоналу.

Мобільний формат є важливим, оскільки смартфон часто є основним пристроєм для швидкого доступу до інформації. Трейдер може переглянути журнал угод, перевірити результати або додати запис без необхідності відкривати комп’ютер. Це робить роботу із системою зручнішою та ближчою до реальних умов щоденного використання.

Ще однією важливою можливістю є автоматичне отримання даних із біржі. Якщо система сама може отримувати дані напряму з біржі, користувачу не потрібно щоразу вручну переносити інформацію про угоди. Це зменшує кількість ручної роботи, знижує ризик помилок і робить журнал більш точним. При цьому система не повинна виконувати торгові операції замість користувача. Її завдання полягає лише в обліку, збереженні та аналізі даних.

Для такого програмного забезпечення важливо також правильно організувати збереження інформації. Оскільки застосунок розробляється як мобільний, доцільно використовувати локальну базу даних. Це дає змогу зберігати угоди безпосередньо на пристрої користувача та переглядати журнал навіть без постійного підключення до мережі Internet. Такий підхід спрощує архітектуру системи та не потребує окремого серверного середовища.

Отже, розробка мобільного програмного забезпечення для обліку торговельної діяльності трейдера є актуальною задачею. Така система має допомогти користувачу вести журнал торгових операцій, зберігати дані локально, отримувати частину інформації з біржі та переглядати основні статистичні показники у зручному вигляді.

Метою бакалаврської кваліфікаційної роботи є розробка мобільного програмного забезпечення для обліку та аналізу торговельної діяльності трейдера.

Об’єктом дослідження є процес обліку та аналізу торговельної діяльності трейдера.

Предметом дослідження є методи та засоби розробки мобільного застосунку для ведення журналу торгових операцій.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- провести аналіз предметної області та існуючих програмних рішень;
- визначити вимоги до програмної системи;
- спроєктувати структуру бази даних;
- спроєктувати архітектуру мобільного застосунку;
- реалізувати збереження торгових записів у локальній базі даних;
- реалізувати можливість ручного додавання торгових записів;
- реалізувати отримання даних із криптовалютної біржі через API;
- реалізувати відображення журналу торгових операцій;
- провести тестування розробленої системи.

Під час виконання роботи було проведено аналіз предметної області, спроєктовано структуру програмної системи та бази даних, а також реалізовано і протестовано мобільний застосунок. Для реалізації програмного продукту використано платформу .NET MAUI, мову програмування C#, локальну базу даних SQLite, засоби безпечного збереження даних SecureStorage та бібліотеки для роботи з API криптовалютних бірж.

Результатом виконання роботи є мобільний застосунок EZstat – Trader’s journal. Він призначений для ведення журналу торгових операцій, збереження інформації про угоди та перегляду основних показників торговельної діяльності. Застосунок не є платформою для торгівлі чи будь-якої маніпуляції активами та не надає торгових рекомендацій.

Пояснювальна записка складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків. У першому розділі розглянуто предметну область, вимоги до програмної системи, моделювання предметної області, існуючі рішення та постановку завдання. У другому розділі виконано

проєктування інформаційного та програмного забезпечення. У третьому розділі описано розробку інформаційної бази, вибір інструментів розробки та реалізацію основних програмних модулів. У четвертому розділі наведено тестування системи, вимоги до апаратного та програмного забезпечення, а також склад інсталяційного пакету.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Трейдингом у межах цієї роботи вважається діяльність користувача, пов’язана з відкриттям і закриттям торгових позицій на криптовалютній біржі з метою отримання фінансового результату. Користувач, який виконує такі операції, далі розглядається як трейдер. У процесі роботи трейдер аналізує зміну ціни активів, обирає напрямок позиції, визначає обсяг угоди та після закриття позиції оцінює її результат.

Криптовалютна біржа є електронною платформою, на якій користувачі можуть виконувати торгові операції з цифровими активами. У межах цієї роботи біржа розглядається не як засіб автоматичної торгівлі, а лише як джерело даних про виконані користувачем операції. Такий підхід дозволяє відокремити облік торговельної діяльності від процесу прийняття торгових рішень.

Одним із поширених напрямів криптовалютної торгівлі є торгівля futures-контрактами. Futures-контракт передбачає угоду щодо купівлі або продажу активу за визначеними умовами. Окремим видом таких інструментів є perpetual futures, тобто безстрокові ф’ючерсні контракти. Їхня основна особливість полягає в тому, що вони не мають дати завершення, тому позиція може утримуватися доти, доки виконуються вимоги щодо маржі [1].

У futures-торгівлі трейдер може відкривати позиції у двох основних напрямах: long і short. Long-позиція відкривається тоді, коли користувач очікує зростання ціни активу. Short-позиція використовується тоді, коли трейдер очікує зниження ціни. Саме можливість працювати в обох напрямах є однією з причин популярності futures-інструментів серед трейдерів [1].

Для відкриття позицій у futures-торгівлі часто використовується кредитне плече. Воно дає змогу відкривати позицію на суму, що перевищує власний баланс користувача. За правилами біржі Bybit, торгівля з плечем дозволяє відкривати

long- і short-позиції на обсяг, більший за баланс гаманця [2]. Через це прибуток або збиток може змінюватися швидше, ніж під час торгівлі без плеча. Водночас кредитне плече не змінює саму логіку обліку угоди: у журналі все одно потрібно зберігати обсяг позиції, напрямок, ціну входу, ціну виходу та фінансовий результат.

Для відкриття позиції користувач повинен мати певну суму забезпечення. У futures-торгівлі таке забезпечення називають маржею. Початкова маржа використовується для відкриття позиції, а підтримувальна маржа визначає мінімальний рівень коштів, необхідний для її утримання [2]. Для програмного забезпечення обліку торговельної діяльності ці поняття важливі тому, що вони пояснюють, чому трейдеру потрібно враховувати не лише результат угоди, а й розмір позиції та використане кредитне плече.

Одним із головних показників результату торгової операції є PnL. Цей показник відображає прибуток або збиток за позицією. У контрактах типу USDT Perpetual прибуток і збиток розраховуються та фіксуються в USDT [3]. Для long-позиції прибуток або збиток залежить від різниці між ціною виходу та ціною входу. У спрощеному вигляді це можна подати формулою:

$$\boldsymbol{PnL} = \left(\boldsymbol{Price}_{\{exit\}} - \boldsymbol{Price}_{\{entry\}}\right) \times \boldsymbol{PositionSize} \quad (1)$$

де

PnL – прибуток або збиток за торговою позицією;

Price_exit – ціна закриття позиції;

Price_entry – ціна відкриття позиції;

PositionSize – обсяг позиції.

Для short-позиції логіка розрахунку є протилежною, оскільки прибуток формується тоді, коли ціна активу знижується. У такому випадку результат залежить від різниці між ціною входу та ціною виходу [4].

Окрім PnL, у трейдингу часто використовується показник ROI. Він показує відсоткове співвідношення результату позиції до початкової маржі. У довідкових

матеріалах Bybit ROI визначається через відношення нереалізованого PnL до початкової маржі позиції [5]:

$$\boldsymbol{ROI} = \left(\frac{\boldsymbol{PnL}}{\boldsymbol{InitialMargin}}\right) \times \mathbf{100}\% \quad (2)$$

де

ROI – відсоткова дохідність позиції;

PnL – прибуток або збиток за угодою;

InitialMargin – початкова маржа позиції.

У межах програмного забезпечення цей показник корисний для швидкого порівняння угод між собою, оскільки дозволяє оцінювати результат не лише в грошовому значенні, а й у відсотках.

Під час виконання торгових операцій трейдер також сплачує комісії. Комісія може стягуватися під час відкриття та закриття позиції. Через це для правильного підрахунку важливо зберігати не лише прибуток або збиток за угодою, а й витрати на комісії. Якщо не враховувати комісії, підсумковий результат торгівлі може бути неточним.

Для обмеження можливого збитку під час відкриття торгової позиції трейдери часто використовують стоп-лос. Стоп-лосом називають заздалегідь визначений ціновий рівень, при досягненні якого позиція повинна бути закрита для мінімізації втрат. Використання стоп-лосу є одним із базових елементів контролю ризику в трейдингу, оскільки дозволяє трейдеру заздалегідь визначити допустимий рівень збитку за угодою. У межах розроблюваної системи значення стоп-лосу використовується як один із параметрів для допоміжних розрахунків у торговому калькуляторі [6].

Для ведення всіх цих даних використовується торговий журнал. Торговий журнал дозволяє зберігати історію угод, переглядати результати за різні періоди та аналізувати власну торгову поведінку. За матеріалами Investopedia, окремий журнал торгівлі дає змогу формувати історію виконаних угод, аналізувати успішність торгівлі, оцінювати торгові підходи та краще розуміти власні результати [7].

У контексті розроблюваної системи торговий журнал є основною частиною програмного забезпечення. Саме в ньому зберігаються дані про торгові операції: символ активу, біржа, напрямок позиції, прибуток або збиток, ROI, обсяг позиції, ціни входу та виходу, розмір комісій, час відкриття і закриття позиції, а також коментар користувача. Наявність таких даних дозволяє не лише переглядати історію угод, а й формувати статистику торговельної діяльності.

Статистика є важливою частиною аналізу результатів трейдера. До основних показників можна віднести загальний PnL, кількість угод, відсоток прибуткових угод, співвідношення long- і short-позицій, суму комісій, найбільший прибуток та найбільший збиток за окремою угодою. Такі показники допомагають користувачу бачити загальну картину своєї діяльності, а не лише окремі результати.

Окремим показником, який часто використовується для оцінки торговельної діяльності, є win rate. Він показує, який відсоток угод завершився з прибутком. У спрощеному вигляді цей показник можна розрахувати так:

$$\boldsymbol{WinRate} = \left(\frac{\boldsymbol{WinningTrades}}{\boldsymbol{TotalTrades}}\right) \times \mathbf{100}\% \quad (3)$$

де

WinRate – відсоток прибуткових угод;

WinningTrades – кількість прибуткових угод;

TotalTrades – загальна кількість угод.

Предметна область обмежується обліком, збереженням та аналізом торговельної діяльності трейдера. Програмне забезпечення не виконує автоматичну торгівлю, не відкриває та не закриває позиції замість користувача і не надає інвестиційних рекомендацій. API біржі використовується лише як джерело даних для імпорту історії угод.

Отже, предметна область роботи охоплює процес ведення журналу торгових операцій, збереження інформації про угоди, отримання даних із біржі та формування статистичних показників. Саме ці процеси є основою для подальшого визначення вимог до програмної системи та її проєктування.

1.2 Аналіз вимог до програмної системи

Аналіз вимог є одним із основних етапів розробки програмного забезпечення, оскільки на цьому етапі визначаються функціональні можливості системи, обмеження її роботи, вимоги до інтерфейсу користувача, збереження даних та безпеки.

Основним користувачем системи є трейдер, який веде журнал власних торгових операцій, переглядає результати угод, додає записи вручну, синхронізує дані з біржами та аналізує статистику за вибраний період.

1.2.1 Функціональні вимоги. Функціональні вимоги визначають основні можливості, які має забезпечувати програмна система. Для мобільного застосунку EZstat – Trader’s journal доцільно визначити такі функціональні вимоги:

- забезпечувати перегляд журналу торгових операцій;
- групувати торгові записи за місяцями;
- відображати повну інформацію про угоду
- забезпечувати ручне додавання торгової операції;
- зберігати торгові записи в базі даних;
- надавати можливість видаляти додані вручну угоди;
- отримувати дані про закриті futures-угоди з бірж через API;
- запобігати дублюванню записів під час повторної синхронізації;
- дозволяти зберігати коментар користувача до торгової операції;
- відображати статистичні показники торговельної діяльності;
- відображати баланс futures-гаманця;
- забезпечувати збереження API-ключів користувача;
- працювати з локально збереженими даними без постійного підключення до мережі Internet.

Кожний торговий запис повинен містити такі дані: біржу, символ активу, напрямок позиції, PnL, ROI, обсяг позиції в доларах США, ціну входу, ціну

виходу, кредитне плече, комісії, час відкриття, час закриття та коментар користувача.

1.2.2 Нефункціональні вимоги. Нефункціональні вимоги описують не окремі функції системи, а характеристики її роботи. Для розроблюваного мобільного застосунку важливими є стабільність, швидкість роботи, безпека збереження ключів та коректність обробки даних.

Система повинна відповідати таким нефункціональним вимогам:

- Основні дані повинні відкриватися без зайвої навігації.
- Застосунок повинен використовувати темну тему інтерфейсу.
- Дані повинні зберігатися локально на пристрої користувача.
- Синхронізація з біржами повинна виконуватися тільки для отримання даних.
- API-ключі не повинні зберігатися у відкритому вигляді.
- Система повинна коректно обробляти ситуації, коли API-ключі не вказані.

Використання локальної бази даних SQLite дозволяє зменшити залежність системи від зовнішньої інфраструктури та забезпечити доступ до журналу угод без постійного підключення до мережі.

1.2.3 Вимоги до інтерфейсу користувача. Інтерфейс користувача повинен забезпечувати швидкий доступ до основних функцій системи та бути зручним для використання на мобільному пристрої. Оскільки система призначена для щоденного перегляду торгових записів і статистики, інтерфейс не повинен бути перевантажений другорядними елементами.

У системі передбачено п’ять основних сторінок:

- сторінка статистики;
- сторінка журналу угод;
- сторінка ручного додавання угоди;
- сторінка калькулятора;
- сторінка налаштувань.

Сторінка статистики призначена для перегляду основних показників торговельної діяльності: загального PnL, win rate, кількості угод, співвідношення long- і short-позицій, комісій та інших підсумкових значень.

Сторінка журналу угод призначена для перегляду історії торгових операцій. На ній угоди повинні відображатися у вигляді карток, згрупованих за місяцями. Користувач повинен мати можливість відкрити детальну інформацію про конкретну угоду.

Сторінка ручного додавання угоди призначена для створення торгового запису користувачем без синхронізації з біржею. На цій сторінці користувач повинен мати можливість ввести біржу, символ активу, напрямок позиції, PnL, ROI, обсяг позиції, ціни входу та виходу, кредитне плече, комісії, час відкриття, час закриття та коментар.

Сторінка калькулятора призначена для розрахунку розміру об’єму та маржі позиції, яку збирається відкрити трейдер, опираючись на такі дані як:

- ціна відкриття угоди;
- ціна стоп-лосу;
- розмір депозиту;
- відсоток допустимого ризику
- кредитне плече

Результати розрахунку мають довідковий характер і використовуються користувачем для самостійного аналізу параметрів майбутньої позиції.

Сторінка налаштувань призначена для введення та збереження API-ключів бірж, а також для керування підключенням джерел даних. Також на сторінці є можливість очищення бази даних.

1.2.4 Вимоги до збереження та синхронізації даних. Система повинна забезпечувати локальне збереження торгових записів на пристрої користувача. Для цього використовується база даних SQLite. Такий підхід відповідає архітектурі мобільного застосунку без окремої серверної бази даних.

Під час синхронізації система повинна отримувати дані з API біржі та зберігати їх у локальній базі даних. Перед додаванням нового запису система повинна перевіряти, чи не існує вже угода з таким самим ідентифікатором. Це потрібно для уникнення дублювання даних.

Синхронізація повинна виконуватися під час відкриття журналу угод. При цьому ручні записи користувача повинні зберігатися незалежно від імпортованих угод і не повинні видалятися під час синхронізації.

1.2.5 Вимоги до безпеки даних. Оскільки система працює з API-ключами криптовалютних бірж, необхідно забезпечити їх безпечне збереження. API-ключі не повинні зберігатися у звичайних таблицях бази даних або у відкритому текстовому вигляді.

Для збереження ключів використовується механізм SecureStorage, який призначений для безпечного локального збереження конфіденційних даних у мобільному застосунку.

Система повинна використовувати API-ключі лише для отримання даних про угоди. У програмному забезпеченні не передбачені будь-які маніпуляції активами трейдера.

Отже, визначені вимоги формують основу для подальшого проєктування структури бази даних, архітектури мобільного застосунку та програмних модулів системи EZstat – Trader’s journal.

1.3 Моделювання предметної області

Моделювання предметної області використовується для формалізації основних процесів системи, визначення взаємодії користувача із застосунком та опису ключових об’єктів, які беруть участь у роботі програмного забезпечення.

У межах цього підрозділу для опису предметної області використовуються чотири моделі: діаграма варіантів використання, діаграма послідовності, діаграма активності та діаграма абстракції предметної області. Такі діаграми дають змогу розглянути систему з різних сторін: з погляду користувача, логіки

виконання процесів, взаємодії між компонентами та структури основних сутностей.

1.3.1 Діаграма варіантів використання. Діаграма варіантів використання (рис. 1.1) призначена для відображення основних сценаріїв взаємодії користувача із програмною системою. Вона дозволяє визначити функції, які доступні користувачу, а також зовнішні джерела даних, з якими взаємодіє застосунок.

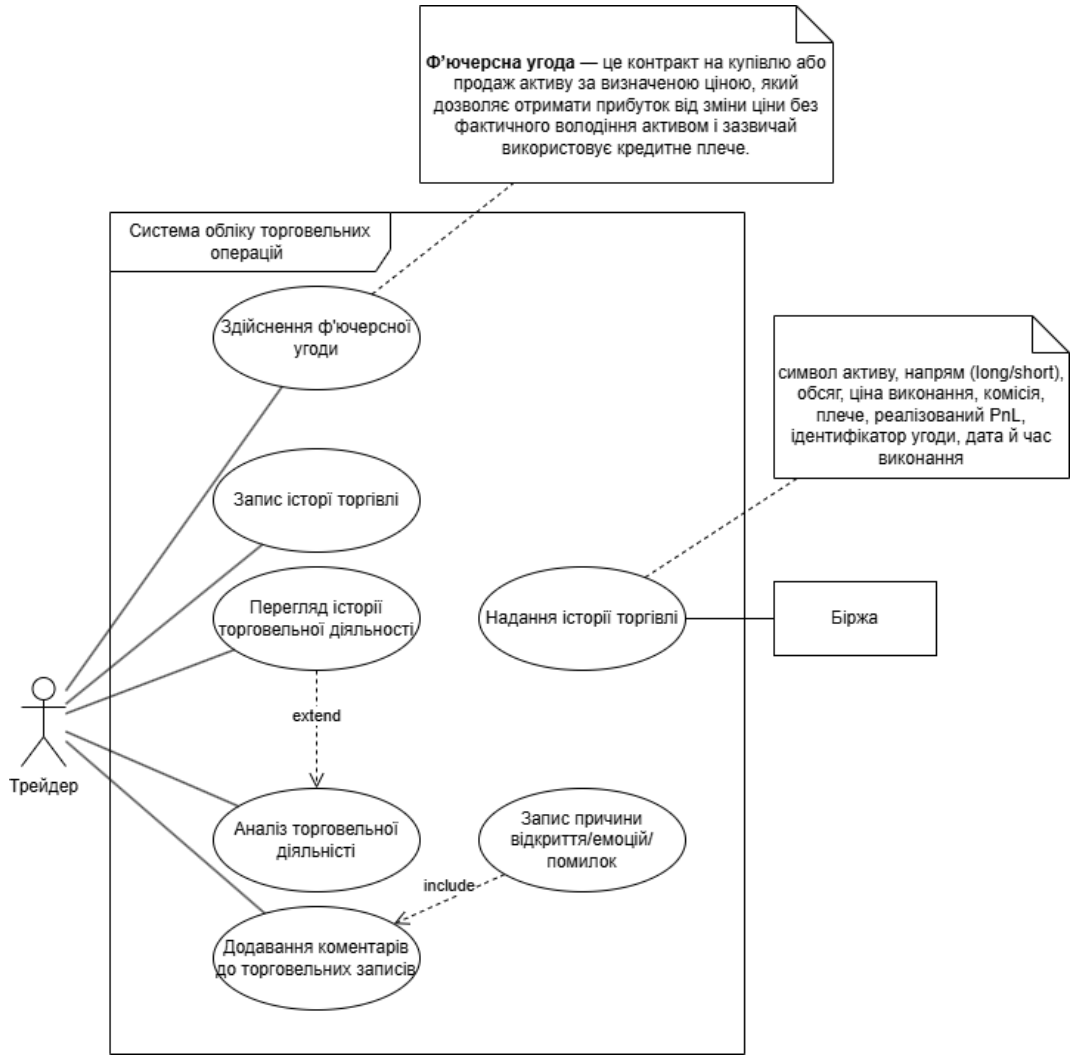

Рис. 1.1 – Діаграма прецедентів

У розроблюваній системі основним користувачем є трейдер. Він може переглядати журнал торгових операцій, додавати угоди вручну, переглядати статистичні показники, використовувати калькулятор позиції, додавати коментарі до угод і налаштовувати підключення до бірж. Криптовалютна біржа на діаграмі виступає зовнішнім джерелом даних, з якого система отримує інформацію про виконані торгові операції.

1.3.2 Діаграма послідовності. Діаграма послідовності (рис. 1.2) використовується для опису порядку взаємодії між об’єктами системи під час виконання певного сценарію. Така модель дозволяє показати, які елементи системи беруть участь у процесі та в якій послідовності між ними передаються дані.

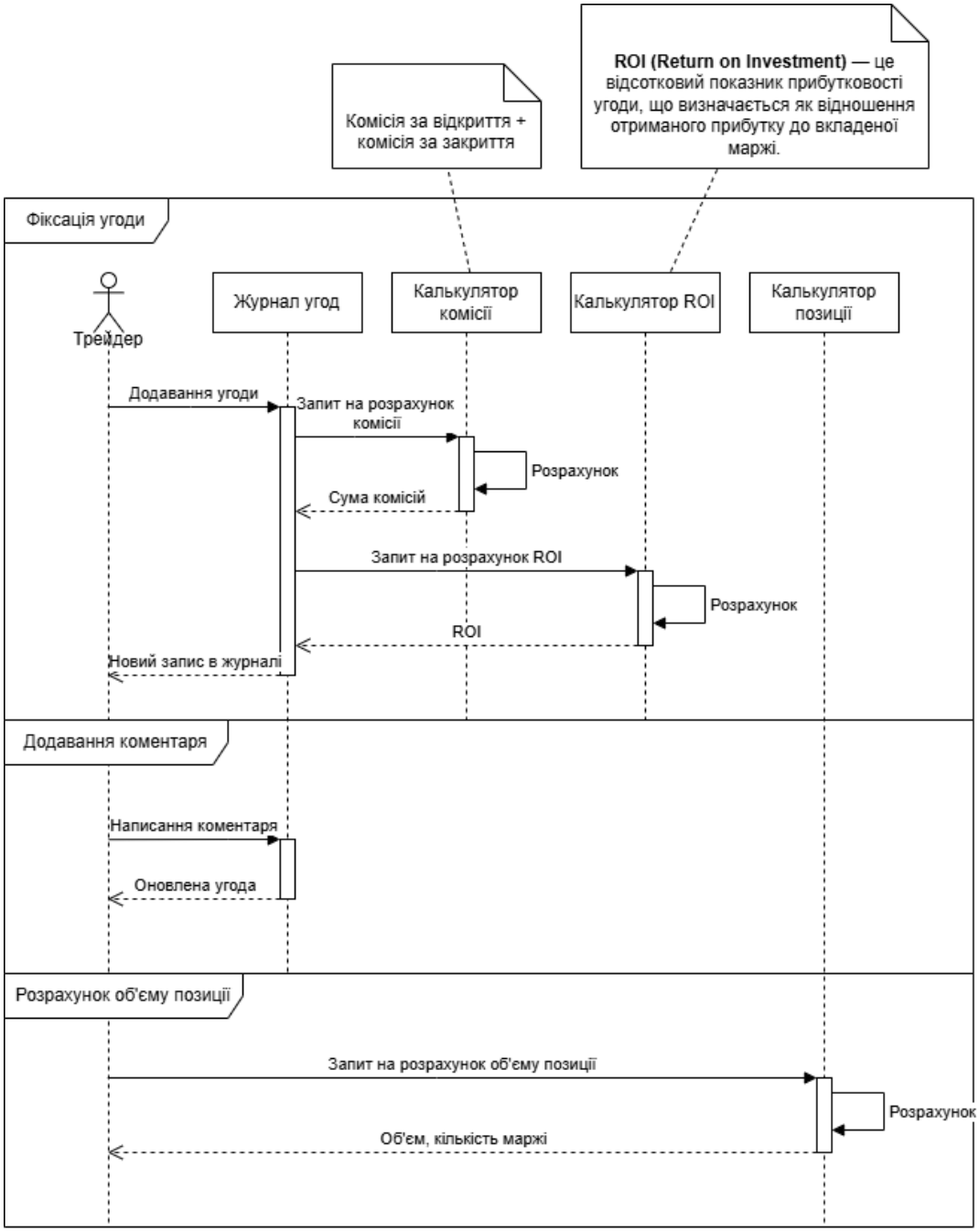

Рис. 1.2 – Діаграма послідовності

У межах цієї роботи діаграма послідовності відображає процес обробки торгової операції та виконання допоміжних розрахунків. Користувач передає дані

про угоду, після чого система виконує розрахунок комісій, ROI та параметрів позиції. Після обробки даних формується торговий запис, який надалі використовується у журналі угод і статистиці.

1.3.3 Діаграма активності. Діаграма активності (рис. 1.3) використовується для опису логіки виконання процесу та переходів між окремими діями. Вона дозволяє показати послідовність операцій, умови переходів і результат виконання процесу.

Рис. 1.3 – Діаграма активності

У розроблюваній системі діаграма активності описує процес обробки торгової операції. Після отримання або введення даних про угоду система

виконує необхідні розрахунки, формує торговий запис і зберігає його в локальній базі даних. Після цього дані можуть бути використані для оновлення журналу та статистики.

1.3.4 Абстракції предметної області. Зображення абстракцій предметної області (рис. 1.4) використовується для узагальненого представлення основних сутностей, які описують логіку роботи системи. На відміну від діаграми класів програмної реалізації, така модель відображає не конкретні програмні класи, а ключові поняття предметної області та зв’язки між ними.

Рис. 1.4 – Абстракції предметної області

У межах розроблюваної системи основними абстракціями є угода, період, статистика, ROI та комісія. Угода є центральною сутністю, оскільки містить основні дані про торгову операцію: торгову пару, напрямок позиції, обсяг, ціну відкриття, ціну закриття, фінансовий результат і комісію. Період використовується для групування угод за часовими межами. Статистика формується на основі збережених угод і містить агреговані показники торговельної діяльності. ROI та комісія є розрахунковими елементами, які використовуються для оцінки результату окремої торгової операції.

1.4 Огляд інформаційних джерел та існуючих рішень

Для визначення вимог до програмного забезпечення було проаналізовано існуючі сервіси для ведення журналу торгових операцій. Такий аналіз дозволяє визначити функціональні можливості сучасних торгових журналів, їхню модель монетизації, наявність мобільних застосунків, підтримку автоматичного імпорту угод та додаткові інструменти для аналізу торговельної діяльності.

Для порівняння було обрано три сервіси: TradeZella, TraderSync та PNL - Simple Trading Journal. Ці рішення мають подібне призначення, оскільки використовуються для збереження торгових записів, аналізу результатів угод та перегляду статистики.

1.4.1 Аналіз сервісу TradeZella. TradeZella є сервісом для ведення торгового журналу, аналізу угод та оцінки результатів торговельної діяльності. Платформа дозволяє імпортувати історію угод за допомогою auto-sync, завантаження файлів або ручного введення даних. Також у сервісі передбачено облік комісій, що дає змогу точніше аналізувати підсумковий результат торгівлі [8].

Окремо TradeZella підтримує автоматичний імпорт угод через підключення брокера. У такому випадку торгові операції імпортуються до журналу без ручного введення даних користувачем [9]. Для користувачів, які не використовують автоматичну синхронізацію, передбачено можливість ручного додавання угод [10].

Сервіс працює за моделлю платної підписки. На сторінці тарифів зазначено кілька планів, зокрема Basic за 29 USD на місяць, Essential за 24 USD на місяць, Premium за 49 USD на місяць та Pro за 33 USD на місяць при річній оплаті [8].

TradeZella – це потужний сервіс для обліку торгівельної діяльності, створений для трейдерів-професіоналів, адже не кожен початківець, який лише відкриває для себе торгівлю, готовий платити такі гроші.

1.4.2 Аналіз сервісу TraderSync. TraderSync є сервісом для ведення торгового журналу та аналізу результатів торговельної діяльності. У Google Play

застосунок описано як торговий журнал для трейдерів, які працюють із різними ринками [11]. Сервіс дозволяє переглядати звіти, відстежувати виконання угод, аналізувати PnL, а також додавати до записів нотатки та скріншоти.

TraderSync має мобільний застосунок для Android та iOS. Крім мобільного формату, сервіс підтримує роботу через web-інтерфейс.

Сервіс працює за моделлю платної підписки. На сторінці тарифів зазначено, що тариф Pro коштує 29.95 USD на місяць або 22.46 USD на місяць при річній оплаті. Також передбачено 7-денний пробний період. Дорожчі тарифи Premium та Elite коштують відповідно 49.95 USD і 79.95 USD на місяць при місячній оплаті [12].

Перевагами TraderSync є підтримка кількох типів ринків, наявність мобільних застосунків, web-версії, статистичних звітів, нотаток і скріншотів до угод. Недоліком є платна модель використання, яка може бути дорогою для починаючого трейдера.

1.4.3 Аналіз сервісу PNL - Simple Trading Journal. PNL - Simple Trading Journal є мобільним застосунком для ведення торгового журналу. Застосунок призначений для відстеження, перегляду та покращення торгових результатів. Сервіс підтримує роботу з криптовалютним та деякими іншими видами ринків [13].

PNL орієнтований переважно на ручне ведення журналу. Користувач може вручну додавати угоди з різних торгових платформ. Застосунок не має підключення до біржі чи будь-якого іншого автоматичного джерела даних.

Серед особливих функцій PNL можна виділити додавання кількох зображень до кожної угоди, встановлення щоденних і місячних цілей прибутку, визначення денних торгових лімітів, експорт PDF та Excel-звітів, кругові діаграми і календарне відстеження угод[13].

Застосунок працює за моделлю підписки. У сервісі доступні кілька варіантів оплати: місячна підписка вартістю 1.29 EUR, підписка на 6 місяців за 6.49 EUR, річна підписка за 10.99 EUR, а також lifetime-доступ за одноразову оплату 28.99 EUR. Також для нових користувачів передбачено пробний період.

Перевагою PNL є мобільний формат а також досить низька ціна підписки у порівнянні з вищезгаданими сервісами. Недоліком є відсутність автоматичного імпорту угод через API, що збільшує обсяг ручної роботи користувача.

1.4.4 Порівняння існуючих рішень. Щоб краще побачити різницю між розглянутими сервісами, їхні основні можливості показано у табл. 1.1. Тут показано, які функції має кожен сервіс, скільки він коштує та які обмеження можуть бути важливими для користувача.

Таблиця 1.1

Порівняльна таблиця

Критерій	TradeZella	TraderSync	PNL - Simple Trading Journal
Тип монетизації	Платна підписка	Платна підписка	Платна підписка
Мінімальна ціна	24 USD/міс	29.95 USD/міс або 22.46 USD/міс при річній оплаті	7-денний пробний період, далі 1.50 USD/міс
Web-версія	Так	Так	Ні
Мобільний застосунок	Ні	Так	Так
Робота без Internet	Ні	Ні	Так
Автоматичний імпорт угод	Так	Так	Ні
Ручне додавання угод	Так	Так	Так
Облік комісій	Так	Так	Так

Проведене порівняння показує, що TradeZella і TraderSync мають розвинені можливості для автоматичного імпорту угод, ведення журналу та аналізу торговельної діяльності. Водночас ці сервіси працюють за моделлю платної

підписки, а їхній функціонал орієнтований на ширше коло трейдерів і різні фінансові ринки.

PNL - Simple Trading Journal більше орієнтований на мобільне ручне ведення журналу. Його перевагою є простота функціонального призначення, але відсутність автоматичного підключення до бірж означає, що користувач повинен самостійно вносити всі торгові записи.

Отже, аналіз існуючих рішень показує доцільність розробки мобільного застосунку, який поєднує базові можливості торгового журналу, локальне збереження даних, ручне додавання угод, автоматичний імпорт торгових операцій через API та перегляд основних статистичних показників.

1.5 Постановка завдання

У результаті проведеного аналізу предметної області та існуючих сервісів для ведення торгових журналів було встановлено, що значна частина сучасних систем обліку торгівельної діяльності працює за моделлю платної підписки та містить велику кількість додаткових функцій, які можуть бути надлишковими для трейдерів-початківців.

У межах бакалаврської кваліфікаційної роботи необхідно розробити мобільний застосунок EZstat - Trader’s journal, призначений для ведення журналу торгових операцій та аналізу результатів власної торгової діяльності на futures-ринку криптовалютних бірж.

Програмний продукт орієнтований насамперед на трейдерів-початківців, яким необхідний простий та зрозумілий інструмент для фіксації торгових операцій, перегляду статистики та оцінки ефективності власної торгової стратегії. На відміну від частини існуючих сервісів, система не повинна бути перевантажена складним професійним функціоналом.

Основними завданнями програмного забезпечення є:

- збереження торгових операцій користувача;
- автоматичний імпорт угод через API криптовалютних бірж;

- ручне додавання торгових записів;
- перегляд статистики торговельної діяльності;
- локальне збереження даних на пристрої користувача;
- забезпечення швидкого доступу до журналу угод через мобільний інтерфейс.

У межах системи необхідно реалізувати журнал угод, сторінку статистики, калькулятор параметрів позиції, сторінку ручного додавання угод та сторінку налаштувань для підключення бірж через API.

Розробка програмного забезпечення повинна виконуватися у форматі мобільного застосунку з використанням платформи .NET MAUI та локальної бази даних SQLite. Для отримання інформації про торгові операції необхідно використати API криптовалютних бірж.

У межах роботи не передбачається реалізація автоматичної торгівлі, відкриття або закриття позицій через API біржі, формування торгових сигналів, інвестиційних рекомендацій, серверної бази даних або хмарної синхронізації.

Результатом виконання бакалаврської кваліфікаційної роботи повинен стати мобільний застосунок EZstat - Trader’s journal, який забезпечує ведення журналу торгових операцій, автоматичний імпорт угод через API криптовалютних бірж, локальне збереження даних та перегляд основних статистичних показників торговельної діяльності трейдера.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних у вигляді ER-діаграми

Для збереження та обробки інформації в мобільному застосунку необхідно визначити основні дані, з якими працює система, а також зв’язки між ними. Логічна модель даних дозволяє показати структуру основних сутностей системи та взаємозв’язки між ними.

Логічну модель даних розроблюваної системи наведено на рис. 2.1.

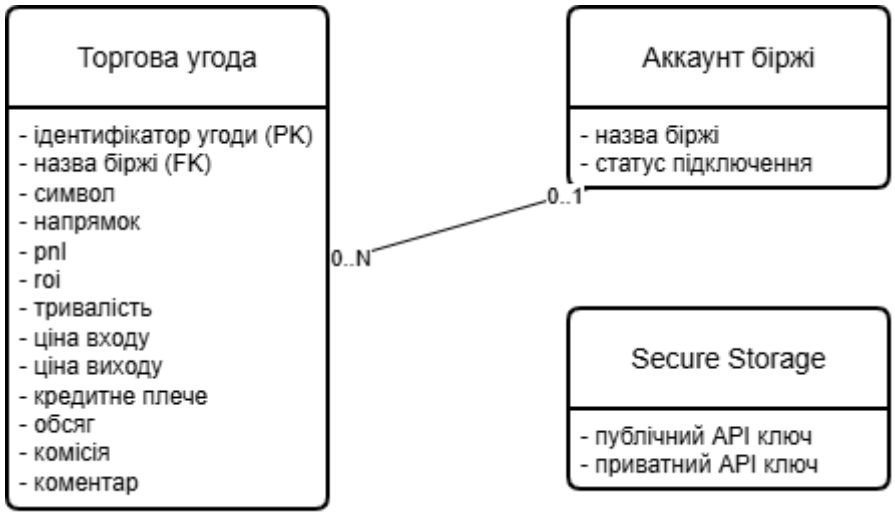

Рис. 2.1 – Логічна модель даних

Основною сутністю системи є торгова угода. Саме вона містить основну інформацію про виконану торгову операцію: символ активу, напрямок позиції, PnL, ROI, ціни входу та виходу, кредитне плече, обсяг позиції, комісію та коментар користувача.

Кожна торгова угода пов’язана з акаунтом біржі, з якого було отримано дані про операцію. Один акаунт біржі може містити багато торгових угод, тому між сутностями реалізовано зв’язок один-до-багатьох.

Сутність «Акаунт біржі» використовується для збереження інформації про підключену біржу та статус підключення користувача.

Окремо в системі використовується Secure Storage для безпечного збереження API-ключів користувача. API-ключі не входять до структури локальної бази даних та не зберігаються у таблицях SQLite. Через це Secure Storage не має прямих зв’язків з іншими сутностями логічної моделі даних.

Такий підхід дозволяє відокремити конфіденційні дані користувача від основної бази даних та підвищити безпечність роботи застосунку.

2.2 Діаграма класів та кооперацій

Для опису структури програмної системи та взаємодії між її основними компонентами в межах роботи використовуються діаграма класів і діаграми кооперації. Вони дозволяють показати, з яких основних елементів складається система, а також як ці елементи взаємодіють між собою під час виконання окремих процесів.

2.2.1 Діаграма класів. Діаграма класів використовується для відображення основних компонентів програмної системи, їхніх властивостей, методів та зв’язків між ними. Вона дозволяє показати логіку побудови застосунку та роль кожного класу в роботі системи.

Діаграму класів програмної системи наведено на рис. 2.2.

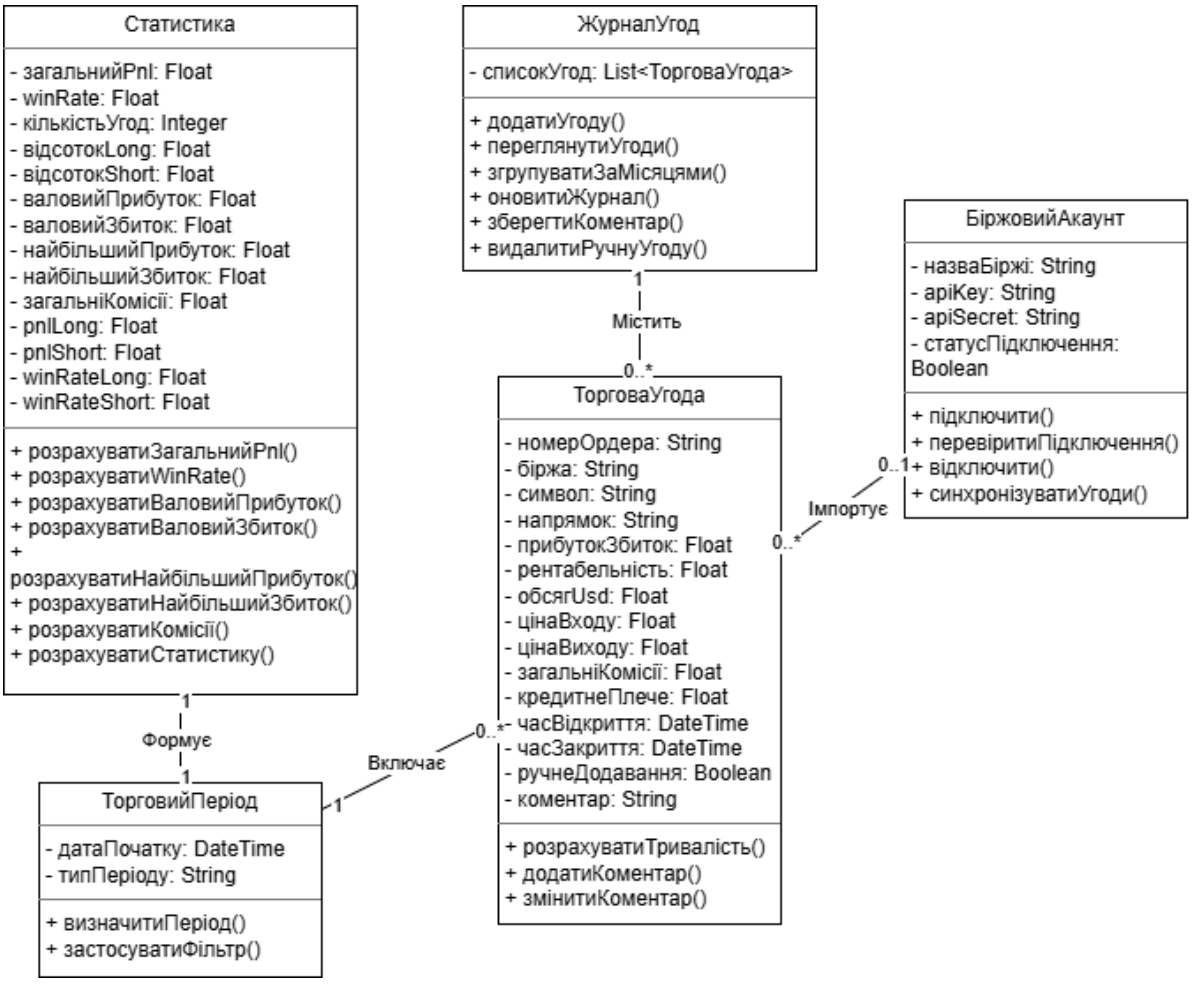

Рис. 2.2 – Діаграма класів

Основним класом системи є клас «ТорговаУгода». Він використовується для збереження інформації про торгову операцію користувача. У класі містяться дані про біржу, торгову пару, напрямок позиції, прибуток або збиток, рентабельність, обсяг позиції, ціни входу та виходу, кредитне плече, комісію, час відкриття та закриття угоди, а також коментар користувача. Крім властивостей, клас містить методи для розрахунку тривалості угоди та роботи з коментарями.

Клас «ЖурналУгод» використовується для роботи зі списком торгових операцій. Через нього виконується перегляд угод, групування записів за місяцями, оновлення журналу та збереження коментарів. Між класами «ЖурналУгод» і «ТорговаУгода» реалізовано зв’язок один-до-багатьох, оскільки журнал містить список торгових операцій.

Клас «БіржовийАкаунт» використовується для підключення криптовалютної біржі та імпорту угод через API. У класі міститься інформація

про назву біржі, API-ключі та статус підключення. Через методи класу виконується підключення біржі, перевірка підключення та синхронізація угод. Між класами «БіржовийАкаунт» і «ТорговаУгода» реалізовано зв’язок один-до-багатьох, оскільки один біржовий акаунт може імпортувати багато торгових операцій.

Клас «Статистика» використовується для розрахунку статистичних показників торговельної діяльності. На його основі система формує загальний PnL, win rate, кількість угод, валовий прибуток, валовий збиток, співвідношення long- і short-позицій, а також інші статистичні показники. Клас містить методи для виконання відповідних розрахунків.

Клас «ТорговийПеріод» використовується для групування угод за певний проміжок часу. На його основі формується статистика за тиждень, місяць або рік. Між класами «ТорговийПеріод» і «ТорговаУгода» реалізовано зв’язок один-до-багатьох, оскільки один торговий період може містити багато угод. Клас «Статистика» формує статистичні показники на основі торгових операцій, які входять до вибраного періоду.

2.2.2 Кооперація імпорту угод з біржі. Для відображення процесу імпорту торгових операцій через API криптовалютної біржі використовується діаграма кооперації імпорту угод.

Кооперацію імпорту угод з біржі наведено на рис. 2.3.

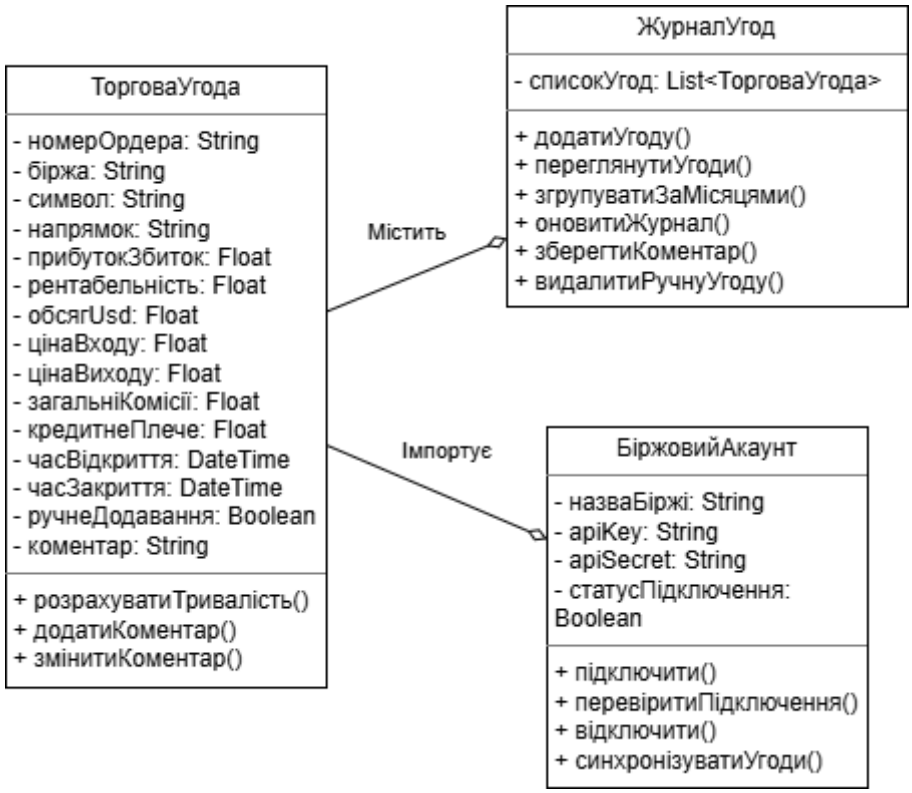

Рис. 2.3 – Проста кооперація «Імпорт угод з біржі»

Під час імпорту угод основна взаємодія відбувається між класами «БіржовийАкаунт», «ТороваУгода» та «ЖурналУгод».

Спочатку клас «БіржовийАкаунт» виконує підключення до API криптовалютної біржі та отримує інформацію про виконані торгові операції користувача. Після отримання даних система створює об’єкти класу «ТороваУгода», які містять інформацію про кожну імпортовану угоду.

Далі створені торгові записи передаються до класу «ЖурналУгод». Саме цей клас використовується для роботи зі списком угод, їх оновлення та відображення у журналі застосунку.

Між класами реалізовано послідовну взаємодію: біржовий акаунт отримує дані з API, торгова угода використовується як модель для збереження інформації про операцію, а журнал угод відповідає за роботу зі списком торгових записів у системі.

Перед додаванням нових угод система виконує перевірку на дублювання записів. Це необхідно для того, щоб під час повторної синхронізації одна й та сама угода не була додана до журналу кілька разів.

2.2.3 Кооперація формування статистики за періодом. Для опису процесу формування статистики торговельної діяльності використовується діаграма кооперації формування статистики за періодом.

Кооперацію формування статистики за періодом наведено на рис. 2.4.

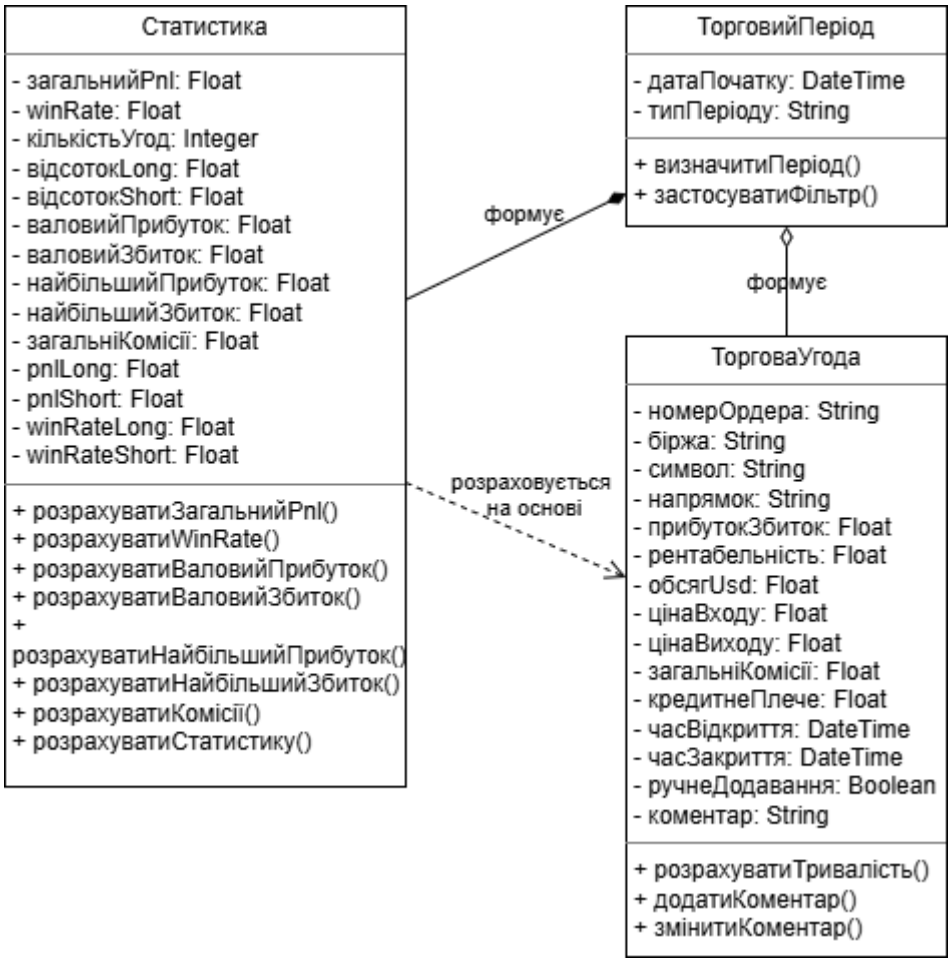

Рис. 2.4 – Формування статистики за періодом

Під час формування статистики основна взаємодія відбувається між класами «ТорговийПеріод», «ТорговаУгода» та «Статистика».

Спочатку користувач обирає необхідний торговий період, наприклад тиждень або місяць. Після цього клас «ТорговийПеріод» визначає список торгових операцій, які входять до вибраного проміжку часу.

Далі система передає список угод до класу «Статистика». Саме цей клас виконує розрахунок статистичних показників на основі отриманих торгових записів.

У процесі взаємодії система розраховує загальний PnL, кількість угод, win rate, співвідношення long- і short-позицій, суму комісій, найбільший прибуток та найбільший збиток.

Таким чином, клас «ТорговийПеріод» відповідає за вибір потрібного проміжку часу, клас «ТорговаУгода» використовується як джерело даних для аналізу, а клас «Статистика» виконує формування статистичних показників для відображення у застосунку.

2.3 Діаграма пакетів

Програмне забезпечення EZstat - Trader’s journal складається з окремих модулів, які відповідають за інтерфейс застосунку, обробку даних, роботу з даними та інтеграцію з API криптовалютних бірж.

Модуль MobileApp використовується для реалізації користувацького інтерфейсу застосунку. У ньому містяться екрани застосунку, компоненти інтерфейсу та навігація між сторінками.

Модуль BusinessLogic містить основну логіку роботи системи. У ньому реалізовано сервіси для синхронізації журналу угод, обробки торгових записів та формування статистичних показників.

Модуль DataAccess відповідає за локальне збереження даних. У ньому реалізовано роботу з SQLite, репозиторій торгових записів та SecureStorage для безпечного збереження API-ключів користувача.

Модуль ExchangeIntegration використовується для роботи з API криптовалютних бірж. У ньому реалізовано сервіси для підключення до біржі, отримання історії угод та синхронізації торгових записів із застосунком.

Структуру програмної системи та взаємодію між основними модулями наведено на рис. 2.5.

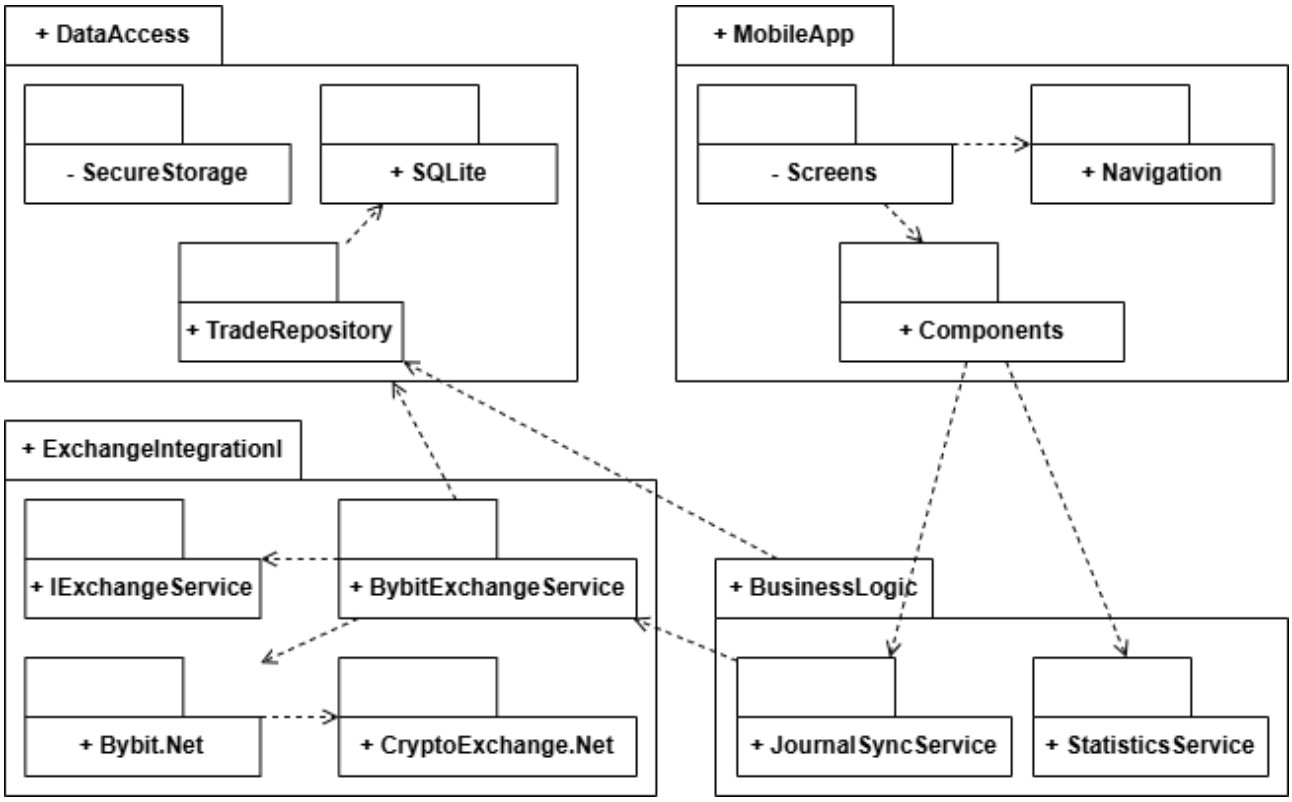

Рис. 2.5 – Діаграма пакетів

Пакет «MobileApp» взаємодіє з пакетом «BusinessLogic» для отримання та обробки даних перед їх відображенням на сторінках застосунку. Компоненти інтерфейсу використовують сервіси бізнес-логіки для оновлення журналу угод та формування статистики.

Пакет «BusinessLogic» взаємодіє з пакетом «DataAccess» для збереження та отримання інформації з локальної бази даних. Через TradeRepository система виконує додавання, оновлення та отримання торгових записів.

Пакет «BusinessLogic» також взаємодіє з пакетом «ExchangeIntegration» для імпорту угод через API криптовалютних бірж. Сервіс BybitExchangeService отримує інформацію про виконані угоди через бібліотеки Bybit.Net та CryptoExchange.Net, після чого передає ці дані до бізнес-логіки застосунку.

SecureStorage використовується окремо від SQLite та призначений для безпечного збереження API-ключів користувача без запису конфіденційних даних у базу даних.

2.4 Діаграма компонентів

Після визначення пакетної структури системи потрібно показати, як окремі частини застосунку взаємодіють між собою під час виконання основних функцій. У розроблюваному програмному забезпеченні інтерфейс користувача не працює напряму з базою даних, API біржі або сховищем ключів. Усі запити від сторінок застосунку спочатку передаються до сервісів прикладної логіки.

Сервіси застосунку виконують основну обробку даних: отримують торгові записи, запускають синхронізацію з біржею, звертаються до локальної бази даних і використовують захищене сховище для доступу до API-ключів. Інтеграція з біржею винесена в окремий компонент, який використовує зовнішні бібліотеки Bybit.Net та CryptoExchange.Net.

Така структура дозволяє розділити відповідальність між частинами системи: інтерфейс відповідає за відображення даних, сервіси - за логіку роботи, доступ до бази даних - за локальне збереження торгових записів, Secure Storage - за ключі доступу, а біржова інтеграція - за отримання даних з API.

Схему взаємодії цих компонентів наведено на рис. 2.6.

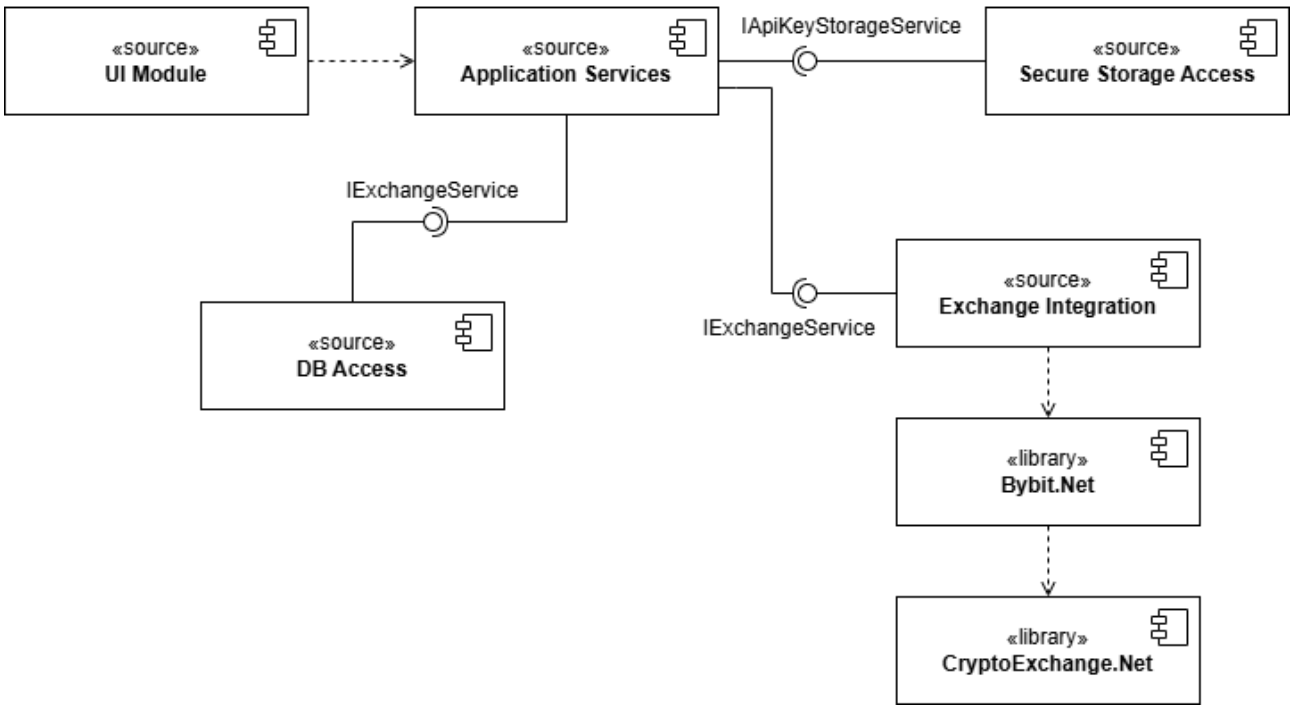

Рис. 2.6 – Діаграма компонентів

На діаграмі видно, що центральним елементом взаємодії є компонент Application Services. Саме він приймає запити від UI Module та передає їх до потрібних частин системи.

Компонент DB Access використовується для роботи з локальною базою даних SQLite. Через нього система зберігає та отримує торгові записи.

Компонент Secure Storage Access використовується окремо від бази даних і відповідає за безпечне збереження API-ключів користувача.

Компонент Exchange Integration відповідає за зв’язок із криптовалютною біржею. Для цього він використовує бібліотеки Bybit.Net та CryptoExchange.Net.

Отже, діаграма показує, що система побудована так, щоб інтерфейс користувача не був напряму пов’язаний із базою даних або API біржі. Уся робота з даними проходить через сервіси застосунку, що робить структуру програмного забезпечення зрозумілішою та зручнішою для подальшої підтримки.

3 РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Система управління інформаційною базою

Для розроблюваної системи було обрано SQLite. SQLite є вбудованою реляційною системою керування базами даних, яка реалізована як бібліотека всередині застосунку та не потребує окремого серверного процесу [14]. Це відповідає архітектурі EZstat – Trader’s journal, оскільки застосунок працює як мобільна система без серверної бази даних, без авторизації та без окремої адміністративної частини.

Особливістю SQLite є те, що база даних зберігається у вигляді одного локального файла [15]. Для мобільного застосунку це є доцільним рішенням, оскільки всі торгові записи користувача можуть зберігатися безпосередньо на пристрої. Завдяки цьому журнал угод залишається доступним навіть без постійного підключення до мережі Internet. Підключення до мережі необхідне лише під час синхронізації з API біржі.

У системі EZstat – Trader’s journal локальна база даних використовується для збереження торгових операцій. До таких даних належать: біржа, символ активу, напрямок позиції, PnL, ROI, обсяг позиції, ціна входу, ціна виходу, комісії, кредитне плече, час відкриття, час закриття, ознака ручного запису та коментар користувача. Саме ці дані надалі використовуються для відображення журналу угод і формування статистичних показників.

У середовищі .NET MAUI робота з SQLite підтримується як типовий підхід для локального збереження даних. У документації Microsoft зазначено, що SQLite дає змогу застосункам .NET MAUI завантажувати та зберігати об’єкти даних у спільному коді [16]. Для реалізації взаємодії із SQLite у проєкті використовується пакет sqlite-net-pcl, який є легкою бібліотекою для збереження даних SQLite у .NET-застосунках [17].

У програмному коді EZstat – Trader’s journal роботу з локальною базою даних винесено в окремий клас AppDatabase. Такий підхід дозволяє не змішувати логіку інтерфейсу користувача з операціями доступу до даних. Сторінки застосунку звертаються до сервісів, а сервіси вже використовують AppDatabase для отримання, додавання, оновлення або видалення торгових записів.

AppDatabase виконує такі основні функції:

- створення локального підключення до SQLite-бази;
- створення необхідних таблиць;
- додавання нових торгових записів;
- отримання списку збережених угод;
- оновлення коментарів користувача;
- видалення ручних записів;
- очищення локальної бази даних.

Основні фрагменти реалізації класу AppDatabase наведено у додатку А.

Окремо потрібно зазначити, що API-ключі бірж не зберігаються в SQLite-базі. Для них використовується SecureStorage, оскільки ці дані є конфіденційними. Такий поділ дозволяє зберігати торгові записи у локальній базі даних, а ключі доступу – в окремому захищеному сховищі.

Отже, SQLite є обґрунтованим вибором для системи управління інформаційною базою в мобільному застосунку EZstat – Trader’s journal. Вона відповідає вимогам локального збереження даних, не потребує серверної інфраструктури, підтримується у .NET MAUI та забезпечує достатню функціональність для ведення журналу торгових операцій.

3.2 Розробка інформаційної бази

У розроблюваній системі інформаційна база складається з двох частин: локальної SQLite-бази даних і захищеного сховища SecureStorage. SQLite використовується для збереження торгових записів користувача, а SecureStorage

– для збереження API-ключів біржі Bybit. Оскільки SecureStorage не є реляційною базою даних і не містить таблиць, він не відображається на фізичній схемі SQLite-бази, а описується окремо як механізм збереження конфіденційних параметрів.

Фізичну структуру таблиці TradeRecord наведено на рис. 3.1.

TradeRecord		
string	OrderId	PK
string	Exchange	
string	Symbol	
string	Direction	
decimal	Pnl	
decimal	Roi	
decimal	VolumeUsd	
decimal	EntryPrice	
decimal	ExitPrice	
decimal	TotalFees	
decimal	Leverage	
datetime	OpenedAt	
datetime	ClosedAt	
bool	IsManual	
string	Comment	
int	DurationMinutes	

Рис. 3.1 – Фізична молель бази даних

Основною таблицею локальної бази даних є TradeRecord. Вона використовується для збереження торгових операцій, які були імпортовані з біржі або додані користувачем вручну. Кожен запис у таблиці відповідає одній закритій торговій угоді.

Первинним ключем таблиці є поле OrderId. Для імпортованих угод воно відповідає ідентифікатору ордера, отриманому з API біржі. Для ручних записів значення формується в застосунку за допомогою Guid.NewGuid().ToString(). Використання OrderId як первинного ключа дозволяє уникати дублювання угод під час повторної синхронізації.

Таблиця TradeRecord містить такі основні групи даних: ідентифікаційні дані угоди, фінансові показники, часові параметри, ознаку походження запису та коментар користувача.

До ідентифікаційних даних належать поля Exchange, Symbol і Direction. Поле Exchange зберігає назву біржі, з якої отримано торговий запис або яку користувач вказав вручну. Поле Symbol містить торгову пару, наприклад BTCUSDT або ETHUSDT. Поле Direction визначає напрямок позиції: Long або Short.

Фінансові показники угоди зберігаються у полях Pnl, Roi, VolumeUsd, EntryPrice, ExitPrice, TotalFees і Leverage. Поле Pnl містить фінансовий результат угоди, поле Roi – рентабельність у відсотках, VolumeUsd – обсяг позиції в доларах США, EntryPrice і ExitPrice – ціни входу та виходу, Leverage – кредитне плече. Поле TotalFees використовується для збереження загальної суми комісій за угодою. Для імпортованих записів це значення формується як сума комісії відкриття та комісії закриття позиції, а для ручних записів вводиться користувачем самостійно.

Часові параметри торгової операції зберігаються у полях OpenedAt, ClosedAt і DurationMinutes. Поле OpenedAt містить дату й час відкриття позиції, ClosedAt – дату й час закриття позиції. Поле DurationMinutes зберігає тривалість угоди у хвилинах. Це значення розраховується один раз перед записом угоди до бази даних на основі різниці між ClosedAt і OpenedAt, після чого використовується без повторного обчислення.

Ознака IsManual показує, чи була угода додана користувачем вручну. Якщо значення дорівнює true, запис створено через сторінку ручного додавання угоди. Якщо значення дорівнює false, запис отримано в результаті синхронізації з API біржі. Поле Comment використовується для збереження текстового коментаря користувача до конкретної угоди.

У програмному коді структура таблиці задається через модель TradeRecord. Фрагмент моделі наведено нижче:

```
public class TradeRecord
{
    [PrimaryKey]
    public string OrderId { get; set; } = "";

    public string Exchange { get; set; } = "Bybit";
    public string Symbol { get; set; } = "";
    public string Direction { get; set; } = "";
```

```
    public decimal Pnl { get; set; }
    public decimal Roi { get; set; }
    public decimal VolumeUsd { get; set; }

    public decimal EntryPrice { get; set; }
    public decimal ExitPrice { get; set; }
    public decimal TotalFees { get; set; }
    public decimal Leverage { get; set; }

    public DateTime? OpenedAt { get; set; }
    public DateTime? ClosedAt { get; set; }

    public bool IsManual { get; set; }

    public string Comment { get; set; } = "";

    public int? DurationMinutes { get; set; }
}
```

Створення таблиці виконується у класі AppDatabase. Для цього використовується метод CreateTableAsync<TradeRecord>(), який формує таблицю на основі властивостей моделі TradeRecord. Приклад наведено нижче:

```
var databasePath = Path.Combine(FileSystem.AppDataDirectory, "ezstat.db3");

_database = new SQLiteAsyncConnection(databasePath);
_database.CreateTableAsync<TradeRecord>().Wait();
```

Шлях до файла бази даних формується через FileSystem.AppDataDirectory, тому файл ezstat.db3 зберігається у локальному сховищі застосунку на пристрої користувача.

Для запису нових угод використовується метод InsertOrReplaceAsync. Він додає новий запис або оновлює наявний запис із таким самим OrderId. Це важливо для синхронізації з біржею, оскільки одна й та сама угода не повинна дублюватися у журналі. Реалізацію репозиторію торгових записів наведено у додатку Б.

Окремо в системі реалізовано оновлення коментаря до угоди. Для цього використовується SQL-запит, який змінює поле Comment у записі з відповідним OrderId.

API-ключі біржі не зберігаються у таблиці TradeRecord і взагалі не входять до структури SQLite-бази. Для них використовується SecureStorage. У системі зберігаються два значення: API Key та API Secret. Вони записуються як пари ключ-значення. Реалізацію сервісу захищеного збереження API-ключів наведено у додатку В.

Отже, інформаційна база системи EZstat – Trader’s journal побудована так, щоб основні торгові записи зберігалися у локальній SQLite-базі, а конфіденційні API-ключі – окремо у SecureStorage. Такий підхід дозволяє розділити звичайні облікові дані системи та чутливі параметри доступу до біржі.

3.3 Вибір інструментарію для створення прикладного програмного забезпечення

Для реалізації мобільного застосунку EZstat – Trader’s journal необхідно було обрати набір програмних засобів, які забезпечують створення мобільного інтерфейсу, роботу з локальною базою даних, інтеграцію з API криптовалютної біржі та підтримку подальшого розширення функціональності системи. Вибір інструментарію виконувався з урахуванням архітектури застосунку, функціональних вимог системи та специфіки мобільної платформи.

Основною платформою розробки було обрано .NET MAUI. .NET MAUI є кросплатформеним фреймворком компанії Microsoft для створення мобільних і настільних застосунків із використанням спільної кодової бази [18]. Використання .NET MAUI дозволяє реалізувати інтерфейс користувача, бізнес-логіку та взаємодію з локальною базою даних у межах одного проєкту.

Важливою причиною вибору .NET MAUI є можливість подальшого розширення застосунку на інші платформи. У межах роботи основною цільовою платформою є Android, однак .NET MAUI дає змогу використовувати спільну кодову базу для Android, iOS, macOS і Windows. Це дає можливість перенести EZstat – Trader’s journal на iOS або desktop-платформи без повної переробки архітектури застосунку.

Для реалізації прикладної логіки використовується мова програмування C#. Дана мова є основною мовою платформи .NET та підтримує об’єктно-орієнтований підхід до розробки програмного забезпечення. Крім цього, C# має велику кількість готових бібліотек і добре підходить для створення мобільних застосунків, які працюють із локальними даними та зовнішніми API.

Користувацький інтерфейс реалізовано за допомогою XAML. Використання XAML дозволяє декларативно описувати елементи інтерфейсу мобільного застосунку, зменшує складність побудови сторінок і спрощує підтримку візуальної частини системи. За допомогою XAML реалізовано сторінки журналу угод, статистики, налаштувань та форму ручного додавання торгових записів.

Для локального збереження даних було обрано SQLite. SQLite є вбудованою реляційною системою управління базами даних, яка не потребує окремого серверного процесу та зберігає базу даних у межах одного локального файла. Такий підхід добре підходить для мобільного застосунку, оскільки забезпечує швидкий доступ до торгових записів і дозволяє працювати із журналом угод без постійного підключення до мережі Internet.

Для взаємодії із SQLite у проєкті використовується бібліотека sqlite-net-pcl. Вона є легкою ORM-обгорткою над SQLite та дозволяє працювати з таблицями через C#-класи [17]. Використання sqlite-net-pcl спрощує створення таблиць, виконання CRUD-операцій та зменшує кількість низькорівневих SQL-запитів у програмному коді.

Інтеграція з криптовалютною біржею Bybit реалізована за допомогою бібліотеки Bybit.Net. Дана бібліотека забезпечує готові механізми роботи з REST API біржі, отримання історії торгових операцій, інформації про баланс та інших даних [19]. Використання готової бібліотеки дозволяє спростити реалізацію мережевої взаємодії та зменшити кількість власного коду для роботи з API.

Бібліотека Bybit.Net побудована на основі CryptoExchange.Net – універсальної бібліотеки для роботи з криптовалютними біржами [20]. CryptoExchange.Net забезпечує механізми HTTP-запитів, десеріалізації відповідей API, обробки помилок та асинхронної взаємодії з біржовими сервісами.

Для захищеного збереження API-ключів використовується SecureStorage. Даний механізм дозволяє зберігати конфіденційні дані окремо від локальної SQLite-бази та використовує захищене сховище мобільної операційної системи.

Середовищем розробки було обрано Visual Studio 2026. Visual Studio підтримує розробку .NET MAUI-застосунків, інтеграцію з Android SDK, керування NuGet-пакетами та засоби налагодження мобільних застосунків. Це дозволяє виконувати розробку, тестування та запуск системи в межах одного програмного середовища.

Для тестування застосунку використовуються Android SDK та Android Emulator. Це дозволяє перевіряти роботу застосунку без використання фізичного мобільного пристрою та спрощує процес налагодження інтерфейсу й функціональних модулів.

Для проєктування користувацького інтерфейсу мобільного застосунку використовувалося середовище Figma. У Figma було створено макети сторінок журналу угод, статистики, налаштувань і форми ручного додавання торгових записів. Це дозволило сформувати єдину структуру інтерфейсу ще до етапу програмної реалізації.

Основні технології, використані під час реалізації EZstat – Trader’s journal, наведено у табл. 3.1.

Таблиця 3.1

Використані технології

Технологія	Призначення
.NET MAUI	Розробка мобільного застосунку
C#	Реалізація прикладної логіки
XAML	Створення інтерфейсу
SQLite	Локальна база даних
sqlite-net-pcl	Робота з SQLite
Bybit.Net	Інтеграція з API Bybit
SecureStorage	Захищене збереження API-ключів
Visual Studio 2026	Середовище розробки

Використані технології забезпечують реалізацію мобільного інтерфейсу, локального збереження торгових записів, інтеграції з API криптовалютної біржі

та захищеного збереження конфіденційних даних. Крім цього, використання .NET MAUI створює можливість подальшого розширення системи для інших платформ без повної переробки програмного забезпечення.

3.4 Алгоритмізація та програмування програмних модулів

Функціонування мобільного застосунку EZstat – Trader’s journal базується на наборі взаємопов’язаних алгоритмів, які забезпечують отримання торгових даних із API криптовалютної біржі, їх локальне збереження, подальшу обробку та відображення статистичної інформації користувачу. Алгоритмічна частина програмного забезпечення є однією з основних складових системи, оскільки саме вона визначає порядок взаємодії між інтерфейсом застосунку, локальною базою даних SQLite та зовнішнім API біржі Bybit.

Під час розробки алгоритмів програмного забезпечення основна увага приділялась:

- мінімізації кількості запитів до API біржі;
- уникненню дублювання торгових записів;
- швидкому формуванню статистики;
- можливості роботи із локально збереженими даними;
- забезпеченню стабільної роботи застосунку при відсутності підключення до Internet;
- коректній обробці помилок API та порожніх наборів даних.

Архітектурно алгоритми системи можна поділити на кілька основних груп:

- алгоритми підключення біржі;
- алгоритми синхронізації торгових угод;
- алгоритми локального збереження даних;
- алгоритми формування статистики;
- алгоритми підготовки даних для відображення у журналі угод;
- алгоритми обробки інтерфейсних подій користувача.

3.4.1 Підключення біржі. Одним із перших алгоритмів, з яким взаємодіє користувач, є алгоритм підключення біржі Bybit. Його основним призначенням є перевірка коректності API-ключів та збереження параметрів підключення у захищеному локальному сховищі SecureStorage.

Після введення користувачем API Key та API Secret застосунок перевіряє, чи всі необхідні поля були заповнені. Якщо хоча б одне поле є порожнім, система припиняє виконання алгоритму та виводить повідомлення про помилку. Така перевірка дозволяє уникнути формування некоректного HTTP-запиту до API біржі.

Після успішної перевірки введених даних застосунок створює HTTP-запит до API Bybit. Для цього використовується бібліотека Bybit.Net, яка реалізує механізми взаємодії із REST API криптовалютної біржі. На цьому етапі виконується перевірка коректності авторизації та можливості доступу до даних користувача. Фрагмент реалізації підключення до Bybit API наведено у додатку Д.

У випадку успішного підключення API-ключі зберігаються у SecureStorage мобільного пристрою. SecureStorage використовується окремо від SQLite та призначений для захищеного локального збереження конфіденційних даних користувача. Після цього у застосунку змінюється статус біржі на «Підключено», а елементи інтерфейсу сторінки налаштувань оновлюються відповідно до поточного стану підключення. Логіку обробки підключення біржі на сторінці налаштувань наведено у додатку Е.

Якщо підключення завершується помилкою, користувачу виводиться повідомлення про неможливість авторизації або відсутність доступу до API біржі.

Блок-схему алгоритму підключення біржі наведено на рис. 3.2.

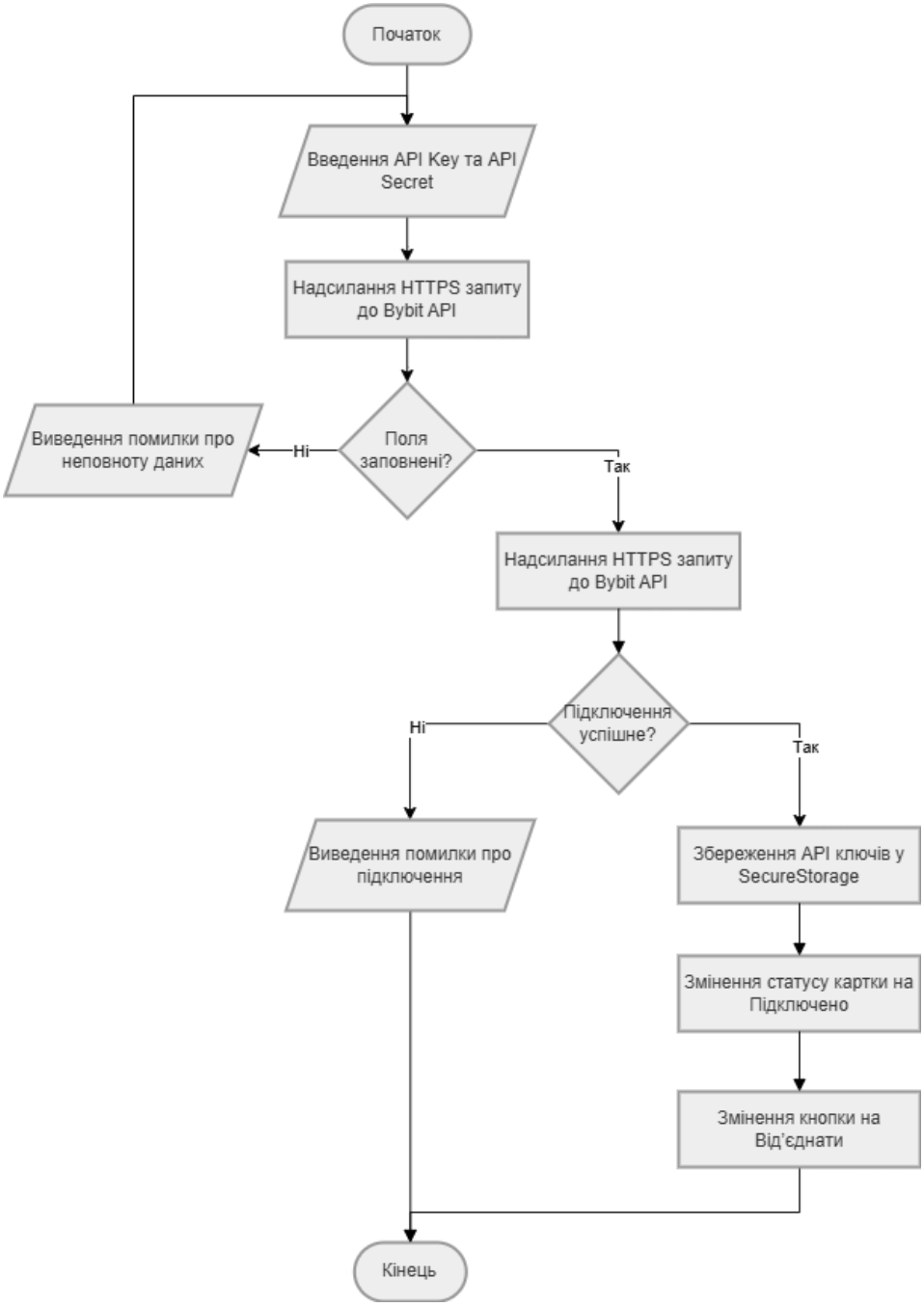

Рис. 3.2 – Блок-схема алгоритму підключення біржі

3.4.2 Синхронізація угод. Після успішного підключення біржі застосунок отримує можливість виконувати синхронізацію торгових угод. Алгоритм синхронізації використовується під час відкриття журналу угод або ручного оновлення списку записів користувачем.

На початковому етапі алгоритм зчитує API-ключі із SecureStorage та формує HTTP-запит до API біржі Bybit для отримання історії закритих Futures-угод користувача. Отримані дані проходять первинну обробку та

перетворюються у внутрішню модель TradeRecord, яка використовується у локальній базі даних SQLite. Фрагмент перетворення даних Bybit API у внутрішню модель системи наведено у додатку Е.

Для кожної торгової операції формуються:

- унікальний ідентифікатор OrderId;
- торговий символ;
- напрямок позиції;
- прибуток або збиток (PnL);
- відсоток прибутковості (ROI);
- обсяг позиції;
- значення кредитного плеча;
- ціни відкриття та закриття;
- сумарна комісія;
- час відкриття та закриття позиції.

Після цього запускається алгоритм перевірки дублювання торгових записів. Система послідовно обробляє кожну отриману угоду та перевіряє наявність запису з таким самим OrderId у локальній базі даних SQLite.

На відміну від повної повторної синхронізації всіх записів, застосунок використовує механізм раннього завершення циклу. Якщо система знаходить угоду, яка вже існує у локальній базі даних, синхронізація нових записів припиняється. Такий підхід використовується через те, що біржа повертає список угод у порядку від новіших до старіших. Відповідно, після знаходження першого дубліката всі наступні записи також уже були раніше збережені у системі.

Після завершення синхронізації застосунок отримує оновлений список угод із локальної бази даних SQLite та виконує їх групування за місяцями для подальшого відображення у журналі торгових операцій. Реалізацію сервісу синхронізації журналу наведено у додатку Ж.

Блок-схему алгоритму синхронізації торгових угод наведено на рис. 3.3.

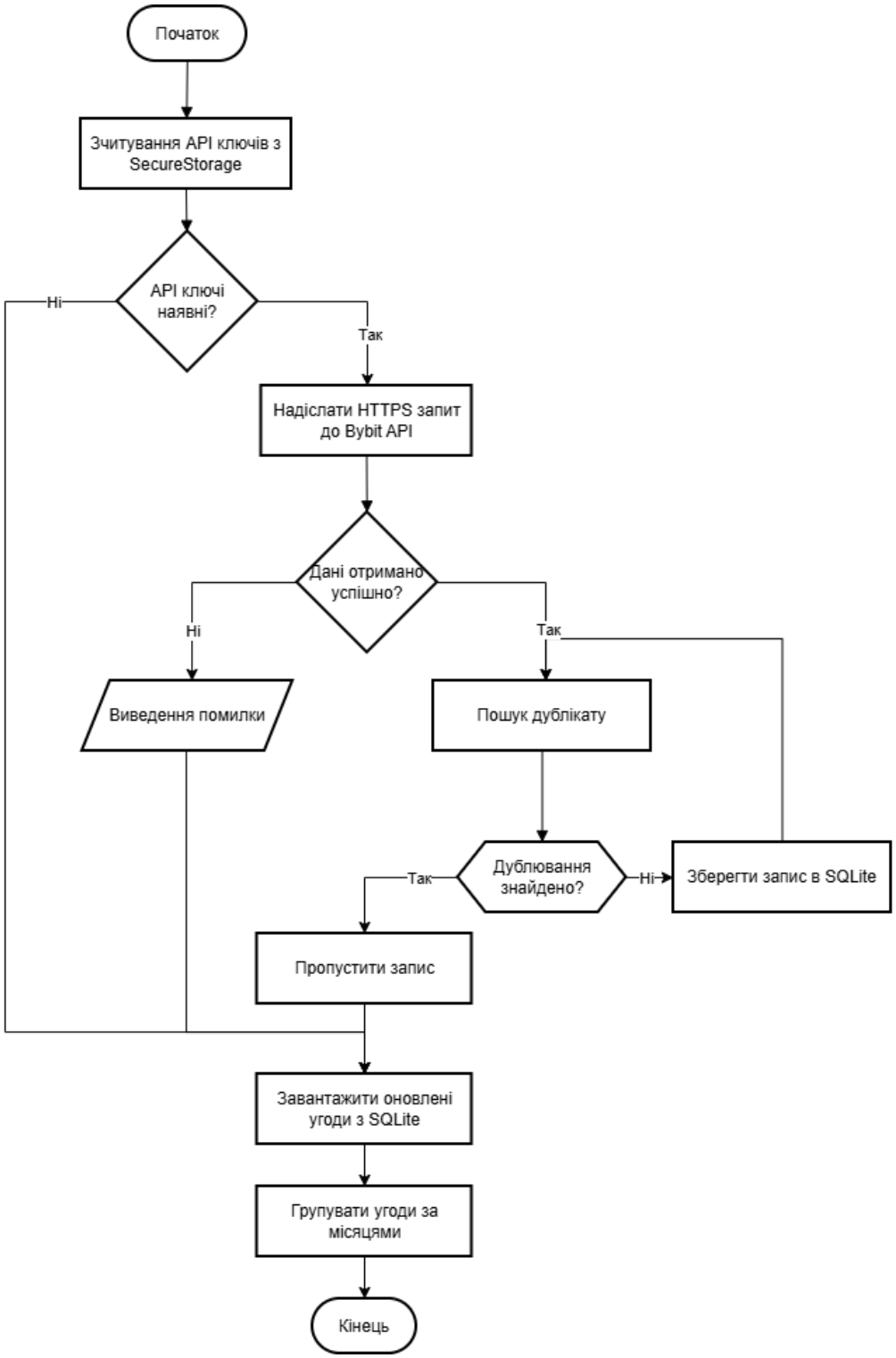

Рис. 3.3 – Блок-схема алгоритму імпорту угод

3.4.3 Ручне додавання угод. Окрім автоматичного імпорту угод із біржі, у застосунку передбачено ручне додавання торгових записів. Цей алгоритм використовується тоді, коли користувач хоче самостійно внести угоду до журналу без отримання даних через API.

На початковому етапі користувач відкриває сторінку додавання угоди та заповнює форму. До обов’язкових даних належать біржа, символ активу, напрямок позиції, PnL, обсяг позиції, ціна входу, ціна виходу, кредитне плече, дата та час відкриття і закриття угоди. Комісія та коментар є додатковими параметрами.

Після натискання кнопки збереження система перевіряє коректність введених даних. Якщо обов’язкові поля не заповнені або значення часу закриття є меншим за час відкриття, користувачу виводиться повідомлення про помилку. У такому випадку запис не створюється.

Якщо введені дані коректні, застосунок формує об’єкт TradeRecord. Для ручної угоди значення OrderId створюється засобами програми, тому запис отримує унікальний ідентифікатор незалежно від біржі. Поле IsManual встановлюється як true, що дозволяє відрізняти ручні записи від імпортованих через API. Фрагмент реалізації ручного додавання торгової угоди наведено у додатку К.

Під час створення ручного запису система також обчислює тривалість угоди в хвилинах на основі часу відкриття та закриття позиції. Після формування об’єкта угода зберігається у локальній SQLite-базі та стає доступною у журналі разом з іншими торговими записами.

Отже, алгоритм ручного додавання угоди забезпечує можливість ведення журналу навіть без підключення біржі та дозволяє користувачу самостійно фіксувати торгові операції.

3.4.4 Завантаження статистики. Одним із центральних алгоритмів системи є алгоритм завантаження статистики. Його основним завданням є підготовка торгових даних для подальшого формування статистичних показників користувача.

Під час відкриття сторінки статистики застосунок виконує очищення локальної бази даних від торгових записів, старших за 180 днів. Після цього визначається початкова дата вибраного статистичного періоду. У поточній реалізації користувач може переглядати статистику за тиждень, місяць або рік.

На основі вибраного періоду із локальної бази SQLite отримується список торгових угод. Далі список передається до сервісу формування статистики. Якщо біржа підключена, додатково виконується отримання поточного балансу Futures-гаманця користувача.

Важливою особливістю алгоритму є те, що статистика формується виключно на основі локальної бази даних SQLite. Це дозволяє:

- зменшити кількість HTTP-запитів до API біржі;
- прискорити відкриття сторінки статистики;
- забезпечити роботу статистики навіть при тимчасовій відсутності Internet-з’єднання.

Блок-схему алгоритму завантаження статистики наведено на рис. 3.4.

Рис. 3.4 – Блок-схема алгоритму завантаження статистики

3.4.5 Формування статистики. Основним алгоритмом програмного забезпечення є алгоритм формування статистичних показників. Саме цей алгоритм реалізує ключову функціональність EZstat – Trader’s journal, пов’язану з аналізом торговельної діяльності користувача.

Алгоритм отримує список торгових угод за вибраний період та визначає загальну кількість записів. Якщо список угод порожній, усім статистичним показникам присвоюються нульові значення. Такий підхід дозволяє уникнути помилок ділення на нуль та забезпечує стабільне відображення інтерфейсу при відсутності торгових операцій.

У випадку наявності торгових записів система формує окремі списки:

- Long-угод;
- Short-угод;
- прибуткових угод;
- збиткових угод.

На основі сформованих списків виконується обчислення статистичних показників. Фрагмент реалізації формування статистичних показників наведено у додатку Л.

Першим етапом є визначення сумарного фінансового результату Net PnL, який обчислюється як сума прибутку та збитку всіх торгових угод за вибраний період.

Після цього визначається кількість прибуткових угод та розраховується показник Win Rate, який відображає відсоток прибуткових угод відносно загальної кількості записів.

Далі система визначає співвідношення Long та Short позицій, а також обчислює окремі статистичні показники для кожного напрямку позиції:

- сумарний прибуток або збиток;
- відсоток прибуткових угод;
- частку угод у загальному обсязі статистики.

Окремо виконується розрахунок:

- загальної суми прибутку (Gross Profit);
- загальної суми збитку (Gross Loss);
- сумарної комісії всіх угод (Total Fees);
- найбільшого прибутку за одну угоду;
- найбільшого збитку за одну угоду.

Після завершення обчислень система виконує форматування грошових та відсоткових значень. Для показників прибутку та збитку додатково визначається колір відображення залежно від знака значення.

Після цього статистичні показники передаються до елементів інтерфейсу користувача та відображаються на сторінці статистики.

Блок-схему алгоритму формування статистичних показників наведено на рис. 3.5.

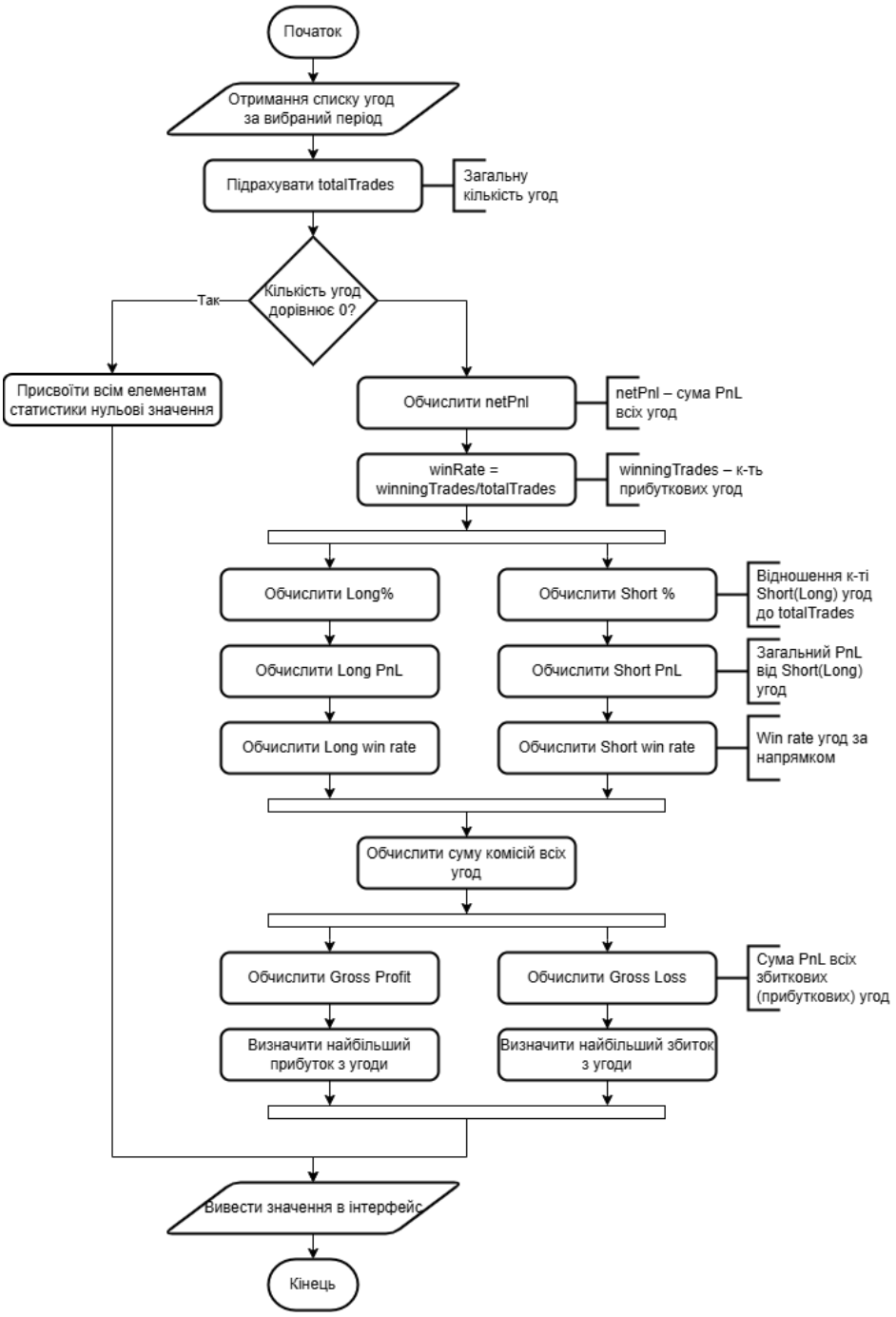

Рис. 3.5 – Блок-схема алгоритму формування статистики

Таким чином, розроблені алгоритми забезпечують повний цикл роботи мобільного застосунку EZstat – Trader’s journal: від підключення біржі та отримання торгових даних до локального збереження, аналізу та відображення статистичних показників користувача. Використання локальної бази даних SQLite, механізму раннього завершення синхронізації та попередньо сформованих статистичних обчислень дозволяє забезпечити високу швидкодію

застосунку та мінімізувати кількість мережевих запитів до API криптовалютної біржі.

3.4.6 Реалізація калькулятора позиції. Окремим модулем мобільного застосунку є калькулятор позиції, призначений для допоміжного розрахунку параметрів торгової угоди перед її відкриттям. Основною задачею калькулятора є визначення допустимого обсягу позиції на основі ризику, який користувач готовий прийняти.

Для виконання розрахунків користувач вводить баланс торгового рахунку, відсоток ризику, ціну входу, ціну стоп-лосу та значення кредитного плеча. Після натискання кнопки розрахунку система перевіряє коректність введених значень та виконує обчислення основних параметрів позиції.

На першому етапі визначається сума допустимого ризику у доларах. Далі система обчислює різницю між ціною входу та стоп-лосом і визначає відносний ризик зміни ціни. На основі отриманих значень розраховується рекомендований обсяг позиції, необхідна маржа та приблизна кількість активу.

Результати розрахунку відображаються безпосередньо на сторінці калькулятора та мають довідковий характер. Отримані значення не є торговою рекомендацією і використовуються лише як допоміжний інструмент для оцінки параметрів угоди.

Під час реалізації калькулятора було передбачено перевірку помилкових даних, зокрема від’ємних значень, нульових параметрів і ситуацій, у яких ціна входу дорівнює ціні стоп-лосу. Для обробки числових значень використовується окремий метод ParseDecimal, який підтримує введення десяткових чисел як через крапку, так і через кому. Фрагмент програмної реалізації калькулятора позиції наведено у додатку М.

3.5 Опис інтерфейсу користувача

Інтерфейс користувача є важливою складовою мобільного застосунку EZstat – Trader’s journal, оскільки саме через нього користувач переглядає торгові записи, аналізує статистику, додає нові угоди та налаштовує підключення до біржі.

Застосунок має темну кольорову тему, що відповідає загальній стилістиці сучасних торгових сервісів і зменшує візуальне навантаження під час тривалої роботи. Навігація між основними сторінками реалізована за допомогою нижньої панелі. Через неї користувач може швидко перейти до статистики, журналу угод, калькулятора та налаштувань.

Основною сторінкою застосунку є журнал торгових операцій. На цій сторінці відображаються угоди користувача, згруповані за місяцями. Для кожного місяця показується кількість угод і сумарний фінансовий результат. Якщо період завершився з прибутком, картка місяця має зеленуватий відтінок, а якщо результат є від’ємним – червонуватий. Це дає змогу швидко оцінити результат торговельної діяльності без детального перегляду кожної угоди.

Інтерфейс сторінки журналу торгових операцій наведено на рис. 3.6.

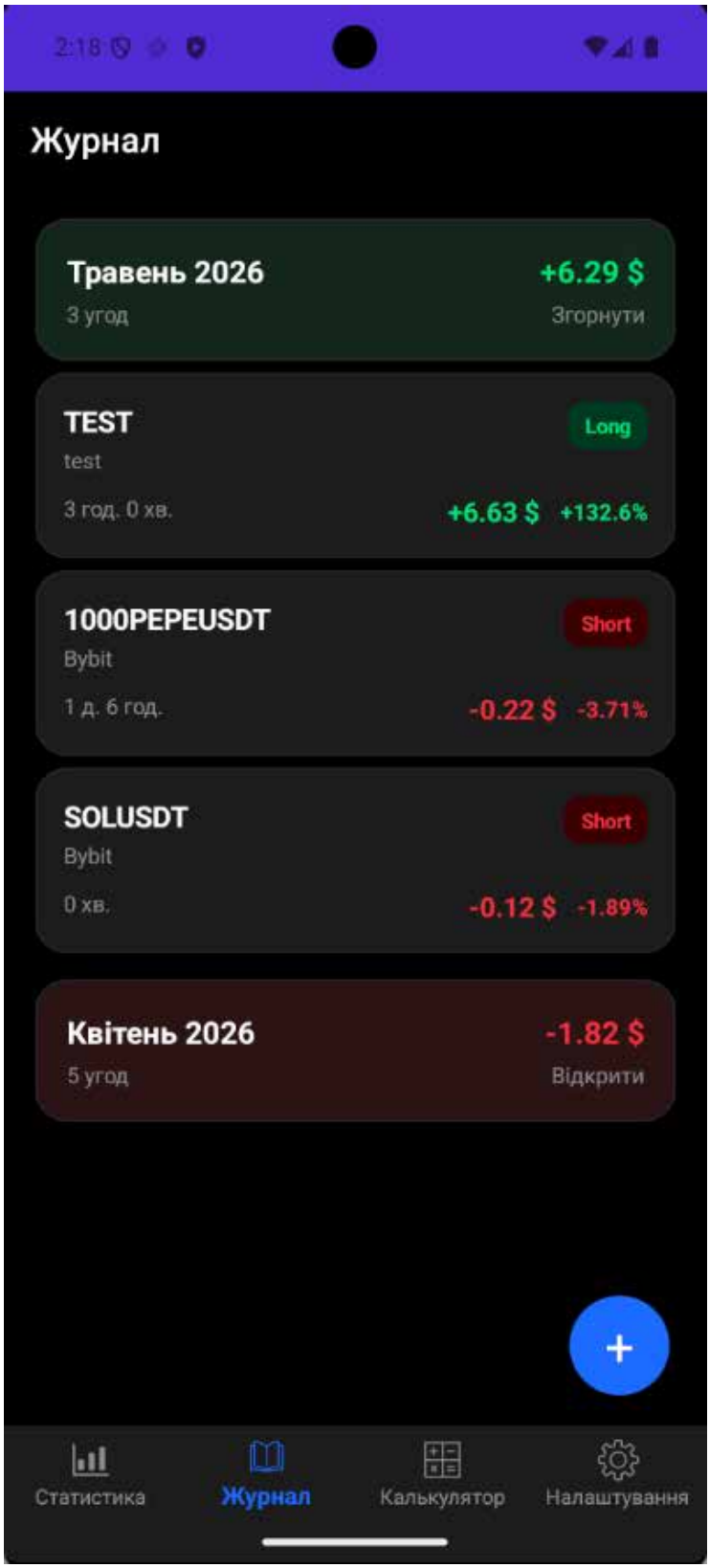

Рис. 3.6 – Журнал угод

Після розгортання місяця користувач отримує доступ до окремих карток угод. Картка торгової операції містить назву торгової пари, біржу, напрямок позиції, PnL, ROI, тривалість угоди, ціни входу та виходу, обсяг позиції, кредитне

плече, комісії та коментар користувача. Такий формат дозволяє показати основну інформацію компактно, але водночас залишити можливість перегляду деталей.

Для ручного внесення торгових операцій у застосунку реалізовано окрему сторінку додавання угоди. Вона використовується тоді, коли користувач хоче самостійно додати запис без синхронізації з біржею. Форма містить поля для введення біржі, торгової пари, напрямку позиції, PnL, ROI, обсягу угоди, ціни входу, ціни виходу, кредитного плеча, комісії, часу відкриття та закриття, а також коментаря. Обов’язкові поля перевіряються перед збереженням запису.

Сторінку ручного додавання угоди наведено на рис. 3.7.

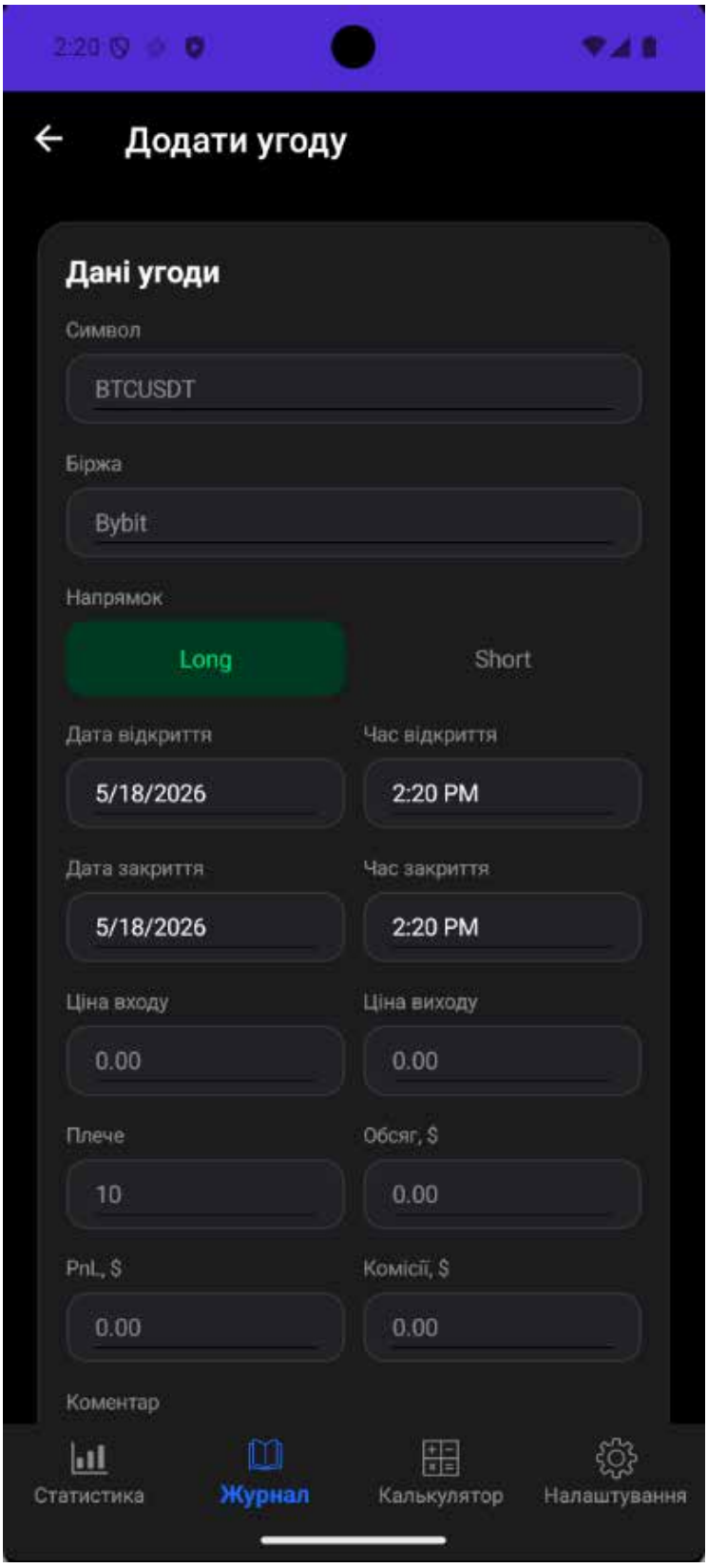

Рис. 3.7 – Додавання угоди

Сторінка статистики призначена для узагальнення результатів торговельної діяльності. На ній відображаються основні показники за вибраний період: загальний PnL, кількість угод, win rate, сума комісій, найбільший прибуток,

найбільший збиток, а також співвідношення Long і Short позицій. Завдяки цьому користувач може швидко оцінити власний результат без ручного підрахунку даних.

Інтерфейс сторінки статистики наведено на рис. 3.8.

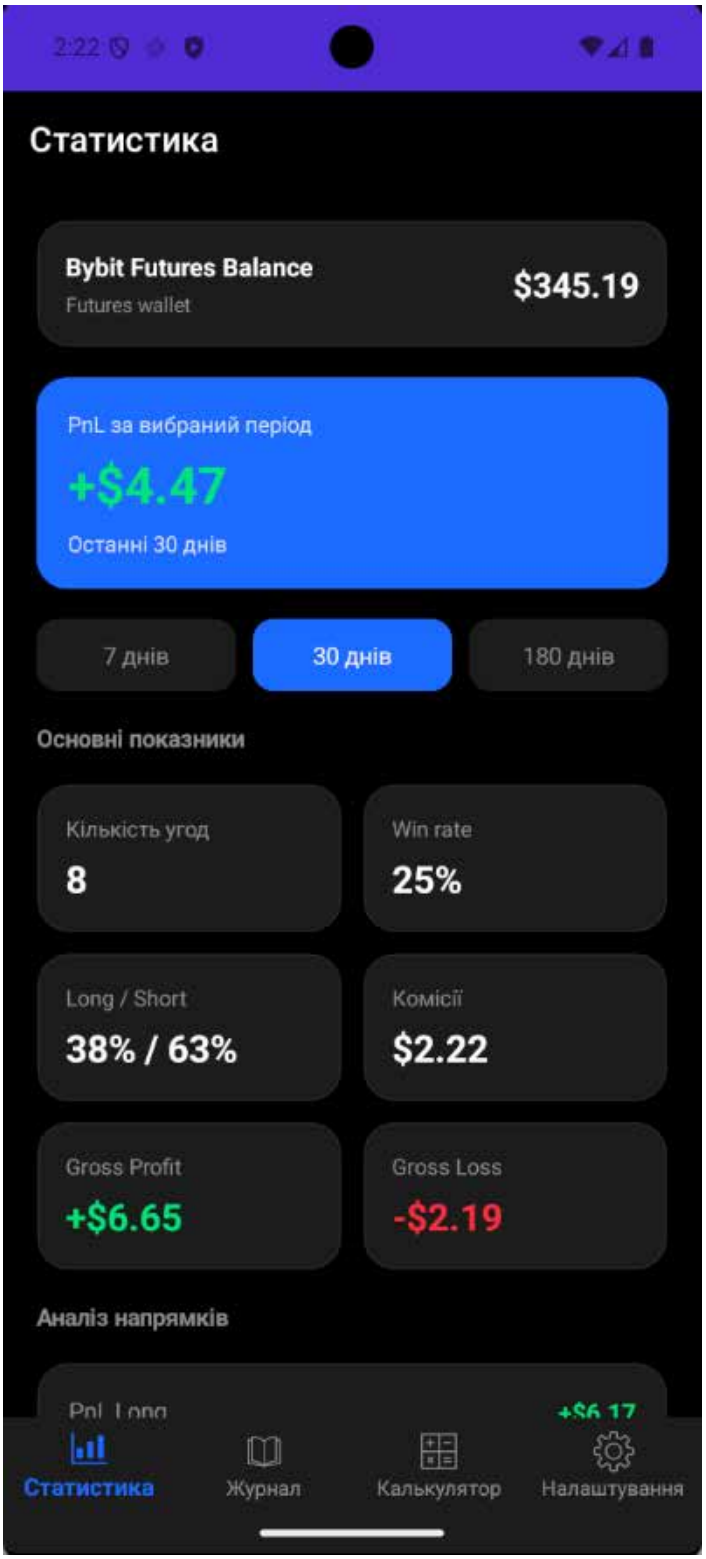

Рис. 3.8 – Статистика

Окремо в застосунку реалізовано торговий калькулятор. Він використовується для допоміжного розрахунку параметрів позиції на основі введених користувачем значень. Калькулятор дозволяє розрахувати об’єм позиції та маржу на основі таких даних як: депозит, ризик, ціну входу, ціну стоп-лосу та кредитне плече. Результати мають довідковий характер і не є торговою рекомендацією.

Інтерфейс сторінки калькулятора наведено на рис. 3.9.

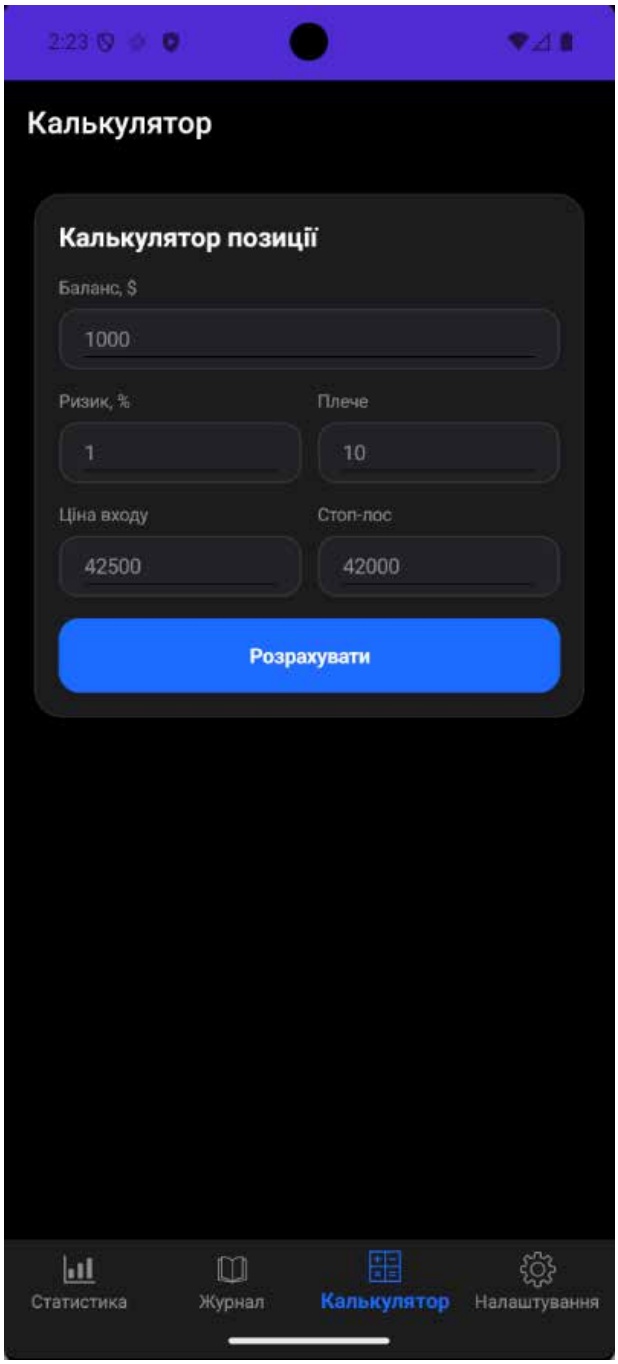

Рис. 3.9 – Калькулятор позиції

Сторінка налаштувань використовується для підключення біржі через API та керування локальними даними застосунку. На цій сторінці користувач може ввести API Key і API Secret, перевірити стан підключення біржі та очистити локальну базу даних. API-ключі не зберігаються у звичайній SQLite-базі, а передаються до SecureStorage.

Інтерфейс сторінки налаштувань наведено на рис. 3.22.

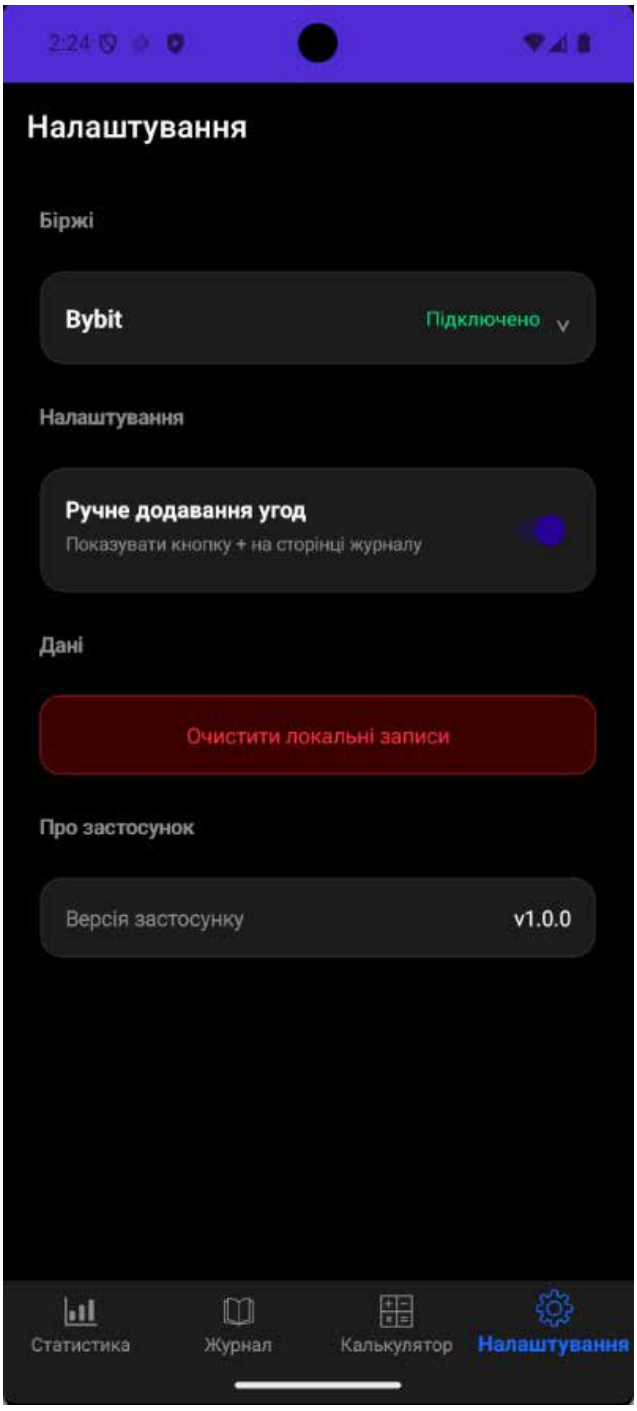

Рис. 3.10 – Налаштування

Під час розробки інтерфейсу використовувалися стандартні компоненти .NET MAUI, зокрема Grid, Border, ScrollView, Label, Entry, Picker і Button. Візуальна структура сторінок описана за допомогою XAML, а обробка дій користувача реалізована у code-behind. Такий підхід відповідає обраній архітектурі застосунку та дозволяє підтримувати просту структуру проєкту без надмірного ускладнення.

Отже, у межах роботи було розроблено користувацький інтерфейс мобільного застосунку EZstat – Trader’s journal, який забезпечує доступ до журналу угод, статистики, калькулятора та налаштувань. Структура інтерфейсу відповідає призначенню системи та забезпечує зручну роботу з основними функціями обліку торговельної діяльності трейдера.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1 Тестування системи

Тестування програмного забезпечення EZstat – Trader’s journal виконувалося з метою перевірки працездатності основних модулів системи, коректності роботи локальної бази даних, правильності обробки торгових записів і стабільності мобільного застосунку під час виконання типових користувацьких сценаріїв. У межах проєкту використовувалися як модульне, так і ручне тестування.

Для реалізації модульного тестування було створено окремий тестовий проєкт EZstat.Mobile.Tests із використанням фреймворку xUnit [21]. Модульні тести використовувалися для перевірки окремих програмних компонентів без запуску графічного інтерфейсу застосунку. Основна частина тестів була спрямована на перевірку моделі TradeRecord, репозиторію локальної бази даних, операцій додавання, видалення та оновлення записів, сортування торгових операцій а також перевірки коректності збереження торгових параметрів. Приклади реалізованих модульних тестів наведено у додатку Н.

Для тестування локальної бази даних використовувалася окрема тимчасова SQLite-база. Це дозволило ізолювати тестове середовище від основних даних користувача. Після завершення виконання тестів тестова база даних автоматично закривалася та видалялася.

Одним із основних модульних тестів був AddAsync_ShouldSaveTrade. Метою цього тесту є перевірка правильності збереження торгової угоди в локальній базі даних. У межах тесту створювався об’єкт TradeRecord із заповненими параметрами торгової операції: біржею, торговою парою, прибутком, ROI та напрямком угоди. Після виклику методу додавання запису виконувався повторний запит до бази даних. Тест вважався успішним, якщо

система повертала один запис із правильним значенням OrderId. Даний тест підтверджує коректність роботи механізму збереження торгових операцій.

Окрему увагу було приділено тесту ExistsAsync_ShouldReturnTrue_WhenTradeExists. Даний тест перевіряє роботу механізму пошуку дубліката торгової операції за її ідентифікатором. Після створення тестового запису система виконувала перевірку існування угоди за значенням OrderId. Результатом тесту мало бути значення true. Ця перевірка використовується під час синхронізації з біржею для уникнення повторного імпорту вже збережених угод.

Тест UpdateCommentAsync_ShouldChangeTradeComment використовувався для перевірки оновлення текстового коментаря користувача. Після створення угоди до запису додавався новий коментар, після чого система повторно отримувала список угод із бази даних. Тест підтверджував правильність збереження текстових приміток до торгових операцій.

Ще одним важливим тестом став DeleteAsync_ShouldRemoveTrade. У межах цього тесту створювався тестовий торговий запис, після чого виконувався виклик методу видалення. Після повторного отримання списку угод система не повинна була містити створений запис. Даний тест підтверджує коректність роботи механізму очищення окремих записів журналу.

Для перевірки очищення локальної бази даних використовувався тест ClearAsync_ShouldRemoveAllTrades. Під час його виконання до бази додавалося кілька тестових угод, після чого виконувався метод очищення. У результаті система повинна була повернути порожній список записів. Такий тест є важливим для перевірки функції очищення локальних даних у налаштуваннях застосунку.

Також було реалізовано тест GetFromDateAsync_ShouldReturnOnlyTradesAfterSelectedDate. Його метою є перевірка правильності фільтрації торгових операцій за датою. До тестової бази даних додавалися угоди з різними датами закриття, після чого система

виконувала вибірку записів лише після визначеної дати. Успішним результатом тесту вважалося повернення лише відповідних угод.

Для перевірки моделі TradeRecord використовувалися окремі модульні тести. Наприклад, тест Pnl_ShouldStoreNegativeValue перевіряв правильність збереження від'ємного значення прибутку або збитку. Це є важливим для журналу торгових операцій, оскільки система повинна коректно працювати як із прибутковими, так і зі збитковими угодами.

Тест IsManual_ShouldBeTrue_WhenTradeAddedManually використовувався для перевірки правильності збереження ознаки ручного додавання угоди. Дана перевірка необхідна для розділення записів, імпортованих із біржі, та записів, створених користувачем вручну.

Після завершення модульного тестування виконувалося ручне функціональне тестування мобільного застосунку. Даний тип тестування використовувався для перевірки поведінки системи під час взаємодії з користувачем через графічний інтерфейс.

У межах ручного тестування перевірялися:

- відкриття сторінок застосунку;
- робота журналу угод;
- додавання нових записів;
- відображення статистики;
- робота калькулятора;
- збереження API-ключів;
- синхронізація торгових операцій;
- очищення локальної бази даних.

План ручного тестування наведено в табл. 4.1.

Таблиця 4.1

План ручного тестування

№	Модуль	Тестова дія	Очікуваний результат	Статус
1	Journal	Відкрити сторінку журналу	Відображається список угод	Успішно
2	Journal	Оновити список угод	Виконується синхронізація	Успішно
3	Journal	Розгорнути картку угоди	Відображаються деталі угоди	Успішно
4	Journal	Зберегти коментар	Коментар зберігається	Успішно
5	Add Order	Додати коректну угоду	Угода додається в БД	Успішно
6	Add Order	Не заповнити Symbol	Система показує помилку	Успішно
7	Statistics	Відкрити сторінку статистики	Відображаються метрики	Успішно
8	Calculator	Виконати розрахунок	Відображається результат	Успішно
9	Settings	Зберегти API-ключі	Ключі зберігаються	Успішно
10	Settings	Очистити локальну БД	Дані видаляються	Успішно

Результати модульного тестування наведено в табл. 4.2.

Таблиця 4.2

Результати модульного тестування

№	Назва тесту	Призначення	Результат
1	DurationMinutes_ShouldStoreCorrectValue	Перевірка збереження тривалості угоди	Успішно
2	Roi_ShouldStorePositiveValue	Перевірка збереження ROI	Успішно
3	Pnl_ShouldStoreNegativeValue	Перевірка роботи зі збитковими угодами	Успішно
4	Direction_ShouldStoreLongValue	Перевірка збереження напрямку Long	Успішно
5	Comment_CanBeEmpty	Перевірка порожнього коментаря	Успішно
6	IsManual_ShouldBeTrue_WhenTradeAdded Manually	Перевірка ручного запису	Успішно
7	TradeRecord_ShouldStoreMainTradeData	Перевірка основних полів TradeRecord	Успішно
8	AddAsync_ShouldSaveTrade	Перевірка збереження угоди	Успішно

Таблиця 4.2 (продовження)

9	GetAllAsync_ShouldReturnTradesNewestFirst	Перевірка сортування угод	Успішно
10	ExistsAsync_ShouldReturnTrue_WhenTradeExists	Перевірка пошуку існуючої угоди	Успішно
11	ExistsAsync_ShouldReturnFalse_WhenTradeDoesNotExist	Перевірка відсутньої угоди	Успішно
12	UpdateCommentAsync_ShouldChangeTradeComment	Перевірка оновлення коментаря	Успішно
13	DeleteAsync_ShouldRemoveTrade	Перевірка видалення угоди	Успішно
14	ClearAsync_ShouldRemoveAllTrades	Перевірка очищення БД	Успішно
15	GetFromDateAsync_ShouldReturnOnlyTradesAfterSelectedDate	Перевірка фільтрації за датою	Успішно
16	DeleteOldTradesAsync_ShouldRemoveTradesOlderThanBorderDate	Перевірка видалення старих записів	Успішно
17	AddAsync_ShouldNotCreateDuplicateOrders	Перевірка уникнення дублікатів	Успішно
18	GetAllAsync_ShouldReturnEmptyList_WhenDatabaseIsEmpty	Перевірка порожньої БД	Успішно

Таблиця 4.2 (закінчення)

19	AddAsync_ShouldSaveManualTrade	Перевірка ручного додавання	Успішно
20	AddAsync_ShouldSaveTradeFees	Перевірка збереження комісій	Успішно
21	AddAsync_ShouldSaveLeverage	Перевірка збереження leverage	Успішно

На момент написання пояснювальної записки всі реалізовані модульні та функціональні тести виконуються успішно. Помилки, які виникали під час розробки системи, були виправлені до завершення етапу тестування.

За результатами тестування встановлено, що мобільний застосунок EZstat – Trader’s journal коректно виконує операції збереження, оновлення, видалення та відображення торгових записів, забезпечує стабільну роботу локальної бази даних SQLite і правильно обробляє основні користувацькі сценарії.

4.2 Вимоги до апаратного та програмного забезпечення

Підрозділ присвячено визначенню середовища, у якому працює мобільний застосунок EZstat – Trader’s journal, а також опису основних програмних компонентів, що беруть участь у його розгортанні. Оскільки система реалізована як мобільний застосунок, основним середовищем виконання є смартфон користувача з операційною системою Android.

Діаграма розгортання показує фізичне розміщення компонентів системи: мобільного застосунку, локальної бази даних, захищеного сховища ключів та зовнішнього API біржі. На відміну від клієнт-серверних систем із власним сервером, EZstat – Trader’s journal не має окремого backend-сервера. Основна логіка виконується локально на пристрої користувача.

Структуру розгортання системи наведено на рис. 4.1.

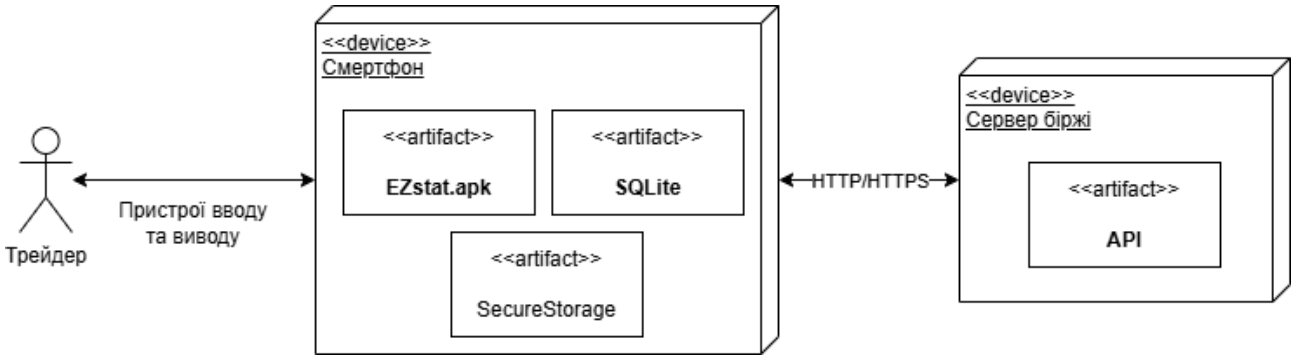

Рис. 4.1 – Діаграма розгортання

Основним вузлом розгортання є смартфон трейдера. На ньому встановлюється застосунок у вигляді APK-файлу. Після запуску застосунок працює в середовищі Android OS та використовує ресурси пристрою для збереження даних, виконання розрахунків і відображення інтерфейсу користувача.

У межах мобільного пристрою розміщуються такі основні артефакти:

- EZstat.apk – інсталяційний файл мобільного застосунку;
- SQLite – локальна база даних для збереження торгових записів;
- SecureStorage – захищене сховище для API-ключів користувача.

SQLite використовується для збереження історії торгових операцій, коментарів, параметрів угод і даних, необхідних для формування журналу та статистики. Це дозволяє переглядати вже збережені записи без постійного підключення до мережі Internet.

SecureStorage використовується окремо від SQLite. У ньому зберігаються API Key та API Secret, необхідні для підключення до біржі. Такий поділ дозволяє не зберігати конфіденційні ключі у звичайній локальній базі даних.

Другим вузлом є сервер біржі. На ньому розміщується API, через який застосунок отримує інформацію про закриті futures-угоди та баланс користувача. З’єднання між смартфоном і сервером біржі виконується через HTTP/HTTPS. Біржа в межах цієї системи використовується тільки як джерело даних.

Під час роботи системи відбувається така послідовність взаємодії: користувач відкриває застосунок, система зчитує збережені API-ключі, надсилає запит до API біржі, отримує дані про угоди, обробляє їх і зберігає у локальній

SQLite-базі. Після цього дані відображаються на сторінках журналу та статистики.

Рекомендовані вимоги до середовища функціонування наведено в табл. 4.3.

Таблиця 4.3

Рекомендовані системні вимоги

Параметр	**Рекомендоване значення**
Тип пристрою	Смартфон
Операційна система	Android 10+
Оперативна пам’ять	4 ГБ
Вільна пам’ять пристрою	200 МБ
Підключення до Internet	Потрібне лише для синхронізації з біржею

Отже, система EZstat – Trader’s journal має просту схему розгортання: мобільний застосунок встановлюється на смартфон користувача, локальні дані зберігаються в SQLite, API-ключі – у SecureStorage, а зовнішній сервер біржі використовується лише для отримання торгової інформації. Така структура відповідає призначенню застосунку як мобільного торгового журналу без власного серверного середовища.

4.3 Склад інсталяційного пакету

Інсталяційний пакет програмного забезпечення EZstat – Trader’s journal призначений для встановлення мобільного застосунку на пристрої під керуванням операційної системи Android. Оскільки система реалізована як автономний мобільний застосунок без окремого серверного середовища, процес встановлення є спрощеним і не потребує додаткового налаштування серверів або зовнішніх баз даних.

Основним компонентом інсталяційного пакету є APK-файл мобільного застосунку. Саме він містить програмний код, графічний інтерфейс, бізнес-

логіку, механізми роботи з локальною базою даних та засоби взаємодії з API криптовалютної біржі.

Після встановлення APK-файлу застосунок автоматично створює необхідні локальні компоненти для роботи системи. До таких компонентів належать локальна база даних SQLite та захищене сховище SecureStorage.

SQLite використовується для локального збереження торгових записів, параметрів угод, коментарів користувача та інших даних журналу. База даних створюється автоматично під час першого запуску застосунку та не потребує окремого встановлення користувачем.

SecureStorage використовується для захищеного збереження API-ключів біржі. Даний компонент використовує вбудовані механізми захисту операційної системи Android і також створюється автоматично під час роботи застосунку.

До складу інсталяційного пакету не входять окремі серверні компоненти, служби баз даних або сторонні програми для адміністрування системи. Уся основна логіка застосунку виконується безпосередньо на мобільному пристрої користувача.

Склад інсталяційного пакету наведено в табл. 4.4.

Таблиця 4.4

Склад інсталяційного пакету

Компонент	**Призначення**
EZstat.apk	Основний файл мобільного застосунку
SQLite	Локальне збереження торгових записів
SecureStorage	Захищене збереження API-ключів

Для встановлення застосунку користувач повинен мати Android-пристрій із підтримкою встановлення APK-файлів. Після інсталяції застосунок готовий до роботи без необхідності додаткового налаштування локальної бази даних або інших програмних компонентів.

Отже, інсталяційний пакет EZstat – Trader’s journal має просту структуру та містить лише компоненти, необхідні для роботи мобільного застосунку. Такий підхід спрощує встановлення системи та зменшує складність її розгортання на мобільному пристрої користувача.

ВИСНОВКИ

У результаті виконаної роботи було розроблено мобільне програмне забезпечення EZstat – Trader’s journal, призначене для обліку, збереження та аналізу торговельної діяльності трейдера на futures-ринку криптовалютних бірж. Розроблена система орієнтована на ведення персонального журналу торгових операцій, перегляд статистичних показників та автоматичне отримання інформації про угоди через API біржі.

Під час виконання роботи було проведено аналіз предметної області та досліджено особливості ведення торгових журналів у сфері криптовалютного трейдингу. У межах аналізу було визначено основні сутності системи, зокрема торгові операції, торгові періоди, статистичні показники, параметри угод і механізми взаємодії з криптовалютними біржами. Також було виконано огляд існуючих програмних рішень для ведення журналу трейдера та встановлено їхні переваги й недоліки.

У результаті аналізу предметної області встановлено, що значна частина існуючих рішень орієнтована переважно на web-платформи а також використовує модель платної підписки. Частина сервісів не підтримує локальне збереження даних або не забезпечує достатньо зручного механізму ручного ведення журналу угод. Це підтвердило доцільність розробки окремого мобільного застосунку з локальним збереженням торгових записів та підтримкою автоматичного імпорту угод через API біржі.

У межах роботи було сформовано функціональні та нефункціональні вимоги до системи. До основних функціональних вимог належать ведення журналу угод, додавання торгових записів вручну, автоматичний імпорт угод із біржі, перегляд статистики, збереження коментарів до торгових операцій і захищене збереження API-ключів користувача. Серед нефункціональних вимог визначено автономність роботи системи та локальне збереження даних.

Під час проєктування програмного забезпечення було виконано моделювання системи із використанням UML-діаграм. У роботі були побудовані діаграма варіантів використання, діаграма активності, діаграма послідовності, діаграма класів, діаграма пакетів, діаграма компонентів та діаграма розгортання. Побудовані моделі дозволили описати структуру програмного забезпечення, взаємодію між компонентами системи та особливості розміщення програмних модулів.

Також у роботі було спроєктовано структуру локальної бази даних для збереження торгових записів. Для цього було побудовано логічну модель даних та реалізовано фізичну структуру бази даних SQLite. У структурі бази даних передбачено збереження інформації про торгові операції, параметри угод, коментарі користувача та службові дані застосунку.

Для реалізації програмного забезпечення було обрано платформу .NET MAUI та мову програмування C#. Вибір .NET MAUI обумовлений можливістю створення сучасних мобільних застосунків із використанням єдиної кодової бази та підтримкою Android-платформи. Для локального збереження даних використано SQLite, а для захищеного збереження API-ключів – SecureStorage.

Для взаємодії з криптовалютною біржею використано бібліотеки Bybit.Net та CryptoExchange.Net. Використання готових бібліотек дозволило реалізувати механізм автоматичного отримання інформації про торгові операції, баланс користувача та історію угод без необхідності низькорівневої реалізації HTTP-запитів.

У процесі реалізації програмного забезпечення було створено основні функціональні модулі системи. Реалізовано сторінку журналу угод із можливістю перегляду торгових записів, розгортання картки угоди та відображення її параметрів. Також було реалізовано форму ручного додавання угоди, сторінку статистики та сторінку налаштувань для збереження API-ключів біржі.

Окрему увагу приділено реалізації механізму синхронізації торгових операцій. Система виконує отримання історії угод через API біржі, перевіряє наявність дублікатів і зберігає нові записи у локальній базі даних SQLite. Це

дозволяє автоматизувати процес наповнення журналу угод та зменшити кількість ручного введення інформації користувачем.

Також у роботі було реалізовано механізми обчислення статистичних показників торговельної діяльності. Застосунок забезпечує розрахунок показників прибутку та збитку, відсотка прибуткових угод, співвідношення Long та Short позицій, загальної суми комісій та інших характеристик торгової діяльності користувача.

Під час виконання роботи було проведено модульне та ручне функціональне тестування системи. Для реалізації модульного тестування створено окремий тестовий проєкт із використанням фреймворку xUnit. У межах тестування перевірялися операції додавання, оновлення, видалення та отримання торгових записів, а також правильність роботи локальної бази даних SQLite.

Крім модульного тестування, було проведено ручне функціональне тестування мобільного застосунку. У межах тестування перевірялися робота журналу угод, додавання нових записів, збереження коментарів, оновлення списку угод, відображення статистики та робота механізму синхронізації з біржею. На момент завершення роботи всі реалізовані тести виконуються успішно.

У роботі також було розглянуто питання розгортання та експлуатації системи. Побудована діаграма розгортання дозволила визначити основні вузли системи, програмні компоненти та взаємодію мобільного застосунку із сервером біржі. Визначено рекомендовані вимоги до середовища функціонування та склад інсталяційного пакету програмного забезпечення.

Отже, у результаті виконання роботи було розроблено мобільний застосунок EZstat – Trader’s journal, який забезпечує ведення журналу торгових операцій, автоматичний імпорт даних через API криптовалютної біржі, локальне збереження інформації та перегляд статистичних показників торговельної діяльності трейдера. Поставлену мету роботи досягнуто в повному обсязі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Binance Academy. What Are Perpetual Futures Contracts? URL: https://www.binance.com/en/academy/article/what-are-perpetual-futures-contracts (дата звернення: 01.04.2026).
2. Bybit. Contract Rules. URL: https://www.bybit.com/en/contract-rules (дата звернення: 01.04.2026).
3. Bybit Help Center. FAQ – USDT Perpetual and Expiry Contracts. URL: https://www.bybit.com/en/help-center/article/FAQ-USDT-Perpetual-and-Expiry-Contracts (дата звернення: 03.04.2026).
4. Bybit Help Center. Profit & Loss Calculations – USDT Contract. URL: https://www.bybit.com/en/help-center/article/Profit-Loss-calculations-USDT-Contract (дата звернення: 03.04.2026).
5. Bybit Help Center. How Leverage Affects ROI. URL: https://www.bybit.com/en/help-center/article/How-Leverage-Affects-ROI (дата звернення: 03.04.2026).
6. U.S. Securities and Exchange Commission. Investor Bulletin: Stop, Stop-Limit, and Trailing Stop Orders. URL: https://www.investor.gov/introduction-investing/general-resources/news-alerts/alerts-bulletins/investor-bulletins-15 (дата звернення: 04.04.2026).
7. Investopedia. 4 Reasons to Keep a Forex Trade Diary. URL: https://www.investopedia.com/articles/forex/11/why-you-need-a-forex-trading-journal.asp (дата звернення: 05.04.2026).
8. TradeZella. Pricing. URL: https://www.tradezella.com/pricing (дата звернення: 14.04.2026).
9. TradeZella. Trading Journal Software That Reveals Your Edge. URL: https://www.tradezella.com/trading-journal (дата звернення: 15.04.2026).

10. TradeZella Help Center. How to Add a Trade Manually in TradeZella. URL: https://help.tradezella.com/en/articles/5829532-how-to-add-a-trade-manually-in-tradezella (дата звернення: 16.04.2026).
11. TraderSync. Trading Journal for Stocks, Forex, Futures, Crypto & Options. URL: https://tradersync.com/ (дата звернення: 17.04.2026).
12. TraderSync. Pricing. URL: https://tradersync.com/pricing/ (дата звернення: 17.04.2026).
13. PNL - Simple Trading Journal: Google Play. URL: https://play.google.com/store/apps/details?id=cypto.trade.manager (дата звернення: 20.04.2026).
14. SQLite. About SQLite. URL: https://sqlite.org/about.html (дата звернення: 25.04.2026).
15. SQLite. Single-file Cross-platform Database. URL: https://sqlite.org/onefile.html (дата звернення: 25.04.2026).
16. Microsoft Learn. .NET MAUI local databases. URL: https://learn.microsoft.com/en-us/dotnet/maui/data-cloud/database-sqlite (дата звернення: 26.04.2026).
17. NuGet. sqlite-net-pcl. URL: https://www.nuget.org/packages/sqlite-net-pcl/ (дата звернення: 26.04.2026).
18. Microsoft. .NET MAUI documentation. URL: https://learn.microsoft.com/en-us/dotnet/maui/ (дата звернення: 30.04.2026).
19. JKorf. Bybit.Net. URL: https://github.com/JKorf/Bybit.Net (дата звернення: 02.05.2026).
20. JKorf. CryptoExchange.Net. URL: https://github.com/JKorf/CryptoExchange.Net (дата звернення: 02.05.2026).
21. xUnit.net. About xUnit.net. URL: https://xunit.net/ (дата звернення: 05.05.2026).

ДОДАТОК А

Реалізація класу AppDatabase

```
public class AppDatabase
{
    private readonly SQLiteAsyncConnection _database;

    public AppDatabase(string? databasePath = null)
    {
        databasePath ??= Path.Combine(FileSystem.AppDataDirectory, "ezstat.db3");

        _database = new SQLiteAsyncConnection(databasePath);
        _database.CreateTableAsync<TradeRecord>().Wait();
    }

    public Task<List<TradeRecord>> GetTradesAsync()
    {
        return _database
            .Table<TradeRecord>()
            .OrderByDescending(x => x.ClosedAt)
            .ToListAsync();
    }

    public async Task SaveTradesAsync(List<TradeRecord> trades)
    {
        foreach (var trade in trades)
        {
            var existingTrade = await _database.FindAsync<TradeRecord>(trade.OrderId);

            if (existingTrade != null)
            {
                break;
            }

            await _database.InsertAsync(trade);
        }
    }

    public Task UpdateCommentAsync(string orderId, string comment)
    {
        return _database.ExecuteAsync(
            "UPDATE TradeRecord SET Comment = ? WHERE OrderId = ?",
            comment,
            orderId);
    }

    public Task<List<TradeRecord>> GetTradesFromDateAsync(DateTime fromDate)
    {
        return _database
            .Table<TradeRecord>()
            .Where(x => x.ClosedAt >= fromDate)
            .OrderByDescending(x => x.ClosedAt)
            .ToListAsync();
    }

    public Task DeleteOldTradesAsync(DateTime borderDate)
    {
        return _database.ExecuteAsync(
            "DELETE FROM TradeRecord WHERE ClosedAt < ?",
            borderDate);
```

```
    }

    public Task SaveTradeAsync(TradeRecord trade)
    {
        return _database.InsertOrReplaceAsync(trade);
    }

    public Task DeleteTradeAsync(string orderId)
    {
        return _database.DeleteAsync<TradeRecord>(orderId);
    }

    public Task ClearTradesAsync()
    {
        return _database.DeleteAllAsync<TradeRecord>();
    }

    public async Task CloseAsync()
    {
        await _database.CloseAsync();
    }
}
```

ДОДАТОК Б

Реалізація репозиторію торгових записів

```
public class TradeRepository : ITradeRepository
{
    private readonly AppDatabase _database;

    public TradeRepository(AppDatabase database)
    {
        _database = database;
    }

    public Task<List<TradeRecord>> GetAllAsync()
    {
        return _database.GetTradesAsync();
    }

    public async Task SaveTradesAsync(IEnumerable<TradeRecord> trades)
    {
        await _database.SaveTradesAsync(trades.ToList());
    }

    public async Task<bool> ExistsAsync(string orderId)
    {
        var trades = await _database.GetTradesAsync();

        return trades.Any(x => x.OrderId == orderId);
    }

    public Task AddAsync(TradeRecord trade)
    {
        return _database.SaveTradeAsync(trade);
    }

    public Task UpdateCommentAsync(string orderId, string comment)
    {
        return _database.UpdateCommentAsync(orderId, comment);
    }

    public Task DeleteAsync(string orderId)
    {
        return _database.DeleteTradeAsync(orderId);
    }

    public Task ClearAsync()
    {
        return _database.ClearTradesAsync();
    }

    public Task<List<TradeRecord>> GetFromDateAsync(DateTime fromDate)
    {
        return _database.GetTradesFromDateAsync(fromDate);
    }

    public Task DeleteOldTradesAsync(DateTime borderDate)
    {
        return _database.DeleteOldTradesAsync(borderDate);
    }
}
```

ДОДАТОК В

Сервіс захищеного збереження API-ключів

```
public class ApiKeyStorageService : IApiKeyStorageService
{
    private const string BybitApiKeyStorageKey = "bybit_api_key";
    private const string BybitApiSecretStorageKey = "bybit_api_secret";

    public Task<string?> GetBybitApiKeyAsync()
    {
        return SecureStorage.GetAsync(BybitApiKeyStorageKey);
    }

    public Task<string?> GetBybitApiSecretAsync()
    {
        return SecureStorage.GetAsync(BybitApiSecretStorageKey);
    }

    public async Task SaveBybitKeysAsync(string apiKey, string apiSecret)
    {
        await SecureStorage.SetAsync(BybitApiKeyStorageKey, apiKey);
        await SecureStorage.SetAsync(BybitApiSecretStorageKey, apiSecret);
    }

    public void RemoveBybitKeys()
    {
        SecureStorage.Remove(BybitApiKeyStorageKey);
        SecureStorage.Remove(BybitApiSecretStorageKey);
    }
}
```

ДОДАТОК Д

Фрагмент реалізації підключення до Bybit API

```
private readonly IApiKeyStorageService _apiKeyStorageService;

public BybitExchangeService(IApiKeyStorageService apiKeyStorageService)
{
   _apiKeyStorageService = apiKeyStorageService;
}

public async Task<bool> ConnectAsync()
{
   using var client = await CreateClientAsync();

   if (client == null)
   {
      return false;
   }

   var result = await client.V5Api.Trading.GetClosedProfitLossAsync(
      category: Category.Linear,
      limit: 1);

   return result.Success;
}

private async Task<BybitRestClient?> CreateClientAsync()
{
   var apiKey = await _apiKeyStorageService.GetBybitApiKeyAsync();
   var apiSecret = await _apiKeyStorageService.GetBybitApiSecretAsync();

   if (string.IsNullOrWhiteSpace(apiKey) ||
      string.IsNullOrWhiteSpace(apiSecret))
   {
      return null;
   }

   return new BybitRestClient(options =>
   {
      options.ApiCredentials = new BybitCredentials(apiKey, apiSecret);
   });
}
```

ДОДАТОК Е

Фрагмент логіки сторінки налаштувань

```
private async void OnConnectButtonClicked(object? sender, EventArgs e)
{
    if (_isBybitConnected)
    {
        _apiKeyStorageService.RemoveBybitKeys();

        ApiKeyEntry.Text = "";
        ApiSecretEntry.Text = "";

        _isBybitConnected = false;
        _isBybitFormExpanded = false;

        UpdateBybitUi();

        await DisplayAlertAsync("Готово", "Bybit від'єднано.", "OK");
        return;
    }

    var apiKey = ApiKeyEntry.Text?.Trim();
    var apiSecret = ApiSecretEntry.Text?.Trim();

    if (string.IsNullOrWhiteSpace(apiKey) ||
        string.IsNullOrWhiteSpace(apiSecret))
    {
        await DisplayAlertAsync("Помилка", "Введіть API Key та API Secret.", "OK");
        return;
    }

    ConnectButton.IsEnabled = false;
    ConnectButton.Text = "Перевірка...";

    await _apiKeyStorageService.SaveBybitKeysAsync(apiKey, apiSecret);

    var isConnected = await TestBybitConnectionAsync();

    ConnectButton.IsEnabled = true;

    if (!isConnected)
    {
        _apiKeyStorageService.RemoveBybitKeys();

        _isBybitConnected = false;
        UpdateBybitUi();
        return;
    }

    _isBybitConnected = true;
    _isBybitFormExpanded = false;

    UpdateBybitUi();

    await DisplayAlertAsync("Готово", "Bybit успішно підключено.", "OK");
}


private async Task<bool> TestBybitConnectionAsync()
{
```

```
    try
    {
        var isConnected = await _exchangeService.ConnectAsync();

        if (!isConnected)
        {
            await DisplayAlertAsync(
                "Помилка підключення",
                "Не вдалося підключитися до Bybit. Перевірте API Key, API Secret та дозволи API.",
                "OK");

            return false;
        }

        return true;
    }
    catch (Exception ex)
    {
        await DisplayAlertAsync("Помилка підключення", ex.Message, "OK");
        return false;
    }
}


private void UpdateBybitUi()
{
    BybitFormLayout.IsVisible = _isBybitFormExpanded;
    ExpandIconLabel.Text = _isBybitFormExpanded ? "^" : "v";

    if (_isBybitConnected)
    {
        ConnectionStatusLabel.Text = "Підключено";
        ConnectionStatusLabel.TextColor = Color.FromArgb("#00E676");

        ConnectButton.Text = "Від'єднати";
        ConnectButton.BackgroundColor = Color.FromArgb("#3A080C");
        ConnectButton.TextColor = Color.FromArgb("#FF3040");
    }
    else
    {
        ConnectionStatusLabel.Text = "Не підключено";
        ConnectionStatusLabel.TextColor = Color.FromArgb("#FF3040");

        ConnectButton.Text = "Підключити";
        ConnectButton.BackgroundColor = Color.FromArgb("#1E6BFF");
        ConnectButton.TextColor = Colors.White;
    }
}

private async void OnClearDatabaseClicked(object? sender, EventArgs e)
{
    var confirm = await DisplayAlertAsync(
        "Очищення локальних записів",
        "Видалити всі угоди з локальної бази даних?",
        "Так",
        "Ні");

    if (!confirm)
    {
        return;
    }

    await _tradeRepository.ClearAsync();

    await DisplayAlertAsync("Готово", "Локальні записи очищено.", "OK");
```

```
}
```

ДОДАТОК Ж

Фрагмент імпорту та перетворення угод Bybit

```
public async Task<List<TradeRecord>> GetClosedTradesAsync()
{
    using var client = await CreateClientAsync();

    if (client == null)
    {
        return [];
    }

    var allTrades = new List<TradeRecord>();

    var finalEndDate = DateTime.UtcNow;
    var currentStartDate = finalEndDate.AddDays(-30);

    while (currentStartDate < finalEndDate)
    {
        var currentEndDate = currentStartDate.AddDays(7);

        if (currentEndDate > finalEndDate)
        {
            currentEndDate = finalEndDate;
        }

        var result = await client.V5Api.Trading.GetClosedProfitLossAsync(
            category: Category.Linear,
            startTime: currentStartDate,
            endTime: currentEndDate,
            limit: 100);

        if (result.Success && result.Data?.List != null)
        {
            foreach (var item in result.Data.List)
            {
                var pnl = ParseDecimal(item.ClosedPnl);
                var volumeUsd = ParseDecimal(item.EntryValue);
                var leverage = ParseDecimal(item.Leverage);

                var roi = volumeUsd == 0m || leverage == 0m
                    ? 0m
                    : pnl * leverage / volumeUsd * 100m;

                var openFee = ParseDecimal(item.OpenFee);
                var closeFee = ParseDecimal(item.CloseFee);
                var openedAt = item.CreateTime;
                var closedAt = item.UpdateTime;
                var durationMinutes = CalculateDurationMinutes(openedAt, closedAt);

                allTrades.Add(new TradeRecord
                {
                    OrderId = item.OrderId,
                    Exchange = "Bybit",
                    Symbol = item.Symbol,
```

```
                Direction = GetDirection(item.Side.ToString()),

                Pnl = pnl,
                Roi = roi,
                VolumeUsd = volumeUsd,

                EntryPrice = ParseDecimal(item.AverageEntryPrice),
                ExitPrice = ParseDecimal(item.AverageExitPrice),
                TotalFees = openFee + closeFee,
                Leverage = leverage,

                OpenedAt = openedAt,
                ClosedAt = closedAt,
                DurationMinutes = durationMinutes,

                IsManual = false,
                Comment = ""
            });
        }
    }

    currentStartDate = currentEndDate;
  }

  return allTrades
    .GroupBy(x => x.OrderId)
    .Select(x => x.First())
    .OrderByDescending(x => x.ClosedAt ?? x.OpenedAt ?? DateTime.MinValue)
    .ToList();
}


  private static string GetDirection(string side)
  {
    return side.Equals("Buy", StringComparison.OrdinalIgnoreCase)
      ? "Long"
      : side.Equals("Sell", StringComparison.OrdinalIgnoreCase)
        ? "Short"
        : side;
  }

  private static decimal ParseDecimal(object? value)
  {
    if (value == null)
    {
      return 0m;
    }

    if (value is decimal decimalValue)
    {
      return decimalValue;
    }

    if (decimal.TryParse(
        value.ToString(),
        NumberStyles.Float,
        CultureInfo.InvariantCulture,
        out var result))
    {
      return result;
    }

    return 0m;
  }
```

```
    private static int? CalculateDurationMinutes(DateTime? openedAt, DateTime? closedAt)
    {
        if (!openedAt.HasValue || !closedAt.HasValue || closedAt.Value < openedAt.Value)
        {
            return null;
        }

        return (int)Math.Round((closedAt.Value - openedAt.Value).TotalMinutes);
    }
}
```

ДОДАТОК И

Сервіс синхронізації журналу угод

```
public class JournalSyncService : IJournalSyncService
{
    private readonly IExchangeService _exchangeService;
    private readonly ITradeRepository _tradeRepository;

    public JournalSyncService(
        IExchangeService exchangeService,
        ITradeRepository tradeRepository)
    {
        _exchangeService = exchangeService;
        _tradeRepository = tradeRepository;
    }

    public async Task<List<TradeRecord>> SyncAsync()
    {
        var apiTrades = await _exchangeService.GetClosedTradesAsync();

        if (apiTrades.Any())
        {
            await _tradeRepository.SaveTradesAsync(apiTrades);
        }

        return await _tradeRepository.GetAllAsync();
    }

    public Task<List<TradeRecord>> GetLocalTradesAsync()
    {
        return _tradeRepository.GetAllAsync();
    }
}
```

ДОДАТОК К

Фрагмент ручного додавання торгової угоди

```
private async void OnSaveClicked(object? sender, EventArgs e)
{
    if (string.IsNullOrWhiteSpace(SymbolEntry.Text))
    {
        await DisplayAlertAsync("Помилка", "Введіть торгову пару.", "OK");
        return;
    }

    if (string.IsNullOrWhiteSpace(ExchangeEntry.Text))
    {
        await DisplayAlertAsync("Помилка", "Введіть назву біржі.", "OK");
        return;
    }

    var openedDate = OpenedDatePicker.Date!.Value;
    var closedDate = ClosedDatePicker.Date!.Value;

    var openedTime = OpenedTimePicker.Time!.Value;
    var closedTime = ClosedTimePicker.Time!.Value;

    var openedAt = openedDate.Date + openedTime;
    var closedAt = closedDate.Date + closedTime;

    var durationMinutes = CalculateDurationMinutes(openedAt, closedAt);

    if (closedAt < openedAt)
    {
        await DisplayAlertAsync("Помилка", "Час закриття не може бути раніше часу відкриття.", "OK");
        return;
    }

    var volumeUsd = ParseDecimal(VolumeEntry.Text);
    var pnl = ParseDecimal(PnlEntry.Text);
    var leverage = ParseDecimal(LeverageEntry.Text);

    var roi = volumeUsd == 0m || leverage == 0m
        ? 0m
        : pnl * leverage / volumeUsd * 100m;

    var trade = new TradeRecord
    {
        OrderId = Guid.NewGuid().ToString(),
        Exchange = ExchangeEntry.Text?.Trim() ?? "",
        Symbol = SymbolEntry.Text.Trim().ToUpperInvariant(),
        Direction = _direction,

        OpenedAt = openedAt,
        ClosedAt = closedAt,

        DurationMinutes = durationMinutes,

        EntryPrice = ParseDecimal(EntryPriceEntry.Text),
        ExitPrice = ParseDecimal(ExitPriceEntry.Text),
        Leverage = leverage,
        VolumeUsd = volumeUsd,

        Pnl = pnl,
```

```
        Roi = roi,
        TotalFees = ParseDecimal(FeesEntry.Text),


        Comment = CommentEditor.Text ?? "",
        IsManual = true
    };

    await _tradeRepository.AddAsync(trade);

    await DisplayAlertAsync("Готово", "Угоду збережено.", "OK");
    await Shell.Current.GoToAsync("..");
}


private decimal ParseDecimal(string? value)
{
    if (decimal.TryParse(value, NumberStyles.Float, CultureInfo.InvariantCulture, out var result))
    {
        return result;
    }

    if (decimal.TryParse(value, NumberStyles.Float, new CultureInfo("uk-UA"), out result))
    {
        return result;
    }

    return 0m;
}


private int? CalculateDurationMinutes(DateTime? openedAt, DateTime? closedAt)
{
    if (!openedAt.HasValue || !closedAt.HasValue || closedAt.Value < openedAt.Value)
    {
        return null;
    }

    return (int)Math.Round((closedAt.Value - openedAt.Value).TotalMinutes);
}
```

ДОДАТОК Л

Фрагмент формування статистики

```
private async Task LoadStatisticsAsync()
{
    var fromDate = DateTime.Now.AddDays(-_selectedDays);
    var trades = await _tradeRepository.GetFromDateAsync(fromDate);

    UpdatePeriodButtons();
    RenderStatistics(trades);
}

private void RenderStatistics(List<TradeRecord> trades)
{
    var totalTrades = trades.Count;

    decimal netPnl = 0m;
    decimal winRate = 0m;
    decimal longPercent = 0m;
    decimal shortPercent = 0m;
    decimal longWinRate = 0m;
    decimal shortWinRate = 0m;
    decimal longPnl = 0m;
    decimal shortPnl = 0m;
    decimal grossProfit = 0m;
    decimal grossLoss = 0m;
    decimal totalFees = 0m;
    decimal largestWin = 0m;
    decimal largestLoss = 0m;

    if (totalTrades > 0)
    {
        var longTrades = trades
            .Where(x => x.Direction.Equals("Long", StringComparison.OrdinalIgnoreCase))
            .ToList();

        var shortTrades = trades
            .Where(x => x.Direction.Equals("Short", StringComparison.OrdinalIgnoreCase))
            .ToList();

        var profitTrades = trades.Where(x => x.Pnl > 0).ToList();
        var lossTrades = trades.Where(x => x.Pnl < 0).ToList();

        var winningTrades = profitTrades.Count;

        netPnl = trades.Sum(x => x.Pnl);
        winRate = (decimal)winningTrades / totalTrades * 100m;
        longPercent = (decimal)longTrades.Count / totalTrades * 100m;
        shortPercent = (decimal)shortTrades.Count / totalTrades * 100m;

        longWinRate = longTrades.Count == 0
            ? 0m
            : (decimal)longTrades.Count(x => x.Pnl > 0) / longTrades.Count * 100m;

        shortWinRate = shortTrades.Count == 0
            ? 0m
            : (decimal)shortTrades.Count(x => x.Pnl > 0) / shortTrades.Count * 100m;

        longPnl = longTrades.Sum(x => x.Pnl);
        shortPnl = shortTrades.Sum(x => x.Pnl);
```

```
        grossProfit = profitTrades.Sum(x => x.Pnl);
        grossLoss = lossTrades.Sum(x => x.Pnl);
        totalFees = trades.Sum(x => x.TotalFees);

        largestWin = profitTrades.Select(x => x.Pnl).DefaultIfEmpty(0m).Max();
        largestLoss = lossTrades.Select(x => x.Pnl).DefaultIfEmpty(0m).Min();
    }

    NetPnlLabel.Text = FormatMoney(netPnl);
    NetPnlLabel.TextColor = GetPnlColor(netPnl);

    PeriodInfoLabel.Text = $"Останні {_selectedDays} днів";

    TradesCountLabel.Text = totalTrades.ToString(CultureInfo.InvariantCulture);
    WinRateLabel.Text = FormatPercent(winRate);
    LongShortLabel.Text = $"{longPercent:0}% / {shortPercent:0}%";

    FeesLabel.Text = FormatMoneyWithoutSign(totalFees);

    GrossProfitLabel.Text = FormatMoney(grossProfit);
    GrossProfitLabel.TextColor = GetPnlColor(grossProfit);

    GrossLossLabel.Text = FormatMoney(grossLoss);
    GrossLossLabel.TextColor = GetPnlColor(grossLoss);

    LongPnlLabel.Text = FormatMoney(longPnl);
    LongPnlLabel.TextColor = GetPnlColor(longPnl);

    ShortPnlLabel.Text = FormatMoney(shortPnl);
    ShortPnlLabel.TextColor = GetPnlColor(shortPnl);

    LongWinRateLabel.Text = FormatPercent(longWinRate);
    ShortWinRateLabel.Text = FormatPercent(shortWinRate);

    LargestWinLabel.Text = FormatMoney(largestWin);
    LargestWinLabel.TextColor = GetPnlColor(largestWin);

    LargestLossLabel.Text = FormatMoney(largestLoss);
    LargestLossLabel.TextColor = GetPnlColor(largestLoss);
}
```

ДОДАТОК М

Фрагмент реалізації торгового калькулятора

```
public partial class CalculatorPage : ContentPage
{
    public CalculatorPage()
    {
        InitializeComponent();
    }

    private async void OnCalculateClicked(object? sender, EventArgs e)
    {
        var balance = ParseDecimal(BalanceEntry.Text);
        var riskPercent = ParseDecimal(RiskEntry.Text);
        var entryPrice = ParseDecimal(EntryPriceEntry.Text);
        var stopLossPrice = ParseDecimal(StopLossEntry.Text);
        var leverage = ParseDecimal(LeverageEntry.Text);

        if (balance <= 0 || riskPercent <= 0 || entryPrice <= 0 || stopLossPrice <= 0 || leverage <= 0)
        {
            await DisplayAlertAsync("Помилка", "Заповніть усі поля коректними значеннями.", "OK");
            return;
        }

        if (entryPrice == stopLossPrice)
        {
            await DisplayAlertAsync("Помилка", "Ціна стоп-лосу має відрізнятися від ціни входу.", "OK");
            return;
        }

        var riskAmount = balance * riskPercent / 100m;

        var priceDifference = Math.Abs(entryPrice - stopLossPrice);
        var priceRiskPercent = priceDifference / entryPrice;

        var volumeUsd = riskAmount / priceRiskPercent;
        var margin = volumeUsd / leverage;
        var quantity = volumeUsd / entryPrice;

        VolumeLabel.Text = FormatMoney(volumeUsd);
        MarginLabel.Text = FormatMoney(margin);
        LossLabel.Text = FormatMoney(riskAmount);
        QuantityLabel.Text = quantity.ToString("0.########", CultureInfo.InvariantCulture);

        ResultCard.IsVisible = true;
    }

    private decimal ParseDecimal(string? value)
    {
        if (decimal.TryParse(value, NumberStyles.Float, CultureInfo.InvariantCulture, out var result))
        {
            return result;
        }

        if (decimal.TryParse(value, NumberStyles.Float, new CultureInfo("uk-UA"), out result))
        {
            return result;
        }

        return 0m;
```

```
    }

    private string FormatMoney(decimal value)
    {
        return value.ToString("$0.##", CultureInfo.InvariantCulture);
    }
}
```

ДОДАТОК Н

Приклади модульних тестів системи

```
public async Task AddAsync_ShouldSaveTrade()
{
    // Arrange
    var repository = CreateRepository(out var database, out var databasePath);

    var trade = new TradeRecord
    {
        OrderId = "order-1",
        Exchange = "Bybit",
        Symbol = "BTCUSDT",
        Direction = "Long",
        Pnl = 100m,
        Roi = 10m,
        ClosedAt = new DateTime(2026, 5, 1, 12, 0, 0)
    };

    // Act
    await repository.AddAsync(trade);
    var trades = await repository.GetAllAsync();

    // Assert
    Assert.Single(trades);
    Assert.Equal("order-1", trades[0].OrderId);

    await database.CloseAsync();
    File.Delete(databasePath);
}

[Fact]
public async Task GetAllAsync_ShouldReturnTradesNewestFirst()
{
    // Arrange
    var repository = CreateRepository(out var database, out var databasePath);

    await repository.AddAsync(new TradeRecord
    {
        OrderId = "old-order",
        Symbol = "ETHUSDT",
        ClosedAt = new DateTime(2026, 5, 1)
    });

    await repository.AddAsync(new TradeRecord
    {
        OrderId = "new-order",
        Symbol = "BTCUSDT",
        ClosedAt = new DateTime(2026, 5, 10)
    });

    // Act
    var trades = await repository.GetAllAsync();

    // Assert
    Assert.Equal(2, trades.Count);
    Assert.Equal("new-order", trades[0].OrderId);
    Assert.Equal("old-order", trades[1].OrderId);

    await database.CloseAsync();
```

```
    File.Delete(databasePath);
}



[Fact]
public async Task ExistsAsync_ShouldReturnTrue_WhenTradeExists()
{
    // Arrange
    var repository = CreateRepository(out var database, out var databasePath);

    await repository.AddAsync(new TradeRecord
    {
        OrderId = "existing-order",
        Symbol = "BTCUSDT"
    });

    // Act
    var exists = await repository.ExistsAsync("existing-order");

    // Assert
    Assert.True(exists);

    await database.CloseAsync();
    File.Delete(databasePath);
}


public async Task UpdateCommentAsync_ShouldChangeTradeComment()
{
    var repository = CreateRepository(out var database, out var databasePath);

    await repository.AddAsync(new TradeRecord
    {
        OrderId = "comment-order",
        Symbol = "BTCUSDT",
        Comment = ""
    });

    await repository.UpdateCommentAsync("comment-order", "Test comment");

    var trades = await repository.GetAllAsync();

    Assert.Single(trades);
    Assert.Equal("Test comment", trades[0].Comment);

    await database.CloseAsync();
    File.Delete(databasePath);
}


public async Task ClearAsync_ShouldRemoveAllTrades()
{
    var repository = CreateRepository(out var database, out var databasePath);

    await repository.AddAsync(new TradeRecord { OrderId = "order-1", Symbol = "BTCUSDT" });
    await repository.AddAsync(new TradeRecord { OrderId = "order-2", Symbol = "ETHUSDT" });

    await repository.ClearAsync();

    var trades = await repository.GetAllAsync();

    Assert.Empty(trades);
```

```
    await database.CloseAsync();
    File.Delete(databasePath);
}
```

ДОДАТОК П

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

Декларація про академічну доброчесність та використання ШІ

ДЕКЛАРАЦІЯ ЗДОБУВАЧА

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

1. Моя бакалаврська кваліфікаційна робота (проєкт) на тему «Програмне забезпечення для обліку торгівельної діяльності трейдера» 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 генеративного ШІ не використовувалися.
    - [✓] інструменти ШІ (вказати назву: ChatGPT, Gemini, Claude, DeepL, _____інші (вказати назву)) були використані для:
        - [ ] перекладу іншомовних джерел;
        - [✓] стилістичного редагування та перевірки граматики;
        - [✓] допомоги у написанні/відлагодженні програмного коду;
        - [ ] інше: _____.

Я розумію, що надання недостовірної інформації може призвести до

скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

«21» Травня 2026 р.  **Максим Авдієвський**

(підпис)