НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ПОГОДЖЕНО **Декан факультету**	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ** **Завідувач кафедри комп'ютерних систем, мереж та кібербезпеки**
_____Ігор БОЛБОТ	**_____Дмитро КАСАТКІН**
(підпис)	(підпис)
«___» _____2026 р.	«___» _____2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка мобільного застосунку для моніторингу та аналітики курсів криптовалют»

Спеціальність «122 Комп'ютерні науки»

Освітньо-професійна програма «Комп'ютерні науки»

Гарант освітньої програми
д.е.н., професор _____ **Роман РУДЕНСЬКИЙ**
(підпис)

Керівник бакалаврської кваліфікаційної роботи _____ **Володимир НАЗАРЕНКО**
(підпис)

Виконав _____ **Даниїл КАРЮК**
(підпис)

Київ-2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних систем, мереж та кібербезпеки
к.е.н., доцент _____Дмитро КАСАТКІН
(підпис)

«___» _____2026 р.

ЗАВДАННЯ
ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
ЗДОБУВАЧУ
Карюку Даниїлу Юрійовичу
(прізвище, ім'я, по батькові)

Спеціальність «122 Комп'ютерні науки»
Освітньо-професійна програма «Комп'ютерні науки» Тема бакалаврської кваліфікаційної роботи
«Розробка мобільного застосунку для моніторингу та аналітики курсів криптовалют»
затверджена наказом від «06» лютого 2026 р. № 46«С» Термін подання завершеної роботи на кафедру «25» _____05___2026 р.
Вихідні дані до бакалаврської кваліфікаційної роботи: дані про курси криптовалют, обсяги торгів, історія зміни цін та користувацькі налаштування.

Перелік питань, що підлягають дослідженню:

1. Аналіз особливостей моніторингу та аналітики курсів криптовалют у мобільних застосунках.
2. Розробка структури та функціональних модулів мобільного застосунку для платформи iOS.
3. Реалізація механізмів отримання, збереження та обробки ринкових даних криптовалют.
4. Створення засобів візуалізації, аналітики та персоналізації роботи користувача з ринковими даними.

Дата видачі завдання «03» лютого 2026 р.

Керівник бакалаврської кваліфікаційної роботи _____ **Володимир НАЗАРЕНКО**
(підпис)

Завдання взяв до виконання _____ **Даниїл КАРЮК**
(підпис)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

API (Application Programming Interface) – програмний інтерфейс застосунку

iOS – операційна система для мобільних пристроїв Apple

UI (User Interface) – інтерфейс користувача

JSON (JavaScript Object Notation) – формат обміну даними

Core Data – технологія локального збереження та керування даними в iOS

Binance API – програмний інтерфейс криптобіржі Binance

SwiftUI – фреймворк для побудови інтерфейсу користувача мовою Swift

URL (Uniform Resource Locator) – уніфікований локатор ресурсу

SQLite – реляційна система керування базами даних вбудованого типу

CRUD (Create, Read, Update, Delete) – базові операції обробки даних

ВСТУП

В умовах розвитку IT криптовалюти стали значною частиною фінансового середовища. Їх ринок характеризується високою волатильністю, швидкою зміною вартості активів, значними коливаннями торгових обсягів і потребою в оперативному отриманні актуальної інформації. У зв’язку з цим зростає потреба у програмних засобах, які дають змогу швидко переглядати ринкові дані, аналізувати зміну цін, формувати власні списки спостереження та зберігати важливу інформацію для подальшого використання [15].

Особливо актуальним є створення мобільного застосунку, оскільки смартфон є зручним інструментом для доступу до фінансової інформації в реальному часі. Мобільний формат дає змогу користувачу швидко отримувати дані про стан ринку, переглядати графіки, працювати з обраними активами, створювати нотатки та цінові сповіщення [16]. Тому розробка застосунку для моніторингу та аналітики курсів криптовалют є актуальним завданням.

Мета бакалаврської кваліфікаційної роботи полягає у розробці мобільного застосунку для моніторингу та аналітики курсів криптовалют на платформі iOS, який забезпечує отримання актуальних ринкових даних, їх візуалізацію, локальне збереження, пошук активів, роботу з обраними монетами, нотатками, сповіщеннями та аналітичними звітами.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- проаналізувати предметну область моніторингу та аналітики курсів криптовалют;
- розглянути існуючі програмні рішення та визначити їх переваги й недоліки;
- сформувати функціональні та нефункціональні вимоги до мобільного застосунку;
- розробити інформаційну модель системи та визначити основні потоки даних;
- обґрунтувати вибір інструментальних засобів розробки;

- реалізувати основні програмні модулі застосунку;
- виконати тестування розробленої системи та оцінити результати її роботи.

Об’єктом дослідження є процеси отримання, обробки, збереження та візуалізації ринкових даних криптовалют у мобільному застосунку.

Предметом дослідження є мобільний застосунок для моніторингу та аналітики курсів криптовалют, розроблений для платформи iOS.

Для розв’язання поставлених завдань було використано методи системного аналізу, моделювання предметної області, проєктування структури даних, об’єктно-орієнтованого програмування, тестування програмного забезпечення та аналізу результатів роботи системи. Під час розробки застосовано мову програмування Swift [1], фреймворк SwiftUI [2], технологію Core Data [3], UserDefaults [4], UserNotifications [5], Charts [6], формат JSON та Binance API як джерело ринкових даних [7]. Також під час підготовки роботи було враховано приклади кваліфікаційних робіт із цифрової бібліотеки НУБіП України [11–13].

Практичне значення роботи полягає у створенні мобільного застосунку, який можна використовувати для зручного моніторингу крипторинку, перегляду актуальних курсів, аналізу історичних даних, формування списку обраних монет, створення нотаток, цінових сповіщень та експорту аналітичних результатів у форматі JSON.

Апробацію результатів роботи здійснено шляхом компіляції та тестового запуску мобільного застосунку в середовищі Xcode [17]. Було перевірено основні функціональні можливості системи: реєстрацію та авторизацію користувача, отримання ринкових даних, пошук криптовалют, побудову графіків, роботу з обраними монетами, нотатками, ціновими сповіщеннями та аналітичним модулем.

Бакалаврська кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел і додатків. У першому розділі проведено системний аналіз предметної області та сформовано вимоги до

системи. У другому розділі розглянуто інформаційне забезпечення, інформаційні потоки та структуру даних. У третьому розділі описано архітектуру, інструментальні засоби та реалізацію основних програмних модулів. У четвертому розділі наведено рекомендації щодо впровадження, експлуатації та тестування застосунку. Робота містить 85 сторінки, 28 використаних джерел і 3 додатки.

1. СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Постановка задачі

Основним завданням є розробка мобільного застосунку на платформі iOS для моніторингу та аналітики курсів криптовалют. У сучасних умовах розвитку цифрових та фінансових технологій користувачам важливо мати зручний та швидкий доступ до актуальної ситуації на ринку криптовалют, інструменти для аналізу даних. Крипторинок має високу волатильність, постійні зміни цін на активи та значні коливання торгових обсягів, тому своєчасне отримання інформації є необхідним для розуміння стану ринку.

Для зручності роботи з ринком було розроблено мобільний застосунок для моніторингу та аналітики курсів криптовалют. Застосунок забезпечує перегляд актуальних курсів, аналіз зміни цін з візуалізацією та інструменти для аналітики.

Можна виділити такі основні переваги:

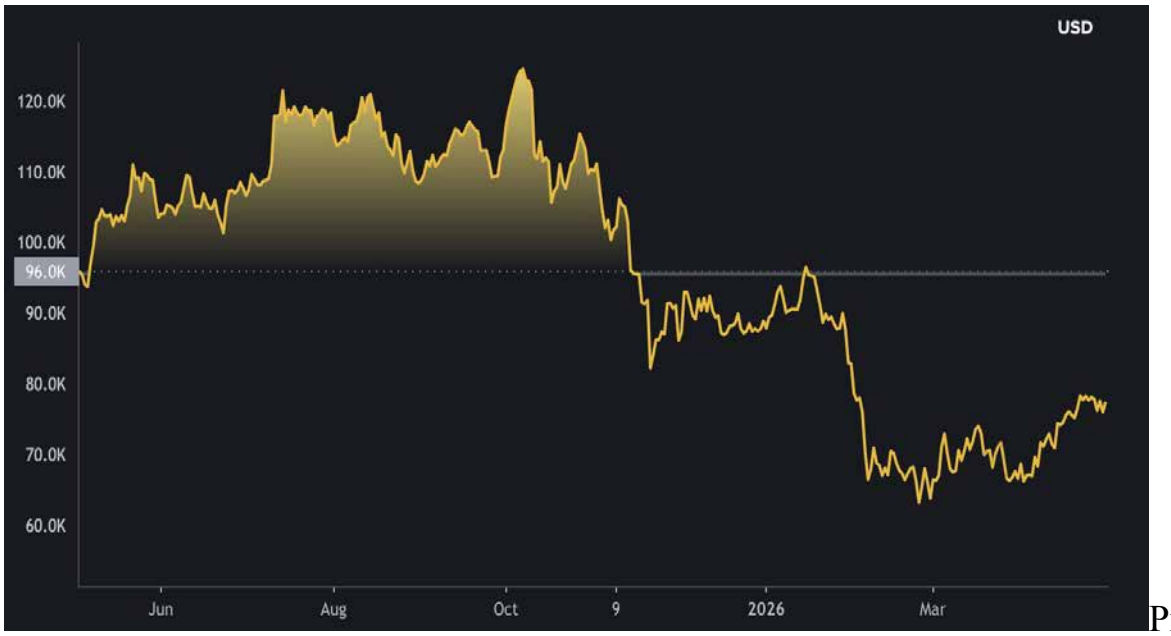
- відображення актуальних курсів криптовалют;
- пошук монет здійснюється не тільки за назвою, але й за символом або торговою парою;
- візуалізація за допомогою графіків для відображення зміни ціни та обсягів торгів;
- можливість створити список обраних монет та створення текстових нотаток для кожного активу;
- налаштування цінових сповіщень;
- перегляд аналітичної складової та можливість порівняння;
- експорт аналітичного звіту для подальшої аналітики.

Застосунок передбачає організовану роботу з ринковими даними, їх локальне збереження та використання аналітичних засобів для підвищення ефективності моніторингу криптовалютного ринку.

1.2. Аналіз предметної області

Крипторинок є складовою цифрового економічного середовища. Поширення технології Blockchain, зростання інтересу до цифрових грошей та використання криптовалют як альтернативного фінансового інструменту зумовлює його розвиток і поширення. Внаслідок цього виникає потреба в застосунку, який забезпечуватиме швидке отримання, обробку та аналіз даних ринку.

Головною особливістю ринку є безперервний режим роботи. Торгівля криптовалютами відбувається цілодобово та без вихідних, на відміну від звичайних фінансових ринків. Ринок має високу волатильність, тобто значні зміни у вартості активів за короткий проміжок часу. Тому можна вважати, що користувачі повинні мати швидкий доступ до актуальної інформації про зміни цін, поточний курс, обсяги та історичні ціни.



Ри

с. 1.1 Приклад динаміки зміни курсу криптовалюти Bitcoin

Предметна область охоплює різні процеси такі як отримання, обробка, збереження, візуалізація та аналіз ринкової інформації. Система працює з основними даними, якими є назва, символ, поточна ціна, зміна за певний період,

мінімальна та максимальна ціни, обсяг торгів і історія зміни вартості. Ці характеристики дають змогу формувати загальне представлення про стан окремої криптовалюти в цілому.

Користувачам важливий не тільки перегляд поточних курсів, а й можливість аналітичної обробки інформації. Основними функціями такої системи є пошук, сортування та фільтрація, перегляд детальної інформації про монету та створення цінових сповіщень з нотатками.

Аналітичний модуль у межах предметної області також має важливе значення. Він дає змогу формувати огляд ринку, бачити лідерів за зростанням, найбільшими обсягами торгів та порівнювати монети між собою. Це дає змогу користувачеві швидко орієнтуватися на ринку та ефективно працювати з великою кількістю інформації.

Отже, аналіз предметної області показав, що моніторинг крипторинку є складною інформаційною задачею, яка потребує сучасних засобів для оперативної роботи з активами. Це є підґрунтям доцільності для розробки мобільного застосунку, який поєднуватиме функції моніторингу, аналітики та персоналізації роботи з цифровими активами.

1.3. Огляд інформаційних джерел та існуючих програмних рішень

Важливим під час розробки є аналіз інформаційних джерел, з яких система отримує дані про ринок. Ще важливим у сфері є вивчення існуючих програмних рішень. Цей огляд дозволить визначити, які функції найбільш потрібні та які підходи до подання інформації зручними для користувача. Також потрібно врахувати недоліки під час створення власного застосунку.

Системи моніторингу крипторинку використовують криптобіржі, аналітичні сервіси та програмні інтерфейси доступу до даних як джерела інформації. Вони надають відомості про зміни вартості, історичні дані для побудови графіків, поточні курси та максимальні та мінімальні значення ціни.

Для таких джерел можна відзначити такі важливі характеристики, як повнота ринкових показників і зручність подальшої програмної обробки.

У бакалаврській кваліфікаційній роботі як джерело даних використовується Binance API. Цей інтерфейс дає змогу отримати свіжі та актуальні дані у структурованому вигляді та просто інтегрується у мобільний застосунок. Висока швидкість оновлення інформації про ринок, доступ до поточних курсів, торгові обсяги та зміни вартості за добу є перевагами такого підходу [7].

Біржові програмні інтерфейси не є єдиним способом отримання інформації, існують спеціалізовані сервіси, що беруть інформацію з декількох торгових платформ і надають узагальнене уявлення про стан ринку. Це дозволяє оцінювати рейтинг активів, аналізувати зміни цін та популярність монет та використовувати додаткові аналітичні інструменти [8–10]. Використання біржового API є більш доцільним завдяки простій інтеграції та достатній кількості інформації, яку він надає адже для мобільного застосунку важливий швидкий моніторинг та персоналізовану взаємодію з ринком.

Існує три основні групи програмних рішень:

- біржові мобільні застосунки,
- універсальні криптовалютні трекери,
- сервіси ринкової аналітики.

Біржові застосунки дають доступ не лише до ринкових даних, а й до торгових операцій. Вони часто містять велику кількість додаткових функцій, тому є потужним інструментом, але можуть ускладнювати використання користувачам, яким потрібен тільки швидкий перегляд та базовий аналіз.

На відміну від біржових застосунків, універсальні криптовалютні трекери орієнтовані насамперед на моніторинг ринку. Вони забезпечують пошук активів та перегляд, сортування за ключовими показниками, створення списків з обраним і відображення історичних графіків. Для повсякденного використання це рішення зручне, але не завжди забезпечує необхідний рівень персоналізації або локального збереження даних.

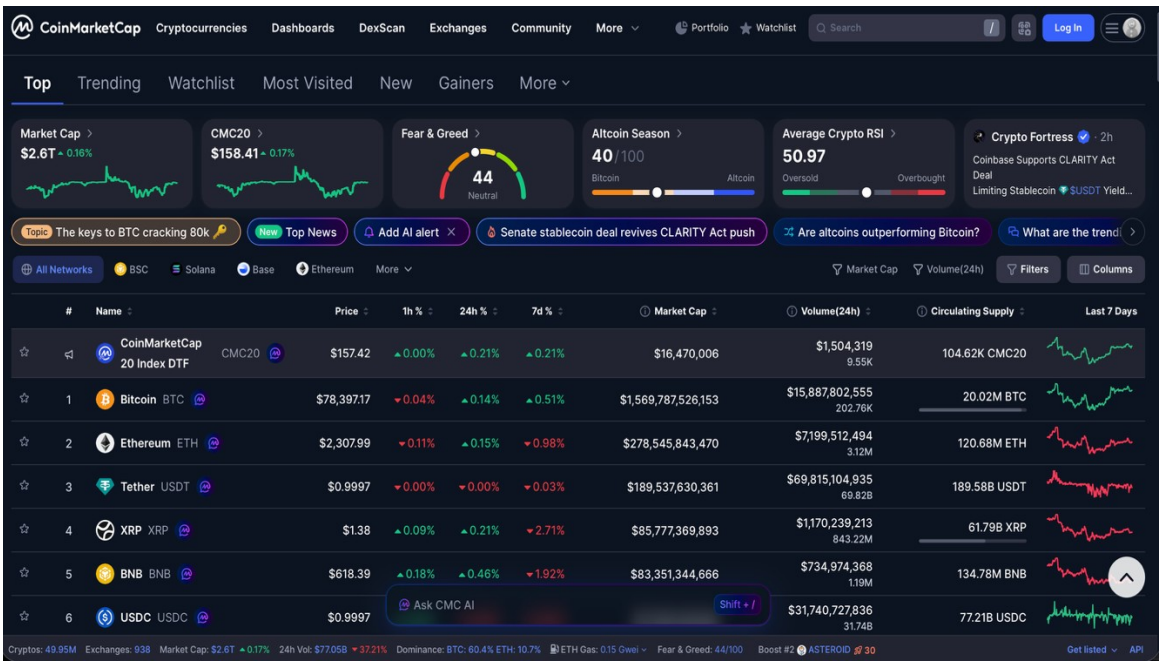


Рис. 1.2 Приклад інтерфейсу сервісу моніторингу криптовалютного ринку

Окремо можна виділити сервіси аналітики для побудови графіків на основі ринкових даних, порівняння активів і використання інструментів технічного аналізу. Їхні переваги – більш глибокі аналітичні можливості та засоби візуалізації. Але для частини користувачів таке рішення може бути складним, оскільки потребує більшої підготовки та орієнтовані на детальний ринковий аналіз.

Аналіз показав, що найбільш потрібними функціями є перегляд поточних курсів, пошук та сортування монет, побудова історичних графіків та налаштування списку обраних зі сповіщеннями. Далеко не всі системи поєднують ці функції з простим інтерфейсом і локальним збереженням даних. Це свідчить про доцільність створення застосунку, що поєднуватиме базові функції моніторингу з елементами аналітики.

Отже, огляд інформаційних джерел та існуючих рішень показав доцільність використання біржового API як основного джерела даних і підґрунтя для створення мобільного застосунку, що поєднуватиме зручний моніторинг, мати локальне збереження даних та аналітичні можливості.

1.4. Аналіз аналогів мобільних застосунків для моніторингу курсів криптовалют

Сучасний ринок насичений застосунками для моніторингу курсів криптовалют, перегляду інформації про ринок та виконання аналітики. Аналіз таких рішень є важливим етапом для розуміння затребуваних функцій, організації інтерфейсу та виявлення слабких та сильних сторін конкурентів.

Найбільш відомими застосунками цієї сфери є: Binance, CoinMarketCap, CoinGecko та TradingView. Вони виконують свої функції, але мають свої особливості використання.

Для порівняння функціональних можливостей мобільних застосунків їх основні характеристики подані у таблиці 1.1.

Таблиця 1.1

Порівняння аналогів мобільних застосунків

Назва застосунку	**Основне призначення**	**Переваги**	**Недоліки**
Binance	Моніторинг ринку та виконання торгових операцій	Актуальні біржові дані, велика кількість активів, детальна інформація про монети, доступ до історичних даних	Перевантажений інтерфейс, орієнтація не лише на моніторинг, а й на торгівлю
CoinMarketCap	Перегляд ринкових показників і загального стану криптовалютного ринку	Зручний перегляд списку активів, рейтинги монет, ринкова капіталізація, наочне подання даних	Обмежена персоналізація, менша гнучкість локальної роботи з даними

Таблиця 1.1 (закінчення)

Назва застосунку	**Основне призначення**	**Переваги**	**Недоліки**
CoinGecko	Моніторинг курсів криптовалют і базова аналітика	Простий інтерфейс, зручний доступ до цін, списків спостереження та змін ринку	Обмежені аналітичні можливості, відсутність повноцінної локальної організації користувацьких даних
TradingView	Графічний і технічний аналіз фінансових активів	Розвинені графіки, інструменти порівняння, індикатори технічного аналізу	Складніший інтерфейс, орієнтація на підготовленого користувача, надмірність для простого моніторингу

Binance поєднує в собі функції моніторингу крипторинку та біржової платформи. Він має доступ до актуальних курсів, широкі можливості роботи з монетами та детальні дані про кожний актив [7]. Також застосунок розрахований на виконання торгових операцій, що робить його інтерфейс складнішим для користувача, яким потрібен швидкий перегляд інформації.

CoinMarketCap – найбільш популярний сервіс для моніторингу ринку. Він надає інформацію про рейтинги криптовалют, дані про капіталізацію, зміни цін за 24 години, торгові обсяги та інші показники [8; 15]. Перевагою є зручний перегляд списку активів та великий обсяг інформації. Застосунок орієнтований

на огляд ринку, але не завжди може забезпечити потрібний рівень персоналізації для користувача.

CoinGecko також належить до універсальних криптотрекерів. Він надає доступ до перегляду поточних курсів, аналізу змін вартості та формування списків спостереження з базовою аналітикою. Його сильною стороною є простий інтерфейс і зручний доступ до базових показників [9; 14]. Але його функції для індивідуальної роботи з даними залишаються обмеженими.

TradingView відрізняється від попередніх застосунків, оскільки має акцент на графічно-технічному аналізі. Перевагами є інструменти для порівняння активів, засоби побудови графіків та аналітичні індикатори [10]. Мінусом є те, що він розрахований на більш підготовленого користувача і може бути складним для щоденного моніторингу.

Підсумовуючи аналіз, можна зазначити, що існуючі рішення забезпечують потрібні функції, але не завжди мають зрозумілий для користувача інтерфейс, локальне збереження даних, вміння працювати з текстовими нотатками та сповіщеннями. Саме це є обґрунтуванням доцільності розробки застосунку для організації зручного моніторингу, аналітики та локальної організації даних користувача в межах одного програмного рішення.

1.5. Формування функціональних та нефункціональних вимог до системи

Важливим етапом є формування вимог до системи, оскільки за рахунок визначених вимог будуть визначатись структура програми, межі функціональності та основні принципи реалізації. Для застосунку в вимогах потрібно враховувати особливості крипторинку, потреби користувача в отриманні, збереженні, аналізі та візуалізації інформації про ринок.

Функціональні вимоги визначають перелік дій і можливостей, які має забезпечувати система. До основних функціональних вимог мобільного застосунку можна віднести:

- реєстрація та авторизація користувача;
- отримання актуальних ринкових даних із зовнішнього джерела;
- відображення списку криптовалют із основними показниками;
- пошук монет за назвою, символом або торговою парою та сортування та фільтрація криптовалют за окремими параметрами;
- перегляд детальної інформації про вибраний актив та побудова графіків зміни ціни та обсягів торгів;
- формування списку обраних монет та створення та збереження текстових нотаток;
- налаштування цінових сповіщень та перегляд аналітичної інформації щодо стану ринку;
- порівняння окремих криптовалют між собою;
- збереження користувацьких налаштувань та перегляд локальних сутностей в адміністративному розділі.

Слід приділити увагу функціям, які пов’язані із роботою користувача. Система повинна мати можливість збереження активів, що становять найбільший інтерес, створення нотаток та налаштування сповіщень про зміну вартості для монет. Такі можливості дозволяють перетворити застосунок на інструмент повсякденної роботи з ринковими даними.

Нефункціональні вимоги визначають якісні характеристики системи та умови її використання. Для розроблюваного застосунку важливими є такі нефункціональні вимоги:

- зручність і зрозумілість користувацького інтерфейсу [16];
- достатня швидкодія під час відкриття та відображення ринкових даних;
- стабільність роботи застосунку та коректне відображення інформації на мобільних пристроях;
- можливість локального збереження даних користувача;
- цілісність і впорядкованість локальної інформації та модульність програмної архітектури;
- можливість подальшого розширення функціональних можливостей;

- надійність взаємодії із зовнішнім API [27];
- підтримка зручної навігації між основними екранами системи.

Оскільки користувач працює з великим обсягом інформації, вимога до зручності візуального подання є не менш важливою. Інтерфейс повинен мати логічне структурування, відображати основні показники та забезпечувати швидкий доступ до головних функцій. Особливе значення має зручне представлення графіків, списків активів, показників змін за певний період і аналітичних звітів.

Крім того, система повинна бути побудована таким чином, щоб забезпечувати можливість її подальшого розвитку. Застосунок має підтримувати розширений функціонал, можливість додавання нових джерел даних, нових аналітичних інструментів і вдосконалення користувацького інтерфейсу без повного перероблення основних компонентів.

Отже, сформовані функціональні і нефункціональні вимоги визначають основні характеристики майбутнього мобільного застосунку та створюють основу для подальшого проєктування інформаційного забезпечення, програмної архітектури та окремих функціональних модулів системи.

1.6. Моделювання предметної області

Важливим етапом розробки є моделювання предметної області, тому що воно дозволяє формувати її межі, функціональні можливості, основні процеси та взаємодію компонентів. Для мобільного застосунку аналітики курсів криптовалют використовується для представлення структури системи, основних сценаріїв роботи користувача, базової логіки обробки даних та взаємодії з API. У межах даного підрозділу застосовано контекстну модель, функціональну модель та UML-модель взаємодії користувача із системою.

1.6.1. Контекстна модель системи

Контекстна модель системи визначає її межі, основних учасників взаємодії та перелік базових функцій, які реалізуються в мобільному застосунку. У розроблюваній системі основними зовнішніми сутностями є користувач та API криптобіржі як зовнішнє джерело ринкових даних. Користувач взаємодіє із застосунком через основні сценарії використання, зокрема реєстрацію та авторизацію, перегляд ринку, пошук монет, роботу з локальними даними, перегляд аналітики та керування обраними монетами і сповіщеннями. Зовнішнє API забезпечує надходження актуальної інформації про криптовалютні активи, яка використовується системою для формування ринкових списків, детального перегляду монет і побудови аналітичних представлень.

Контекстна модель у вигляді діаграми прецедентів дає змогу визначити, які саме функції є ключовими для користувача та які зовнішні сутності беруть участь у роботі системи. Вона відображає систему як єдиний функціональний об’єкт, який забезпечує доступ до ринкових даних, підтримує збереження локальної інформації та надає користувачу засоби аналітичної взаємодії з криптовалютним ринком.

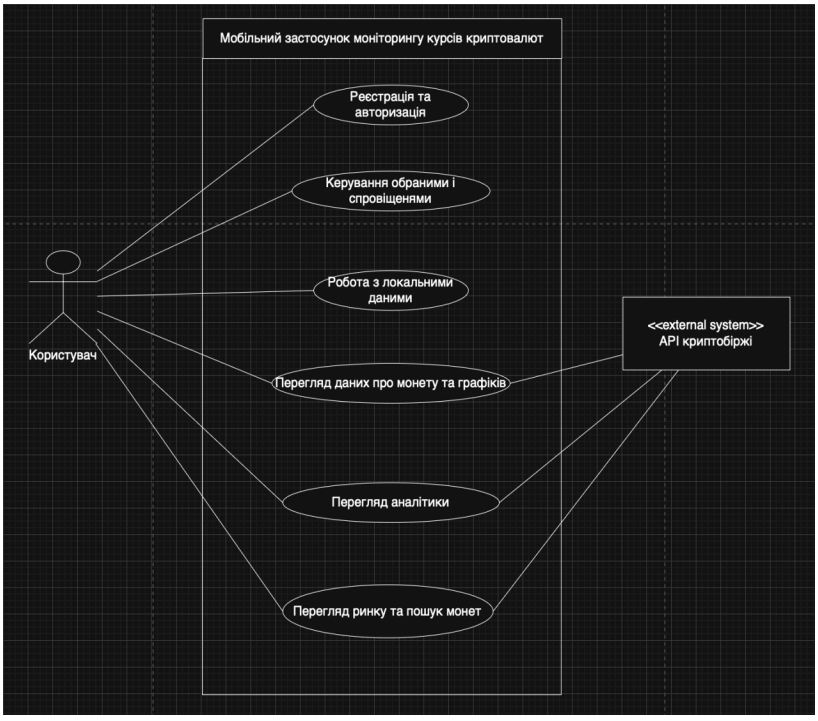


Рис. 1.3 Контекстна модель у вигляді діаграми прецедентів

Як видно користувач є основним учасником взаємодії із системою, коли API криптобіржі виступає джерелом ринкових даних. Діаграма також відображає основні функції застосунку, що визначають його функціональні межі та загальне призначення.

1.6.2. Функціональна модель системи

Функціональна модель показує послідовність дій, що виконуються коли користувач взаємодіє з системою. За допомогою неї можна побачити логіку функціонування системи від запуску до отримання користувачем ринкових даних, аналітики або виконання інших дій з монетами.

Функціональна модель застосунку показує етапи відкриття застосунку, робота з ринком, перевірка локальних даних та збереження їх, звернення до API, якщо це потрібно, а також виконання інших дій таких як додавання в обране та створення нотаток або сповіщення. Таким чином, функціональна модель дозволяє представити узагальнену логіку роботи системи та основні переходи між її станами.

Найбільш доцільним засобом подання такої моделі є діаграма діяльності, оскільки вона наочно відображає послідовність операцій, умови переходів та альтернативні сценарії поведінки системи залежно від наявності локальних даних або вибору користувача.

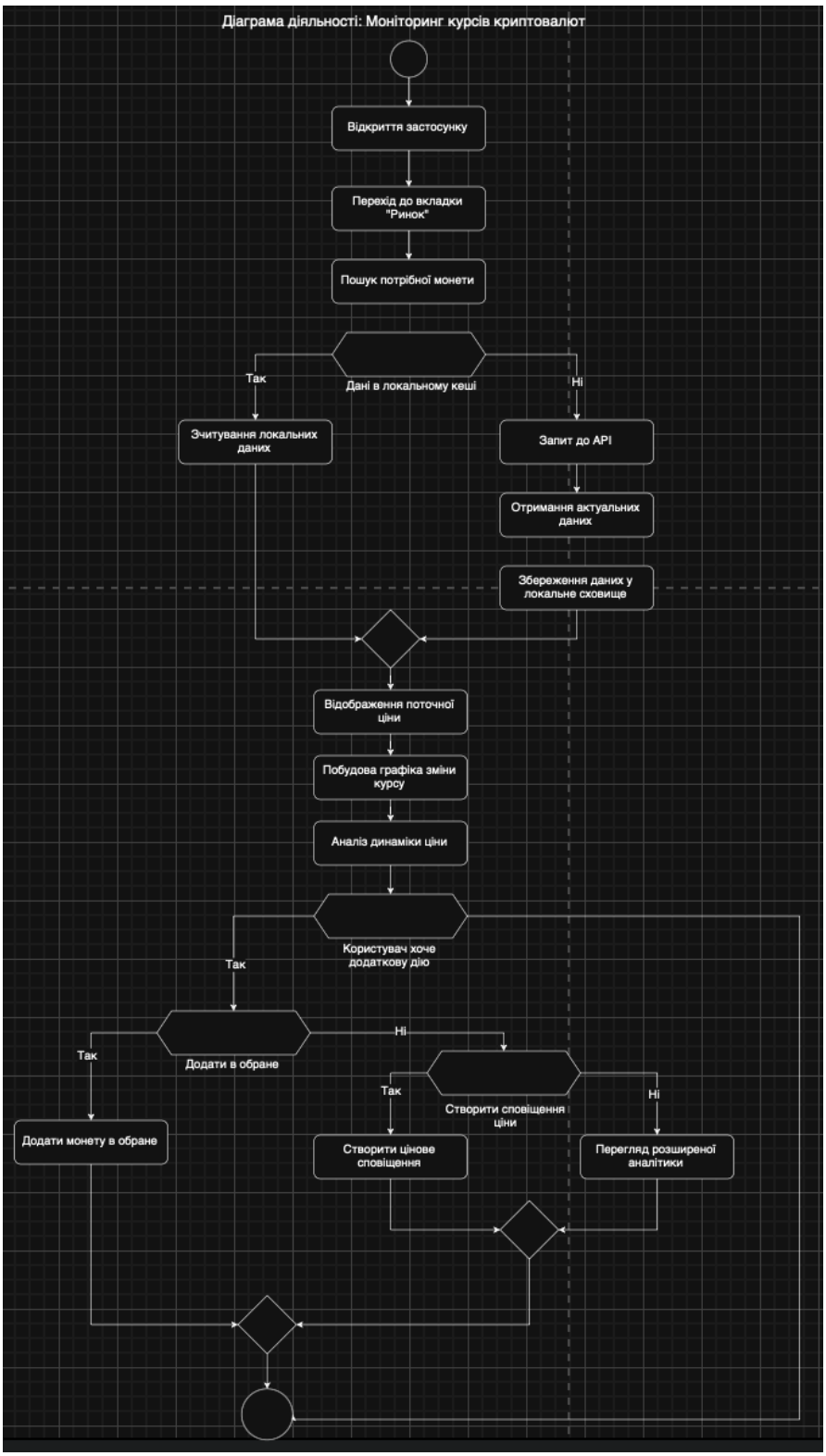


Рис. 1.4 Функціональна модель у вигляді діаграми діяльності

Як видно з рисунка 1.4, система підтримує як роботу з локально збереженою інформацією, так і отримання даних із зовнішнього API. Діаграма діяльності також показує, що після перегляду основної інформації користувач

може виконати додаткові дії, пов’язані з персоналізацією роботи із системою та переглядом аналітичних даних.

1.6.3. UML-моделі взаємодії користувача із системою

Представлення внутрішньої логіки того, як користувач взаємодіє з системою, зображене у вигляді діаграми послідовності. Ця модель дозволяє відобразити компоненти системи, які виконують певний сценарій або у якій послідовності передаються запити з результатами між ними.

У даній роботі діаграма показує отримання курсів криптовалют. За сценарієм користувач відкриває сторінку вибраної ним монети, потім сам застосунок формує запит до сервісного модуля. Модуль перевіряє, чи є дані в сховищі чи ні; якщо дані знаходяться, вони повертаються без звернення до самого API. Але якщо локальних даних немає, то виконується запит до Binance API, потім ця інформація зберігається в БД та передається до інтерфейсу для відображення.

Такий підхід збільшує швидкодію та зменшує навантаження на зовнішні сервіси, оптимізує використання локального кешу. Діаграма послідовності відображає ролі користувача, застосунку, сервісного модуля, самого API та БД у межах одного сценарію.

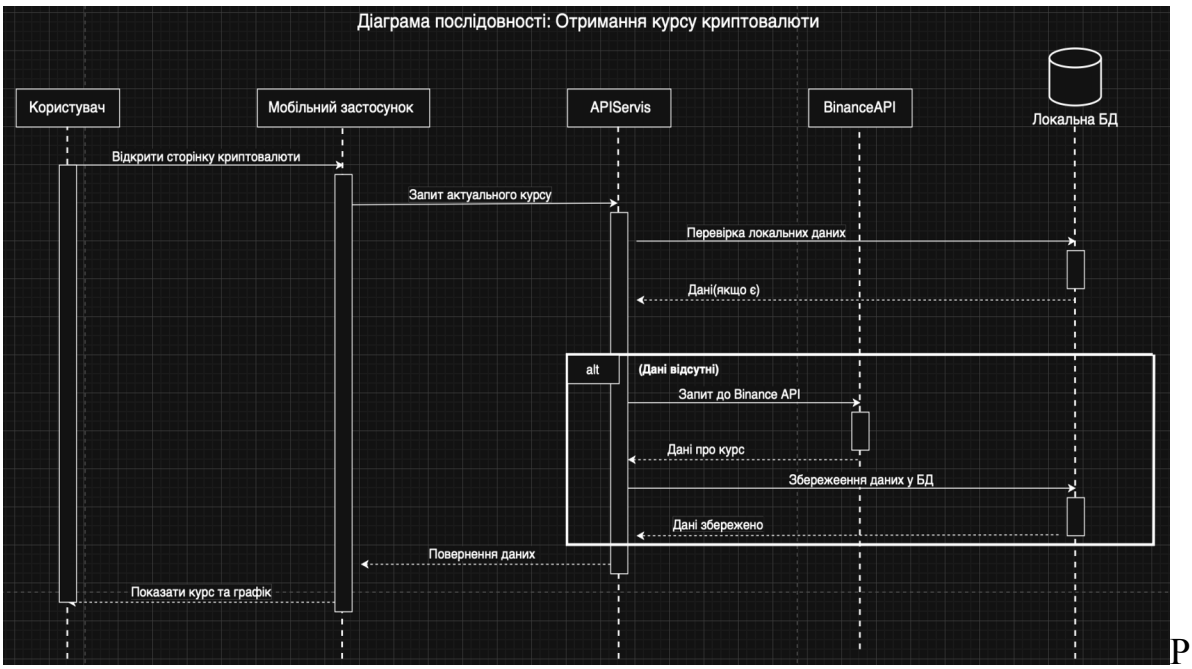


Рис. 1.5 UML-діаграма послідовності взаємодії користувача із системою при отриманні курсу криптовалюти

З рисунка 1.5 видно, що отримання ринкових даних здійснюється за участю кількох логічно пов’язаних компонентів. Діаграма демонструє перевірку локального сховища, умову для звернення до API та повертає результат до застосунку, що дає змогу більше зрозуміти організацію системи.

2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

2.1. Аналіз інформаційних потоків системи

Для визначення джерела даних, способів збереження та обробки інформації був проведений інформаційний аналіз потоків. У цій роботі потоки охоплюють зовнішні дані про ринок, локальні дані користувача та результати їх обробки.

Ринкові дані є основним потоком, який надходить від Binance API [7]. До них належать курси криптовалют та зміни їх вартості, обсяги торгів та інші показники для відображення ринку. Дані надходять до застосунку для відображення списків монет, детальної інформації про них та побудови графіків.

Ще можна виділити такі потоки, які йдуть від користувача. Такі як пошукові запити, вибрані криптовалюти, локальні нотатки та сповіщення та налаштування самого застосунку. Після обробки відповідних дій система повертає користувачеві результати у вигляді відображених ринкових показників, аналітичної інформації, повідомлень і збережених локальних даних.

Для підвищення швидкодії застосунку та зменшення кількості повторних звернень до зовнішнього API у системі передбачено локальне сховище [3]. Частина отриманих ринкових даних зберігається локально та може бути повторно використана у вигляді кешу. Це дозволяє швидше відображати інформацію, а також забезпечує зручніше використання системи при повторному зверненні до тих самих активів.

Окремий напрям інформаційних потоків пов’язаний із роботою аналітичного модуля. Мобільний застосунок передає до нього дані для аналізу, на основі яких формуються результати аналітичної обробки. До таких результатів можуть належати узагальнені показники ринку, порівняння

криптовалют, статистичні розрахунки та інші аналітичні представлення. Крім того, аналітичний модуль забезпечує формування JSON-звіту, який використовується для експорту результатів роботи системи.

Рис. 2.1 Схема інформаційних потоків мобільного застосунку

На рисунку 2.1 видно, що центральним елементом усієї системи є мобільний застосунок. Від модулів системи, таких як користувач, зовнішній API, аналітичний модуль та сховище, йдуть основні інформаційні потоки.

Отже, інформаційні потоки розроблюваної системи охоплюють отримання зовнішніх ринкових даних, роботу з локальною інформацією користувача, аналітичну обробку та відображення результатів. Саме узгоджена робота цих потоків забезпечує повноцінне функціонування мобільного застосунку та зручну взаємодію користувача з криптовалютним ринком.

2.2. Визначення складу вхідних та вихідних даних

Для того щоб мобільний застосунок міг коректно виконувати свої функції, потрібно чітко визначити, які саме дані він отримує на вході та що формує на виході. У випадку системи моніторингу та аналітики курсів криптовалют це особливо важливо, оскільки застосунок одночасно працює і з ринковою інформацією із зовнішніх джерел і з даними, які створює або задає сам користувач.

До вхідних даних можна віднести дані API такі як: курси криптовалют та зміни їх вартості за певний час, обсяги торгів, мінімум та максимум ціни, та дані

для побудови графіків. Ця інформація є мінімально необхідною для бачення ринку та його аналітики.

Дані, що вводить користувач, можна віднести до окремої групи. В них входять дані реєстрації та авторизації, запити пошуку, обрані монети, текстові нотатки зі сповіщеннями. Дані мають індивідуальний характер, на відміну від ринкових даних, та використовуються для персоналізації середовища під користувача.

Окрім попередніх груп даних, система ще працює з локальними вхідними даними. Вони зберігаються окремо в кеші або локальній БД. До них можна віднести список обраних монет, створені нотатки, параметри сповіщень, дані про користувача та частину раніше отриманої ринкової інформації. Повторне використання таких даних дозволяє прискорити роботу застосунку та зменшити кількість повторних звернень до зовнішнього API.

Вихідними даними системи є результати, які застосунок надає користувачеві після обробки вхідної інформації. До них належать списки криптовалют із поточними показниками, детальна інформація про вибраний актив, графіки зміни вартості, обсягів торгів, результати аналітичного порівняння монет, сформовані сповіщення та повідомлення інтерфейсу. Також до вихідних даних можна віднести JSON-звіт, який створюється в процесі експорту результатів аналітичної обробки.

Таким чином, вхідні дані системи формуються із трьох основних джерел: зовнішнього API, дій користувача та локального сховища. Вихідні дані є результатом обробки цієї інформації та подаються у вигляді ринкових показників, аналітичних результатів, графічних представлень і службових повідомлень. Чітке визначення складу вхідних і вихідних даних є необхідним для подальшого проєктування структури збереження інформації та організації роботи програмних модулів застосунку.

2.3. Логічна модель даних

Для відображення основних сутностей системи, атрибутів та зв'язків між ними, була використана логічна модель даних. Вона показує, як дані пов'язані між собою, в самому застосунку. Це дає можливість побачити, як будуть організовані дані під час побудови самої БД.

У майбутній базі даних можна відокремити такі сутності як: сутність Криптовалюта, що використовується для збереження інформації про монету (назва, символ, ціна та ін.), та Історія зміни ціни - дані про зміну ціни за проміжок часу та також для побудови графіків.

Для персоналізації роботи користувача в системі передбачено сутності Обране, Сповіщення та Нотатка. Вони використовуються для збереження вибраних монет, параметрів цінових повідомлень і текстових коментарів користувача. Окремо може використовуватися сутність, пов’язана з кешуванням ринкових даних. Вона дає можливість повторно використовувати раніше отриману інформацію без зайвих звернень до зовнішнього API.

Основні сутності логічно пов’язані між собою. Один користувач може мати кілька записів у списку обраного, кілька сповіщень і кілька нотаток. Одна криптовалюта може бути пов’язана з багатьма записами історії ціни, та може входити до списків обраного або бути об’єктом сповіщень. Структура такого виду дає можливість згрупувати дані та забезпечити коректне використання в системі.

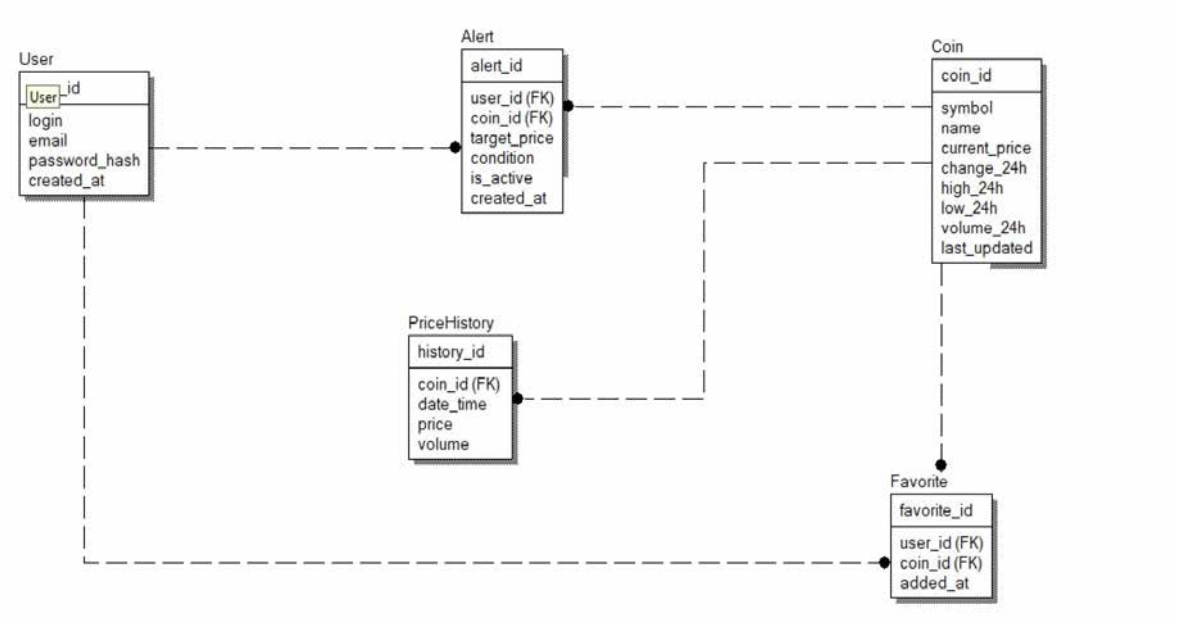


Рис. 2.2 Логічна модель даних у вигляді ER-діаграми

Отже, можна сказати, що логічна модель відображає основні сутності та зв'язки між ними та є основою для подальшого проєктування БД й реалізації програмних модулів системи.

2.4. Організація локального збереження та обробки даних

Для мобільного застосунку важливо не тільки отримувати дані, а й зберігати, оскільки під час взаємодії із застосунком формується багато різних даних. До них можна віднести облікові записи, обрані монети, сповіщення та нотатки. Локальне збереження дозволить менше звертатися до API збільшивши швидкодію, та дасть змогу користувачам зберігати їхні персональні параметри.

Для бази даних було віддано перевагу Core Data. Він є стандартним для платформи iOS і добре зберігає сутності, які мають зв'язки між собою. Ще однією перевагою можна зазначити зручну інтеграцію з мовою програмування Swift без ускладнення архітектури застосунку.

Також використовується для збереження кешованих ринкових даних формат JSON. Оскільки відповіді від зовнішнього API також надходять у цьому форматі, це спрощує збереження отриманих даних і зменшує необхідність

додаткових перетворень. Також цей формат є зручним для експорту та подальшої аналітики.

Такий підхід дозволяє розділити навантаження та не перевантажувати БД тимчасовою інформацією. Core Data використовується для роботи, де є структура та зв'язки між сутностями, а JSON для швидкого кешування та експорту.

Після того, як система отримала інформацію, вона перетворює її у внутрішні моделі, зберігає, сортує та підготовлює до відображення в інтерфейсі.

Отже, поєднання таких методів є рішенням, яке забезпечує локальне збереження, швидкодію та зручну роботу з різними типами даних у застосунку.

2.5. Формування структури інформаційної бази системи

Для впорядкованого збереження формується БД, яка повинна охоплювати ринкову інформацію із API, локальні дані користувачів. При формуванні було враховано логічну модель даних та взаємодію між сутностями. База даних містить відомості про користувачів, криптовалюти та зміни їх цін, обрані монети користувачів, їх сповіщення та нотатки та інші допоміжні дані. Така кількість інформації для роботи основних функцій застосунку забезпечує цілісність інформації під час обробки та збереження.

Основу інформаційної бази становлять дані про криптовалюти та пов’язані з ними ринкові показники. Саме вони використовуються для відображення поточного стану ринку, формування детальної інформації про окремі активи, побудови графіків і виконання аналітичних операцій. Інша частина бази пов’язана з користувачем і зберігає ті дані, які забезпечують персоналізацію роботи застосунку. До них належать записи про обрані монети, параметри сповіщень, локальні нотатки та службова інформація.

Побудована така структура, щоб різні типи даних зберігалися окремо, але водночас були логічно пов'язані між собою. За рахунок такого методу можна уникнути дублювання, спростити доступ до інформації для програмних модулів та забезпечити зручне оновлення даних під час роботи системи. Крім цього, він створює основу для подальшого розширення функціональних можливостей застосунку без суттєвого перегляду вже сформованої структури.

Таким чином, сформована структура інформаційної бази системи забезпечує впорядковане збереження ринкових, користувацьких і службових даних, необхідних для стабільної роботи мобільного застосунку. Вона є завершальним елементом інформаційного забезпечення системи та створює основу для подальшої реалізації прикладного програмного забезпечення.

3. ПРИКЛАДНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Організаційна структура програмного забезпечення

Для визначення основних частин застосунку, їх призначення та як вони взаємодіють між собою була визначена організаційна структура. Для даної системи це важливо для коректної роботи інтерфейсу, обробки даних та їх збереження, взаємодії із зовнішнім ресурсом та формування аналітики.

Під час побудови програмного забезпечення було використано модульний підхід. Це означає, що застосунок поділено на окремі логічні частини, кожна з яких виконує власну функцію. Крім того, структура застосунку орієнтована на принципи архітектурного підходу MVVM, у межах якого інтерфейс користувача, логіка обробки даних та моделі предметної області розділяються між відповідними програмними компонентами.

Структуру можна розділити на декілька модулів. Першим можна виділити інтерфейсні, які відповідають за коректне відображення екранів і взаємодію з

ними. Також є модулі бізнес-логіки, вони відповідають за стани екранів, обробку даних та зв'язки між сервісами з інтерфейсом. Сервісні модулі відповідають за роботу із зовнішнім API, локальним сховищем, кешуванням і сповіщеннями. Окремо виділяються моделі даних та допоміжні компоненти, що використовуються для збереження, форматування, пошуку та інших службових операцій.

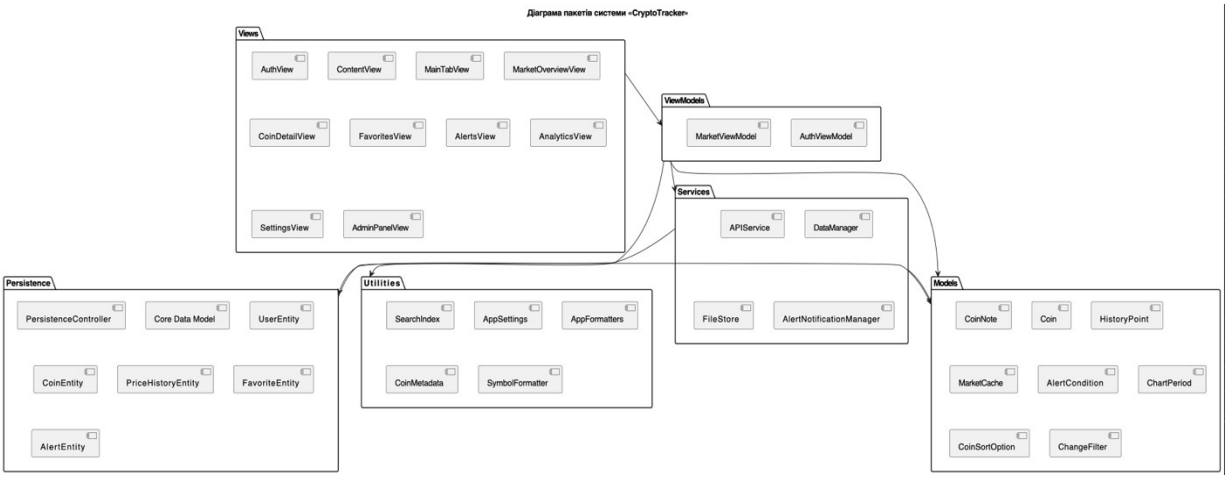


Рис. 3.1 Діаграма пакетів програмного забезпечення мобільного застосунку моніторингу криптовалют

На рисунку 3.1 показано поділ програмного забезпечення на основні пакети та зв’язки між ними. Основні фрагменти програмного коду, що використовуються під час реалізації модулів застосунку, наведено у додатку А. Детальну структуру програмного проєкту та призначення його основних файлів наведено у додатку Б. Така організація дозволяє зробити структуру застосунку більш зрозумілою, спростити супровід програмного коду та забезпечити можливість подальшого розширення функціональності системи.

3.2. Вибір інструментальних засобів розробки

Важливим етапом розробки є вибір інструментів, які будуть відповідати особливостям розробки, давати зручну можливість реалізації функціоналу і не ускладнювати подальшу модернізацію. Розроблений застосунок орієнтований на операційну систему iOS, тому вибір інструментів був здійснений з урахуванням

сучасного інтерфейсу, роботи з локальними даними та нативного підходу до створення.

Перевагу було віддано мові програмування Swift, фреймворку SwiftUI для створення інтерфейсу, Core Data та JSON для збереження та обробки даних. Такий набір інструментів дає змогу зробити сучасний інтерфейс, правильно реалізувати логіку, інтегрувати зовнішній API та локально зберігати дані.

3.2.1. Обґрунтування вибору мови програмування Swift

Для розробки була вибрана мова програмування Swift [1]. Ця мова була обрана через нативність для екосистеми Apple та призначена для застосунків для платформи iOS. Вона має добру сумісність із системними бібліотеками, зручне середовище розробки та сучасні підходи для створення ПЗ [28].

Також перевагою можна відзначити простий синтаксис та мовні конструкції, це спрощує написання та супровід коду. Це є важливим для проєкту, оскільки система поділена на логічні модулі, які мають без ускладнення взаємодіяти між собою. Також Swift дозволяє будувати інтерфейси, працювати з асинхронними запитами та локальними моделями даних.

Враховуючи вищезазначене, Swift є найкращою мовою для створення застосунку для платформи iOS завдяки зручній розробці та підтримці сучасної архітектури застосунку.

3.2.2. Обґрунтування вибору фреймворку SwiftUI

Для побудови користувацького інтерфейсу застосунку було обрано SwiftUI [2]. Використання цього фреймворку дає можливість створювати інтерфейси нативно для iOS, використовуючи декларативний підхід до опису екранів і елементів взаємодії.

Однією з причин вибору SwiftUI є зручність побудови інтерфейсів, що складаються з багатьох взаємопов’язаних екранів. У межах розроблюваного застосунку це особливо важливо, оскільки потрібно реалізувати перегляд ринку, сторінки окремих монет, аналітичні екрани, роботу з обраним, сповіщеннями та

налаштуваннями. SwiftUI дозволяє будувати такі екрани більш компактно, а також простіше організовувати оновлення даних у відповідь на дії користувача або зміну стану системи.

Ще можна відзначити перевагу хорошої взаємодії з архітектурним підходом MVVM, який був використаний у роботі. Це дає можливість відокремити відображення даних від логіки їх обробки та поєднати інтерфейс з ViewModel-компонентами. Отже, вибір SwiftUI є обґрунтованим завдяки зручності побудови інтерфейсу, хорошій інтеграції з платформою iOS та підтримці сучасних підходів до організації мобільних застосунків.

3.2.3. Обґрунтування використання Core Data та JSON

Для збереження даних та їх обробки використано Core Data [3] і JSON. Вибір формується на понятті того, що в системі використовуються різні типи даних з різними підходами до збереження.

Core Data використовується для збереження локальних даних, які мають сутності та зв'язки між ними. До них можна віднести користувачів, криптовалюти та їх історію цін, обрані монети, нотатки та сповіщення. Цей засіб добре підходить для роботи з локальним сховищем на iOS, оскільки дозволяє організувати впорядковане збереження даних, їх оновлення та вибірку.

За допомогою JSON кешуються дані про ринок і працює частина допоміжної інформації. Через те, що дані із API надаються цьому форматі спрощується їх використання та збереження. Крім того, такий формат є зручним для експорту аналітичних результатів.

Поєднання Core Data і JSON є доцільним технічним рішенням, оскільки дозволяє зручно працювати як зі структурованими локальними сутностями, так і з кешованими або експортованими даними.

3.3. Проєктування архітектури мобільного застосунку

Архітектуру системи було створено для визначення структури та основних частин системи. Вона також повинна забезпечувати правильну обробку ринкових даних, коректну роботу API, коректне відображення ринкових даних та коректне зберігання інформації в базі даних.

Було застосовано модульний підхід до проєктування системи (з орієнтацією на MVVM); тому інтерфейс користувача відокремлений від логіки обробки даних; модель домену відокремлена від інтерфейсу користувача; а сервіси відокремлені від усіх вищезгаданих розділів архітектури системи [18].

Користувацький інтерфейс (UI) у SwiftUI разом із ViewModels, сервісними модулями (для взаємодії з API), локальним сховищем, кешами, сповіщеннями, моделями даних та допоміжними компонентами є найважливішим елементами архітектури застосунку. Локальні дані зберігаються за допомогою Core Data/SQLite, тоді як дані кешу та дані експорту зберігаються у форматі JSON.

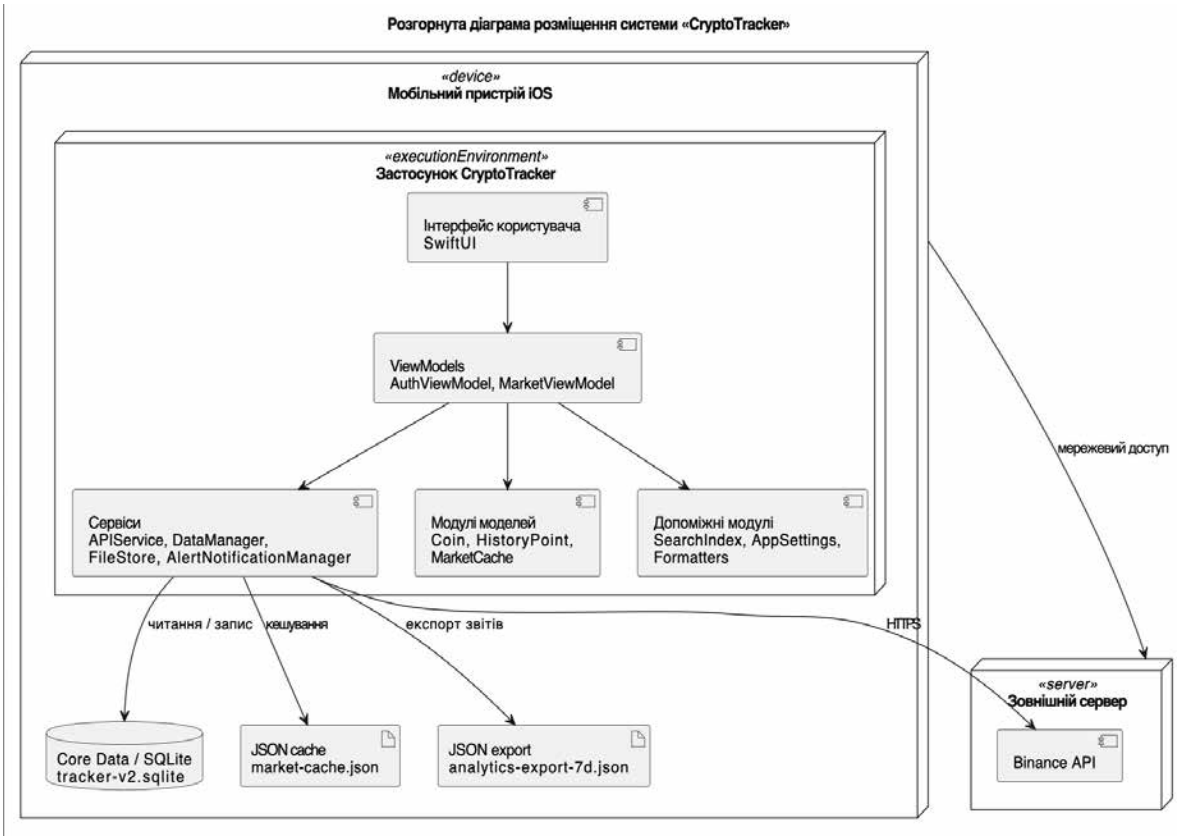


Рис. 3.2 Розгорнута діаграма розміщення з використанням елементів діаграми компонентів

Отже, створена архітектура забезпечує чіткий розподіл обов'язків основних компонентів програми та створює добру основу для майбутнього зростання.

3.4. Реалізація модуля авторизації та реєстрації користувачів

За вхід користувача в систему та створення його запису відповідає модуль авторизації та реєстрації. Цей модуль було реалізовано локально без використання зовнішніх серверів. Інтерфейс модуля реалізовано у файлі AuthView.swift. Екран підтримує два режими: вхід до вже створеного облікового запису та реєстрацію нового користувача. Під час реєстрації перевіряється, чи заповнені всі поля, чи збігаються паролі, чи має пароль мінімальну довжину та чи містить тільки допустимі символи. Це дозволяє ще на рівні інтерфейсу відсіяти некоректні дані.

AuthViewModel відповідає за основну логіку, а в DataManager виконується робота з локальними записами. Після того як користувач виконав реєстрацію або вхід до застосунку система зберігає стан користувача, тому після повторного запуску застосунку його сесія продовжується. Зберігання пароля виглядає у вигляді хешу для безпеки даних.

Фрагмент коду з AuthViewModel наведено у додатку А, який показує перевірку введених даних і виклик методів register(...) та login(...):

У цих фрагментах видно, що перед збереженням або перевіркою облікового запису дані спочатку нормалізуються: із рядків прибираються зайві пропуски та переноси рядка. Далі виконуються базові перевірки. Якщо якесь поле не заповнене або паролі не збігаються, користувач одразу отримує повідомлення про помилку. Якщо перевірка проходить успішно, AuthViewModel звертається до DataManager, який уже виконує реєстрацію або пошук користувача в локальному сховищі.

Окремо варто відзначити, що збереження пароля реалізовано без прямого запису його текстового значення. У DataManager використовується хешування пароля перед збереженням [20; 26], а під час входу перевіряється відповідність введеного значення збереженому хешу. Це робить локальну авторизацію безпечнішою і відповідає базовим вимогам до роботи з обліковими даними.

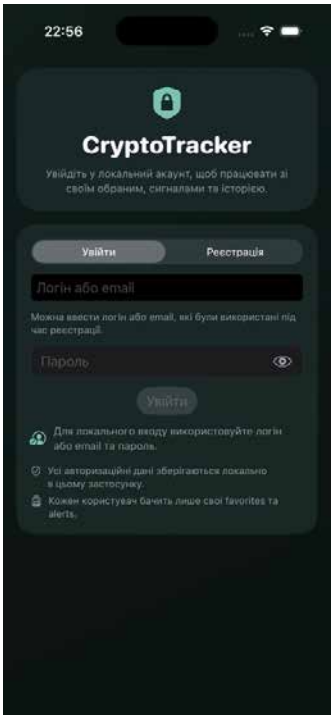


Рис. 3.3 Екран авторизації користувача

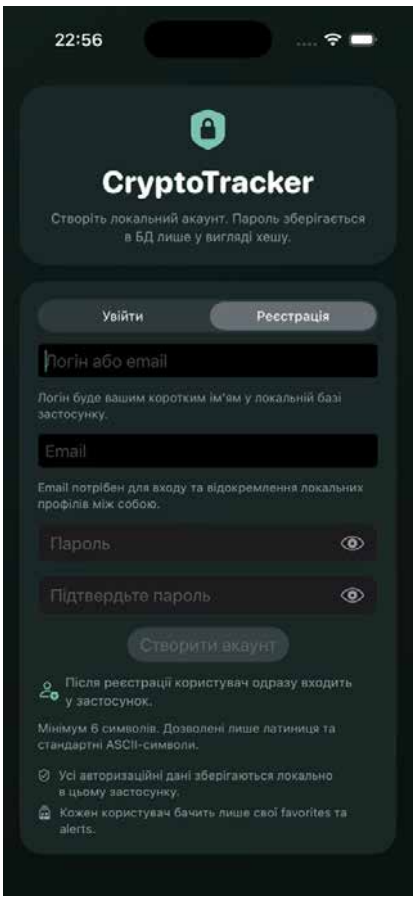


Рис. 3.4 Екран реєстрації нового користувача

Таким чином, модуль авторизації та реєстрації забезпечує локальну ідентифікацію користувача, перевірку коректності введених даних і подальший доступ до персональних можливостей застосунку.

3.5. Реалізація модуля перегляду ринку криптовалют

Модуль перегляду є центральною частиною застосунку, тому що саме з нього починається основна робота користувача. Цей екран відображає список монет, їх ціни та зміни цін за добу, ринкові показники. Саме цей модуль дає користувачеві загальне уявлення про стан ринку та дозволяє швидко перейти до більш детального перегляду окремого активу.

Робота модуля побудована на взаємодії MarketOverviewView, MarketViewModel та APIService. Після запуску застосунку ViewModel звертається до сервісного рівня, отримує ринкові дані з Binance API, відбирає

потрібні торгові пари та формує внутрішні моделі типу Coin, з якими вже працює інтерфейс. Такий підхід дозволяє відокремити відображення даних від логіки їх завантаження й обробки.

Інтерфейс модуля реалізовано у вигляді списку карток монет. Користувач може оновити ринок вручну, перейти до детального перегляду вибраного активу, а під час завантаження даних на екрані відображаються placeholder-елементи. Це робить роботу інтерфейсу більш плавною та зрозумілою для користувача.

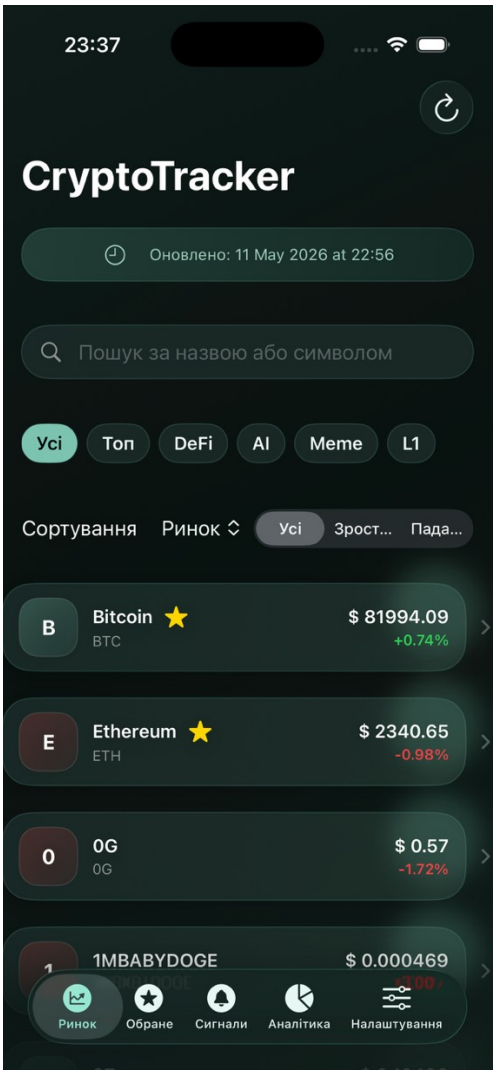


Рис. 3.5 – Основний екран перегляду ринку криптовалют

Фрагмент коду з методу fetchCoins(...) наведено у додатку А, який використовується для завантаження та відбору монет із зовнішнього API:

У цьому фрагменті спочатку визначається кількість монет, яку потрібно показати на екрані. Далі виконується отримання всіх доступних USDT-пар через допоміжний метод fetchEligibleTickers(). Після цього частина монет потрапляє

до списку пріоритетних, а решта сортується за обсягом торгів і силою зміни ціни. На завершальному етапі вибрані записи перетворюються у внутрішні об’єкти Coin, з якими далі працює застосунок.

Такий підхід є зручним, оскільки дозволяє ще на рівні сервісного модуля відфільтрувати зайві дані та підготувати для інтерфейсу вже готовий список монет. Завдяки цьому модуль ринку забезпечує швидкий доступ до актуальної інформації та зручну навігацію по списку криптовалют.

3.6. Реалізація модуля пошуку та фільтрації монет

Модуль пошуку та фільтрації потрібен для того, щоб користувач міг швидко знайти потрібну монету в загальному списку. У застосунку це особливо важливо, оскільки навіть після відбору ринкових даних користувач працює з великою кількістю активів. Саме тому пошук повинен бути достатньо швидким, а фільтрація – простою і зрозумілою.

У розробленій системі пошук реалізовано на двох рівнях. Перший рівень – локальний, коли користувач працює з уже завантаженим списком монет. Другий рівень – віддалений, коли для точнішого результату застосунок звертається до Binance API. Крім цього, модуль підтримує сортування, тематичні категорії та історію останніх пошукових запитів, що робить взаємодію зі списком монет зручнішою.

В модулі APIService побудована логіка пошуку, а через MarketViewModel стан фільтрації та її відображення. Після введення запиту текст нормалізується, потім обчислюється умовний ранг для кожної монети та сортуються за точністю збігу та додатковими ринковими показниками.

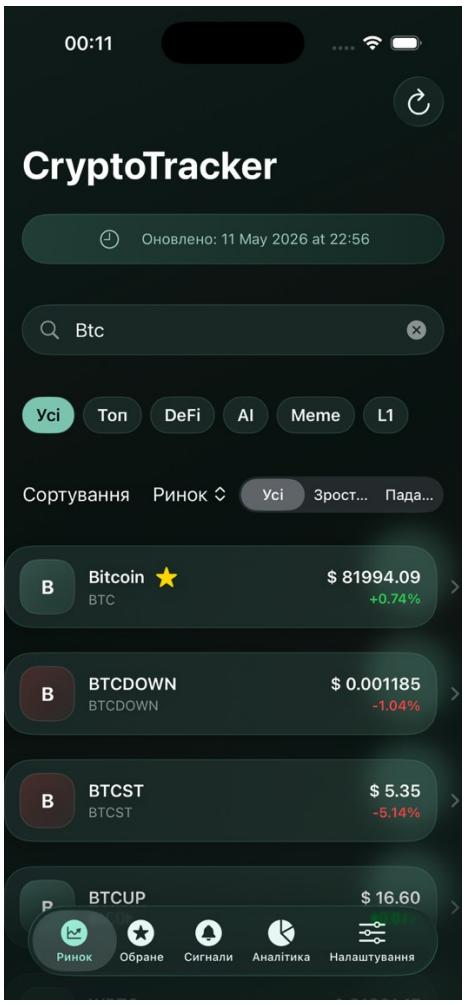


Рис. 3.6 Пошук, сортування та фільтрація монет у застосунку

Фрагмент коду з методу searchCoins(matching:) наведено у додатку А, який використовується для віддаленого пошуку монет:

У цьому фрагменті введений користувачем запит спочатку очищається від зайвих пропусків і переводиться у верхній регістр. Якщо рядок порожній, система одразу повертає порожній результат. Далі завантажується перелік доступних монет, для кожної з них обчислюється ранг відповідності запиту, а потім результати сортуються спочатку за релевантністю, а потім за ринковими показниками. На останньому етапі записи перетворюються у внутрішні об’єкти Coin.

Такий підхід дозволяє отримувати не просто збіг за назвою чи символом, а більш коректно впорядкований список результатів. У підсумку модуль пошуку

та фільтрації значно спрощує роботу користувача зі списком монет і робить взаємодію з ринком більш зручною.

3.7. Реалізація модуля перегляду детальної інформації про актив

Після того, як користувач вибрав монету та хоче переглянути її детально він переходить від списку монет до аналізу конкретної криптовалюти. Тут можна побачити поточну ціну, зміну вартості за добу, ринкові показники та додаткові елементи, які пов'язані з аналізом і персоналізацією.

У CoinDetailView.swift реалізовано інтерфейс самого модуля. У верхній частині екрана розташована картка монети з її назвою, символом, поточною ціною та зміною за 24 години. Нижче відображається блок із ключовими ринковими характеристиками, серед яких максимальна та мінімальна ціна за добу, обсяг торгів, а також допоміжні елементи керування, що дозволяють працювати з графіками, нотатками та сповіщеннями.

Особливість цього модуля полягає в тому, що під час його відкриття система одночасно завантажує історію цін, яка потрібна для побудови графіків. Тобто екран детального перегляду поєднує в собі відразу кілька функцій: показ актуальної інформації про монету, доступ до історичних даних і перехід до подальших аналітичних дій. Завдяки цьому користувач отримує всю основну інформацію про вибраний актив в одному місці, без необхідності переходити між кількома окремими екранами.

Також модуль детального перегляду тісно пов’язаний із модулями історичних даних, обраного, нотаток і сповіщень. Саме з цього екрана користувач може не лише переглядати ринкові показники, а й виконувати дії над конкретною монетою: додавати її до списку обраного, створювати локальні нотатки або налаштовувати цінові сигнали. Це робить даний модуль одним із ключових елементів взаємодії користувача із системою.

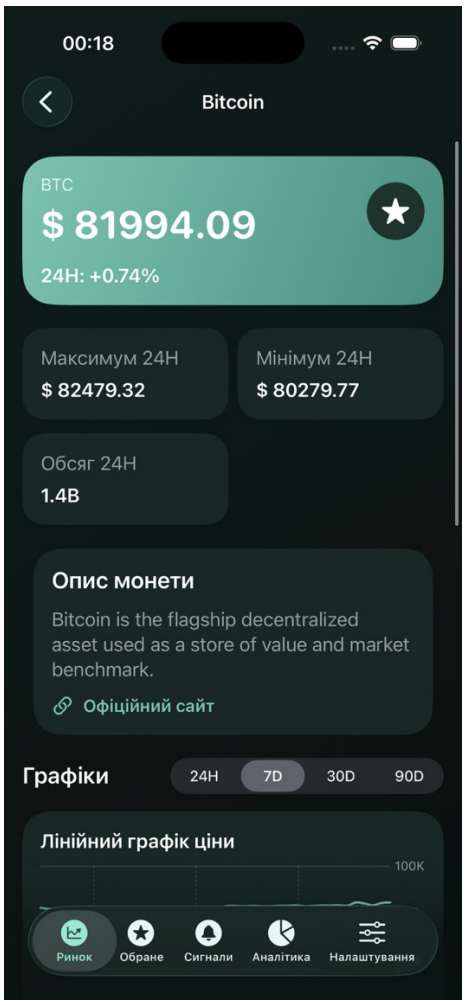


Рис. 3.7 Екран перегляду детальної інформації про криптовалютний актив

У результаті модуль детального перегляду дозволяє перейти від загального огляду ринку до аналізу конкретного активу та забезпечує користувачеві зручний доступ до всієї основної інформації, пов’язаної з вибраною монетою.

3.8. Реалізація модуля побудови графіків та відображення історичних даних

Для аналізу ринку недостатньо бачити лише поточну ціну, тому в застосунку реалізовано модуль побудови графіків та відображення історичних даних. Він дає змогу побачити, як змінювалися ціна монети та обсяг торгів за вибраний період, а отже, дозволяє оцінювати ринок не лише за одним поточним значенням, а й в динаміці.

За допомогою Binance API завантажуються дані, після чого формується запит до біржової кінцевої точки API для отримання свічкових даних [22], а потім отримана відповідь перетворюється на масив точок типу HistoryPoint за допомогою сервісного модуля APIService. Далі дані можуть зберігатися локально для повторного використання, що зменшує кількість запитів і збільшує швидкість застосунку.

Якщо зовнішнє API тимчасово недоступне, система не припиняє роботу відразу. Спочатку перевіряється наявність потрібної інформації в оперативному кеші, а після цього — у локальному сховищі. Такий підхід дозволяє не втрачати вже отримані дані, зменшує залежність від мережевого з’єднання і робить модуль більш стійким у реальному використанні. Для візуалізації графіків у застосунку використовується фреймворк Charts [6].

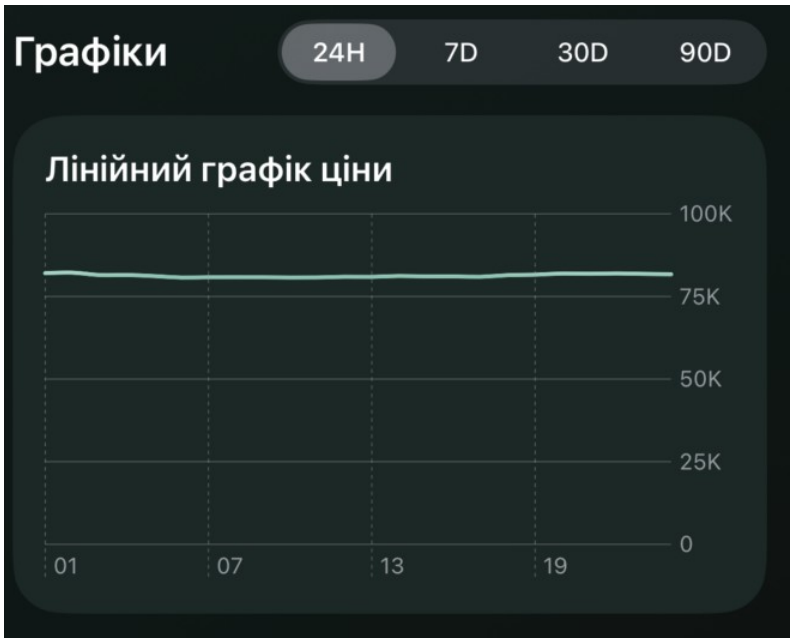


Рис. 3.8 Лінійний графік зміни ціни криптовалюти

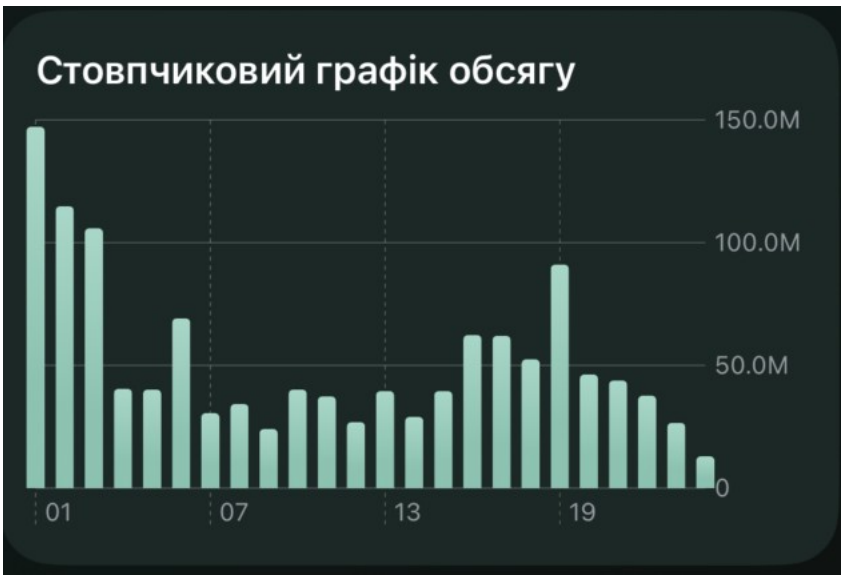


Рис. 3.9 Стовпчиковий графік обсягу торгів

Фрагмент коду з методу loadHistory(...) наведено у додатку А, який відповідає за завантаження історичних даних, збереження результату та резервне відновлення з локального сховища:

У цьому фрагменті спочатку формується ключ для доступу до історії конкретної монети за вибраний період. Далі система намагається завантажити історичні дані через apiService.fetchHistory. Якщо запит виконується успішно, отриманий масив зберігається в оперативному словнику histories, записується в локальне сховище та потрапляє до кешу. Якщо ж під час запиту виникає помилка, застосунок спочатку перевіряє наявність історії в поточному кеші, а потім звертається до локальної бази.

Такий механізм дозволяє забезпечити більш стабільну роботу модуля графіків навіть у разі тимчасових проблем із мережею або зовнішнім API. У результаті модуль історичних даних робить застосунок кориснішим для реального спостереження за ринком і дає змогу оцінювати зміну ціни та торгової активності за певний проміжок часу.

3.9. Реалізація модуля обраних монет, нотаток та цінових сповіщень

Для персоналізації роботи користувача в застосунку реалізовано модуль обраних монет, нотаток та цінових сповіщень. Завдяки цьому користувач може зберігати активи, які його цікавлять, робити власні короткі позначки та налаштовувати сигнали за ціною. Саме ці можливості роблять застосунок не просто переглядачем ринку, а більш зручним інструментом для щоденного використання.

Механізм обраного реалізовано через MarketViewModel і DataManager. Якщо монета вже є у списку, запис видаляється, а якщо її ще немає – створюється новий FavoriteEntity, який пов’язується з поточним користувачем і конкретною монетою. Завдяки цьому користувач може швидко формувати власний набір активів для постійного спостереження.

Локальні нотатки зберігаються через FileStore у JSON-кеші [19; 24]. Для кожної монети користувач може залишити короткий коментар, який потім відновлюється під час повторного відкриття екрана. Це дозволяє фіксувати власні спостереження, не використовуючи сторонні засоби. Цінові сповіщення, у свою чергу, зберігаються локально й перевіряються системою під час оновлення ринку. Якщо ціна досягає заданої умови, формується локальне сповіщення.

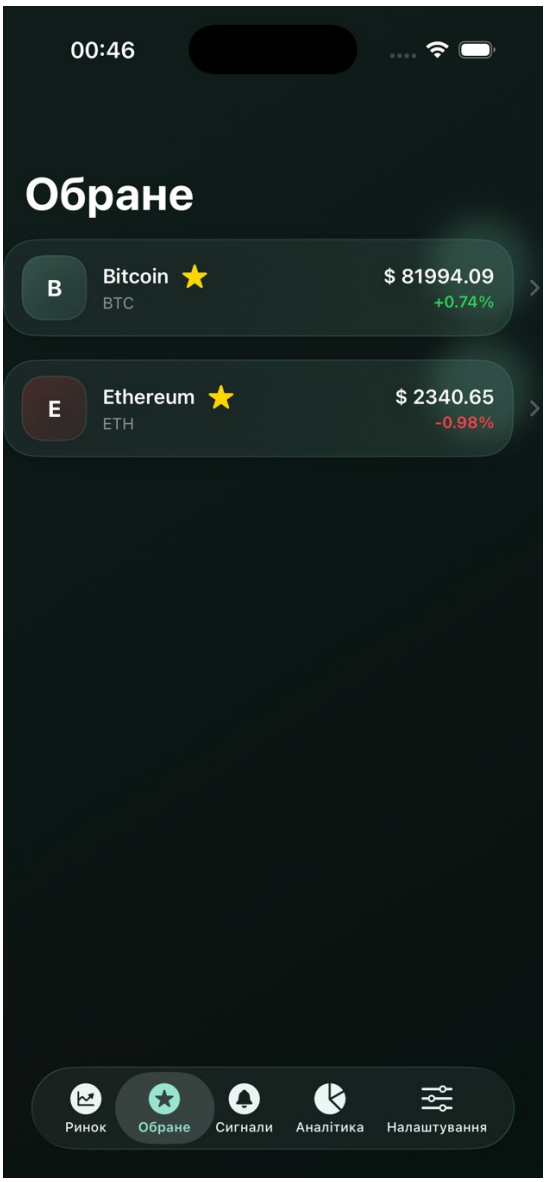


Рис. 3.10 Робота зі списком обраних монет

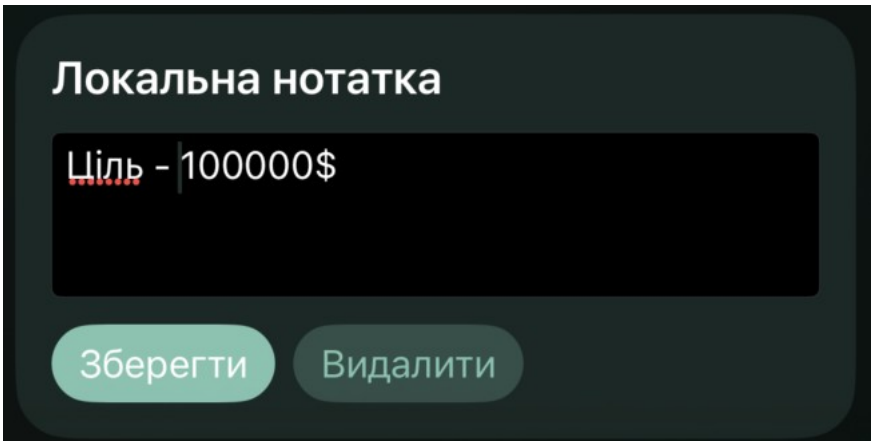


Рис. 3.11 Блок локальної нотатки для монети

Фрагмент коду з методів toggleFavorite(...) та saveAlert(...) наведено у додатку А, які відповідають за додавання монети до списку обраного і створення цінового сповіщення:

У наведеному фрагменті метод toggleFavorite(...) перевіряє, чи є монета обраною. Якщо вона вже додана, запис видаляється; якщо ні – створюється новий. Після цього список обраних монет оновлюється, щоб інтерфейс одразу показав зміни. Метод saveAlert(...) [5] відповідає за створення сповіщення. Спочатку він перевіряє правильність введеної цільової ціни, а потім передає дані до DataManager, де виконується локальне збереження сигналу.

Окремо слід вказати, що перевірка сповіщень виконується автоматично під час оновлення ринку. Це означає, що користувачеві не потрібно вручну контролювати момент досягнення потрібної ціни. Локальні нотатки, обране та сповіщення разом формують персоналізований рівень роботи із системою.

Цей модуль збільшує практичну цінність застосунку для користувача, щоб мати можливість не лише переглядати ринок, а й організовувати власну роботу з активами.

3.10. Реалізація аналітичного модуля

Аналітичний модуль потрібен для того, щоб користувач бачив не лише ок ремі ціни, а й загальну картину по ринку. У застосунку він формує коротке зведення, показує лідерів зростання, монети з найбільшими обсягами торгів та дає з могу порівнювати два активи між собою. За допомогою цього користувач отримує змістовне уявлення про стан ринку замість простого набору чисел.

AnalyticsView.swift реалізує технічно сам модуль. Для функцій порівняння монет використовується нормалізація історичних даних відносно першої точки ряду, після чого обчислюється відсоткова зміна ціни. Завдяки цій функції можна коректно порівнювати активи з різною початковою вартістю. Також є модуль, який робить коротке ринкове зведення у вигляді списку лідерів за зростанням за найбільшим обсягом торгів.

Також є підтримка експорту JSON-файлу в окремому модулі, який формує звіт, що містить дані про ринок, списки лідерів та результати порівняльного аналізу [24; 25]. Такий підхід дає змогу не лише переглядати аналітику в інтерфейсі, а й зберігати результати для подальшого використання.

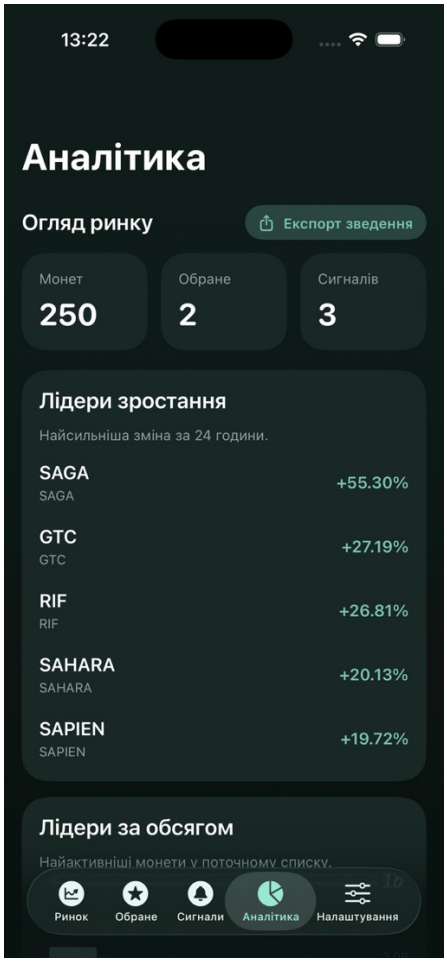


Рис. 3.12 Екран аналітичного модуля мобільного застосунку

Наведений фрагмент коду у додатку А формує об'єкт AnalyticsExportPayload, який об'єднує результати аналітичної обробки до якої входять дата формування звіту, коротке підсумкове зведення, список монет-лідерів за зростанням, список активів з найбільшими обсягами торгів і результат вибраних порівняних монет. Потім готова структура передається до методу exportAnalytics(...) для формування JSON-файлу. Після чого можна цей файл зберегти та використовувати для подальшої аналітики. Приклади JSON-

структур, що використовуються для отримання, кешування та експорту даних, наведено у додатку В.

Такий спосіб реалізації є зручним, оскільки дозволяє зібрати результати аналітики в одному місці та представити їх у структурованому вигляді. У підсумку аналітичний модуль дає користувачеві узагальнену інформацію про ринок і дозволяє зберігати результати аналізу у файлі.

3.11. Реалізація модуля налаштувань та адміністративного розділу

Для керування параметрами застосунку та перегляду сутностей БД було розроблено два окремі модулі, такі як налаштування та адміністрування у вигляді окремих блоків. Перший блок виконаний для звичайного користувача, а інший для контролю стану даних системи. Користувач отримує прості параметри для персоналізованої роботи, а розробник чи адміністратор може переглядати збережені записи застосунку.

Клас AppSettings реалізує основні користувацькі налаштування. Він зберігає мову, тему, пріоритет обраних монет, розмір графіків (компактний чи стандартний), глибину списку ринку та інші параметри застосунку. Збереження виконується через UserDefaults [4], що дає змогу після перезапуску застосунку відновлювати користувацькі налаштування.

Адміністративний розділ реалізовано у вигляді окремого екрана AdminPanelView. У ньому використано кілька FetchRequest для перегляду локальних сутностей Core Data: користувачів, монет, історії цін, обраного та сповіщень. Це зручно для перевірки локального стану системи під час розробки та тестування, оскільки дає можливість швидко бачити, які саме записи зараз зберігаються в базі.

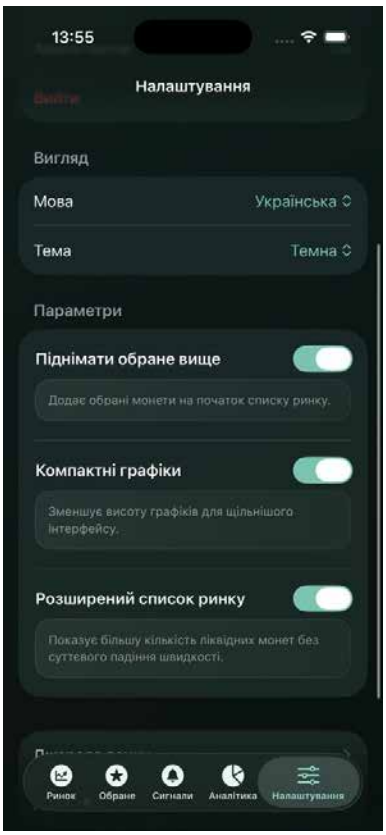


Рис. 3.13 Екран налаштувань мобільного застосунку

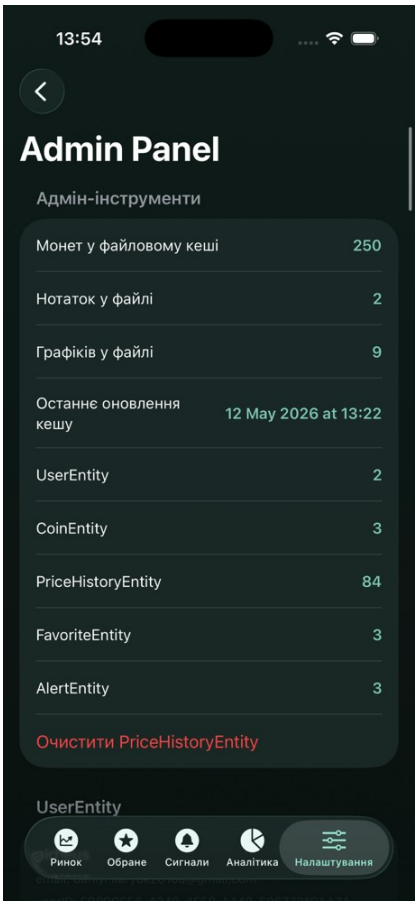


Рис. 3.14 Адміністративний розділ для перегляду локальних сутностей

Фрагмент коду з класу AppSettings наведено у додатку А, який показує збереження основних параметрів користувача. Він показує, що кожна зміна налаштувань записується в UserDefaults, що дає можливість не скидати параметри після закриття застосунку. Завдяки такому простому підходу користувач отримує просту структуру збереження налаштувань локально. В адміністративному розділі використано FetchRequest для отримання локальних сутностей Core Data. Дані користувачів і монет виводяться у відсортованому вигляді за датою створення або оновлення.

У результаті модуль налаштувань та адміністративного розділу забезпечує персоналізацію роботи користувача і контроль за локальними даними, що зберігаються в застосунку.

3.12. Алгоритмізація та програмування основних програмних модулів

Застосунок працюватиме не лише через графічний інтерфейс, а й за допомогою двох специфічних типів алгоритмів: алгоритмів обробки даних, які належним чином визначені. Протягом процесу розробки застосунку було створено кілька основних сценаріїв користувача для таких функціональних елементів: оновлення ринку, імпорт/оновлення історії цін, додавання монет до списку обраного, перевірка сповіщень про ціни та створення аналітичних файлів. На основі цих сценаріїв буде згенеровано основну логіку системи.

Один з основних алгоритмів у цій системі можна назвати «Оновлення ринкових тікерів». Він починається з того, що система запитує інформацію про ринковий тікер від Binance API, який повертає колекцію об'єктів тікерів. Система вибирає, які дані тікерів потрібні для подальшої обробки на основі власних внутрішніх критеріїв, створює модель Coin для кожного з цих вибраних тікерів, оновлює своє локальне сховище/кеш цими даними та перевіряє наявність активних сповіщень, які можуть стосуватися щойно створених Coin. Зрештою,

користувач побачить список доступних Coin для відображення в інтерфейсі користувача системи.

Ще один алгоритм у системі передбачає завантаження історичних даних із зовнішнього API, якщо він доступний. Якщо з цим методом завантаження виникнуть проблеми, система спробує завантажити історичні дані, використовуючи локальний операційний кеш або локальну базу даних. Ці резервні джерела збережуть функціональність діаграм, незважаючи на можливе нестабільне мережеве з'єднання.

Процеси додавання монети до списку обраного, збереження локальної ноти, створення/перевірки сповіщення про ціну та генерації аналітичного звіту JSON – це окремі алгоритми. Однак вони не є незалежними один від одного. Усі вони є частиною спільної логіки програми, тому їх алгоритмізація є життєво важливою для забезпечення стабільної роботи системи.

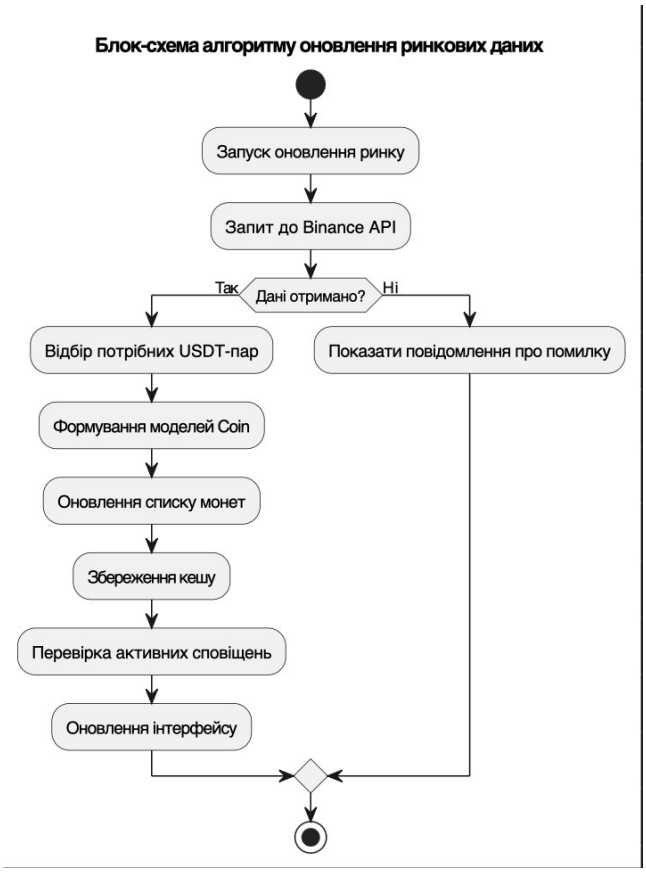


Рис. 3.15 Блок-схема алгоритму оновлення ринкових даних

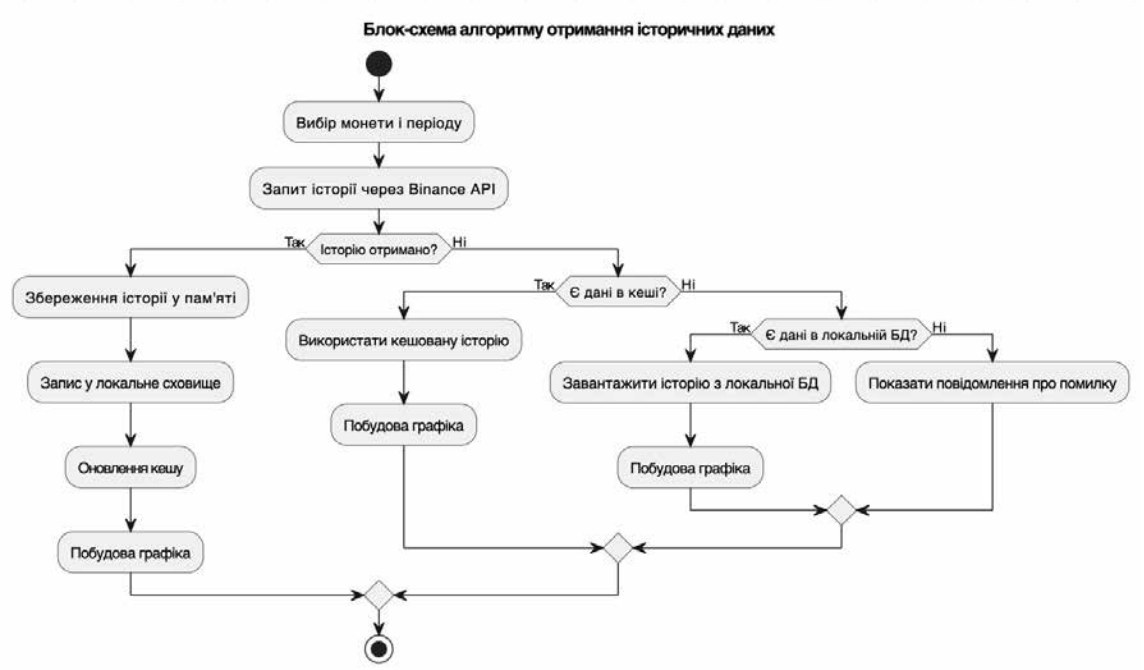


Рис. 3.16 Блок-схема алгоритму отримання історичних даних із використанням кешу та локального сховища

Уривок коду з методу refresh(...) наведено у додатку А, який демонструє алгоритм оновлення ринку:

Процес оновлення починається із завантаження системи, після чого з APIService запитуються поточні ринкові дані. Якщо запит успішний, існуючий список монет буде оновлено, новий об'єкт MarketCache буде створено та збережено локально, а також будуть перевірені активні сповіщення. Якщо із запитом виникла проблема, користувач отримає повідомлення про помилку.

Другий ключовий приклад для демонстрації принципу – це завантаження історичних даних за допомогою методу loadHistory(...). Перша частина цієї функціональності звертається до зовнішнього API, і якщо це не вдається, завантаження відбувається з локальних джерел.

Згаданий код показує, що система намагається зв'язатися з API для отримання історії цін цього символу. Якщо це не вдається, система перевіряє історію в поточному кеші та запитує, чи зберігається він локально. Якщо після перевірки всіх трьох можливих місць розташування дані все ще відсутні, система

відображає користувачеві повідомлення про помилку. Як результат, цей метод робить модуль побудови графіків більш стійким до будь-яких зовнішніх збоїв.

Таким чином, алгоритмізація основних модулів дозволила систематизувати логіку роботи застосунку та забезпечити його стабільну роботу як у штатному, так і в резервному сценаріях.

4. РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1. Тестування мобільного застосунку

Застосунок було протестовано після впровадження всіх основних функціональних модулів, щоб перевірити, чи правильно функціонують інтерфейс користувача, локальне сховище даних, отримання даних із зовнішнього API та аналітика. Це дало змогу виміряти рівень стабільності для ключових сценаріїв користувача, а також чи відповідають встановленим вимогам користувача.

Функціональність проєкту було перевірено шляхом успішної компіляції коду проєкту в середовищі розробки Xcode на відповідність бажаній цільовій платформі iOS. Також було виконано ручну перевірку кількох основних варіантів використання: реєстрація та вхід, оновлення з ринку, рендеринг графіків, додавання монет до списку обраного, створення нотаток та налаштування сповіщень для вищезгаданих варіантів використання. Крім того, вони виконали ручне тестування локального сховища шляхом успішного перезапуску програми, в результаті чого обліковий запис користувача, список обраних монет, історія цін, кешовані ринкові дані, сигнали та будь-які локальні нотатки залишилися недоторканими.

Тестування показало, що основні модулі працюють коректно, а загальна логіка системи реалізована стабільно.

4.1.1. Перевірка функціональних можливостей системи

Щоб оцінити функціональність програми, було проведено кілька тестів на основі набору поширених сценаріїв, які найбільше відповідають тому, як фактичний користувач використовуватиме програму (наприклад, реєстрація,

вхід, перегляд ринку, пошук монет/активів, перевірка сторінок з детальною інформацією про активи, побудова графіків, керування обраним, керування нотатками, налаштування цінових сповіщень). Виконуючи кожен із цих сценаріїв, можна повніше оцінити систему з точки зору її поведінки в реальному використанні.

На основі результатів тестування було перевірено, чи відповідають результати, отримані системою, очікуванням. Наприклад, коли користувач успішно завершує реєстрацію, він потрапляє до свого головного інтерфейсу. Після оновлення ринку користувач бачить найактуальніший список монет (на відповідному ринку). Коли користувач виконує дію зі створення сповіщення, він зможе знайти створене сповіщення у відповідному розділі. Нижче наведено таблицю для уточнення результатів основної функції системи, як зазначено в плані тестування: Таблиця 4.1.

Таблиця 4.1

Результати перевірки функціональних можливостей системи

№	**Тестовий сценарій**	**Що має статися**	**Що сталося**	**Висновок**
1	Реєстрація нового користувача	Створено новий локальний обліковий запис, і новий користувач буде на головному екрані	Реєстрація пройшла успішно, і користувач перейшов на головний екран	Успішно
2	Вхід	Після введення правильної інформації користувач увійде в систему	Вхід пройшов успішно	Успішно
3	Оновлення ринку	Користувач отримує поточні дані ринку криптовалют через API	Користувач успішно отримав та відобразив поточні дані ринку.	Успішно
4	Пошук монети	Система може знайти монету, шукаючи її за назвою або символом	Пошук успішно завершено та повернуто	Успішно

			результати	

Таблиця 4.1 (закінчення)

№	Тестовий сценарій	Що має статися	Що сталося	Висновок
5	Переглянути детальну інформацію	Сторінка «Монета» відображає ринкові індикатори для вибраної монети	Дані про активи відображаються без помилок	Успішно
6	Створення графіка цін	Успішно відображено історію минулих цін за вибраний період часу. Зображення завершеного графіка	Графік успішно створено	Успішно
7	Додавання монети до обраного	Вибрану монету успішно додано до списку обраного	Нотатку, створену під час того ж сеансу, успішно збережено та отримано	Успішно
8	Збереження нотаток	Локальні нотатки успішно створено та збережено з вибраною монетою	Пошук успішно завершено та повернуто результати	Успішно
9	Створення цінового сповіщення	У системі успішно створено новий ціновий сигнал із заданим рівнем сповіщення	Сповіщення успішно згенеровано та розміщено у правильному місці	Успішно
10	Перезапустіть програму	Уся локальна інформація про користувача повернеться до останнього збереженого стану після видалення та повторної інсталяції	Усі дані, що повертаються користувачеві після запуску, – це вся відповідна інформація щодо його облікового запису.	Успішно

Таким чином, після успішного завершення тестів можна зробити висновок, що кожен з основних сценаріїв користувача був виконаний правильно та відповідав необхідним критеріям для системи, як описано в документації

проєкту. Це дає вагомі підстави вважати, що реалізована функціональність забезпечує стабільне та логічне функціонування програми для всіх основних випадків використання за її цільовим призначенням.

4.1.2. Аналіз результатів тестування

Тести показали, що реалізовані модулі мобільного застосунку працюють стабільно та забезпечують очікувану функціональність застосунку. Були протестовані важливі сценарії користувача, такі як авторизація, отримання ринкових даних, побудова графіків, робота з вибраними монетами або валютами, використання нотаток, отримання сповіщень та використання аналітичного модуля. У більшості випадків застосунок працював належним чином та поводився стабільно.

Також було підтверджено працездатність локального сховища після перезапуску програми та перевірки збереження облікового запису користувача, а також усіх вже збережених даних. Таким чином, локальне збереження та відновлення стану системи, схоже, працюють належним чином для звичайного використання.

Результати кожного з трьох проведених тестів показали, що зовнішній API та внутрішні модулі програми успішно взаємодіють один з одним. Було успішно завершено процес отримання даних з Binance API, їх відображення в інтерфейсі та збереження в локальному кеші або базі даних, якщо необхідно. Крім того, було помічено, що деякі функції програми залежать від зовнішнього сервісу (місця призначення API) та від стабільності мережевого з'єднання. Якщо API тимчасово недоступний, застосунок використовуватиме кеш або локальну копію; однак це може відображати деякі дані, які не будуть найточнішими для користувача.

Отже, тестування також пройшло успішно. Мобільний застосунок, що розробляється, правильно реалізував усі основні функції, необхідні для локального зберігання інформації, зв'язку із зовнішнім API та надає користувачеві доступ до зручних даних про ринок та аналітику. Це дає достатню

впевненість у тому, що застосунок працездатний і що робота над розвитком цього програмного рішення може продовжуватися в майбутньому.

4.2. Вимоги до апаратного та програмного забезпечення

Під час розробки та використання мобільного застосунку слід враховувати кілька моментів, що стосуються як програмних середовищ, так і апаратних платформ. Оскільки програмне забезпечення для цієї системи було створено для роботи в екосистемі Apple, важливо зібрати, скомпілювати та розгорнути його за допомогою відповідних інструментів (розробка) та пристроїв (апаратне забезпечення). Як середовище розробки, так і робоче середовище повинні відповідати мінімальним технічним вимогам.

Розробка операційної системи macOS вимагатиме використання Xcode та встановленого iOS SDK. Розробка застосунку буде здійснюватися в Swift 5.0, тоді як мінімальна версія iOS буде 17.0 [21]. Застосунок буде розроблено на Core Data/SQLite для зберігання структурованих даних та використовуватиме JSON-файли для кешування частини інформації для використання під час її експорту. Можна протестувати застосунок за допомогою симулятора iOS, який постачається з Xcode, або протестувати його на мобільному пристрої.

Мобільний пристрій iPhone з операційною системою iOS версії 17.0 або новішої з доступом до Інтернету є мінімальною вимогою для безпосереднього використання програми. Окрім забезпечення підключення до Інтернету для отримання даних ринку в реальному часі з Binance API, пристрій повинен мати певний обсяг вільної пам'яті для локального зберігання кешу, повідомлень, сповіщень та файлів, пов'язаних з послугами. Крім того, оскільки застосунок працює з графіками, ринковими списками та локальними записами, наполегливо рекомендується, щоб пристрій, який використовується, мав хорошу продуктивність під час запуску сучасних нативних програм iOS, без помітної затримки між введенням та виведенням.

Як результат, потреби в апаратному та програмному забезпеченні практично такі ж, як і для будь-якого іншого сучасного мобільного застосунку корпоративного рівня, що розробляється з нуля (тобто стандартного клієнт-серверного рішення). Вимоги до складної багаторівневої серверної системи або будь-якої сторонньої інфраструктури не включені до вимог для запуску цього застосунку, що робить як початкову розробку, так і подальшу експлуатацію програмного застосунку простішою та економічно ефективнішою.

4.3. Рекомендації щодо встановлення та використання застосунку

Застосунок буде встановлено в середовищі розробки за допомогою Xcode. Потрібно відкрити файл CryptoTracker.xcodeproj, вибрати схему застосунку, визначити цільовий пристрій або симулятор і виконати процес збірки та запуску. Якщо запускати TestFlight на фізичному iPhone, він повинен бути підключений до комп'ютера та мати дозвіл на використання тестових програм.

Коли користувач вперше відкриє застосунок, він побачить екран авторизації. Потім він зможе створити новий локальний обліковий запис або увійти до існуючого. Після успішного входу він отримає доступ до основного інтерфейсу застосунку, який містить п’ять вкладок: «Ринок», «Улюблені», «Сповіщення», «Аналітика» та «Налаштування». Таке розташування забезпечує легкий доступ до основних функцій застосунку з мінімальною кількістю додаткових кроків.

Потрібно постійно оновлювати інформацію про свій ринок для глобального відстеження даних за допомогою цього інструменту програми. Подумати про створення власного списку монет, за яким користувач хоче бути в курсі, щоб вони були легко доступні. Можна використовувати функцію локальних нотаток програми для відстеження спостережень. Налаштувати сповіщення для монет, які найбільше цікавлять. Якщо використовувати

аналітику з цієї програми, можна експортувати ці результати у форматі JSON, щоб зберегти їх або проаналізувати пізніше.

Ще одне: встановлення та використання програми буде простим, оскільки вона має визначену структуру, подібну до того, як працюють інші програми на пристроях Apple (тобто вона дотримується того самого базового процесу). Довідкові документи, доступні через програму, допоможуть забезпечити її належну роботу, і користувачі зможуть легко ставити запитання щодо ринкових даних.

4.4. Склад інсталяційного пакета

Тип інсталяційного пакета, який використовується для встановлення або подальшої розробки, є фактором, що визначає склад інсталяційного пакета, оскільки він базується на способі передачі програмного рішення користувачеві. Наприклад, стосовно бакалаврського проекту може бути корисним враховувати не лише завершену програму після її компіляції, але й усі файли проекту, необхідні для роботи або розробки програми після компіляції.

Все необхідне для встановлення цієї програми на Mac OS можна знайти у пакеті інсталяції: файли проекту для Xcode, весь вихідний код Swift, усі ресурси інтерфейсу користувача, включаючи моделі Core Data (для яких будуть зберігатися та підтримуватися будь-які постійні дані); допоміжні модулі/підтримувані файли для локального зберігання, кешування та експорту даних; файли конфігурації; графічні ресурси; та інші файли, необхідні для компіляції та успішного запуску в середовищі розробки.

Після розробки застосунку є можливість запустити його або через симулятор iOS, або на фактичному мобільному пристрої. Якщо є бажання перенести проєкт для майбутньої розробки, слід переконатися, що була збережена повна структура каталогів і файлів; зміна або відсутність частин цих структур може призвести до неправильної роботи модулів або до помилки під час спроби компіляції будь-якого або всіх модулів проєкту.

Таблиця 4.2

Склад інсталяційного пакета

№	Компонент	Призначення
1	CryptoTracker.xcodeproj	Файл проекту Xcode, необхідний для відкриття, створення та роботи з програмою
2	Вихідні файли Swift	Реалізація графічного інтерфейсу, бізнес-логіки, сервісів, моделей та модулів підтримки
3	Файли інтерфейсу користувача (UI)	Елементи екрана програми, а також елементи, що дозволяють взаємодію та навігацію
4	Основна модель даних Core Data	Представлення локальних записів та зв’язків, що використовуються для зберігання даних
5	Локальне сховище SQLite	Зберігання інформації користувача, улюблених монет, сповіщень, історичних даних та інших даних, організованих у структури
6	Файли кешу JSON	Зберігання ринкових даних, нотаток у вашому обліковому записі та іншої допоміжної інформації
7	Файли експорту JSON	Отримання результатів аналітичних процесів через файл
8	Сервіс та графічні ресурси	Піктограми/Кольорові елементи/Додаткові налаштування та різні інші типи ресурсів для використання додатком
9	API для зовнішніх даних Binance	Доступне місце для отримання поточних ринкових даних від Binance для використання додатком.
10	Середовище Xcode та iOS SDK	Інструменти, необхідні для створення, запуску та тестування програми.

Дані, представлені в таблиці 4.2, показують, що окрім самої програми, інсталяційний пакет також містить усі необхідні програмні компоненти (програму, файли тощо), необхідні для належної роботи програми та забезпечення її успішної роботи в майбутньому. Повну структуру програмного проєкту подано у додатку Б.

Таким чином, інсталяційний пакет охоплює всі основні компоненти, необхідні для запуску програми, роботи з її локальними даними та подальшої підтримки в середовищі розробки.

4.5. Можливості подальшого розвитку системи

Новий мобільний застосунок містить основні необхідні функції для відстеження та аналізу цін на криптовалюту; проте існує значна можливість для покращення функціональності та технічних аспектів різних компонентів застосунку. Постійна волатильність на ринках криптовалют та зростаючий попит користувачів на такі додатки свідчать про те, що існує постійна потреба в їх розробці.

Однією з перспективних для подальшого розвитку сфер є підтримка кількох зовнішніх джерел ринкових даних замість одного API Binance [23]. Це надасть більше даних з різних бірж і більше можливостей для порівняння інформації між біржами, що призведе до підвищення надійності отримання даних і меншої залежності від одного сервісу. Зрештою, це може призвести до можливості автоматичного перемикання між джерелами даних, якщо одне з них стане недоступним.

Розробка аналітичного модуля. Необхідно створити більш надійний набір графічних та порівняльних інструментів, а також використовувати альтернативні способи відображення динамічних змін ринку за допомогою цих інструментів. Крім того, слід включити додаткові статистичні вимірювання для глибшого вивчення ефективності різних класів активів. Це допоможе покращити функції програми як для щоденного моніторингу, так і для детальної візуалізації ринкових тенденцій з плином часу.

Важливо докласти додаткових зусиль до сторони кінцевого користувача програми. Серед областей, які мають великий потенціал для покращення, є: синхронізація даних на кількох платформах, можливість зберігати свої облікові записи в хмарному сховищі, створення розширених push-сповіщень, підтримка віджетів на головному екрані iOS та розробка простого способу керування

портфелями активів для користувачів. Додавання цих функцій зробить програму простішою у використанні в реальному житті та забезпечить більшу гнучкість для користувачів.

Це означає, що існує багато можливостей для подальшого вдосконалення як функціональних, так і технічних аспектів. Його базова структура забезпечує спосіб поступового збільшення функціональності програми, а не вимагає фундаментального перероблення існуючих елементів, що є головною перевагою для можливих удосконалень програмного рішення.

ВИСНОВКИ

Під час написання бакалаврської кваліфікаційнорї роботи було створено мобільний застосунок, який би дозволив користувачам відстежувати та аналізувати ціни на криптовалюти спеціально для пристроїв iOS. Створення проходило шляхом ретельного аналізу протягом усього процесу. Було проведено аналіз теми, визначення того, як працює ринок криптовалют, звідки отримувати дані з ринку, а також раніше створених програмних рішень, що використовуються для відстеження цифрових активів. У результаті це допомогло розробити обґрунтовані вимоги та основні функції, які будуть частиною майбутньої системи.

Теоретична частина дослідження визначила як функціональні, так і нефункціональні вимоги до застосунку, побудувала моделі, що описують предметну область, а також встановила, як має бути структурована інформація для підтримки системи. Аналіз потоку інформації, типи даних, які використовуватимуться для введення в систему, логічне моделювання даних та способи організації локального зберігання даних були важливими компонентами, пов'язаними разом у проектуванні архітектури для підтримки реалізації програмного модуля.

Розробка програмного забезпечення використовувала Swift як мову програмування, SwiftUI Framework як основу для макета інтерфейсу користувача програми, MVVM для архітектурного дизайну, а також серверні системи (Core Data) та інструменти JSON для ефективного локального зберігання та обробки даних. Цей набір інструментів дозволив створити справді нативний мобільний застосунок, який має чітко визначену структуру, а також можливість локального зберігання даних, підтримку кешування та можливість підключення до зовнішнього API Binance.

Розроблений застосунок містить усі основні функціональні модулі для задоволення потреб користувача, встановлених офіційними вимогами. Ці модулі

дозволяють користувачу зареєструватися та увійти в програму, переглядати список криптовалют, якими зараз торгують, шукати певну монету, фільтрувати/шукати за типом торгівлі, проводити поглиблену оцінку активів, переглядати активи на графіках, а також перераховувати улюблені активи та створювати нотатки про них локально. Також можна створювати цінові сповіщення на основі встановлених критеріїв. Крім того, застосунок містить аналітичну програму, яка експортуватиме всі аналітичні результати у вигляді даних JSON. Ці функції надають користувачеві актуальну інформацію про курси валют, а також найефективніший спосіб взаємодії з ринковими даними відповідно до індивідуальних потреб користувача.

Щоб забезпечити можливість отримання як поточної інформації із зовнішнього API, так і попередньо збереженої локальної інформації, особливу увагу було приділено локальному зберіганню інформації, а також підтримці різних сценаріїв резервного копіювання під час процесу оцінювання, що покращило продуктивність системи, а також зменшило залежність від мережевого підключення. Це надзвичайно важливий фактор для мобільних застосунків, оскільки вони повинні бути максимально зручними для використання в щоденному контексті.

Згідно з результатами тестування, усі ключові модулі (включно з логічним блоком) працюють належним чином. Також було перевірено, що модуль авторизації, оновлення ринку, побудова графіків, обране, нотатки, сповіщення/оголошення та аналітичні модулі системи працювали належним чином. Загалом, результати тестування показують, що нове розроблене програмне рішення повністю функціональне, надійне та готове до використання.

Мета цього дослідження була досягнута шляхом розробки робочого мобільного , який дозволить його подальший розвиток, включаючи покращення аналітичних можливостей, підключення різних типів потоків ринкових даних, покращення процесів сповіщення та розробку спеціальних інструментів для користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Apple. Swift Documentation [Електронний ресурс]. Режим доступу: https://www.swift.org/documentation/

2. Apple Developer. SwiftUI [Електронний ресурс]. Режим доступу: https://developer.apple.com/documentation/swiftui/

3. Apple Developer. Core Data [Електронний ресурс]. Режим доступу: https://developer.apple.com/documentation/coredata/

4. Apple Developer. UserDefaults [Електронний ресурс]. Режим доступу: https://developer.apple.com/documentation/foundation/userdefaults/

5. Apple Developer. UserNotifications [Електронний ресурс]. Режим доступу: https://developer.apple.com/documentation/usernotifications/

6. Apple Developer. Charts [Електронний ресурс]. Режим доступу: https://developer.apple.com/documentation/charts/

7. Binance Developers. Binance Spot API Documentation [Електронний ресурс]. Режим доступу: https://developers.binance.com/docs/binance-spot-api-docs

8. CoinMarketCap. About CoinMarketCap [Електронний ресурс]. Режим доступу: https://coinmarketcap.com/about/

9. CoinGecko. About CoinGecko [Електронний ресурс]. Режим доступу: https://www.coingecko.com/en/about

10. TradingView. About TradingView [Електронний ресурс]. Режим доступу: https://www.tradingview.com/about/

11. DGlibrary НУБіП України. Цифрова бібліотека НУБіП України [Електронний ресурс]. Режим доступу: https://dglib.nubip.edu.ua/home

12. Якимов О.С. Система аналізу загроз інформаційним ресурсам мобільних пристроїв : дипломна робота ... магістра : 122 «Комп'ютерні науки». Київ, 2021. 56 с. [Електронний ресурс]. Режим доступу: https://dglib.nubip.edu.ua/handle/123456789/2742

13. Громик Н.В. Моделювання та аналітична підтримка поведінки клієнтів оператора мобільного зв'язку : дипломна робота ... магістра : 051 «Економіка». Київ, 2021. 64 с. [Електронний ресурс]. Режим доступу: https://dglib.nubip.edu.ua/handle/123456789/3106

14. CoinGecko. Cryptocurrency Prices by Market Cap [Електронний ресурс]. Режим доступу: https://www.coingecko.com/

15. CoinMarketCap. What are cryptocurrencies? [Електронний ресурс]. Режим доступу: https://support.coinmarketcap.com/hc/en-us/articles/360018692832-What-are-cryptocurrencies

16. Apple Developer. Human Interface Guidelines [Електронний ресурс]. Режим доступу: https://developer.apple.com/design/human-interface-guidelines

17. Apple Developer. Xcode Documentation [Електронний ресурс]. Режим доступу: https://developer.apple.com/documentation/xcode

18. Apple Developer. URLSession [Електронний ресурс]. Режим доступу: https://developer.apple.com/documentation/foundation/urlsession

19. Apple Developer. FileManager [Електронний ресурс]. Режим доступу: https://developer.apple.com/documentation/foundation/filemanager

20. Apple Developer. CryptoKit [Електронний ресурс]. Режим доступу: https://developer.apple.com/documentation/cryptokit

21. Apple Developer. App Review Guidelines [Електронний ресурс]. Режим доступу: https://developer.apple.com/app-store/review/guidelines/

22. Binance Developers. Market Data endpoints [Електронний ресурс]. Режим доступу: https://developers.binance.com/docs/binance-spot-api-docs/rest-api/market-data-endpoints

23. Binance Developers. WebSocket Streams [Електронний ресурс]. Режим доступу: https://developers.binance.com/docs/binance-spot-api-docs/web-socket-streams

24. IETF. RFC 8259. The JavaScript Object Notation (JSON) Data Interchange Format [Електронний ресурс]. Режим доступу: https://datatracker.ietf.org/doc/html/rfc8259

25. Ecma International. ECMA-404. The JSON Data Interchange Format [Електронний ресурс]. Режим доступу: https://ecma-international.org/publications-and-standards/standards/ecma-404

26. OWASP. Mobile Application Security Verification Standard [Електронний ресурс]. Режим доступу: https://mas.owasp.org/MASVS

27. ISO. ISO/IEC 25010:2023 Systems and software engineering — Product quality model [Електронний ресурс]. Режим доступу: https://www.iso.org/standard/78176.html

28. Swift.org. API Design Guidelines [Електронний ресурс]. Режим доступу: https://www.swift.org/documentation/api-design-guidelines

ДОДАТОК А

ЛІСТИНГ ОСНОВНИХ ПРОГРАМНИХ МОДУЛІВ

Лістинг А.1 – Реєстрація та вхід користувача

```
func register(login: String, email: String, password: String, confirmPassword: String)
{
   guard !login.isEmpty, !email.isEmpty, !password.isEmpty else { return }
   guard password == confirmPassword else { return }

   do {
      currentUser = try dataManager.register(
         login: login,
         email: email,
         password: password
      )
      authError = nil
   } catch {
      authError = error.localizedDescription
   }
}

func login(identifier: String, password: String) {
   guard !identifier.isEmpty, !password.isEmpty else { return }

   do {
      currentUser = try dataManager.login(
         identifier: identifier,
         password: password
      )
      authError = nil
   } catch {
      authError = error.localizedDescription
   }
}
```

Лістинг А.2 – Отримання та пошук ринкових даних

```
func fetchCoins(limit: Int? = nil) async throws -> [Coin] {
   let tickers = try await fetchEligibleTickers()

   let sorted = tickers.sorted {
      $0.quoteVolume > $1.quoteVolume
   }
```

```
    return Array(sorted.prefix(limit ?? maxVisibleCoins)).map(makeCoin)
}

func searchCoins(matching query: String) async throws -> [Coin] {
    let normalizedQuery = query.uppercased()
    let tickers = try await fetchEligibleTickers()

    return tickers
        .filter { $0.symbol.contains(normalizedQuery) }
        .sorted { $0.quoteVolume > $1.quoteVolume }
        .map { makeCoin(from: $0) }
}
```

Лістинг А.3 – Оновлення даних та кешування

```
func refresh(settings: AppSettings) async {
    isLoading = true
    defer { isLoading = false }

    do {
        let freshCoins = try await apiService.fetchCoins()
        coins = freshCoins

        let cache = MarketCache(
            updatedAt: .now,
            coins: freshCoins,
            histories: histories,
            notes: Array(notes.values)
        )

        try fileStore.saveCache(cache)
        lastUpdated = cache.updatedAt
        evaluateAlerts()
    } catch {
        errorMessage = settings.text(.apiRefreshError)
    }
}
```

Лістинг А.4 – Завантаження історії цін

```
func loadHistory(for coin: Coin, period: ChartPeriod, settings: AppSettings) async {
```

```
    let key = historyKey(for: coin.symbol, period: period)

    do {
        let history = try await apiService.fetchHistory(
            symbol: coin.symbol,
            period: period
        )

        histories[key] = history
        try dataManager.replacePriceHistory(for: coin, with: history)
        try persistCache()
    } catch {
        if let storedHistory = try? dataManager.loadPriceHistory(for: coin) {
            histories[key] = storedHistory
        } else {
            errorMessage = settings.format(.historyLoadError, coin.name)
        }
    }
}
```

Лістинг А.5 – Обране та цінові сповіщення

```
func toggleFavorite(for coin: Coin, settings: AppSettings) {
    do {
        if dataManager.isFavorite(symbol: coin.symbol) {
            try dataManager.removeFavorite(symbol: coin.symbol)
        } else {
            try dataManager.saveFavorite(for: coin)
        }

        favorites = try dataManager.loadFavorites()
    } catch {
        errorMessage = settings.text(.favoriteSaveError)
    }
}

func saveAlert(for coin: Coin, targetPrice: Double, condition: AlertCondition) {
    do {
        try dataManager.saveAlert(
            for: coin,
            targetPrice: targetPrice,
            condition: condition
```

```
        )

        alerts = try dataManager.loadAlerts()
    } catch {
        errorMessage = error.localizedDescription
    }
}
```

Лістинг А.6 – Експорт аналітичного звіту

```
private func makeAnalyticsExportFile() -> URL? {
    let payload = AnalyticsExportPayload(
        exportedAt: .now,
        period: comparisonPeriod.rawValue,
        marketSummary: .init(
            totalCoins: viewModel.coins.count,
            favorites: viewModel.favorites.count,
            alerts: viewModel.alerts.count
        ),
        gainers: gainers.map(exportCoin),
        volumeLeaders: volumeLeaders.map(exportCoin),
        comparison: comparisonSeries.map(exportPoint)
    )

    let encoder = JSONEncoder()
    encoder.outputFormatting = [.prettyPrinted, .sortedKeys]

    let data = try? encoder.encode(payload)
    let url = FileManager.default.temporaryDirectory
        .appendingPathComponent("analytics-export.json")

    try? data?.write(to: url)
    return url
}
```

Лістинг А.7 – Збереження налаштувань користувача

```
final class AppSettings: ObservableObject {
    @Published var language: AppLanguage {
                        didSet { UserDefaults.standard.set(language.rawValue, forKey:
Keys.language) }
    }
```

```
    @Published var themeMode: AppThemeMode {
                    didSet { UserDefaults.standard.set(themeMode.rawValue, forKey:
Keys.themeMode) }
    }

    @Published var prioritizeFavorites: Bool {
                        didSet { UserDefaults.standard.set(prioritizeFavorites, forKey:
Keys.prioritizeFavorites) }
    }

    @Published var compactCharts: Bool {
        didSet { UserDefaults.standard.set(compactCharts, forKey: Keys.compactCharts)
}
    }
}
```

ДОДАТОК Б

СТРУКТУРА ПРОГРАМНОГО ПРОЄКТУ

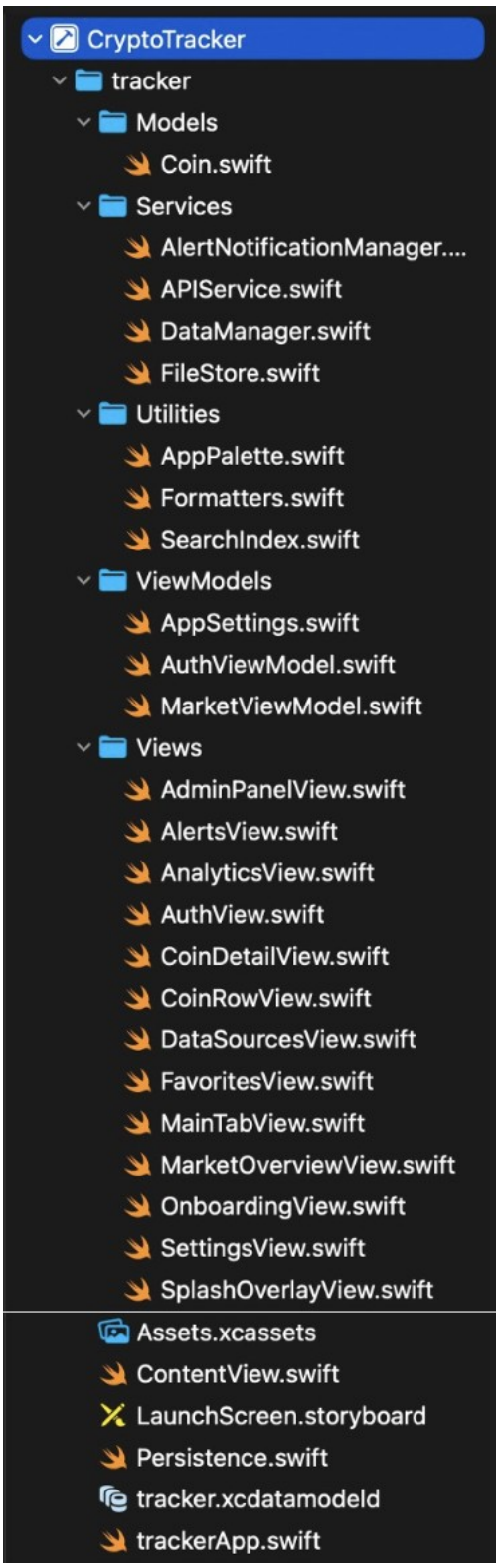


Рис. Б.1 Структура програмного проєкту CryptoTracker у середовищі Xcode

На рисунку наведено структуру програмного проєкту CryptoTracker у середовищі Xcode. Проєкт поділено на окремі каталоги відповідно до призначення файлів: Models містить моделі даних, Services відповідає за роботу з API, локальним сховищем і сповіщеннями, Utilities містить допоміжні

інструменти форматування та пошуку, ViewModels реалізує логіку взаємодії між даними й інтерфейсом, а Views містить екрани мобільного застосунку.

Окремо в структурі проєкту розміщено ресурси застосунку Assets.xcassets, стартовий екран LaunchScreen.storyboard, файл Persistence.swift для налаштування Core Data, модель локальної бази даних tracker.xcdatamodeld та головний файл запуску trackerApp.swift.

ДОДАТОК В

ПРИКЛАДИ JSON-ДАНИХ ТА API-ВІДПОВІДЕЙ

Приклад відповіді Binance API:

```
{
 "symbol": "BTCUSDT",
 "lastPrice": "104250.50",
 "priceChangePercent": "2.31",
 "highPrice": "105100.00",
 "lowPrice": "101800.00",
 "quoteVolume": "3250000000.00"
}
```

Приклад локального JSON-кешу:

```
{
 "updatedAt": "2026-05-15T12:30:00Z",
 "coins": [
  {
   "symbol": "BTCUSDT",
   "name": "Bitcoin",
   "price": 104250.50,
   "change24h": 2.31,
   "high24h": 105100.00,
   "low24h": 101800.00,
   "volume24h": 3250000000.00
  }
 ],
 "histories": {},
 "notes": [
  {
   "symbol": "BTCUSDT",
```

```
      "text": "Основний актив для спостереження",
      "updatedAt": "2026-05-15T12:35:00Z"
    }
  ]
}
```


Приклад експорту аналітичного звіту:


```
{
  "exportedAt": "2026-05-15T12:40:00Z",
  "period": "24H",
  "marketSummary": {
    "totalCoins": 250,
    "favorites": 3,
    "alerts": 2
  },
  "gainers": [
    {
      "symbol": "ETHUSDT",
      "name": "Ethereum",
      "price": 3850.20,
      "change24h": 4.12
    }
  ],
  "volumeLeaders": [
    {
      "symbol": "BTCUSDT",
      "name": "Bitcoin",
      "price": 104250.50,
      "volume24h": 3250000000.00
```

```
        }
      ]
    }
```

ДОДАТОК Д

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій Декларація про академічну доброчесність та використання ШІ

Я, Карюк Даниїл Юрійович, здобувач(ка) НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота на тему

«Розробка мобільного застосунку для моніторингу та аналітики курсів криптовалют» 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 (обрати необхідне):

[ ] інструменти генеративного ШІ не використовувалися.

[✓] інструменти ШІ (ChatGPT) 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.

«22»_____травня____ 2026 р. _____ Даниїл КАРЮК