КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра теоретичної кібернетики

Кваліфікаційна робота

на здобуття освітнього ступеня бакалавра
за спеціальністю 122 Комп'ютерні науки
на тему:

РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ ДЛЯ ВІДСТЕЖЕННЯ СТАНУ БАНКІВСЬКИХ РАХУНКІВ

Виконав студент 4-го курсу
Антон ЛЯННОЙ

(підпис)

Науковий керівник:
доцент кафедри теоретичної кібернетики
кандидат фіз.-мат. наук,
Андрій СТАВРОВСЬКИЙ

(підпис)

Засвідчую, що в цій роботі
немає запозичень з праць інших авторів
без відповідних посилань.
Студент

(підпис)

Роботу розглянуто й допущено до захисту
на засіданні кафедри теоретичної
кібернетики
«1» червня 2022 р.,
протокол № 11
Завідувач кафедри
доктор фіз.-мат. наук, професор
Юрій КРАК

(підпис)

КИЇВ-2022

РЕФЕРАТ

Обсяг роботи 54 сторінки, 37 зображень, 13 джерел посилань.

МОБІЛЬНИЙ ДОДАТОК, ОНЛАЙН-БАНКІНГ, ФІНТЕХ, РЕАКТИВНЕ ПРОГРАМУВАННЯ, КРОСПЛАТФОРМНІСТЬ, API-ІНТЕРФЕЙС.

Об’єктом роботи є процес проєктування та програмування мобільного додатка з використанням найбільш сучасних підходів розробки програмного забезпечення. Предметом роботи є кросплатформний додаток, основною функцією якого є відстеження стану банківських рахунків користувача. Метою роботи є створення додатка із заявленою функціональністю.

Методи розробки: UX/UI-проєктування, програмування. Інструменти розроблення: онлайн-середовище для розробки інтерфейсів та прототипування Figma, мова програмування широкого призначення TypeScript, кросплатформний фреймворк для розробки нативних мобільних додатків React Native, редактор вихідного коду VS Code, інтегровані середовища розробки Xcode для платформи iOS та Android Studio для платформи Android Studio. Результати роботи: створено кросплатформний додаток, що має можливість відстежувати стан під’єднаних банківськіх рахунків, а також стан уведених вручну користувацьких активів та пасивів.

ЗМІСТ

ВСТУП

Актуальніть роботи. За минуле десятиріччя сфера онлайн-банкінгу та фінтеху — фінансових технологій — зазнала вкрай швидкого розвитку, помітним результатом цього є те, що в наш час майже кожен відомий банк обов’язково має свій веб- та мобільний додаток для клієнтів.

Зручність у проведенні банківських операцій у режимі онлайн має настільки багато переваг, що у багатьох країнах з’являється чимало банків, які працюють виключно дистанційно і навіть не мають жодних відділень.

Як правило, кожна особа володіє більше ніж одним банківським рахунком, більше того, рахунками, оформленими більше, ніж у одному банку, що ускладнює аналіз її витрат та підрахунок сумарного бюджету, яким ця особа володіє. Виникає потреба в автоматизації процесу збирання даних про залишки коштів на її рахунках та зведення інформації про них в єдиному місці задля зручності та економії часу.

Мета й завдання роботи. Головна ціль даної роботи - спроєктувати мобільний додаток, який за допомогою підключення до відкритих API-інтерфейсів банків буде відображати інформацію про залишки на рахунках користувача у цих банках. Планується також додати ручні інтерфейси для введення інформації про небанківські активи і пасиви, що дозволить користувачу повністю слідкувати за усіма каналами розподілу свого бюджету всередині єдиного додатка.

Сфери застосування. Готовий мобільний додаток потенціально матиме вкрай широке застосування серед користувачів, зацікавлених у відстеженні своїх банківських рахунків — тобто як фізичних, так і юридичних осіб.

1 ЗАГАЛЬНИЙ ОГЛЯД ТА ПОПЕРЕДНЄ ПЛАНУВАННЯ МАЙБУТНЬОЇ АРХІТЕКТУРИ

1.1 Взаємодія з серверними структурами банків

Принцип роботи будь-якого додатка-агрегатора — збір інформації із різних джерел та представлення її у вигляді єдиної сутності, зазвичай списку, оформленого у будь-якій формі. Для підключення цих джерел додатки використовують спеціальні програмні інтерфейси (API-інтерфейси), запропоновані джерелом, які можуть бути як відкритими, так і закритими.

Банки з великою кількістю клієнтів, як правило, мають відкриті публічні API-інтерфейси, які використовуються як у платіжних, так і моніторингових сервісах. Ці інтерфейси зручно підключити до мобільного додатка, оскільки вони використовують звичайні HTTP-запити для обміну інформацією між сервером та клієнтом.

Серед українських банків найбільшими за кількістю клієнтів є державний банк Приватбанк та мобільний банк Монобанк, що є пакетом послуг акціонерного товариства Універсал Банк. Кожен з них має публічні API, що дозволяють перегляд балансів на рахунках користувачів. Монобанк має єдиний інтерфейс для рахунків фізичних осіб та для рахунків фізичної особи-підприємця зареєстрованих на цих користувачів. [1] Приватбанк має 2 різні інтерфейси — для карткових рахунків фізичних осіб [2] та для будь-яких рахунків юридичних осіб [3].

Для авторизації і виконання запитів користувача кожен з цих сервісів вимагає створення користувачем спеціальних ключей доступу (токенів) у форматі текстового рядка і подальшого їх включення у кожний запит. Отже, щоб підключити до відстеження у додатку один або декілька рахунків у кожного банку, користувачу потрібно буде згенерувати відповідні токени і

зберегти їх у постійній пам’яті додатку. Додаток повинен мати сторінку для додавання нового рахунку, яка міститеме необхідні поля для вводу токенів, яких потребує банк, а також інстукцію для користувача з генерування цих токенів.

Після отримання ключів, для отримання інформації про баланси додаток має автоматично підставляти збережені токени у всі наступні запити до банку, які робить користувач.

1.2 Отримання, зберігання та відображення інформації

Оскільки уся інформація, отримана додатоком як від користувача, так і від банків, є конфіденційною і має бути доступна лише безпосередньо користувачу, додаток має бути захищений паролем. Додатково, за бажанням користувача, вхід у додаток може проводитися за біометричними даними — такими як відбиток пальця або модель обличчя.

Щоб уникнути несанкціонованого доступу третіх осіб до інформації додатка, що зберігається у постійній пам’яті — тобто пам’яті мобільного пристрою — вона має бути збережена у зашифрованому вигляді в спеціальних сховищах, наданих для цього операційною системою. Доступ до цих даних має бути лише у самого додатка.

Під час обміну даними між банком та додатком дані не мають бути доступні третім особам, тому вся інформація, що міститимется у всіх вхідних та вихідних запитах має обов’язково бути зашифрована. Можливість розшифрувати її повинні мати тільки безпосередньо сам користувач, а також банк, із яким здійснюється поточний обмін даними.

Для введення токенів банків, налаштування параметрів додатка та введення іншої інформації у додатку має бути імплементовано сторінку налаштувань. Відображення згрупованої інформації, що отримана від банків — оскільки це є головною функцією додатка — має знаходитися на головному екрані. Цей екран має містити окремо розбиті дані за балансами

банківських рахунків, уведені за бажанням користувача, дані активів та пасивів, а також підрахований сумарний бюджет, яким, за даними додатка, володіє користувач.

Оскільки рахунок користувача може бути оформлено у тому числі в іноземних валютах, додаток має вміти їх обробляти, конвертувати між собою за поточними курсами цих валют, тощо. Поточні курси валют також мають оновлюватися при кожному запуску додатка.

Кожного разу, як користувач заходить у додадок, інфомація від банків має автоматично оновлюватися шляхом повторних запитів. Також користувач повинен мати можливість запросити оновлення інформації безпосередньо під час користування додатком.

2 ОГЛЯД ІНСТРУМЕНТІВ ДЛЯ ПРОЄКТУВАННЯ ТА РОЗРОБКИ СИСТЕМИ

2.1 Візуальний інтерфейс

Для проєктування візуального інтерфейсу додатка було обрано Figma — векторний онлайн-сервіс розробки інтерфейсів та прототипування з можливістю організації спільної роботи. Сервіс працює у двох форматах: як браузерний або ж як десктопний додаток. Усі файли, які створює та редагує користувач, зберігаються онлайн. [4] До переваг сервісу можна віднести швидкість, зручність у користуванні, велику кількість сторонніх плагінів для різних потреб, а також те, що створеними прототипами можна ділитися за посиланням.

2.2 Програмна розробка

2.2.1 Загальна архітектура

Оскільки поставленою метою роботи є розробка кросплатформного додатка, для програмування було використано один із найбільш сучасних фреймворків для розробки додатків на мові JavaScript React Native. Цей фреймворк дозволяє використовувати можливості бібліотеки React для програмування нативних мобільних додатків, що мають доступ до системних програмних інтерфейсів платформ. [5] Програмування додатків на React Native дозволяє не програмувати різні версії під кожну платформу, натомість дає можливість використовувати один і той самий код під різні платформи.

Для покращення якості розробки коду було використано типізовану версію мови JavaScript — TypeScript. Серед переваг цієї мови над

класичною JavaScript є можливість статичної типізації, підтримка використання повноцінних класів та підтримка підключення модулів. Ці характеристики мають підвищувати швидкість розробки, прочитність, рефакторинг і повторне використання коду, швидкість пошуку помилок на етапі розробки та компіляції, а також загальну швидкодію програми. [6]

Для досягнення цілей, яких неможливо досягнути лише за допомогою TypeScript коду та бібліотек для React Native, було проведено роботу із нативними файлами для платформ із використанням мов програмування, призначених відповідно для кожної платформи. Це, зокрема, об'єктно-орієнтовані мови Java для платформи Android та Objective-C для платформи iOS.

Для компіляції додатків кожна платформа має своє спеціальне власне середовище. Android використовує Android Studio — середовище, що побудовано на базі вихідного коду продукту IntelliJ IDEA Community Edition, який розвивається компанією JetBrains. [7] Для розробки під iOS компанією Apple розроблено середовище Xcode. Обидва середовища містять в своєму арсеналі набори емуляторів пристроїв відповідних платформ для можливості тестування зовнішнього вигляду та функціоналу додатків на різних пристроях.

Для відладки додатків на React Native можна використовувати інструменти відладки браузера Google Chrome, який віддалено підключається до JavaScript-потоку.

2.2.2 Зберігання даних

Оскільки дані мають зберігатися у зашифрованому вигляді та бути недоступними для третіх осіб, найкраще зберігати їх у спеціально відведених для цього кожною платформою сховищах — це Keychain для платформи iOS та EncryptedSharedPreferences для платформи Android.

Keychain — це система управління паролями, основною задачею якої є зберігання конфіденційних даних. Зазвичай вона використовується для зберігання паролів, приватних ключів, сертифікатів та захищених записів. [8]

EncryptedSharedPreferences — це інтерфейс для отримання та редагування інформації у якому ключі і значення є зашифрованими. [9][10]

Для взаємодії з цима сховищами за допомогою TypeScript напряму з React Native було обрано бібліотеку react-native-encrypted-storage. Ця бібліотека експортує об'єкт, який має лише 2 методи: запис значення до зашифрованого сховища та його зчитування.

2.2.3 Управління компонентами

Програмування із використанням бібліотеки React базується на створенні компонентів — окремих частин інтерфейсу, які є ієрархічно пов'язаними між собою. Компонент можна оголосити двома способами — як клас або ж як функцію. Довгий час функціональні компоненти мали обмежені можливості у порівнянні із класовими, але після того, як у бібліотеці з'явилася можливість використання хуків — спеціальних функцій всередині компонента, які можуть бути або вбудованими, або створеними власноруч, функціональні компоненти набули більш значного поширення і стали своєрідним стандартом програмування з React.

Використання хуків у функціональних компонентах дозволяє легко керувати їх станами, процесом рендерингу та доступом до зовнішніх об'єктів, а також дозволяє скоротити кількість коду всередині оголошення компонента шляхом винесення функціоналу, що часто повторюється, в окремий хук. Багато бібліотек для React та React Native експортують власні хуки для свого функціоналу.

2.2.4 Управління станами додатка

У будь-яких комплексних додатках є присутньою проблема керування станами, оскільки при великій кількості компонентів та сторінок (які в свою чергу теж є компонентами) всередині цього додатка дуже складно підтримувати постійний доступ до інформації про ті чи інші стани шляхом її прямої передачі між компонентами. Виникає суттєва потреба в наявності спеціального — загального — стану додатка, в якому б цетралізовано зберігалася уся інформація, і яку можна було б отримати або змінити із будь-якої частини цього додатка.

JavaScript має декілька бібліотек, що надають таку можливість. Серед них однією з найкращих і найпоширеніших у комерційних проєктах є бібліотека Redux. Основною концепцією Redux є так зване єдине джерело істини — стан всього застосунку зберігається у вигляді деревіа об’єктів в єдиному сховищі. Цей стан можна змінити лише за допомогою виокремлення конкретних дій, що, в свою чергу, гарантує, що ні перегляди, ні зворотні виклики мережі ніколи не будуть змінювати стан. Безпосередньо зміни стану вібуваються за допомогою редьюсера — функції, що приймає попередній стан, дію та повертає наступний стан. Редюсери визначають, як стан додатка змінюється у відповідь на запущені дії. Оскільки редьюсери — це лише функції, можна контролювати порядок їх надсилання, передавати додаткові дані або створювати повторювані редьюсери для однотипних послідовностей дій. [11]

3 РОЗРОБКА СИСТЕМИ

3.1 Визначення функціоналу для імплементації

Для проєктування візуального прототипу додатка і зокрема самого додатка потрібно описати чіткий перелік функцій, які мають бути в ньому наявні.

Зважаючи на доступні можливості, а також доцільність імплементації, було виділено наступні фунції, що мають міститися у додатку:

а) вхід за допомогою пароля та/або біометричних даних;

б) відображення поточних курсів валют НБУ та підключених банків;

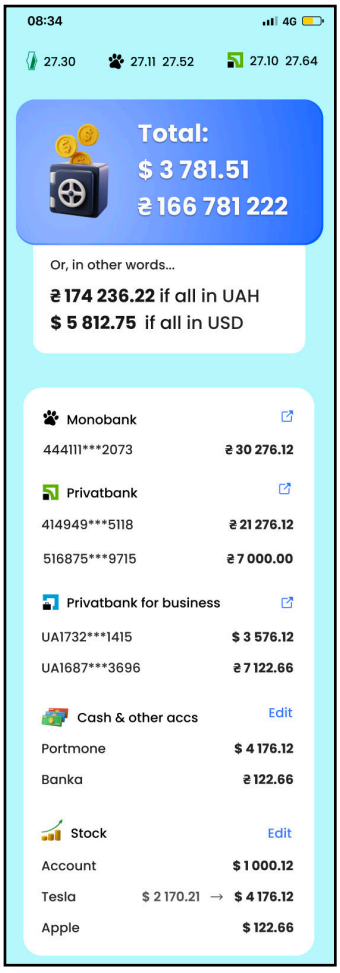

Рисунок 3.1 — Головна сторінка, звичайна тема

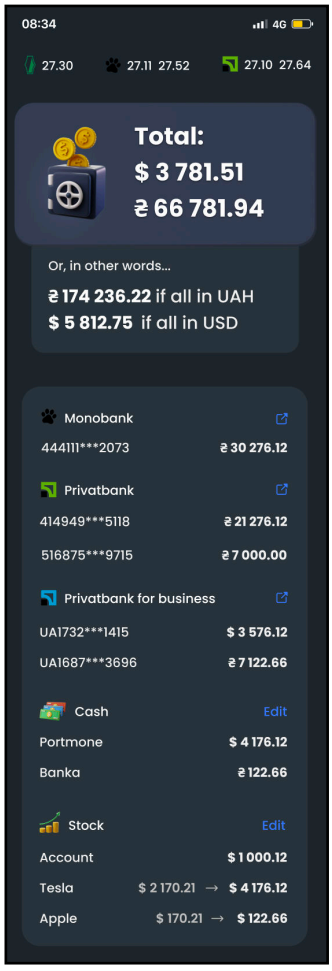

Рисунок 3.2 — Головна сторінка, нічна тема

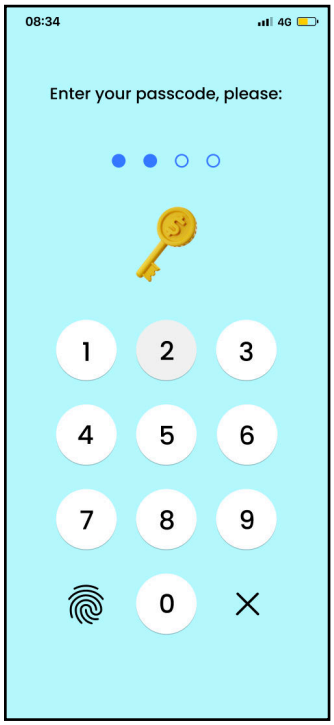

Рисунок 3.3 — Сторінка вводу пароля

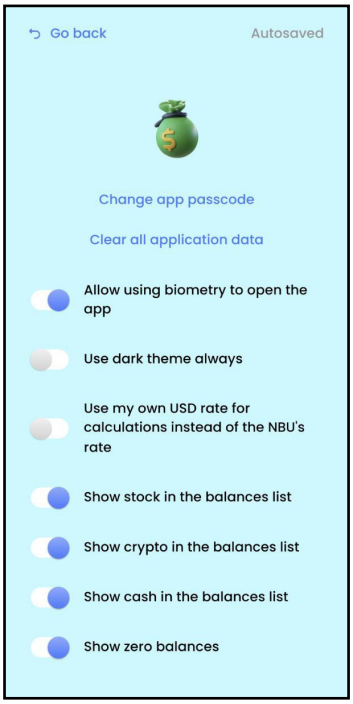

Рисунок 3.4 — Сторінка налаштувань

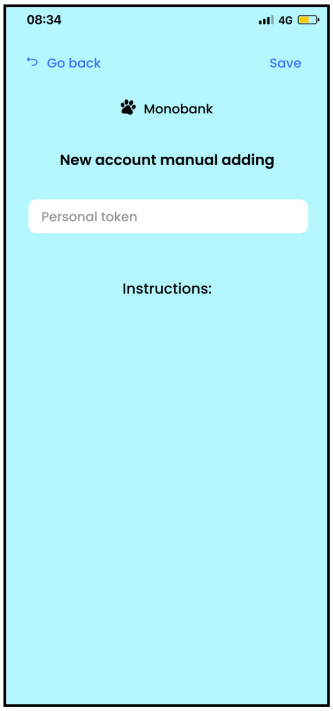

Рисунок 3.5 — Сторінка вводу токена банка

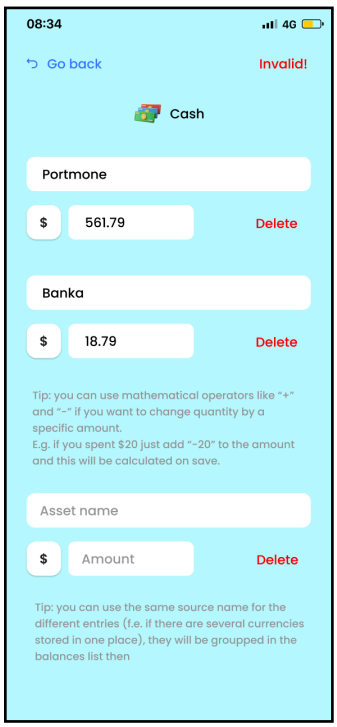

Рисунок 3.6 — Сторінка вводу даних про готівку

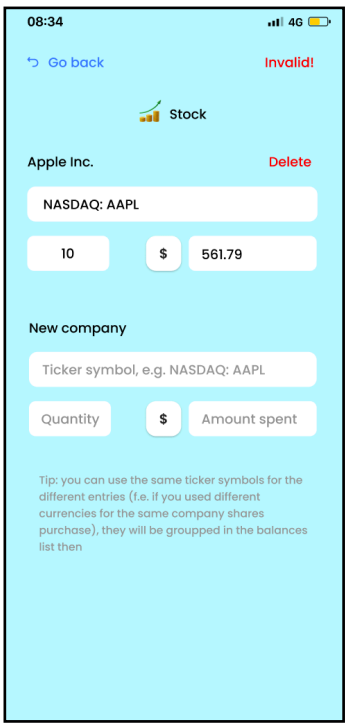

Рисунок 3.7 — Сторінка вводу даних про акції

Рисунок 3.8 — Сторінка відображення помилок з серверів банку

в) відображення списку балансів користувача;

1) баланси банківських рахунків користувача (за наявності);

- банку Монобанк (для фізичних осіб та ФОП);
- банку Приватбанк (для фізичних осіб);
- банку Приватбанк (для юридичних осіб);

2) баланси активів та пасивів користувача;

- поточна вартість акцій компаній, якими володіє користувач;
- поточна вартість криптовалют, якими володіє користувач;
- наявна готівка та інші джерела (додається вручну);

г) відораження підрахованого сумарного балансу за всіма залишками на рахунках та вартості активів/пасивів;

ґ) інтерфейс підключення та видалення банківських рахунків;

д) інтерфейс уведення даних про активи та пасиви;

е) інтерфейс встановлення та зміни пароля;

є) налаштування відображення балансів (можливість приховувати непотрібні);

ж) налаштування дозволу використовувати біометричні дані;

з) наявність нічної теми та можливість її примусового використання;

і) можливість переходу пезпосередньо до офіційних додатків банку;

к) відображння помилок, отриманих від сервера банка при завантаженні балансів;

л) можливість стерти усі дані та налаштування додатка

3.2 Проєктування візуального прототипу

Опираючись на вищенаведені вимоги до наявного функціоналу, за допомогою сервісу Figma та зображень із відритою ліцензією, наданих сервісом Freepik [12], було створено візуальний прототип майбутнього додатка (рис. 3.1 — 3.6).

Кольори тексту, заднього фону сторінок, полів вводу даних та більшості інших елементів оформлено з огляду на використовувану світлову тему, а також загальний візуальний стиль.

Кнопки, із якими може взаємодіяти користувач, оформлено у вигляді звичайних блоків тексту. При цьому, задля їх візуального виділення як активних елементів, як у світлому так і у темному режимі додатка усі звичайні кнопки оформлено яскравим синім кольором, натомість кнопки, що попереджують користувача або ведуть до невідворотньої дії (наприклад, видалення) оформлено яскравим червоним.

Використання яскравих та єдиних для усіх кнопок і обох світлових тем кольорів на менш яскравому та змінюваному відповідно до теми задньому фоні дозволяє користувачу інтуітивно розпізнавати кнопки як елементи, що потенційно можуть бути натиснуті.

Кнопки, що наразі неактивні, використовують сірий колір.

3.3 Ініціалізація проєкту та піключення базових бібліотек

Було встановлено наступні необхідні компоненти розробки:

а) фреймворк Node.js;

б) систему управління пакетами для Node.js Yarn

б) середовище розробки Android Studio;

в) середовище розробки Xcode;

г) менеджер залежностей iOS-проєкту CocoaPods;

Далі за допомогою командного рядку було ініціалізовано новий React Native проєкт із використанням мови TypeScript.

Наступним кроком було встановлено бібліотеки із необхідним для проєкта функціоналом:

а) axios для керування HTTP запитами;

б) react-native-encrypted-storage для зберігання зашифрованих даних;

в) react-native-splash-screen для встановлення екрану заставки;

г) react-native-svg для відображення векторних зображень;

ґ) redux та react-redux для керування загальним станом додатка;

д) styled-components для легкого керування стилями компонентів;

е) @react-navigation/native та @react-navigation/native-stack для навігації між сторінками додатка;

є) react-native-fingerprint-scanner для входу у додаток із використанням біометричних даних;

3.4 Формування стуктури проєкту

Базова струкрура стандартного проєкту React Native це: обов'язкова папка node_modules, що містить усі встановлені бібліотеки; папки ios та android, кожна з яких є самостійним проєктом, вже придатним до компіляції у відповідному середовищі розробки своєї платформи та папка src, яка містить вихідний TypeScript код, тобто є основним джерелом наповнення проєкту.

Задля майбутньої кращої зчитуваності ієрархії коду та файлів проєкту та зважаючи на порядок імпорту й експорту, структуру директорій усередині папки src було поділено на 4 основні папки наступним чином (рис. 3.9):

а) папка "assets" — для так званих активів: шрифтів, зображень, оголошених модулів та загальних константних виразів й методів;

б) папка "components" для файлів оголошень усіх React-компонентів; ця папка, в свою чергу, ієрархічно сортує всі компоненти за принципами атомного дизайну у 5 директорій:

1) "atoms" для атомів — тобто найспростіших компонентів, таких як текст або контейнери, що можуть використовуватися багаторазово і необмежено

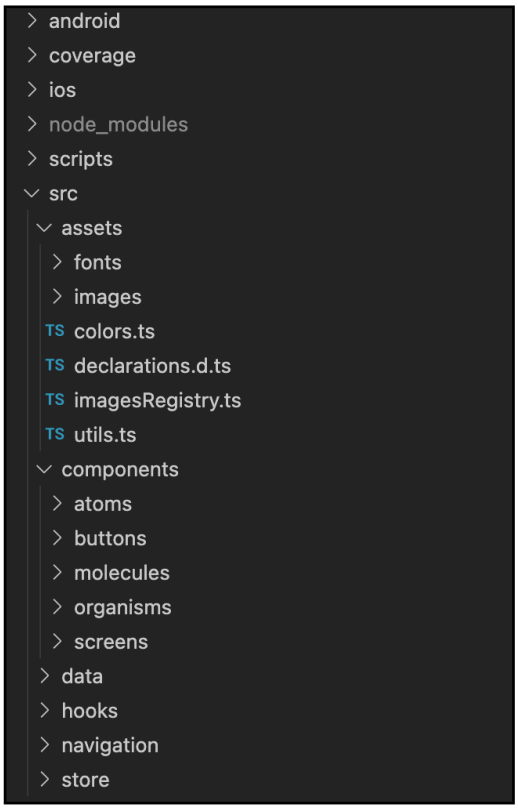

Рисунок 3.9 — Структура проєкту

2) “buttons” лише для кнопок, оскільки вони, хоч і є атомами за сутністю, вирізняються серед них окремою логікою, а також можуть з них складатися,

3) “molecules” для молекул — компонентів, що мають більш складну логіку, ніж атоми, і зазвичай з них складаються,

4) “organisms” для так званих організмів — компонентів із складною логіку, які зазвичай не використовуються у проєкті більше одного разу,

5) “screens” для компонентів, що є повноцінними сторінками додатку;

в) папка “data” для зберігання константних даних, таких як тексти, списки кольорів, регулярні вирази, тощо;

г) папка “hooks” для файлів оголошень хуків;

ґ) папка “navigation” для зберігання конфігурацій маршрутів, сторінок та типів міжсторінкових параметрів;

д) папка “store” для конфігурації Redux сховища, що відповідає за стан додатка, його редьюсерів та початкового стану.

3.5 Загальна архітектура проєкту та створення конфігурацій

3.5.1 Керування станами

Для керування загальним станом програми було сворено Redux сховище (почаковий стан сховища наведено у додатку А) і редьюсер, що відповідає за зміни стану цього сховища (наведено у додатку Б).

Редьюсер приймає 2 типи дій, що за принципом з офіційної документації бібліотеки [13] є звичайними рядками: “changeState” для часткової зміни стану за допомогою колбека та “setState” для повної заміни стану на інший.

Сховище зберігає всю інформацію, що:

- була отримана з мережі (наприклад, баланси рахунків, курси валют), при цьому зберігатиметься також статус отримання цієї інформації — завантаження, помилка або успіх;
- була введена і збережена користувачем (наприклад, пароль від додатку, токени доступу та інша налаштування).

Оскільки усі ці дані мають зберігатися після перезавантаження додатку, а Redux сховище може існувати лише під час активної роботи JavaScript, є доцільним зберігати його у пам’яті пристрою після кожних змін, а при новому вході у додаток зчитувати збережений стан і встановлювати його у сховищі як поточний.

За допомогою API бібліотеки react-native-encrypted-storage для вищенаведених дій було створено 2 методи:

- encryptStore — конвертує об’єкт стану у формат JSON у вигляді рядка та записує цей рядок у зашифровану пам’ять операційної системи;

- decryptStore — зчитує зашифровану пам’ять та, при наявності збереженого стану, робить його синтаксичних аналіз і записує значення отриманого об’єкта до сховища.

Метод decryptStore викликатиметься лише при запуску додатку, а encryptStore кожен раз, коли доцільно зберегти зміни стану саме до постійної пам’яті.

Оскільки зміну в загальний стан доволі часто вносяться зсередини React-компонентів, були вирішено створити хук useChanges, що повертає функцію change. Ця функція приймає аргументом колбек зміни стану. Під час виклику вона змінює стан сховища, після чого для збереження інформації викликає функцію encryptStore.

3.5.2 Навігація

Для навігації між сторінками додатку, за допомогою API бібліотеки @react-navigation/native-stack, було створено компонент Navigation. Цей компонент містить масив сторінок (а точніше компонентів, що ставлять у відповідність ідентифікатори сторінок з їх React-компонентами) і відмальовує їх при переходах з однієї на іншу за допомогою стеку: рендеринг нових сторінок відбувається поверх старих, формально, сторінки накладаються одна на одну при навігації. Це дозволяє зберігати стани вже відкритих сторінок при переходах і поверненні.

Для передачі додаткової інформація при переході з однієї сторінки на іншу використовуються параметри навігації, що містяться в об’єкті, який створюється на сторінці відправлення та зчитується на сторінці призначення. Для уникнення помилок, пов’язаних з помилковою передачею або зчитуванням параметрів, було проведено типізацію ідентифікаторів-найменувань сторінок, які має містити готовий додаток за наданим вище візуальним дизайном, та параметрів, які ці сторінки можуть приймати.

Готові типізації представлені у додатку В. Окремі пояснення щодо зовнішніх типів, які вони використовують, будуть надані пізніше.

Бібліотека @react-navigation/native експортує два основних хуки, що дозволяють керувати навігацією:

- useNavigation — повертає об’єкт із методами керування навігацією, зокрема navigate — для переходу на бажану сторінку використовуючі бажані параметри (приймає як аргументи ідентифікатор сторінки та об’єкт параметрів) — та goBack — для переходу на 1 сторінку назад, в даному випадку — по стеку;
- useRoute — повертає об’єкт із даними маршрута, зокрема цей об’єкт містить поле params — об’єкт параметрів, встановлених під час навігації.

Оскільки обидва хуки у TypeScript використовують узагальнення для типізацій, їх незручно використовувати звичайним викликом всередині компонента, як у JavaScript, оскільки кожен раз вони будуть вимагати передачі одних й тих самих параметрів. Внаслідок цього було прийняте рішення створити статично типізовані версії цих хуків: useNav — типізовану версію useNavigation та useParams — типізовану версію useRoute, яка, на відміну від прототипа, одразу повертає параметри навігації як результат виконання. Такий підхід до використання хуків є дуже поширеним в TypeScript, адже зменшення кількості коду без погіршення якості розробки є його суттєвою перевагою.

3.5.3 Кольорова схема

Оскільки, по-перше, усі нові версії як платформи iOS, так і платформи Android, а також багато сучасних додатків під ці платформи мають можливість підлаштовувати забарвлення теми інтерфейсу під поточний час доби; а, по-друге, за візуальним дизайном додатка багато різних елементів використовують одні й ті самі кольори (при чому однакові

кольори у світлій темі є однаковими і у темній), наявною стала проблема грамотного управління цими кольорами.

Для управління фунціональними компонентами, що є залежними від світлової теми пристрою, React Native має вбудований хук useColorScheme. Цей хук оформлює підписку компонента на зміни теми та повертає рядок, що приймає значення “light” або “dark”.

Було вирішено, що найефективнішим способом управління забарвленням є безпосереднє отримання рядка із HEX-кодом потрібного кольору всередені компонента завдяки функції, яка б приймала аргументом ідентифікатор елемента для забарвлення, а повертала готовий колір забарвлення в залежності від використовуваної на даний момент світлової теми.

Взявши за основу хук useColorScheme, було імплементовано власний хук useColors, що повертає об’єкт із 2 записами: методом colors, принцип роботи якого описано вище, та полем isDarkTheme, що за допомогою булевого значення вказує, чи використовується у даний час нічна тема. Якщо світлова тема зміниться під час користування додатком (наприклад, зайде сонце, або тему навмисно змінить користувач), функція colors та змінна isDarkTheme одразу повернуть нові значення, оскільки будуть підписаними на зміни.

Метод colors працює наступним чином: отримавши ідентифікатор елемента для забарвлення, за яким він має повернути поточний колір, метод знаходить значення цього ідентифікатра у регістрі кольорів — об’єкта, у якого ключами є ідентифікатори, а значеннями — масиви з двох елементів: перший — HEX-код кольору для світлої теми, — другий — HEX-код длятемної теми. Далі в залежності від значення, повернутого викликом хука useColorScheme, метод повертає або перший, або другий елемент масиву.

Надалі результат уже безпосередньо самим компонентом додається у відповідні стилі і елемент отримує необхідне забарвлення.

3.5.4. Рендер зображень

Оскільки за візуальним дизайном додаток має дві версії для кожного зображення: темну та світлу, ці версії мають змінюватися таким же чином, як і кольори елементів — за підпискою.

React Native надає можливість рендерингу статичних зображень більшості форматів. Для цього потрібно лише скористатися стандартним компонентом Image із бібліотеки, передавши йому імпортований локальний файл.

Досить легким рішенням зміни зображень за підпискою могла би бути звичайна заміна посилання на файл із зображенням в залежності від значення поля isDarkTheme із раніше створеного хука useColors. Нажаль, таке ріщення неможливе у парадигмі React Native, оскільки він не підтримує динамічного імпорту файлів, в тому числі зображень. Внаслідок цього обмеження, всі версії зображень мають бути спочаку імпортованими, і лише потім можуть змінюватися в залежності від побажань. Для уникнення повторного використання логіки заміни зображень при зміні теми та для спрощення процедури імпорту зображень в цілому, було вирішено створити спеціальний регістр, який містить імпорти усіх наявних в проєкті файлів зображень.

Даний регістр є звичайним об’єктом, ключами якого є ідентифікатори зображеннь, а значеннями — масиви з двома елементами: перший — це імпортованим файл зображення для світлої теми, — другий — файл для темної теми.

Для уникення великої кількості ручної роботи при додаванні усіх необхідних файлів зображень в регістр, було написано спеціальний Node.js скрипт. При запуску скрипт сканує наявні у проєкті файли зображень, організовує їх імпорт у вигляді описаного вище об’єкта та записує цей об’єкт у файл регістру. Для оновлення регістру після додавання нових зображень у проєкт, скрипт потрібно перезапустити.

Після того, як було створено даний регістр, було також додатково імлементовано власний компонент CustomImage на заміну вбудованому. Цей компонент приймає ідентифікатор зображення, знаходить його в регістрі, обирає потрібну версію в залежності від світлової теми та рендерить цю версію за допомогою нативного вищезгаданого компонента Image.

3.5.5 Рендер тексту

Оскільки React Native по суті генерує 2 різних додатки під плаформи iOS і Android, використовуючи нативні API цих платформ, деякі компоненти, внаслідок різних параметрів, встановлених за замовчуванням у цих платформах, можуть виглядати по-різному. Текст є одним із таких компонентів, тому, задля його однакового вигляду як на одній, так і на іншій платформі, потрібно задати повну конфігурацію усіх його характеристик, таких як висота літер, висота рядка і т.д. Більше того, бажаною поведінкою тексту є автоматичне підлаштовування під світлову тему (за дизайном, текст у нічній темі має ставати білим) та використання за замовчування єдиного шрифту визначеного дизайном для усього додатка одразу.

Вирішити проблеми наведених пунктів можливо за допомогою сворення власного компонента для екту, який буде одразу містити усі потрібні параметри і не потребуватиме додаткового налаштування стилів при використанні. Натомість, його використання у будь-якій частині додатку одразу повертатиме текст, оформлений відповідно наявних до нього вимог функціональності.

Керування забарвленням цього компоненту імплементовано за допомогою функції colors, повертаємої викликом хука useColors, інші стилі було встановлено константами та підключено до нативного компонента для рендеру текстових рядків.

3.5.6 Виконання HTTP-запитів

Невід'ємною частиною цього додатка є взаємодія із серверами — в першу чергу банківськими, а також і інших сервісів. Як вже було сказано вище, уся інформація, отримана з мережі, має одразу потрапляти в сховище і зберігатися лише там. Зважаючи на це, є доцільним виконувати всі запити за допомогою функції, що приймає як аргументи об'єкт параметрів запиту (URL-посилання, дані для відправки, метод та заголовки запиту) і колбек, що описує порядок запису інформаціїї в сховище. Також функція має приймати опціональний третій параметр — колбек зміни стану завантаження. Цей стан може бути трьох типів — успіх, завантаженя або помилка. У разі отримання помилки її текст має бути збережено.

У додатку ця функція отримала назву handleRequest і діє наступним чином:

- за наявності колбеку зміни стану завантаження, спочатку виклакається цей колбек; він скасовує збережену у сховищі помилку завантаження, якщо така є, і встановлює статус "завантаження";
- далі, за допомогою засобів бібліотеки axios, виконується HTTP запит із вказаними параметрами, що були передані у функцію;
- після того, як запит було виконано, статус "завантаження" прибирається і в залежності від результату виконання проводиться одна з двох дій: якщо запит виконано успішно, отримані дані записуються у сховище за допомою переданого колбека, якщо застосунок отримав помилку — текст помилки записується.

Більшість компонентів, наявна інформація в яких залежить від даних, отриманих в результаті виконання запиту, мають демонструвати користувачу статус виконання цього запиту. Тому за даними зі сховища кожен такий компонент матиме змогу це зробити.

3.6 Імплементація прямого функціоналу програми

3.6.1 Іконка застосунку та сплеш-скрін

Іконка додатка — це картинка, що репрезентує його (як правило, в меню, а також в інших місцях, де це потрібно).

Сплеш-скрін — це екран заставки додатка, що відображається декілька секунд після запуску. Мета такого екрану — скрити фонові завантаження самого додатку і його налаштувань. Платформи iOS та Android (починаючи з 12 версії) потребують обов'язкову наявність сплеш-скріну, замінюючи його на стандартну заставку за відсутності.

Іконку додатка та версії сплеш-скрінів для обох світлових тем було створено на етапі дизайну, вони представлені на рисунках 3.10 та 3.11 — 3.12 відповідно.

Рисунок 3.10 — Іконка додатку з
її можливими варіантами
вигляду в меню

Як сплеш-скрін, так і іконку можна налаштувати лише через нативні середовища розробки для платформ — оскільки вони керуються операційною системою відповідної платформи, а не потоком JavaScript-

Рисунок 3.11 — Екран заставки, звичайна тема

Рисунок 3.12 — Екран заставки, нічна тема

коду. Тому використовуючи інструменти розробки, надані середовищами Xcode та Android Studio, іконка та екрани заставки були окремо встановлені у проєктах додатка для кожної платформи.

Тим не менш, створеної конфігурації сплеш-скріну для React Native недостатньо, оскільки для повної готовності до своєї роботи додаток має запустити JavaScript-код, провести рендер компоненту Navigation, який відповідає за навігацію сторінками та їх відображення, а також дочекатися, доки Navigation покаже початкову сторінку.

Всі ці маніпуляції відбуваються вже після того, як додаток фактично запущено, тобто після того, як налаштований нативно сплеш-скрін зникає. Для того, щоб завадити його завчасному зникненню, було використано спеціально спроєктовану під цю потребу бібліотеку react-native-splash-screen. Завдяки налаштуванням бібліотеки, заставка не зникає одразу після того, як додаток завантажився, а залишається доти, поки її явно не скриють за

допомогою методу SplashScreen.hide всередині коду JavaScript. Цей метод встановлено як значення пропа onReady компонента Navigation. Завдяки цьому екран заставки зникатиме лише тоді, коли відбудеться рендер стартової сторінки.

3.6.2 Пароль від застосунку

Компонент сторінки вводу пароля (рис. 3.3) фактично виконує роль одразу двох сторінок: стартової сторінки, яка запитує в користувача пароль, та сторінки зміни пароля у налаштуваннях. Дізнатися, який з цих двох режимів має бути використаний у поточній ситуації, сторінка може завдяки параметру isNewPinCreating, значення якого вона отримує з описаного вище хука useParams. Значення цього параметра, встановлене як true, буде передаватися лише при навігації зі сторінки налаштувань. В усіх інших випадках параметр дорівнюватиме false за замовчуванням.

Якщо навіть при значенні false параметра isNewPinCreating компонент сторінки не знайде збереженого пароля у сховищі — це означатиме, що користувач відкрив додаток вперше, і пароль ще не було встановлено, тому сторінка автомачно увімкне режим сворення нового пароля.

Ці два режими відрізняються лише за логікою роботи. Якщо увімкнено режим створення нового пароля, то після першого вводу додадок запросить користувача ввести пароль ще раз і порівняє результат другого вводу із першим. Якщо пароль співпаде, додаток запише новий пароль у сховище за допомогою описаного вище хука useChanges.

Якщо ж режим створення нового пароля буде вимкнено, то після його вводу додаток порівняє пароль зі збереженим у сховищі. При вдалому вводу пароля користувача буду перенаправлено на головну сторінку. При невдалій спробі процедуру можна будет повторити до максимально встановленої кількості спроб (що наразі дорівнює 8), при цьому коли в

користувача залишиться менше 5 спроб, додаток буде попереджувати його про кількість спроб, що залишилася. Щоб кількість залишених спроб не анулювалася після перезавантаження додатка, вона зберігається у сховищі. Після вичерпання усіх спроб додаток заблокується.

Якщо користувач увімкнув функцію “дозволити використання біометричних даних” у налаштуваннях, то він зможе заходити у додаток за допомогою відбитка пальця або моделі обличчя. Для написання функціоналу проведення валідації біометричних даних було використано API бібліотеки react-native-fingerprint-scanner. Вона експортує 2 основні методи: authenticate, що відправляє в операційну систему запит на валідацію й повертає результат у виляді промісу, та isSensorAvailable, що теж у вигляді промісу повертає дані про те, чи може пристрій провести аутентифікацію такого типу.

Для більш легкого користування цима функціями було додаткого створено хук useBiometrics. Цей хук повертає масив з трьома елементами: 0 — булеве значення, що вказує, чи можна наразі скористуватися біометрію; 1 — функція, що запускає системну аутентифікацію відбитка пальця або моделі обличчя та повертає проміс із булевим значенням, при цьому результат true означає, що аутентифікація відбулась успішно; 2 — компонент модального вікна для відображення на старих версія платформи Android, оскільки вони не мають нативного системного діалогу.

Всередині компонента сторінки вводу пароля результат виклику описаного хука деструктуризується (кожен елемент масиву отримує власну змінну), після чого усі елементи використовуються відповідно до своїх ролей. Тобто як тільки сторінка завантажена і значення isNewPinCreating дорівнює false, відбувається перевірка того, чи увімкнена опція “дозволити використання біометричних даних” у налаштуваннях та чи підтримує пристрій використання біометрії. Якщо обидва резутати матимуть значення true, то у лівому нижньому кутку з’явиться кнопка для проведення

біометричної аутентифікації, а також воно буде одразу запущено автоматично.

3.6.3 Головна сторінка

Виляд головної сторінки додатка зображено на рисунках 3.1 та 3.2. Вона містить наступні елементи:

- курси покупки й продажу валюти від НБУ, Монобанку та Приватбанку відповідно;
- дошка із підсумковим бюджетом користувача, який рахується як сума усіх його банкіських рахунків, а також активів із пасивами; при цьому на основній частині дошки окремо відображається бюджет накопичень у гривні та бюджет накопичень у інших валютах (в еквіваленті до американського долара); на нижній частині відображається повний сумарний бюджет користувача, що виражений в еквіваленті як до гривні, так і до американського долара;
- список усіх накопичень, за категоріями, зокрема це: баланси рахункіх у банках Монобанк, Приватбанк для фіз. осіб та Приватбанк для юр. осіб., вартість доданих користувачем акцій компаній, вартість доданих користувачем криптомонет та готівкові кошти користувача.

Списки банківських накопичень мають з правої сторони кнопки, що за допомогою діплінкування відкривають мобільний клієнт даного банка або, за відсутності такого, посилання на цей клієнт у магазині додатків платформи. Списки інших накопичень (тобто тих, що користувач редагує вручну) в свою чергу мають знизу кнопку “Змінити”, що перенаправляє на відповідний екран редагування.

Нижня частина сторінки містить кнопку для переходу на екран налаштувань.

При потраплянні на головну сторінку вперше після відкриття застосунку, за допомою функції handleRequest та методу Promise.all одночасно буде відправлено HTTP-запити на отримання таких даних:

а) загальних курсів валют (використовуються для підрахунків на головній дошці) — з бешкоштовної публічної бази даних;

б) курсу долара, наданого НБУ, Монобанком та Приватбанком — напряму з серверів відповідних банків за допомогою їх відкритого API;

в) балансів рахунків за усіма збереженими токенами доступу користувача для кожного банку;

г) вартості наявних в користувача цінних паперів;

ґ) вартості наявних в користувача криптовалют;

Аналогічний список запитів відправляється при примусовому оновленні сторінки користувачем шляхом змахування її донизу (жест “pull to refresh”).

Під час завантаження біля відповідних курсів валют та записів у списку накопичень з’являється спінер, вказуючий на це. Якщо якийсь запит виконався з помилкою, спінер зміниться на червону іконку помилки. При натисканні на цю іконку користуваца буде перенаправлено на сторінку із текстом отриманої помилки (рис. 3.8). Якщо всі запити виконалися із однаковою помилкою “Nerwork Error” або “timeout exceeded”, це означатиме, що користувач не має доступу до інтернету. В такому разі іконки помилок не відображатимуться біля записів, натомість користувач отримає повідомлення у верхній частині екрану про те, що в нього відсутній зв’язок та наразі він бачить лише дані із кешу.

3.6.4 Сторінка налаштувань

Оскільки за дизайном і логікою сторінка налаштувань містить однотипні компоненти для виконання однотипних функцій (наприклад кнопка ввімкнути/вимкнути опцію або поле для вводу рядка), немає сенсу

додавати кожен такий компонент на цю сторінку вручну. Натомість, ефектившим є створити масив із об’єктів, що репрезентують кожну опцію налаштуання і міститять наступні поля:

- type — рядок-ідентифікатор типу налаштування (наприклад ‘toggler’, ‘input’, ‘button’, тощо);
- key — рядок, що є ключем до значення цього налаштування у сховищі; при зміні значення налаштування, нове значення буде одразу записане у сховище за цим ключем;
- data — об’єкт із даними, що різняться в залежності від типу налаштування, визначеному у полі type; це може бути підпис до поля вводу, функція для виклику при натисканні кнопки, тощо; ознайомитися із наявними у проєкті типами налаштувань та їх релевантними даними можна у додатку Г.

Передавши цей масив, а також функцію, що вказує, яким чином має рендеритися компонент кожного типу, у вбудований компонент FlatList, який використовується в React Native для ефективного рендеру списків, було отримано екран налаштувань, вигляд елементів якого є стандартизованим, а також легко змінюваним. Для того щоб змінити зовнішній вигляд одразу всіх компонентів деякого типу, достатньо змінити функцію рендеру; в той же час, для створення нової опції налаштування потрібно лише додати об’єкт із необхідними полями у масив, після чого компонент налаштування автоматично з’явиться на сторінці та одразу буде містити усю необхідну логіку функціоналу, оскільки її вже прописано у функції рендеру.

Для зручності користувача масив налаштувань було поділено на візуальні блоки. Кожен блок задля візуального розділення має над собою зображення, яке для ефективного редагування даних сторінки теж міститься саме у об’єкті блоку.

Сторінка налаштувань містить усі опції, заявлені в першому пункті цього розділу. Оскільки кожна зміна налаштувань одразу записується у сховище, дії користувача не потребують додаткового збереження.

3.6.5 Редагування списків

Для редагування інформації про наявні в користувача криптовалюти, цінні папери та готівку (рис. 3.6 — 3.7), дизайн пропонує використувати редагований списіок, елементами якого є комплексні поля вводу даних, релевантних для кожної сутності, а логіка якого є однаковою для усіх сторінок і включає видалення елементів, їх редагування, валідацію та додавання нових із подальшим збереженням списку.

Для багаторазоваго використання спільної логіки такого списку на різних сторінках було створено хук useEditedList. Цей хук отримує на вхід початкові дані, доступні для редагування, дозволяє їх змінювати будь-яким чином, зберігаючи внесені зміни лише всередині власного стану, а потім зберегти у сховище за бажанням користувача. Результатом виконнаня хука є функція рендеру сторінки із редагованим списком. Рендер списку елементів у цій функції відбувається за допомногою компонента FlatList, що є спеціально оптимізованим для рендерингу списків із великим набором даних.

Ознайомитися із типами параметрів, які приймає даних хук, та результатом, який він повертає, можна у додатку Ґ.

3.6.6 Конфігурація банків та відстеження балансів

Алгоритм підключення кожного із банків та подальший моніторинг їх стану за своєю логікою є єдиним для усіх банків: як і тих що наявні в додатку на даний момент, так і тих, що можуть з’явитися пізніше. Він включає в себе:

1) отримання користувачем токенів доступу до рахунків через банківський API;

2) уведення цих токенів на сторінці додавання рахунку відповідного банку;

3) здійснення пробного запиту балансів за цими токенами; в разі успіху такого запиту токени зберігаються у сховище і автоматично використовуються для наступних запитів, поки користувач їх не видалить; в разу невдалого запиту користувачу буде повідомлено про помилку та запропоновано перевірити дані й здійсними запит ще раз;

Відрізнятиметься на сторінках підключення банків та при HTTP-запитах для отримання їх балансів лише:

- логотип та назва банку,
- інструкція з отримання токенів,
- формат, кількість та назви токенів для підключення,
- URL-адреса, метод запиту та форма включення токенів доступу у хедери,
- формат списку балансів, отриманого від банка.

Для уніфікації процесу здійснення запитів та стандартизації формату збереження інформації про баланси задля її подальшого зручного опрацьовування та відображення, було вирішено сворити об’єкт, що є своєрідним регістром із інформацією, характерною для кожного з банків, тобто тими даними, що наведені вище.

Ключами регістру є ідентифікатори банків а значеннями об’єкт із наступними полями:

- icon — логотип банку у форматі svg,
- name — назва банку для відображення,
- deeplink — діплінк, тобто глубоке посилання на мобільний клієнт банку; діплінк кожного потрібного додатка можна знайти або в Інтернеті, або шляхом декомпіляції інсталяційного файлу цього додатка;

- credentials — масив, що містить об’єкти конфігурацій токенів підключення банка, а саме назву токена, ключ токена для сховища, максимально можливу довжину та регулярний вираз для валідації;
- getRequestData — функція, що отримує на вхід значення токенів, а повертає параметри запиту балансів рахунків, підключених до цих токенів; параметри включають URL-посилання, метод, хедери, а також опціональний об’єкт із даними;
- convertResponseToBalances — функція, що переводить баланс із формату, надісланого банком, в єдиний формат балансів, який використовує додаток;
- connectionGuide — інструкція з отримання токенів доступу для API банку;

Ознамитися із значеннями цього регістра для, наприклад, банку “Монобанк” можна у додатку Д (окрім інструкції отримання токенів).

Підставляючи відповідні дані для кожного банку з цього регістру, додаток поєднує різну логіку програмних взаємодій із банківськими API в єдину і не залежну від конкретного банку логіку підключення й відстеження рахунків для користувача.

Переглянути і видалити підключені банківські рахунки, а також додати нові для відстеження, користувач може зі сторінки налаштувань.

3.6.7 Особливості відстеження вартості цінних паперів та криптовалют користувача

Інтерфейси відстеження вартості акцій публічних компаній та криптовалют мають схожу логіку у додатку.

Для відслідковування вартості власних акцій компанії користувач має ввести у відповідний редагований список тікер компанії, за яким вона котирується на біржі, та кількість наявних в нього акцій, а для криптовалюти — її код та кількість наявних монет. При натисканні на

кнопку “Зберегти” додаток здійснить запит за вказаними даними до публічних баз даних, і в разі успіху збереже дані й отриману із запиту назву компанії чи валюти та вартість акцій/монет для відображення на головній сторінці. Якщо інформація, надана користувачем, виявиться невірною (тобто запит відбудеться із помилкою), додаток сповістить користувача про те, що надані дані є хибними.

3.6.8 Додавання даних про готівку

Додаючи дані про готівку та інші пасивні накопичення у редагований список, користувач має вказати назву накопичення та його суму, обравши валюту накопичення за допомогою спеціальної кнопки, доданої в дане комплексне поле вводу.

Для зручності користувачів в це поле було також додано функцію використання арифметичних виразів — це означає, що коли користувач витратив, наприклад, 20 грн зі 100 грн, які були записані у списку, редагуючи цей список користувач може дописати “-20” у поле вводу. Після цього додаток автоматично порахує значення наданого виразу при збереженні.

4 ДЕМОНСТРАЦІЯ РОБОТИ ДОДАТКА

4.1 Компіляція проєкту і встановлення додатка на мобільний пристрій

У середовищі Xcode було скопмільовано автозгенерований у папці ios проєкт у режимі “production”. Цей режим будує остаточну оптимізовану версію додатка, призначену для App Store. Отриманий додаток було встановлено на пристрій iPhone 12 та запущено.

4.2 Перший запуск

Після запуску додаток запропонував створити пароль для входу (рис. 4.1). Отримавши комбінацію пароля, введену двічі, додаток одразу перенаправив на екран налаштувань (рис. 4.2), з якого можна додати рахунки для відстеження.

4.3 Підключення банківських рахунків

Для демонстрації процесу підключення банку до додатка був обраний Монобанк. Після натискання на кнопку “Додати новий рахунок для відстеження” відкрився екран додавання нового рахунку (рис. 4.3), що містить логотип банка, поле для вводу токена доступу та інструкцію з отримання токена.

При натисканні на посилання з інструкції, додаток відкрив онлайн-портал керування токенами доступу до рахунків Монобанк, який в свою чергу запустив офіційний додаток Монобанка для проведення аутентифікації користувача (рис. 4.4). Після натискання кнопки “Підтвердити”, портал згеренував токен доступу (рис. 4.5).

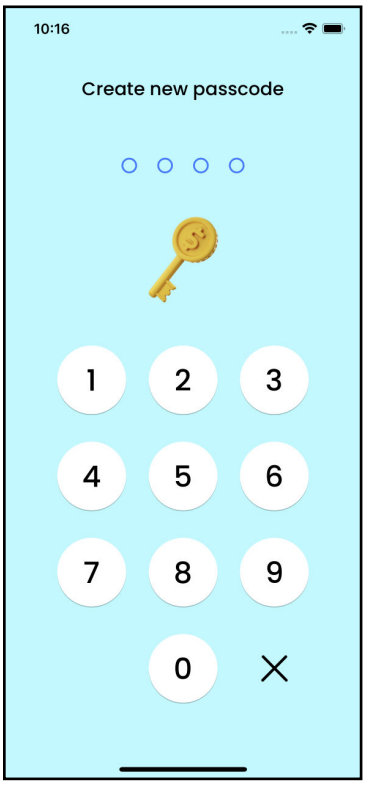

Рисунок 4.1

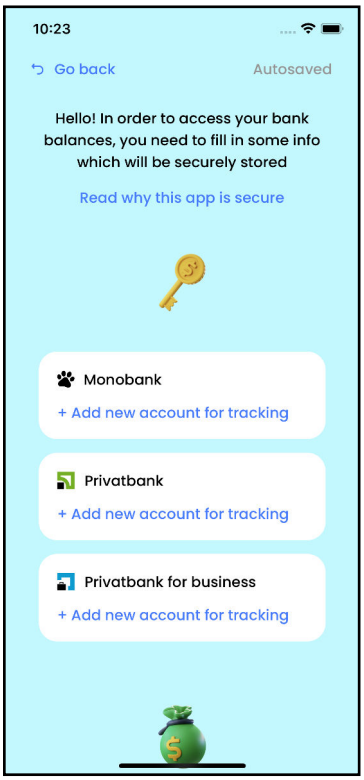

Рисунок 4.2

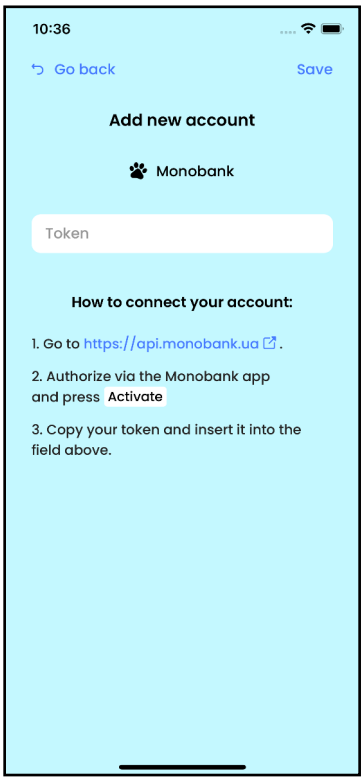

Рисунок 4.3

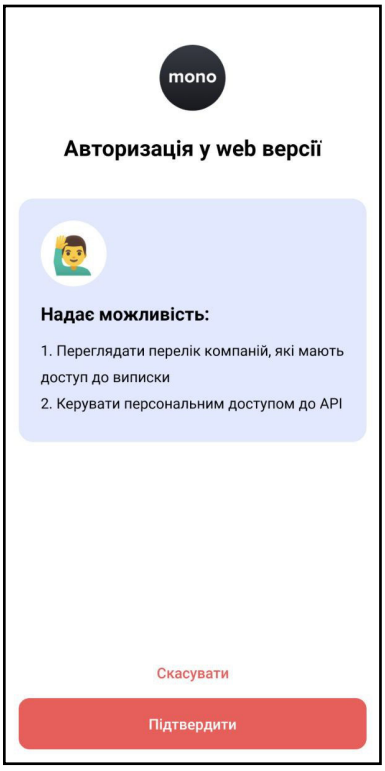

Рисунок 4.4

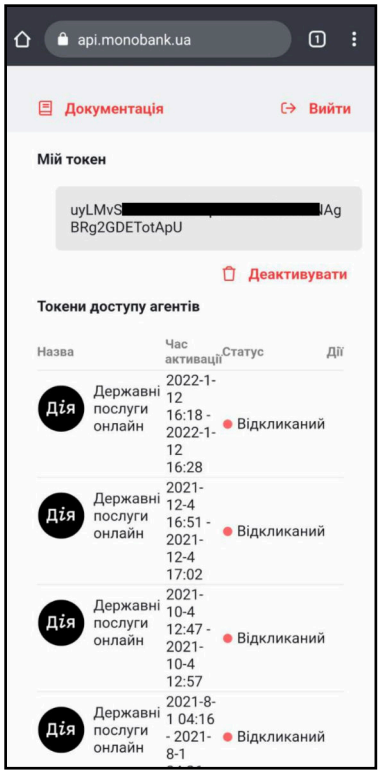

Рисунок 4.5

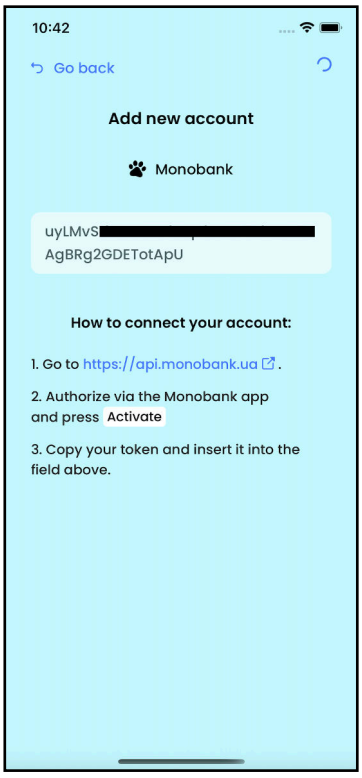

Рисунок 4.6

Дотримуючись інструкції, токен було скопійовано та вставлено у відповідне поле. Після натискання кнопки “Зберегти” та односекундного завантаження (рис. 4.6), відбулося перенаправлення на екран налаштувань і з’явилося сповіщення про те, що рахунок успішно додано (рис. 4.7).

Під кнопкою додачі нового рахунку з’явилася кнопка для редагування підключених рахунків. При натисканні на кнопку, додаток відкрив сторінку із переліком рахунків, що прив’язані до підключеного токена (рис. 4.8).

Повернувшись на головну сторінку, було помічено, що підключені щойно рахунки з’явилися у списку балансів (рис. 4.9), а також, що сума накопичень на цих рахунках з’явилася на верхній дошці.

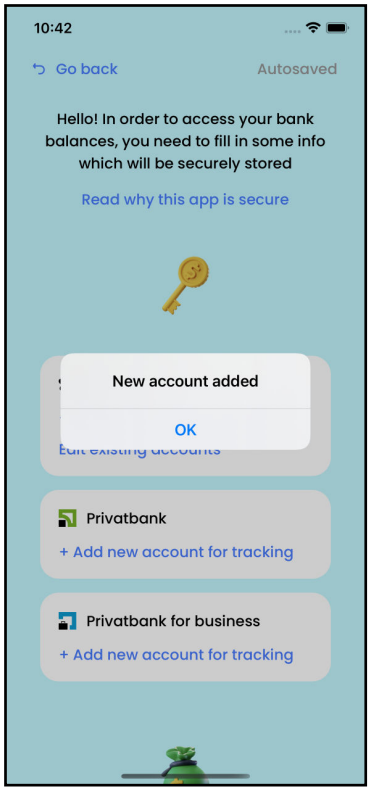

Рисунок 4.7

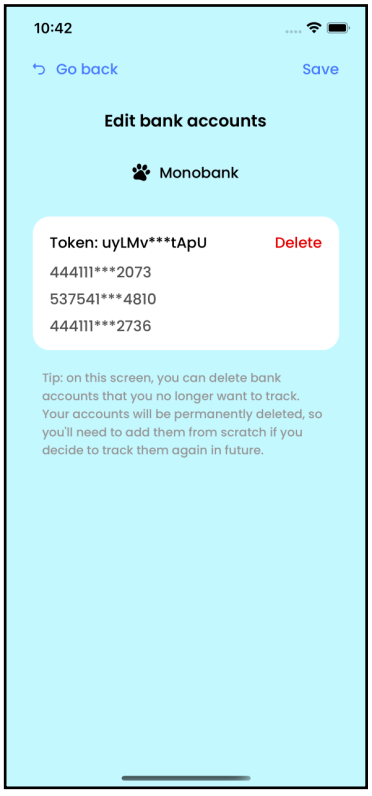

Рисунок 4.8

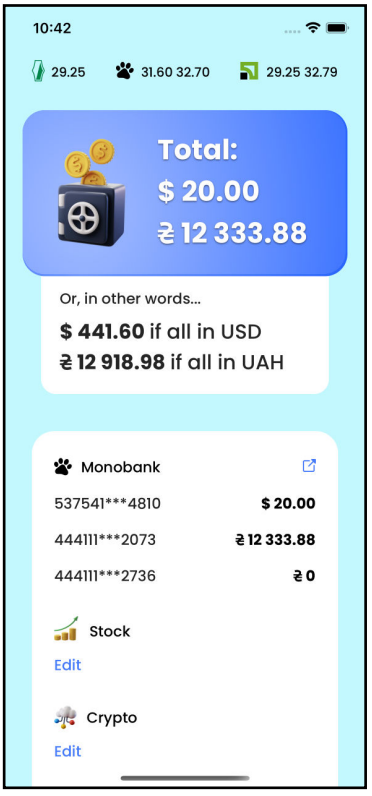

Рисунок 4.9

Відкривши екран налаштувань, вимкнувши опцію “Відображати нульові баланси” (рис. 4.10) та повернувшись на головний екран, було

помічено, що рахунок із нульовим балансом більше не демонструється (рис. 4.11).

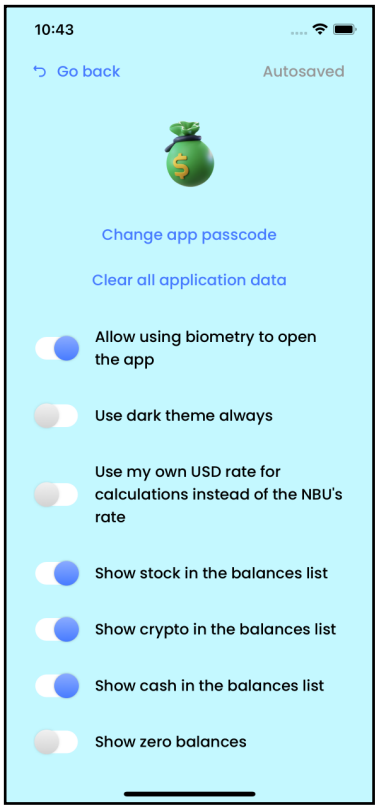

Рисунок 4.10

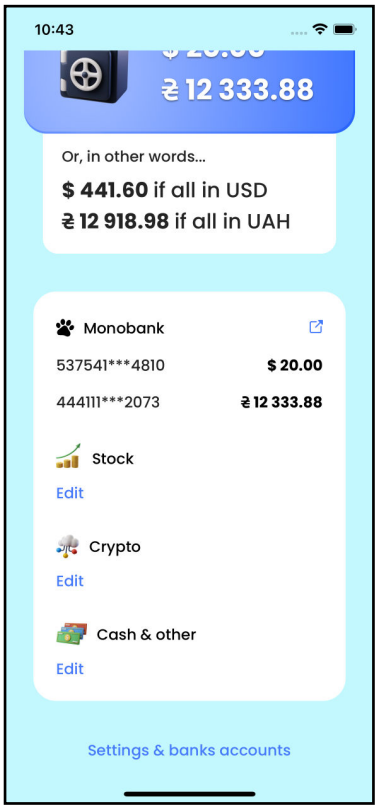

Рисунок 4.11

4.4 Додавання акцій компаній

Натиснувши кнопку “Редагувати” під розділом із назвою “Акції”, було відкрито екран редагування наявних у користувача акцій (рис. 4.12). Для демонстрації роботи було вирішено спробувати додати у список 2 акції компанії Apple, яка має тікер “AAPL” на біржі, без початкової ціни покупки, та 2 акції компанії Coca-Cola, яка має тікер “KO”, із початковою вартістю цих акцій 52 долари США (рис. 4.13). Після збереження та нетривалого завантаження на головному екрані у відповідному списку зв’явилися назви доданих компаній та поточна вартість доданих акцій (рис. 4.14). При цьому для компанії Coca-Cola також відобразалася додана початкова ціна.

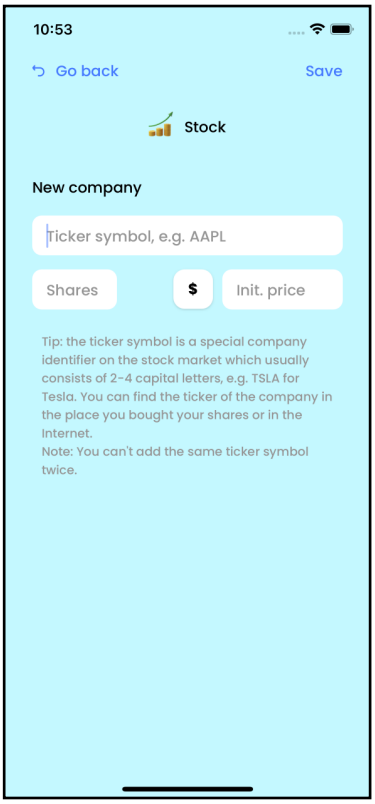

Рисунок 4.12

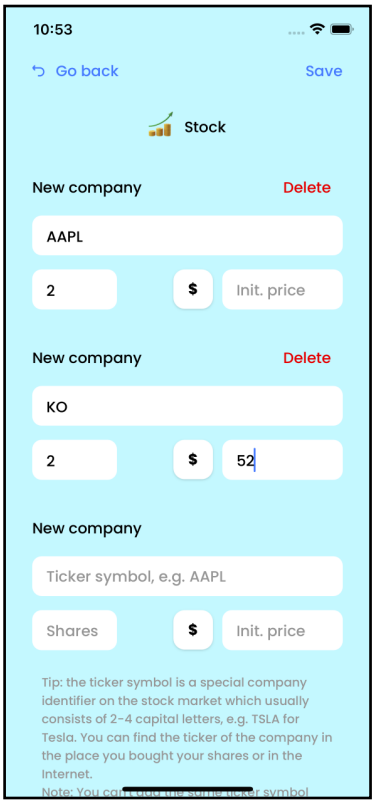

Рисунок 4.13

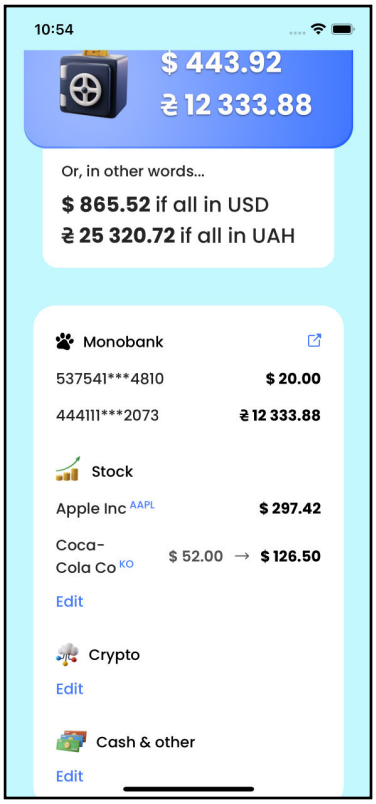

Рисунок 4.14

4.5 Додавання криптовалют

Процес додавання криптовалюти аналогічний процесу додавання акцій компаній. Для демонстрації було обрано біткоїн. На відповідному екрані було додано код валюти, тобто “BTC”, кількість монет 0.003 та початкову вартість 70 доларів США (рис. 4.15).

Після збереження назва, початкова вартість та поточна вартість доданих монет з’явилися на головній сторінці (рис. 4.16).

4.6 Додавання даних про готівку

Натиснувши на кнопку “Редагувати” під написом “Готівка та інше”, було відкрито екран редагування даних про готівку. На цьому екрані було додано накопичення із назвою “Portmone” та сумою 150 гривень (рис. 4.17).

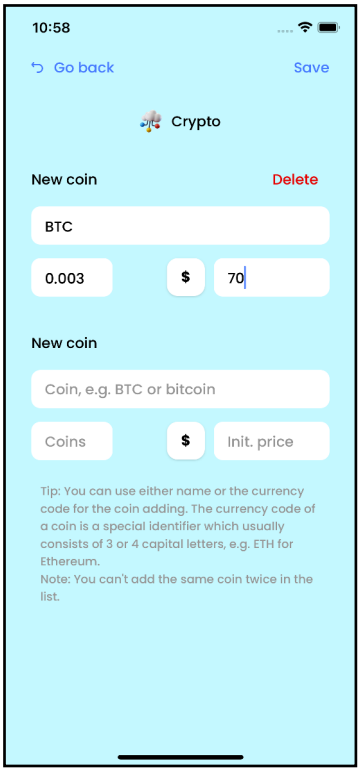

Рисунок 4.15

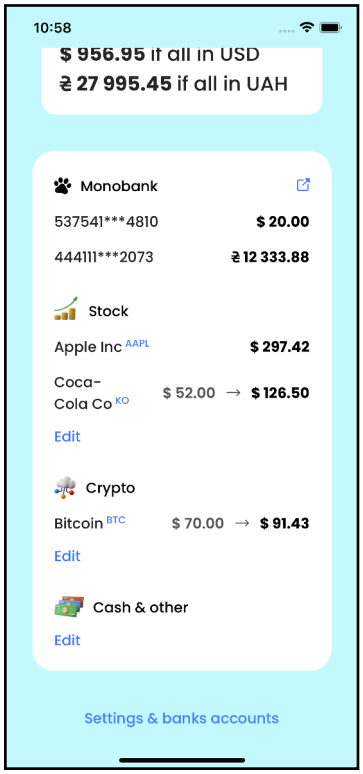

Рисунок 4.16

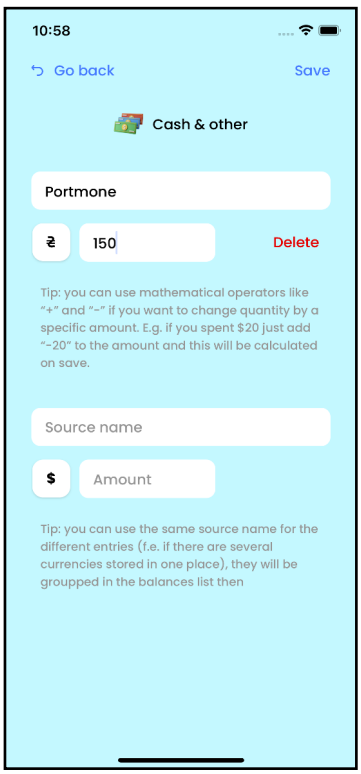

Рисунок 4.17

Після збереження інформації та її відображення на екрані (рис. 4.18), кнопку "Редагувати" було натиснуто ще раз. Суму накопичення було відредаговано, додрукувавши "- 25.76" (рис. 4.19), а потім збережено. Як видно на рисунку 4.20, додаток самостійно підрахував значення введеного виразу.

4.7 Зміна світлової теми

Відкривши сторінку налаштувань і прогортавши її до опції "Використовувати нічну тему завжди" (рис. 4.21), було написнуто на перемикач біля неї. Одразу після цього додаток змінив нему на нічну (рис. 4.22 — 4.23).

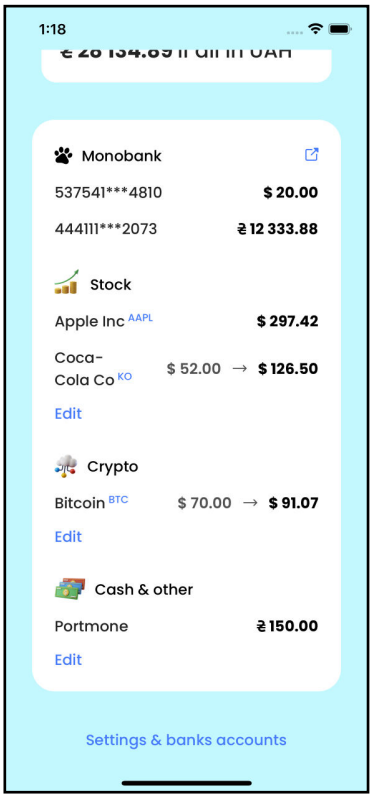

Рисунок 4.18

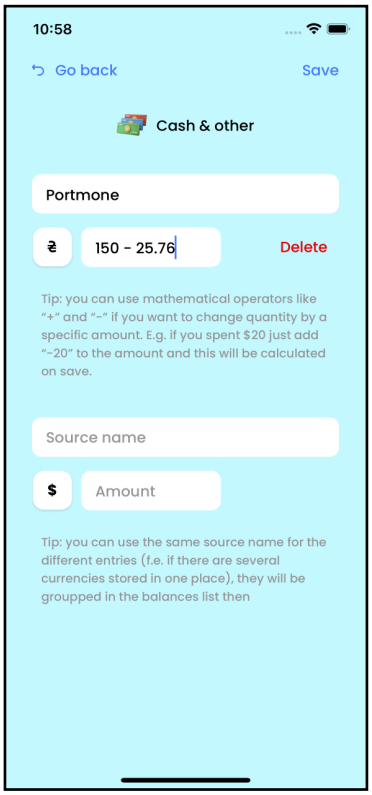

Рисунок 4.19

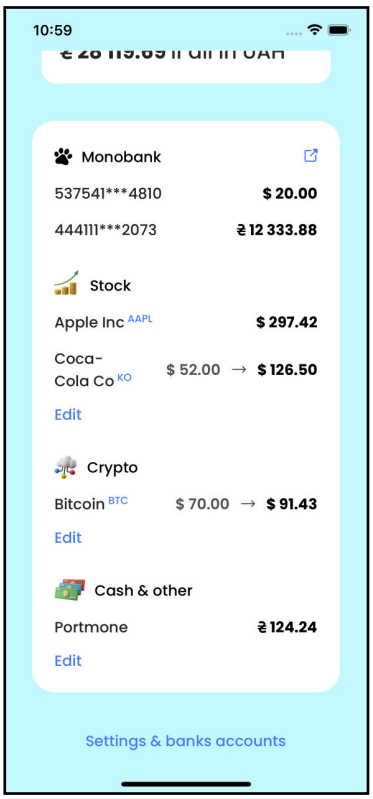

Рисунок 4.20

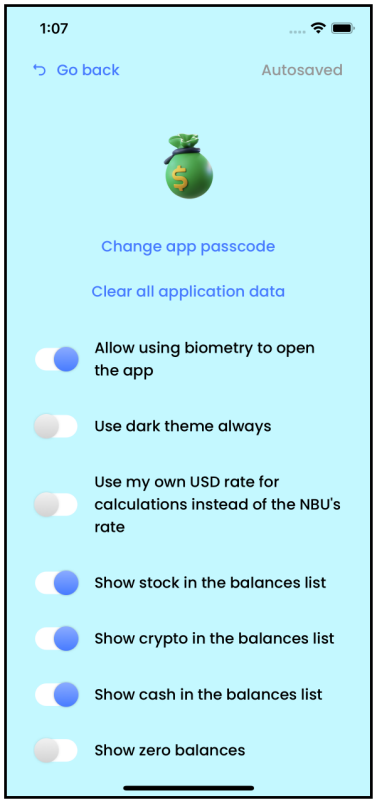

Рисунок 4.21

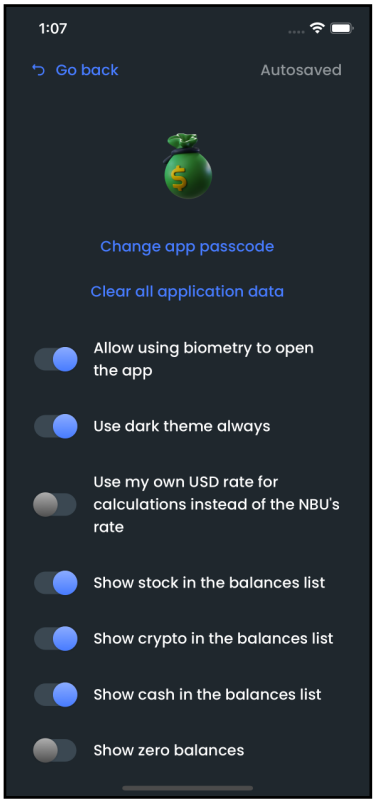

Рисунок 4.22

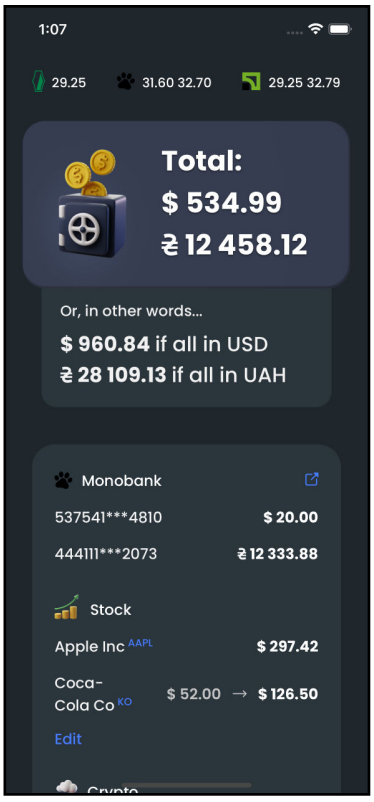

Рисунок 4.23

4.8 Збереження даних у пам’яті

Для перевірки, того, що дані вірно зберігаються у пам’яті, додаток було перезапущено. Завдяки успішній аутентифікації через FaceID, додаток одразу ж відкрив головний екран, не запитуючи пароль. Як видно на малюнках 4.24 — 4.25 усі дані, введені під час минулої сесії, збереглися. Також збереглися значення уведених налаштувань щодо приховання нульових балансів та примусового використання нічної теми.

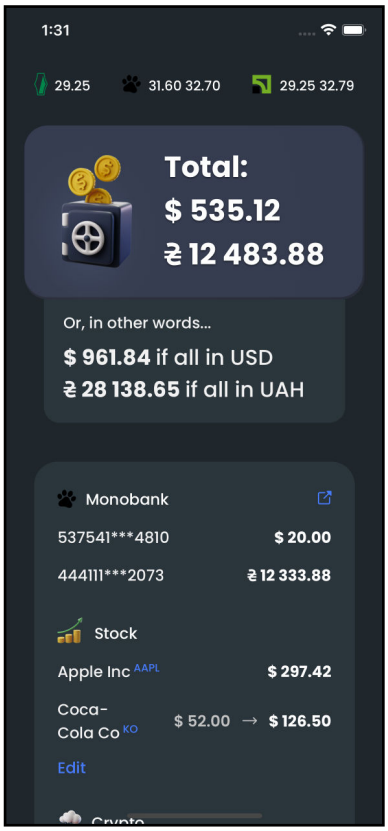

Рисунок 4.24

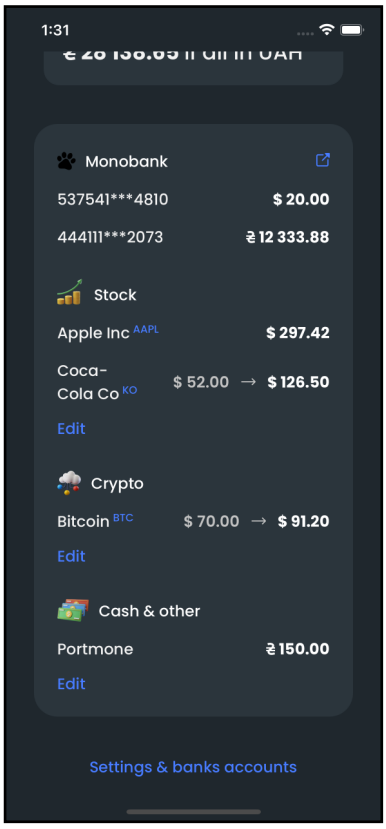

Рисунок 4.25

ВИСНОВКИ

В результаті цієї роботи було створено кросплатформний додаток, що має можливість відстежувати стан під'єднаних банківських рахунків, а також інших накопичень користувача — криптовалют, цінних паперів та готівкових коштів. Опираючись в підрахунках на ці дані, додаток дозволяє користувачу також відстежувати свій сумарний бюджет та завдяки чому краще ним керувати.

Спроєктований додаток є простим у застосуванні, при цьому застосовуючи усі сучасні стандарти функціональності — такі як наявність темної теми та сплеш-скріну. Усі дані, що зберігаються у додатку або передаються між ним та серверами є захищеними від несанкціонованого доступу третіх осіб завдяки використанню пароля, біометричних даних та шифрування. В цілому додаток містить усі попередньо заплановані функції, що працюють або так само, або краще, ніж планувалося.

Результат роботи може бути представлено на ринку сучасних мобільних додатків для конкурування з іншими платформами такого ж напряму. Оскільки функціональність, імплементована додатком, має попит серед користувачів, його створення було цілком виправданим.

Доцільним рішенням є продовжити роботу над додатком для збільшення кількості вбудованих у нього функцій, оскільти сфера фінансів і фін-трекінгу пропонує чимало додаткових можливостей для автоматизації.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Monobank open API [Електронний ресурс] – Режим доступу до ресурсу: https://api.monobank.ua/docs/
2. PrivatBank API [Електронний ресурс] – Режим доступу до ресурсу: https://api.privatbank.ua/#p24/main
3. Опис API для взаємодії з серверною частиною Автоклієнта версія 3.0.0 [Електронний ресурс] – Режим доступу до ресурсу: https://bit.ly/399lBnk
4. Figma – Вікіпедія [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Figma
5. React Native – Вікіпедія [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/React_Native
6. TypeScript – Вікіпедія [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/TypeScript
7. Android Studio – Вікіпедія [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Android_Studio
8. Keychain (software) – Вікіпедія [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Keychain_(software)
9. SharedPreferences | Android Developers [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/reference/android/content/SharedPreferences
10. EncryptedSharedPreferences | Android Developers [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/reference/androidx/security/crypto/EncryptedSharedPreferences
11. Redux – Вікіпедія [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Redux

12. Free Vectors, Stock Photos & PSD Downloads | Freepik [Електронний ресурс] – Режим доступу до ресурсу: https://www.freepik.com/
13. Redux Essentials, Part 1: Redux Overview and Concepts | Redux [Електронний ресурс] – Режим доступу до ресурсу: https://redux.js.org/tutorials/essentials/part-1-overview-concepts

ДОДАТОК А

```
export const INITIAL_STATE: RootState = {
  enterAttempts: 0,
  noInternet: false,

  banksCurrencyRates: {
    nbu: {
      rates: [0],
    },
    mono: {
      rates: [0, 0],
    },
    privat: {
      rates: [0, 0],
    },
  },

  generalCurrencyRates: {},

  banksBalances: {
    mono: {},
    privat: {},
    privatFop: {},
  },

  commonBalances: {
    stock: [],
    cash: [],
    crypto: [],
  },

  settings: {
    passcode: '',
    banksData: {
      mono: [],
      privat: [],
      privatFop: [],
    },
    allowBiometry: true,
    alwaysDark: false,
    doNotUseNbu: false,
    ownUsdRate: '',
    showZero: true,
    showStock: true,
    showCrypto: true,
    showCash: true,
  },

  stockRates: {},
  cryptoRates: {},
};
```

ДОДАТОК Б

```
import { createStore, Reducer } from 'redux';
import { INITIAL_STATE } from './initial';
import type { RootState } from '../data';

export type Action =
  | {
      type: 'setState';
      newState: RootState;
    }
  | {
      type: 'changeState';
      fn: (state: RootState) ⇒ void;
    };

export const mainReducer: Reducer<RootState, Action> = (
  state = INITIAL_STATE,
  action,
) ⇒ {
  switch (action.type) {
    case 'changeState':
      const stateDeepCopy = JSON.parse(JSON.stringify(state));
      action.fn(stateDeepCopy);
      return stateDeepCopy;
    case 'setState':
      return action.newState;
    default:
      return state;
  }
};

export const store = createStore(mainReducer);
```

ДОДАТОК В

```
import type { CommonLabelProps } from '../components/molecules';
import type { BankId, InfoPageNames } from '../data';

export type ParamsList = {
  Locked: undefined;
  Passcode: {
    isNewPinCreating: boolean;
  };
  Main: undefined;
  Settings: undefined;
  NewAccount: {
    bankId: BankId;
  };
  EditAccounts: {
    bankId: BankId;
  };
  CashEdit: {
    labelProps: CommonLabelProps;
  };
  StockEdit: {
    labelProps: CommonLabelProps;
  };
  CryptoEdit: {
    labelProps: CommonLabelProps;
  };
  Error: { data: any; screenLabelComp: () ⇒ JSX.Element };
  Info: { pageName: InfoPageNames };
};
```

ДОДАТОК Г

```
type SettingsTypes =
  | {
      type: ‘bankAccounts'; // кнопка управління рахунками банку
      data: { bankId: BankId }; // ідентифікатор банку
    }
  | {
      type: ‘toggler'; // перемикач
      data: { text: string }; // текст біля перемикача
    }
  | {
      type: ‘input’; // поле вводу
      data: {
        placeholder: string; // плейсхолдер
        label?: string; // підпис поля зверху
        tip?: string; // підказка для кристувача
        keyboardType?: TextInputProps[‘keyboardType'];//тип клавіатури вводу
        maxLength: number; // максимальна довжина рядка
        regex: RegExp; // регулярний вираз для перевірки
      };
    }
  | {
      type: ‘button'; // кнопка
      data: {
        text: string; // текст кнопки
        onPress: (nav: NavProp) ⇒ () ⇒ void; // дія при натисканні
      };
    }

export type Setting =
  | SettingsTypes & {
      key?: keyof Settings;
      showIf?: (s: Settings) ⇒ boolean;
    };

export type SettingsBlock = {
  image: ImageData;
  settingsList: Setting[];
};
```

ДОДАТОК Ґ

```
// useEditedList params

export const useEditedList = <
  InputData extends InputValidationData & { [field: string]: any },
  InputValidationData extends { [field: string]: string },
>(
  id: string,
  writeToStoreFn: (list: InputData[]) => (s: RootState) => void,
  initState: InputData[],
  validationMap: { [field in keyof InputValidationData]: keyof typeof
REGEX },
  defaultValue: InputData,
  InputComp: React.FC<ComplexInputProps<InputData, InputValidationData>>,
  renderFooter?: (() => JSX.Element) | null | undefined,
  onSaveInputAction?: (
    changeInputState: (changeFn: (inputState: InputData) => InputData) =>
void,
    setFieldError: (field: keyof InputValidationData) => void,
    currState: InputData,
    index: number,
    allState: InputData[],
  ) => Promise<void> | void,
)


// useEditedList result
{
    renderList: () => (
      <BgView>
        <TopButtons
          onSavePress={onSavePress}
          errorText={isWrongData(errorsList) && 'Wrong data'}
          loading={isLoading}
        />
        <FlatList
          ListHeaderComponent={renderHeader}
          ListFooterComponent={
            renderFooter ? FooterPadding(renderFooter) : DEFAULT_FOOTER
          }
          data={currState}
          renderItem={renderInput}
          keyExtractor={(_, index) => `${id}-input-${index}`}
          showsVerticalScrollIndicator={false}
          keyboardShouldPersistTaps="handled"
        />
      </BgView>
    ),
  };
```

ДОДАТОК Д

```
{
    icon: MonoIcon,
    name: 'Monobank',
    deeplink: {
      url: 'mono://app',
      playStoreId: 'com.ftband.mono',
      appStoreId: 1287005205,
    },
    credentials: [
      {
        name: 'Token',
        key: 'token',
        maxLength: 60,
        validRegex: /^(\w|\/|=|\+|\?|\.|⊣|#|\$|@|!|%|&|\*){20,}$/,
      },
    ],
    getRequestData: ({ token }) => ({
      method: 'get',
      url: 'https://api.monobank.ua/personal/client-info',
      headers: { 'X-token': token },
    }),
    convertResponseToBalances: response => {
      const balances =
        response.accounts?.map((acc: any) => ({
          source: acc.maskedPan[0],
          currency: CURRENCIES_CODES[acc.currencyCode],
          amount: (acc.balance - acc.creditLimit) / 100,
        })) || [];

      if (response.jars) {
        balances.push(
          ...response.jars.map((jar: any) => ({
            source: jar.title,
            currency: CURRENCIES_CODES[jar.currencyCode],
            amount: jar.balance / 100,
          })),
        );
      }

      return balances;
}
```