Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

Кваліфікаційна робота

освітній ступінь – бакалавр

на тему: «**РОЗРОБКА ANDROID БІБЛІОТЕКИ З ГРАФІКАМИ ЗМІНИ КУРСУ ВАЛЮТ ТА КРИПТОВАЛЮТ В СВІТІ**»

Виконав: студент 4-го року навчання,

Освітньої програми «Інженерія програмного забезпечення», 121

Джосан Володимир Олегович

Керівник: Борозенний О.С.,

Рецензент _____

Кваліфікаційна робота захищена
з оцінкою _____

Секретар ЕК _____

«____» _____ 2023 р.

Київ 2023

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ
Зав.кафедри мультимедійних систем,
доцент, к.ф-м.н.
_____О. П. Жежерун (підпис)
„____”_____2023 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на кваліфікаційну роботу
студенту Джосану Володимиру Олеговичу
Факультету інформатики 4 р.н. бакалаврської програми
ТЕМА: Розробка Android бібліотеки з графіками зміни курсу валют та криптовалют в світі

Зміст ТЧ до кваліфікаційної роботи:

Індивідуальне завдання
Вступ
1. Аналіз предметної області
2. Архітектура бібліотеки. Теоретичні відомості щодо використаних технологій
3. Опис реалізації Android-бібліотеки
Висновки
Список літератури
Додатки (за необхідністю)

Дата видачі „___” _____ 2022 р. Борозенний О.С. _____ (підпис)

Завдання отримав _____ (підпис)

Календарний план виконання роботи

Тема: Розробка Android бібліотеки з графіками зміни курсу валют та криптовалют в світі

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на дипломну роботу.	07.07.2022	
2.	Аналіз матеріалів за темою.	25.02.2023	
3.	Розробка Android-бібліотеки.	30.04.2023	
4.	Написання текстової частини роботи	10.05.2023	
5.	Створення презентації та написання доповіді для попереднього захисту	10.05.2023	
6.	Попередній захист кваліфікаційної роботи	11.05.2023	
7.	Редагування текстової частини роботи, презентації та доповіді	24.05.2023	
8.	Здача кваліфікаційної роботи на перевірку	24.05.2023	
9.	Захист кваліфікаційної роботи.	31.05.2023	

Джосан В. О. _____

Борозенний С. О. _____

“_____”_____

Зміст

Анотація 5

Вступ 6

Основна частина 9

1. Аналіз предметної області 9

1.1. Аналіз важливості валют та криптовалют в сучасному світі 9

1.2. Аналіз та вибір публічних API 12

1.3. Висновки до розділу 1 14

2. Архітектура бібліотеки. Теоретичні відомості щодо використаних технологій 15

2.1. Архітектура бібліотеки 15

2.2. Чому Kotlin? 18

2.3. Використання бібліотеки Retrofit в Android проекті 20

2.4. Використання бібліотеки RxJava в Android проекті 22

2.5. Використання технології WebSocket в Android проекті 25

2.6. Висновки до розділу 2 28

3. Опис реалізації Android-бібліотеки 29

3.1. Особливості реалізації “core” модулю 30

3.2. Особливості реалізації модулю криптовалют 31

3.3. Особливості реалізації модулю валют 40

3.4. Особливості реалізації головного модулю бібліотеки 43

Висновок 56

Список використаних джерел 57

Анотація

Метою бакалаврської роботи є створення зручної та функціональної Android-бібліотеки, яка надаватиме інструменти для роботи з даними про зміни курсу валют ти криптовалют в світі. Бібліотека дозволить розробникам легко додавати до свого мобільного застосунку графіки зі статистичною інформацією про валюти та криптовалюти (як елементи інтерфейсу користувача), кастомізувати їх, а також надаватиме API для роботи з цим даними.

Ключові слова: валюта, криптовалюта, графік, Android, Android-бібліотека, Binance, Exchangerate.host, багатомодульність, Kotlin, Retrofit, RxJava, RxAndroid, WebSocket, OkHttp.

Вступ

Актуальність теми та її практичне значення

Сьогодні смартфони - це незамінна річ в повсякденному житті для кожної людини. Ми постійно використовуємо смартфони та мобільні додатки для різних цілей. Широке розповсюдження мобільних пристроїв створило підвищений попит на інструменти, які забезпечують швидкий і легкий доступ до інформації. Також це стосується сфери валютних і криптовалютних ринків. З огляду на зростаюче значення як традиційних, так і цифрових валют, розробка Android-бібліотеки з графіками курсів валют і криптовалют є дуже актуальною і має важливе практичне значення.

Валюта завжди відігравала важливу роль у світовій економіці, а зі стрімким зростанням криптовалют все більше приватних осіб та підприємств входять у світ торгівлі цифровими активами та інвестування. Враховуючи це, існує велика потреба в надійних інструментах, які надають користувачам точну інформацію про обмінні курси. Це особливо актуально в мобільній сфері, де користувачі очікують, що матимуть можливість швидко отримувати доступ до ринкових даних і приймати обґрунтовані рішення на ходу.

Крім того, зростаюча кількість мобільних додатків, пов'язаних з операціями з валютою та криптовалютою, свідчить про практичну значущість цієї теми. Розробники постійно шукають інноваційні рішення, які спрощують інтеграцію фінансових даних у свої додатки, зберігаючи при цьому точність і зручність для користувача. У зв'язку з цим розробка бібліотеки, що полегшує доступ до інформації про курси валют і криптовалют, набуває особливого значення.

Враховуючи вищенаведену актуальність теми, *метою даної роботи* була поставлена розробка Android-бібліотеки, що надаватиме інструменти для візуалізації зміни курсу валют та криптовалют в світі у вигляді графіків, а також можливість роботи з цими даними, через надане бібліотекою API.

Розробка такої бібліотеки матиме позитивний вплив не лише на користувачів мобільних додатків, але й на розробників, які прагнуть інтегрувати дані про валютні та криптовалютні обмінні курси в свої мобільні застосунки. Завдяки простоті інтеграції та налаштування бібліотеки, розробники зможуть швидко створювати та модифікувати додатки, що відповідають конкретним вимогам користувачів.

Об'єкт дослідження даної роботи — процес створення андроїд бібліотеки з графіками зміни курсу валют та криптовалют у світі, процеси використання публічних API для валют та криптовалют для отримання даних та їх застосування як джерел даних в бібліотеці, процеси створення графіків як ui-елементів для візуалізації статистики зміни курсу валют та криптовалют.

Предмет дослідження даної роботи — використання бібліотек Retrofit, RxJava та технології WebSocket для створення Android-бібліотеки з графіками зміни курсу валют та криптовалют у світі з використанням публічних API для валют та криптовалют як джерел даних.

Структура роботи

Робота складається з анотації, вступу, основної частини, яка в свою чергу складається з трьох розділів, висновків, списку літератури та додатків.

Постановка задачі

1. Проаналізувати, як саме можуть бути представлені різні статистичні дані, пов’язані зі зміною курсу валют та криптовалют. Обрати та використати публічні API для валют та криптовалют.
2. Розглянути і дослідити бібліотеки Retrofit та RxJava та технологію WebSocket для створення Android-бібліотеки з використанням публічних API для валют та криптовалют.
3. Створити Android-бібліотеку для візуалізації статистичних даних про зміни курсів валют та криптовалют.

Основна частина

1. Аналіз предметної області

1.1. Аналіз важливості валют та криптовалют в сучасному світі

Гроші завжди були невід'ємною частиною життя людей протягом усієї історії. Сьогодні у світі існує безліч валют, і кожна країна випускає свою власну грошову одиницю. Ці валюти мають свої особливості та вартість, яка визначається такими факторами, як економіка країни, політична стабільність та світовий ринок.

Також, останнім часом криптовалюти набувають все більшої популярності як альтернативна форма грошей. На відміну від традиційних валют, криптовалюти є децентралізованими і не пов'язані з жодною конкретною країною чи центральною владою. Вони базуються на криптографічних алгоритмах та технології розподіленого реєстру (блокчейн), для своєї цілісності та безпеки. Деякими з ключових характеристик криптовалют є їх цифрова природа, анонімність та відсутність посередників, таких як банки, що призводить до швидших, безпечніших та прозоріших транзакцій.

Розуміння статистики валют і криптовалют, обізнаність в їх розвитку та історії має вирішальне значення для багатьох людей, чия діяльність тісно пов’язана з фінансами. Це, наприклад, трейдери, інвестори, банкіри, фінансові аналітики та консультанти, та й навіть звичайні люди, які планують вкладати свої гроші або відкривати власний бізнес. Розуміння характеристик валют і криптовалют, аналіз їх статистики та графіків може допомогти у прийнятті обґрунтованих рішень, зменшенні ризиків, збільшенні прибутку та ефективному

плануванні фінансових дій. Ця статистика може бути представлена за допомогою різних графіків, таких як лінійні, гістограми або свічкові діаграми, які відображають зміни курсу валют та криптовалют та їх тенденції протягом певних відрізків у часі. Такий спосіб їх візуалізації дозволяє людям ефективно аналізувати коливання та динаміку цін та історичні патерни валют і криптовалют, полегшуючи розуміння складної фінансової інформації та прийняття оптимальних рішень.

При аналізі валют та криптовалют різні типи графіків можуть відображати різні статистичні дані, які можуть бути корисними для оцінки змін курсів, торгівельних обсягів та іншої інформації. Основні та найпопулярніші типи графіків - це лінійні, гістограми та свічкові.

Лінійні графіки представляють собою послідовність точок, що з'єднані між собою лініями, де кожна точка відповідає ціні валюти чи криптовалюти в певний час. Вони часто використовується для відображення курсів обміну валют та криптовалют, а також їх коливань протягом часу. Лінійні графіки можуть бути корисні для виявлення трендів, зокрема зростання чи спаду курсів, а також для порівняння різних валют або криптовалют.

Графіки-гістограми представляють собою послідовність вертикальних стовпців, що відображають розподіл числових даних. Наприклад, у випадку криптовалют, такі графіки можуть бути корисними для відображення обсягу торгів за певний період часу, демонструючи динаміку ринку.

Свічкові графіки є особливо популярними для представлення історії цін криптовалют, оскільки вони надають детальне уявлення про ринкову діяльність. Кожен період часу в такому графіку зображений у вигляді так званої “свічки”, яка відображає 4 ключові показники: ціну відкриття і

закриття періоду та мінімальну і максимальну ціну криптовалюти за цей період часу. Свічкові графіки допомагають виявляти мінливість цін, прогнозувати можливі майбутні рухи цін, а також розробляти торговельні стратегії, реагуючи на зміни на ринку криптовалют.

Різні типи графіків можуть відображати різні статистичні дані для валют та криптовалют. Кожен з них має свої особливості та може бути корисним для різних аналітичних потреб.

1.2. Аналіз та вибір публічних API

Як було зазначено раніше, за мету кваліфікаційної роботи було обрано розробку Android-бібліотеки, яка надаватиме розробникам, що її використовуватимуть, такі UI-елементи як графіки зміни курсу валют та криптовалют у світі, а також API для роботи з цими даними. Для отримання даних про курси валют і криптовалют, а також їх відповідну репрезентацію за допомогою розробленої Android-бібліотеки, було заплановано використати публічні API, які надають таку інформацію.

Отже, було проведено аналіз більшості публічних сервісів, які надають статистичну інформацію про валюти і криптовалюти в світі за наступними критеріями:

- Наявність необхідної інформації, яку надає сервіс.
- Надійність та швидкість сервісу, його популярність.
- Відмовостійкість сервісу (час безвідмовної роботи).
- Кількість символів (валют/криптовалют), для яких сервіс надає необхідну інформацію.
- Діапазон часу, для якого сервіс надає інформацію про історію зміни курсу відповідних символів.
- Відсутність обмеження на кількість запитів до сервісу на місяць.
- Безкоштовне використання сервісу.

Після детального дослідження доступних сервісів, які відповідають усім вищезазначеним умовам, було обрано два публічних API: для валют - exchangerate.host API [1], для криптовалют - Binance API [2].

Exchangerate.host API - це простий та безкоштовний сервіс для отримання поточних курсів валют і криптовалют та історії їх змін. Даний сервіс було обрано для отримання інформації про валюти світу.

На відміну від інших API, Exchangerate.host API відповідає усім умовам, за якими було проведено пошук сервісів. Він надає інформацію про понад 170 валют за останні 20 років, з моменту запуску сервісу у 2020 році працює 99.99% часу, повністю безкоштовний та без обмежень по кількості запитів на місяць.

Binance API - надійний, швидкий та безкоштовний сервіс, який надає інформацію про різні валютні та криптовалютні пари. Цей сервіс надається однією з найпопулярніших бірж криптовалют у світі – Binance [3], що і визначає його популярність та надійність.

Кількість криптовалютних пар, які підтримує Binance API постійно змінюється, оскільки нові криптовалюти постійно добавляються, а старі - іноді вилучаються зі списку. На момент створення бібліотеки, сервіс підтримував понад 2000 криптовалютні пари. Стосовно статистики безвідмовної роботи сервісу - вона публічно не розголошується, проте, як заявляє команда Binance, вони прагнуть забезпечити стабільну та ефективну платформу для своїх користувачів. Також у сервісу немає обмеження на кількість запитів в місяць, проте є ліміт на кількість запитів у хвилину, що було враховано при розробці Android-бібліотеки. В цілому - цей варіант був найкращим серед усіх інших сервісів, що надають дані про криптовалюти, тому обрано було саме його.

1.3. Висновки до розділу 1

У першому розділі було проаналізовано важливість валют та криптовалют у сучасному світі, з чого і випливає актуальність даної теми. Також було розглянуто типи графіків, які можуть найкраще підійти для відображення різної статистики про зміни курсу валют та криптовалют: лінійні графіки, графіки-гістограми та свічкові графіки.

Крім цього, було досліджено доступні публічні API, для отримання історичних даних та даних в реальному часі для валют та криптовалют, а також описані критерії, за якими було зроблено вибір даних сервісів. В результаті було обрано два публічних API: для валют – Exchangerate.host API, та для криптовалют – Binance API.

2. Архітектура бібліотеки. Теоретичні відомості щодо використаних технологій

2.1. Архітектура бібліотеки

Перед початком розробки Android-проекту (застосунку чи бібліотеки), перед розробником зазвичай стоїть задача – визначитись з його архітектурою, а саме чи його проект буде одномодульним, чи багатомодульним. Це рішення залежить від багатьох факторів, які треба передбачити наперед: розмір проекту та команди розробників, складність проекту, потреба в його розширенні в майбутньому, підтримка та супровід.

Нижче буде розглянуто основні переваги і недоліки обох архітектурних підходів.

Одномодульний Android-проект є зазвичай гарним вибором для невеликих застосунків, які мають обмежений функціонал, оскільки їх набагато легше налаштовувати та розробляти. Також, якщо над проектом працює невелика команда, або й взагалі одна людина, це також може спростити процес розробки та контролю версій. Ще однією перевагою одномодульності, є швидший час збірки проекту у випадку, якщо він містить небагато залежностей та ресурів. Проте такий підхід містить також немало недоліків. Оскільки один модуль з часом може надто сильно розростись, це може призвести до обмеженості в масштабуванні, зменшенні можливості перевикористання коду та погіршення підтримки проекту.

Інший архітектурний підхід - це розділення проекту на окремі автономні частини - модулі. Кожен модуль є незалежним і служить якійсь чіткій визначеній меті. Така модуляризація проекту має багато переваг:

- Масштабування. В одномодульному проекті, де присутня тісна зв’язаність залежностей, одна зміна може спричинити каскад змін у, здавалося б, не пов'язаних між собою частинах коду. Проект, належним чином розбитий на модулі, відповідатиме принципу розподілу відповідальності і, таким чином, обмежуватиме тісну зв’язаність.
- Інкапсуляція. Кожна частина коду знає мінімально про інші частини. Такий ізольований код легше читати і розуміти.
- Повторне використання коду. Оскільки кожен модуль вузькоспеціалізований та незалежний, їх можна повторно використовувати в інших проектах.
- Легке тестування. Розбиваючи проект на модулі, кожен компонент можна протестувати окремо, цим самим гарантуючи, що функціонал та поведінка кожного модуля є правильною, перш ніж інтегрувати його з іншими частинами застосунку.
- Швидша збірка проекту. Такі особливості Gradle [4], як інкрементна збірка, кешування та паралельна збірка, також покращують продуктивність збірки багатомодульного проекту.

Звісно, багатомодульність також має і свої недоліки, які також необхідно враховувати під час планування архітектури Android-проекту. Це, наприклад, більш складне налаштування проекту, повільніша початкова збірка, накладні витрати у вигляді підвищеної складності збірки та більш складне ознайомлення з проектом нових розробників.

Отже, рішення стосовно архітектури проекту є дуже важливим початковим етапом перед його створенням. Перед його прийняттям, розробникам важливо ретельно проаналізувати унікальні характеристики та вимоги до своїх майбутніх проектів. Це рішення допоможе якісно

сформувати структуру проекту, що сприятиме більш ефективному процесу розробки.

Перед розробкою Android-бібліотеки, як практичної частини даної роботи, було зважено усі переваги і недоліки одномодульних та багатомодульних Android-проектів описаних вище. Функціонал бібліотеки можна розділити на три чітких частини: робота з даними про валюти, криптовалюти та візуалізація цих даних у вигляді графіків. Більше того, як джерела даних про валюти і криптовалюти було обрано два різних публічних API, які потребують окремого підходу. Таким чином, було прийнято рішення виокремити логіку роботи з даними про валюти та криптовалюти в два окремі модулі і створити один основний модуль, який об’єднуватиме увесь функціонал бібліотеки та в якому будуть реалізовані самі графіки. Також, для визначення основних сутностей та інтерфейсів, як протоколів взаємодії, було створено окремий “core” модуль.

Така архітектура бібліотеки є гарним рішенням, оскільки кожен модуль відповідає лише за одну задачу та ізольований один від одного. Таким чином, у випадку, якщо в майбутньому, наприклад, буде потреба замінити реалізацію одного з модулів або змінити джерело даних, через закриття або зміну в політиці роботи того чи іншого сервісу, через те, що проект працює за визначеним протоколом і в ньому невисока зв’язаність залежностей, це буде достатньо легко зробити.

2.2. Чому Kotlin?

Kotlin — це статично типізована мова програмування з відкритим кодом, яка підтримує як об’єктно-орієнтоване, так і функціональне програмування. Kotlin має схожі синтаксичні риси і концепції з інших мов, включаючи C#, Java і Scala. Kotlin, як і Java, компілюється в байт-код та виконується JVM.

Як було сказано раніше, Kotlin - це об’єктно орієнтована мова, проте вона також має багато елементів функціонального програмування, такі як функції вищого порядку та лямбда-вирази.

У липні 2011 року компанія JetBrains [8] представила Project Kotlin, нову мову програмування, яка виконується JVM. Випущеною мова була 15 лютого 2016 року, що вважається першим офіційно стабільним релізом. До появи Kotlin, основною мовою програмування для Android була Java. Проте в 2017 році компанія Google почала підтримувати Kotlin у своєму середовищі розробки Android Studio, що зумовило перехід багатьох Android-розробників на нову мову програмування. У 2019 році, на щорічній конференції розробників Google I/O [9], компанією Google було оголошено, що Kotlin стає основною мовою програмування для Android.

Kotlin має певні значні переваги над Java.

Синтаксис Kotlin сильно спрощує написання коду, що робить його більш лаконічним і читабельним, ніж код, написаний на Java. Також в Kotlin менше шаблонного коду, що робить процес програмування набагато ефективнішим і приємнішим. Це значить, що написавши менше строчок коду, та потративши на їх написання менше часу, можна досягти тих самих цілей, що і за допомогою Java.

Система типів у Kotlin спрямована на усунення небезпеки null-посилань. Однією з найпоширеніших проблем у багатьох мовах програмування, включаючи Java, є те, що можливий доступ до посилань, для яких відсутнє значення (тобто null-посилання), що призводить до виняткових ситуацій та некоректного завершення роботи програми. У Kotlin система типів розрізняє посилання, які можуть мати значення null (nullable посилання), і ті, які не можуть (non-null посилання) під час компіляції, що і забезпечує безпеку коду.

Мова програмування Kotlin повністю сумісна з Java. Розробники мають доступ до всіх фреймворків і бібліотек Java.

Вищезазначені переваги та підтримка Google є одними із найголовніших причин, чому Kotlin вибирають так багато розробників, як мову розробки своїх Android-застосунків. Тому, саме мову програмування Kotlin було обрано для написання Android-бібліотеки, як практичної частини даної роботи.

2.3. Використання бібліотеки Retrofit в Android проекті

Бібліотека Retrofit [10] - це типобезпечний HTTP-клієнт для Android та Java, розроблений компанією Square [11]. Це потужна та ефективна бібліотека, яка спрощує процес створення мережевих запитів та обробки відповідей від RESTful API. Retrofit стала основною бібліотекою для Android завдяки простоті використання, розширюваності та надійності.

Retrofit в основному використовується для перетворення кінцевих точок API в інтерфейси Java. За допомогою анотацій можна легко визначати HTTP методи і параметри запитів. Після чого бібліотека згенерує типобезпечний клієнт, за допомогою якого можна буде виконувати запити та отримувати відповіді. Основна перевага бібліотеки Retrofit полягає в тому, що вона усуває необхідність написання громіздкого коду для взаємодії з RESTful API, дозволяючи більше зосередитися на реалізації функціоналу проекту.

Бібліотека Retrofit має кілька особливостей, які виділяють її серед інших конкурентів. Це, наприклад, підтримка синхронних та асинхронних запитів, інтеграція з популярними бібліотеками для серіалізації, такими як Gson [12] та Moshi [13], кастомні конвертери та можливість легко обробляти помилки за допомогою перехоплювачів (interceptors). Крім того, Retrofit підтримує різні HTTP методи, такі як GET, POST, PUT, DELETE і PATCH, а також за допомогою цієї бібліотеки можна завантажувати і вивантажувати файли.

Під час виконання практичного завдання, дана бібліотека була використана для взаємодії з двома публічними API: Binance API та exchange.host API. Обидва ці сервіси надають RESTful API, саме тому ця бібліотека гарно підійшла для взаємодії з ними. Для серіалізації відповідей від серверу, була використана бібліотека Gson, адже, як було зазначено

раніше, Retrofit легко з нею інтегрується. Також, оскільки запити до серверів повинні відбуватись асинхронно, Retrofit було використано в комбінації ще з однією бібліотекою, яка надає таку можливість – RxJava.

```
class BinanceRetrofit {

    companion object {

        private const val BASE_URL = "https://api.binance.com/"

        fun makeRetrofitService() : BinanceRetrofitService {
            val client = OkHttpClient.Builder()
                .addInterceptor(NoSpamInterceptor())
                .addInterceptor(WeightLimitInterceptor())
                .build()
            return Retrofit.Builder()
                .baseUrl(BASE_URL)
                .addConverterFactory(GsonConverterFactory.create(GsonBuilder().setLenient().create()))
                .addCallAdapterFactory(RxJava3CallAdapterFactory.create())
                .client(client)
                .build()
                .create(BinanceRetrofitService::class.java)
        }

    }

}
```

Рис. 1 Приклад використання бібліотеки Retrofit

```
interface BinanceRetrofitService {

    @GET("/api/v3/exchangeInfo")
    fun getExchangeInfo() : Single<ExchangeInfoResponse>

    @GET("/api/v3/klines")
    fun getCandlestickHistory(
        @Query("symbol") symbol: String,
        @Query("interval") interval: String,
        @Query("limit") limit: Int,
        @Query("endTime") startTime: Long? = null,
        @Query("endTime") endTime: Long? = null
    ) : Single<List<List<Any>>>

}
```

Рис. 2 Інтеграція бібліотеки RxJava в Retrofit

2.4. Використання бібліотеки RxJava в Android проекті

RxJava [14] - це бібліотека, яка використовує функціонально-реактивний стиль програмування, для створення асинхронних потоків даних у реальному часі та подійно-орієнтованих програм за допомогою спостережуваних послідовностей. Дана бібліотека використовує патерн “Спостерігач” (“Observer”) та наслідує його концепцію спостерігачів (Observers) та об’єктів спостереження (Observables). Таким чином, в даній бібліотеці можна визначити два основних компоненти: Observable та Observer.

Observable - це потік даних, який надсилає елементи даних (або події) протягом певного часу. Він може видавати будь-яку кількість елементів (або взагалі не видати жодного), а також може надсилати помилку або сигнал завершення. Observer’и є джерелами даних у моделі реактивного програмування. Вони представляють асинхронний потік даних, на який можуть підписатися інші компоненти. Таким компонентом є Observer.

Observer - це споживач даних, які надсилає об'єкт спостереження. Він підписується на Observable, щоб отримувати елементи даних, помилки або сигнали завершення.

Усі дані, які надсилаються об’єктом спостереження, можуть бути певним чином опрацьовані перед споживанням спостерігачами, наприклад трансформовані (оператор map), об’єднані (оператор merge), відфільтровані (оператор filter) або інакшим чином змінені.

Ще одним важливим елементом даної бібліотеки є також планувальник (Scheduler). Він відповідає за визначення потоку, в якому має бути виконана певна операція (наприклад, підписка на Observable або обробка відправлених ним елементів). RxJava надає декілька стандартних

планувальників, наприклад для задач з великим навантаженням на процесор (computation), задач пов’язаних з введенням/виведенням (io), а також спеціальний планувальник для Android, для задач пов’язаних з оновленням інтерфейсу (mainThread), що повинні виконуватись на головному потоці.

Останній планувальник не входить до стандартної бібліотеки RxJava, а до її спеціального розширення для Android - бібліотеки RxAndroid [15]. RxAndroid - це розширена бібліотека від RxJava, призначена для задоволення специфічних вимог Android та надання додаткових утиліт, які спрощують розробку додатків. RxAndroid додає новий, специфічний функціонал для Android платформи, такий як вищезгаданий планувальник для основного потоку Android, підтримку управління підписками, що залежать від життєвого циклу активностей чи фрагментів, View Bindings [16] та Backpressure handling [17], пристосовані для платформи Android.

У практичній частині роботи, бібліотека RxJava (та її розширення RxAndroid) грають дуже важливу роль, оскільки проект повністю пронизаний її функціоналом. Як уже зазначалось раніше, RxJava була використана у комбінації з іншою бібліотекою – Retrofit, для можливості асинхронно виконувати запити та отримувати результати їх виконання. В основному, для отримання необхідної інформації про валюти і криптовалюти з веб-сервісів, було використано клас Single, який реалізує патерн реактивного програмування для повернення єдиного результату або помилки.

Також, для належної реалізації логіки підписок UI-елементів на зміни поточних даних валют та криптовалют в реальному часі, були використані класи BehaviorProcessor та Flowable. Клас BehaviorProcessor, наприклад, характеризується тим, що він може одночасно бути і

спостерігачем, і об’єктом спостереження. Він може виконувати такі задачі: кешувати останнє надіслане значення і робити його повторне надсилання новим підписникам, бути підписаним на один інший об’єкт спостереження та транслювати елементи, які він надсилає, або власні елементи, своїм підписникам. Клас Flowable, у свою чергу, особливий тим, що підтримує Backpressure. Це механізм для вирішення ситуацій, коли об’єкт спостереження надсилає об'єкти швидше, ніж спостерігач може їх споживати (обробляти). Використовуючи Flowable, можна керувати швидкістю надсилання об'єктів і запобігти перевантаженню спостерігачів. Клас Flowable надає декілька методів вирішення цієї проблеми. Наприклад, можна відкидати нові надіслані елементи, якщо попередні ще не були оброблені, або ж буферизувати їх.

Завдяки таким особливостям класів Single, BehaviorProcessor та Flowable, їх і було успішно та ефективно використано при розробці Android-бібліотеки.

```
fun getAllSymbols(): Single<List<CryptoSymbol>> {
    return retrofit.getExchangeInfo().map { response ->
        val symbols = response.symbols.map { it: SymbolInfo
            CryptoSymbol(
                it.name.lowercase(),
                it.baseAsset.lowercase(),
                it.quoteAsset.lowercase()
            )
        }
        cacheAllSymbols(symbols)
        symbols ^map
    }.subscribeOn(Schedulers.io())
}
```

Рис. 3 Приклад використання бібліотеки RxJava (виклик запиту Retrofit сервісу з типом результату Single та зміна потоку виконання операції)

2.5. Використання технології WebSocket в Android проекті

WebSocket [18] - це комунікаційний протокол, який забезпечує повнодуплексні канали зв'язку через одне TCP-з'єднання. Даний протокол забезпечує взаємодію між клієнтом і веб-сервером із зовсім невеликими накладними витратами, полегшуючи передачу даних в обидві сторони в режимі реального часу. Це можливо завдяки стандартизованому способу надсилання сервером інформації клієнту без окремого на це запиту з його боку, а також завдяки тому, що повідомлення можуть передаватися в обидві сторони, під час того, як з’єднання залишається відкритим. Таким чином, між клієнтом і сервером може відбуватися постійний двосторонній діалог.

Комунікація клієнта та сервера через даний протокол здійснюється через TCP-порт 80 (для незашифрованих даних) і TCP-порт 443 (для зашифрованих даних SSL), які є тими самими портами, що використовуються для HTTP і HTTPS. Це дозволяє WebSocket-з'єднанням обходити багато корпоративних брандмауерів і проксі-серверів. Дві URI схеми, визначені для протоколу WebSocket, це ws (WebSocket) і wss (WebSocket Secure) для незашифрованих і зашифрованих з'єднань відповідно.

Далі розглянемо основний принцип роботи за протоколом WebSocket. Можна виділити 3 основні етапи: відкриття з’єднання, процес передачі даних та закриття з’єднання.

Процес встановлення з’єднання між клієнтом і сервером через WebSocket, відомий як рукостискання, складається з обміну HTTP-запитами/відповідями між ними.

Після рукостискання, клієнт та сервер можуть обмінюватися повідомленнями (фреймами) через постійне WebSocket з'єднання. Ці

повідомлення можуть бути як звичайним текстом, так і двійковими даними. Також, протягом етапу обміну повідомленнями між клієнтом на сервером, використовуються контрольні фрейми, для керуванням з’єднанням. Це “Ping” та “Pong” фрейми, за допомогою яких сервер може дізнатись, чи досі клієнт його “слухає”. Час від часу, сервер надсилатиме “Ping” фрейми, на які клієнт повинен відповісти “Pong” фреймами. Зазвичай, якщо клієнт протягом певного часу не відповідає серверу, сервер ініціює закриття WebSocket з’єднання. Проте, це також залежить від самого сервера, як він опрацьовуватиме дану ситуацію.

Закриття WebSocket з’єднання відбувається після того, як воно виконало свої функції, а саме, коли більше немає необхідності для клієнта чи сервера обмінюватись повідомленнями. Ініціювати закриття з’єднання можуть як клієнт, так і сервер, відправивши спеціальне повідомлення про закриття.

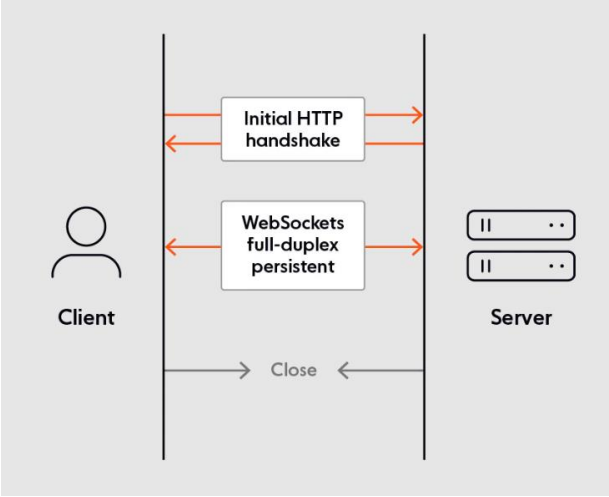


Рис. 4 Принцип роботи протоколу WebSocket

Як уже було зазначено раніше, WebSocket’и забезпечують зв'язок між клієнтом і сервером в режимі реального часу без необхідності частих HTTP-запитів/відповідей. Така особливість роботи даного протоколу забезпечує зменшення затримок, підвищення продуктивності та швидкості спілкування клієнта та сервера. Завдяки своїй постійній і двонаправленій

природі WebSocket протокол є більш гнучким, ніж HTTP, і може бути дуже корисним при створенні застосунків, які повинні працювати в режимі реального часу та які вимагають частого обміну даними. Це, наприклад, чат-додатки, ігри в реальному часі, застосунки з можливістю одночасного редагування (Google Docs та різні онлайн дошки), програми з онлайн трансляціями та інші.

У практичній частині даної роботи, WebSocket протокол було використано, для того, щоб отримувати оновлення поточної інформації про ціну криптовалют в реальному часі через Binance WebSocket API. Для цього було використано досить відому бібліотеку OkHttp [19] від компанії Square. OkHttp - це популярний HTTP-клієнт для Java та Android, який також забезпечує надійну підтримку з'єднання через WebSocket протокол.

```
class BinanceConnection(
    private val request: Request,
    private val client: OkHttpClient,
    private val onConnectionOpened: (() -> Unit)? = null,
    private val onMessageReceived: ((String) -> Unit)? = null,
    private val onConnectionFailure: ((Throwable, Response?) -> Unit)? = null,
    private val onConnectionClosing: (() -> Unit)? = null,
    private val onConnectionClosed: (() -> Unit)? = null
) : WebSocketListener() {

    private var connection: WebSocket? = null

    override fun onOpen(webSocket: WebSocket, response: Response) {
        super.onOpen(webSocket, response)
        onConnectionOpened?.invoke()
    }

    override fun onMessage(webSocket: WebSocket, text: String) {
        super.onMessage(webSocket, text)
        onMessageReceived?.invoke(text)
    }

    override fun onFailure(webSocket: WebSocket, t: Throwable, response: Response?) {
        super.onFailure(webSocket, t, response)
        this.connection = null
        onConnectionFailure?.invoke(t,response)
    }

    override fun onClosing(webSocket: WebSocket, code: Int, reason: String) {
        super.onClosing(webSocket, code, reason)
        onConnectionClosing?.invoke()
    }

    override fun onClosed(webSocket: WebSocket, code: Int, reason: String) {
        super.onClosed(webSocket, code, reason)
        onConnectionClosed?.invoke()
    }
```

Рис. 5 Приклад використання технології WebSocket за допомогою бібліотеки OkHttp

2.6. Висновки до розділу 2

У другому розділі було описано два підходи для організації архітектури Android-проекту: одномодульний та багатомодульний проект. Також було аргументовано причини вибору багатомодульної архітектури для розробки Android-бібліотеки як практичної частини кваліфікаційної роботи.

Крім цього, було розглянуто та описано особливості інструментів, таких як мова програмування, бібліотеки та технології, які були використані під час розробки Android-бібліотеки.

3. Опис реалізації Android-бібліотеки

Як було зазначено раніше, розроблена Android-бібліотека – це мультимодульний проект, який складається з чотирьох частин: “core” модуль, модулі для логіки роботи з криптовалютами та валютами та основний модуль бібліотеки, який об’єднує увесь функціонал.

Далі, розглянемо особливості реалізації кожного з модулів, а також проблеми, які виникали по мірі їх розробки та шляхи їх вирішення.

3.1. Особливості реалізації “core” модулю

Назва цього модулю говорить сама за себе. Він був створений для того, щоб містити в собі основний функціонал, загальні сутності та інтерфейси, які повинні бути доступні в будь-якому місці Android-бібліотеки.

По-перше, це визначення інтерфейсів, як протоколів роботи головного модуля бібліотеки з модулями валют та криптовалют. Для кожного з цих модулів, було визначено “core” інтерфейс, який надає відповідні API для роботи з валютами та криптовалютами: звичайне та Stream API. Звичайне API можна використовувати для отримання статичних даних, які запитуються у відповідного RESTful веб-сервісу, відповідно до модуля. Stream API, у свою чергу, потрібен для підписки на оновлення символів у реальному часі, що здійснюється за допомогою WebSocket протоколу. Звичайне API реалізоване обидвома модулями для валют і криптовалют, в той час як Stream API доступне лише для підписки на зміни криптовалют у реальному часі. Оскільки веб-сервіс для валют exchangerate.host API оновлює дані лише один раз в день, немає необхідності у підтримуванні Stream API для валют.

Також в “core” модулі були визначені основні сутності, які визначають окремі символи валют і криптовалют (CryptoSymbol та CurrencySymbol), інтервали, які використовуються, для запитів історії різних символів (Interval), сутності запитів та результатів для різних методів API (наприклад, запити історії цін, коливань цін, об’єму торгів) та інші.

3.2. Особливості реалізації модулю криптовалют

Даний модуль об’єднує в собі увесь функціонал, пов’язаний з отриманням даних про криптовалюти, їх обробкою та підготовкою до використання кінцевим користувачем та віджетами, визначеними в основному модулі бібліотеки.

У попередніх розділах було описано, що для отримання даних про криптовалюти було використано публічні API – Binance API та Binance WebSocket API.

Розпочнемо з реалізації Stream API для криптовалют. Його можна розбити на чотири шари:

1. CryptocurrencyStreamApiImpl - реалізація інтерфейсу з “core” модулю (CryptocurrencyStreamApi), для визначення основного функціоналу Stream API для криптовалют.
2. BinanceStreamApi - сінглтон клас, який по своїй суті реалізовує патерн програмування “Спостерігач”. В ньому відбувається управління підписками об’єктів на потоки даних різних криптовалютних пар. Підписка на той чи інший потік відбувається за допомогою наступного шару - класу BinanceConnectionManager.
3. BinanceConnectionManager - це клас, за допомогою якого відбувається управління запитами на підписку та відписку від різних потоків, адже Binance WebSocket API має певні обмеження на користування даним сервісом, які будуть описані пізніше.
4. BinanceConnection - клас, в якому відбувається безпосередня взаємодія із Binance WebSocket API. Цей клас реалізує інтерфейс WebSocketListener бібліотеки OkHttp та має методи відкриття та закриття з’єднання з сервером та надсилання йому повідомлення.

```
class CryptocurrencyStreamApiImpl(
    private val binanceStreamApi: BinanceStreamApi
) : CryptocurrencyStreamApi {

    override fun listenToCandlestickUpdates(request: CryptoStreamRequest, listenHolder: Any,
                                            listenOnUi: Boolean, listener: CandlestickListener) {
        binanceStreamApi.listenToCandlestickUpdates(request, listenHolder, listenOnUi, listener)
    }

    override fun removeCandlestickListenHolder(listenHolder: Any) {
        binanceStreamApi.removeCandlestickListenHolder(listenHolder)
    }

    override fun removeCandlestickListener(request: CryptoStreamRequest, listenHolder: Any) {
        binanceStreamApi.removeCandlestickListener(request, listenHolder)
    }

}
```

Рис. 5 Клас CryptocurrencyStreamApiImpl

Якщо принцип роботи останнього шару (BinanceConnection) досить очевидний, то причини створення та деталі роботи інших шарів, управління підписками та запитами (BinanceStreamApi та BinanceConnectionManager), варто пояснити.

Як було сказано вище, Binance WebSocket API певним чином обмежує об’єми користування сервісом з однієї IP адреси:

1. WebSocket з’єднання має ліміт в 5 повідомлень на секунду. Повідомленням вважається “Ping” фрейм, “Pong” фрейм та будь-яке повідомлення на підписку на потік, чи відписку від нього. WebSocket з’єднання, яке виходить за ці межі, буде розірвано. IP-адреси, які порушуватимуть це обмеження та примусово відключатимуться, можуть бути заблоковані.
2. Одне WebSocket з’єднання може одночасно “слухати” не більше ніж 1024 потоки даних.

Через автоматизацію бібліотекою OkHttp відповідей на “Ping” фрейми та неможливістю змінити чи розширити дану логіку, було

прийнято рішення зменшити ліміт повідомлень на секунду, які міг би відправляти BinanceConnectionManager до 4, щоб випадково не порушити обмеження Binance WebSocket API.

Вважаючи на ці обмеження, було прийнято рішення зробити менеджер (BinanceConnectionManager), який зміг би слідкувати за кількістю повідомлень, що надсилаються до сервера кожної секунди та забороняти підписуватись на більше, ніж 1024 потоки.

Першу проблему було вирішено за допомогою створення черги повідомлень на відправлення. Якщо ж за останню секунду було відправлено менше ніж 4 повідомлення, воно надсилається одразу ж, якщо більше - добавляється в чергу, яка перевіряється кожну секунду з окремого потоку, та за можливості повідомлення надсилається, коли часове обмеження пройшло.

Друга проблема була вирішена звичайною перевіркою на кількість уже існуючих підписок та створенням виняткової ситуації, якщо ліміт було досягнуто.

Ще один важливий клас, який виконує роль управління підписками об’єктів на потоки даних різних криптовалютних пар - це BinanceStreamApi. По суті, з Binance WebSocket API розроблена бібліотека слухає лише один тип потоків (проте для різних криптовалютних пар) - klines (candlestick bars), а тобто інформацію про зміну статистики про певний символ за певний період часу, у вигляді так званих свічкових стовпців. Одне повідомлення, яке отримується з WebSocket з’єднання містить різну інформацію, яка відповідно розбивається у різні класи та постачається усім підписникам потоку.

В бібліотеці було виділено 3 типи підписок на основну статистику криптовалюти:

1. CryptoCandlestickData - свічкові дані з цінами відкриття та закриття періоду і мінімальною та максимальною ціною криптовалюти протягом цього періоду.
2. CryptoLineData - дані з ціною криптовалюти в певну точку часу з певним інтервалом.
3. CryptoVolumeData - дані з обсягом торгів криптовалютою в певну точку часу з певним інтервалом.

Враховуючи такі визначені види підписок на потоки сервісу та типи даних, на які можна розділити отриману інформацію, було обрано наступний підхід для максимально ефективного управління підписками.

Як було сказано раніше, BinanceStreamApi реалізує всім відомий патерн програмування “Спостерігач”. Таким чином, використовуючи Stream API для криптовалют, можна підписатись на потік даних певного типу (CryptoCandlestickData, CryptoLineData, CryptoVolumeData) з будь-якого місця в програмі, зазначивши об’єкт, до якого буде прив’язано підписку.

В класі BinanceStreamApi визначено дві хеш-мапи, за допомогою яких і реалізовано патенр “Спостерігач”. Одна з них зберігає як ключ назву потоку в Binance WebSocket API , що відповідає певному символу та інтервалу, та як значення - BehaviorProcessor, клас бібліотеки RxJava, який реалізовує вищезгаданий інтерфейс Observable та вміє повідомляти підписаних на нього спостерігачів про нові елементи, які до нього надходять.

Друга хеш-мапа - відповідає за збереження інформації про усіх спостерігачів, які підписані на певний потік даних. Ключем цієї мапи є знову ж таки назва потоку (символ та інтервал), а значення - інша мапа, яка розподіляє усіх спостерігачів на 3 типи даних, які вони слухають

(CryptoCandlestickData, CryptoLineData та CryptoVolumeData). Значення цієї мапи - ще одна мапа, яка ставить у відповідність об’єкт, до якого прив’язується підписка, до значення - об’єкта класу Disposable, що є так званим контрактом підписки спостарігача на певний об’єкт спостереження. У даному випадку це підписка на потік даних, що представляється об’єктом спостереження типу BehaviorProcessor з першої описаної хеш-мапи. Особливість класу Disposable - це можливість розірвати контракт підписки, коли він більше не потрібен, за допомогою методу “dispose”. Це дуже корисно, оскільки коли користувачу більше не потрібно буде слухати оновлення даних з певного потоку, він зможе відписатись від цих оновлень і контракт буде розірвано. Це значить що даний спостерігач більше не буде підписаний на даний потік (об’єкт спостереження - BehaviorProcessor) та не отримуватиме оновлень.

За допомогою такої структури даних, було досягнуто ефективного способу управляння підписками на потоки даних, оскільки таким чином, для об’єктів, які повинні слухати різні типи даних з одного потоку (для одного символу з однаковим інтервалом), не створюються окремі об’єкти спостереження, а використовується лише один BehaviorProcessor. І уже відповідно до підписки на конкретний тип даних (CryptoCandlestickData, CryptoLineData та CryptoVolumeData) певного спостерігача, інформація що надходить на основний об’єкт спостереження (BehaviorProcessor) парситься відповідно до контракту підписки в правильну структуру даних.

Проблемою, яку необхідно було вирішити, це можливість витоку пам’яті, у випадку якщо користувач бібліотеки не відпишеться від оновлень, прив’язаних до певного спостерігача, оскільки спостерігачем може бути абсолютно будь-який об’єкт. В Android це можуть бути, наприклад, об’єкти з певним життєвим циклом та об’єкти, пов’язані з контекстом (Context). І якщо таких спостерігачів не відписати від

оновлень перед кінцем їх життєвого циклу - це може призвести до значних витоків пам’яті.

Ця проблема була вирішена використанням колекції WeakHashMap, для зберігання спостерігачів та контрактів підписок на потоки (Disposables), яка використовує для ключів слабкі посилання (weak references), що не перешкоджають збирачу сміття видаляти недосяжні об’єкти, а тобто об’єкти, для яких не існує жорстких посилань (hard references). Таким чином, навіть, якщо не відписати від оновлень такого спостерігача, він буде видалений з колекції підписок. З першого погляду таким чином проблема витоку пам’яті буде вирішена, проте хоч входження даного спостерігача і буде видалено з WeakHashMap, відповідне йому значення - контракт підписки на потік, не буде коректно розірвано, і оновлення продовжитимуть опрацьовуватись для даного об’єкту Disposable.

Для вирішення цієї проблеми, було створено класи WeakDisposable та ReferenceQueueMonitor, суть яких розривати контракт підписки на потік в той момент, коли збирач сміття визначає, що об’єкт спостерігача стає слабодоступним. У сінглтон класі ReferenceQueueMonitor було створено чергу посилань (ReferenceQueue), щоб мати можливість відслідковувати, коли об’єкти спостерігачів ставатимуть слабодоступними, а також мапу фантомних посилань на об’єкти спостерігачів з відповідними їм контрактами підписок (Disposables). Коли створюється об’єкт класу WeakDisposable, за допомогою ReferenceQueueMonitor спостерігач і його контракт реєструються на відслідковування моменту, коли спостерігач стане слабодоступним. Це реалізовано за допомогою створення фантомного посилання на даний об’єкт з указанням ReferenceQueue класу ReferenceQueueMonitor. На даній черзі посилань в нескінченному циклі викликається блокуючий метод remove(), який повертає посилання на

об’єкт, який збирач сміття визначив як слабодоступний. Саме в цей момент клас ReferenceQueueMonitor дістає з мапи відповідний контракт підписки спостерігача і розриває його. Таким чином, і спостерігачі видаляються збирачем сміття, і їхні контракти коректно розриваються, коли в них більше немає потреби.

Бібліотека також надає API для отимання історичних даних про криптовалюти. Для цього, як уже зазначалось раніше, як джерело інформації було використано публічне API від Binance, а також бібліотеку Retrofit для отримання інформації з кінцевих точок веб-сервісу. Також було інтегровано бібліотеку RxJava, для використання її типів для асинхронного отримання результатів запитів, а також бібліотеку Gson, як конвертера результатів запитів. API для криптовалют, що надається бібліотекою, складається з трьох шарів:

1. CryptocurrencyApiImpl - клас, що реалізує інтерфейс, який визначає протокол взаємодії з API для криптовалют, - CryptocurrencyApi.
2. BinanceApi - основний клас, який відповідає за реалізацію взаємодії з публічним API.
3. BinanceRetrofitService - інтерфейс, який визначає запити до кінцевих точок Binance API, для реалізації сервісу бібліотекою Retrofit.

Основним класом, який відповідає за реалізацію взаємодії з публічним API - є сінглтон BinanceApi. Під час його ініціалізації надсилається запит на отримання усіх підтримуваних символів до сервісу, після чого ці дані кешуються. Також даний клас визначає статичний список усіх підтримуваних сервісом інтервалів.

Даний клас надає наступні методи:

- Синхронний метод для отримання усіх закешованих підтримуваних символів
- Синхронний метод для отримання списку підтримуваних інтервалів
- Асинхронний метод для отримання усіх підтримуваних символів
- Три асинхронні методи для отримання історії зміни курсу криптовалюти у вигляді різних типів даних (CryptoCandlestickData, CryptoLineData та CryptoVolumeData).

```
class CryptocurrencyApiImpl(
    private val binanceApi: BinanceApi
) : CryptocurrencyApi {

    override fun getAllCachedSymbols(): List<CryptoSymbol> {
        return binanceApi.getAllCachedSymbols()
    }

    override fun getAllSymbols(): Single<List<CryptoSymbol>> {
        return binanceApi.getAllSymbols()
    }

    override fun getCandlestickHistory(request: CryptoHistoryRequest): Single<CryptoHistoryResponse<CryptoCandlestickData>> {
        return binanceApi.getCandlestickHistory(request)
    }

    override fun getCandlestickLineHistory(request: CryptoHistoryRequest): Single<CryptoHistoryResponse<CryptoLineData>> {
        return binanceApi.getCandlestickLineHistory(request)
    }

    override fun getCandlestickVolumeHistory(request: CryptoHistoryRequest): Single<CryptoHistoryResponse<CryptoVolumeData>> {
        return binanceApi.getCandlestickVolumeHistory(request)
    }

    override fun getSupportedIntervals(): List<Interval> {
        return binanceApi.getSupportedCandlestickIntervals()
    }

    override fun getCandlestickIntervalById(id: String): Interval? {
        return binanceApi.getCandlestickIntervalById(id)
    }
}
```

Рис. 6 Клас CryptocurrencyApiImpl

Використаний сервіс для отримання історичних даних про криптовалюти, Binance API, має певні обмеження на його використання. Для кожного запиту до API визначається його вага, залежно від складності запиту. Всього, для одної IP-адреси, виділяється 1200 одиниць ваги, яку користувач може використати за хвилину. Якщо користувач порушить

даний ліміт, запит не буде виконано, а буде повернена помилка з кодом 429 про перевищення можливої кількості запитів і заголовок “retry-after” з кількістю секунд, яку потрібно зачекати, щоб продовжити звертатись до API. І якщо продовжити надсилати запити з цього IP, буде повернена помилка 418 з сервісу, та в результаті IP-адреса буде заблокована на певний період часу.

Отже, для того щоб запобігти такій ситуації, були прийняті певні міри, щоб обмежити можливість користувачів надсилати надто велику кількість запитів і бути в результаті заблокованими. Це було реалізовано за допомогою двох перехоплювачів запитів (Interceptors) при створенні OkHttp клієнта для Retrofit сервісу. Один з них, WeightLimitInterceptor, перехоплює відповіді API та перевіряє їх на наявність кодів 429 та 418. Якщо було перевищено ліміт запитів або клієнта було заблоковано, перехоплювач виставляє певний прапорець на сінглтон класі BinanceApi. Він визначає, що ліміт на кількість запитів було порушено. Через отриману з відповіді кількість секунд даний прапорець змінить своє значення, щоб можна було продовжувати надсилати запити. Інший перехоплювач, NoSpamInterceptor, перехоплює запити до публічного API та перевіряє цей прапорець на можливість надсилання запиту. І в тому випадку, якщо ліміт було порушено, перехоплювач не дозволяє надіслати запит до сервісу і повертає помилку, що сповіщає користувачів про перевищення обмеження. Таким чином, Binance API не блокуватиме IP-адреси користувачів, що робить дану бібліотеку більш надійною.

3.3. Особливості реалізації модулю валют

Наступний модуль - це модуль, який містить усю логіку роботи з отриманням та обробкою інформації про валюти.

Як уже було зазначено раніше, як джерело інформації використовується публічне API - exchangerate.host API. Так само, як і модуль з криптовалютами, даний модуль надає API для отримання історичних даних про валюти. Для цього так само використовується бібліотека Retrofit в інтеграції з бібліотеками RxJava та Gson.

API для валют, що надається бібліотекою, складається з трьох шарів:

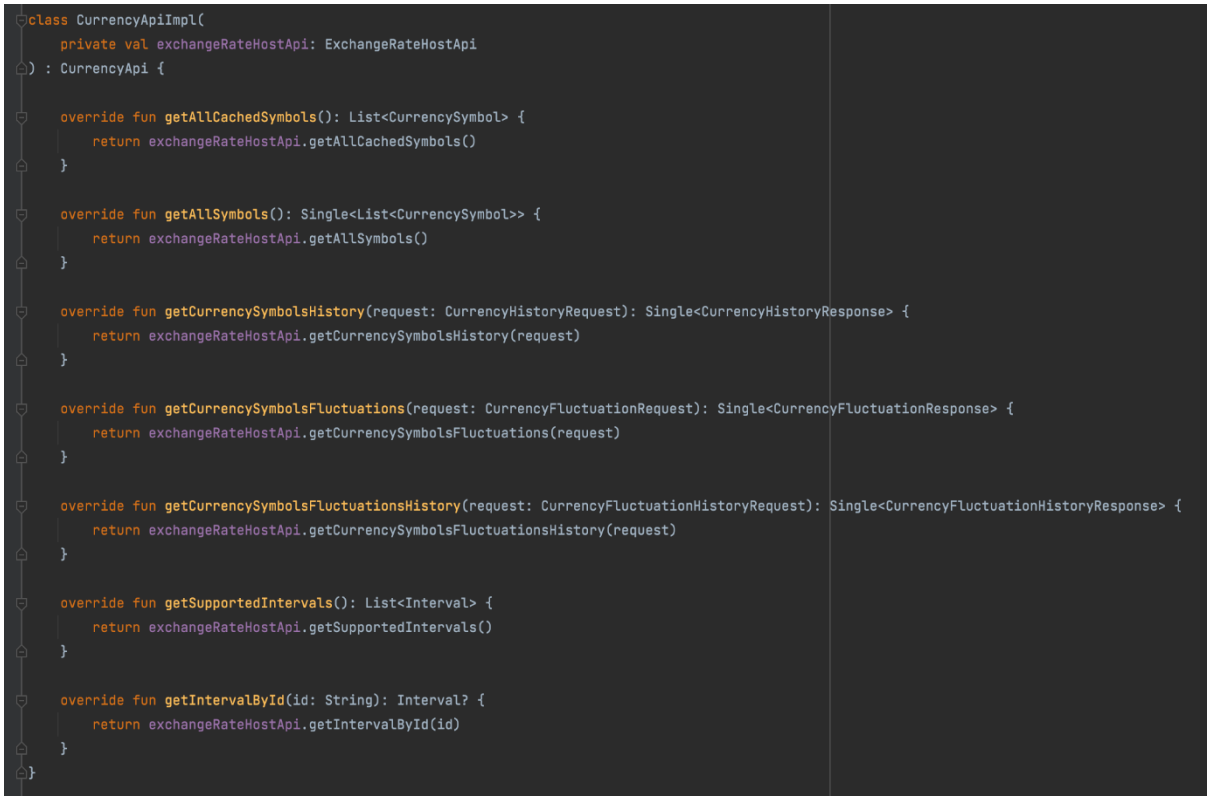
1. CurrencyApiImpl - клас, що реалізує інтерфейс, який визначає протокол взаємодії з API для валют, - CurrencyApi.
2. ExchangeRateHostApi - основний клас, який відповідає за реалізацію взаємодії з публічним API.
3. CurrencyRetrofitService - інтерфейс, який визначає запити до кінцевих точок exchangerate.host API, для реазілації сервісу бібліотекою Retrofit.

Принцип роботи класу ExchangeRateHostApi дуже схожий з принципом роботи класу BinanceApi в модулі криптовалют. ExchangeRateHostApi також під час ініціалізації надсилає запит на отримання усіх підтримуваних символів, після чого кешує їх для можливості користувачів отримувати їх синхронно. Також, як і BinanceApi, даний клас визначає статичний список усіх підтримуваних інтервалів.

Даний клас надає наступні методи:

- Синхронний метод для отримання усіх закешованих підтримуваних символів

- Синхронний метод для отримання списку підтримуваних інтервалів
- Асинхронний метод для отримання усіх підтримуваних символів
- Асинхронний метод для отримання історій зміни курсів валют протягом певного періоду часу за наданим інтервалом
- Асинхронний метод для отримання історії коливань цін валют протягом певного періоду часу з певним інтервалом
- Асинхронний метод для отримання значення коливання цін валют за певний період часу.



```
class CurrencyApiImpl(
    private val exchangeRateHostApi: ExchangeRateHostApi
) : CurrencyApi {

    override fun getAllCachedSymbols(): List<CurrencySymbol> {
        return exchangeRateHostApi.getAllCachedSymbols()
    }

    override fun getAllSymbols(): Single<List<CurrencySymbol>> {
        return exchangeRateHostApi.getAllSymbols()
    }

    override fun getCurrencySymbolsHistory(request: CurrencyHistoryRequest): Single<CurrencyHistoryResponse> {
        return exchangeRateHostApi.getCurrencySymbolsHistory(request)
    }

    override fun getCurrencySymbolsFluctuations(request: CurrencyFluctuationRequest): Single<CurrencyFluctuationResponse> {
        return exchangeRateHostApi.getCurrencySymbolsFluctuations(request)
    }

    override fun getCurrencySymbolsFluctuationsHistory(request: CurrencyFluctuationHistoryRequest): Single<CurrencyFluctuationHistoryResponse> {
        return exchangeRateHostApi.getCurrencySymbolsFluctuationsHistory(request)
    }

    override fun getSupportedIntervals(): List<Interval> {
        return exchangeRateHostApi.getSupportedIntervals()
    }

    override fun getIntervalById(id: String): Interval? {
        return exchangeRateHostApi.getIntervalById(id)
    }
}
```

Рис. 7 Клас CurrencyApiImpl

Exchangerate.host API не надає готової інформації про історію змін статистики курсу валюти зі змінним інтервалом, а тільки з єдиним інтервалом в один день. Тому, мінімальний допустимий інтервал, який підтримується API, що надає розроблена бібліотека, є один день, а максимальний - один місяць. Також через таке обмеження сервісу, клас ExchangeRateHostApi відповідальний за обробку результатів запитів на

історію змін статистики про курс валюти таким чином, щоб вона відповідала вказаному в запиті інтервалу. Тобто, денна історія фільтрується таким чином, щоб повернути результат з більшим інтервалом.

Також, Exchangerate.host API не надає готової історії про коливання курсу, а тільки значення коливання між двома датами. Тому клас ExchangeRateHostApi містить логіку для отримання даних такого типу. Це реалізовано за допомогою отримання історії змін курсів валют та трансформацію цієї історії в історію коливань. А саме, значення коливання для одного дня із історії визначається різницею цін між двома днями з певним інтервалом.

Ще одне обмеження даного сервісу, це те, що він надає можливість отримання історії зміни курсів валют з найбільшим проміжком часу в 366 днів. Оскільки необхідно мати можливість отримувати історію з більшим періодом часу, клас ExchangeRateHostApi також вирішує дану проблему. Якщо проміжок часу, для якого треба отримати історію, надто великий, він розбивається на максимально допустимі проміжки, і до сервісу одночасно надсилаються декілька паралельних запитів, які в результаті об’єднуються, та формується єдиний результат. Це реалізовано за допомогою методу zip(…) класу Single, з бібліотеки RxJava, що об’єднує результати виконання декількох запитів, що повертають тип Single.

3.4. Особливості реалізації головного модулю бібліотеки

Останній модуль - оце основний модуль бібліотеки, який об’єднує в собі увесь функціонал: надає можливість доступатись до API та визначає графіки валют та криптовалют. До цього модулю підключені усі інші модулі бібліотеки: “core” та модулі для валют та криптовалют.

“Core” модуль під’єднаний до основного модулю, як “api” залежність для того, щоб дати змогу користувачам створювати та доступатись до основних сутностей бібліотеки.

Основним класом даного модулю є клас CryptoGraphsLib. За допомогою нього, користувачі бібліотеки можуть її ініціалізувати та отримати основні API для валют та криптовалют: CurrencyCoreApi та CryptocurrencyCoreApi. В свою чергу, клас CurrencyCoreApi надає користувачам API для роботи з історичними даними валют, а CryptocurrencyCoreApi - API для роботи з історичними даними та даними в реальному часі для криптовалют.

Для того, щоб ініціалізувати бібліотеку, необхідно викликати метод initialize(…), в який передати об’єкт класу Application, який визначає глобальний стан Android-застосунку. Якщо коректно не ініціалізувати бібліотеку і спробувати її використати, наприклад, добавити на екран якийсь графік, або використати одне з API, буде створена виняткова ситуація, яка сповіщатиме користувача, що бібліотека не ініціалізована.

Бібліотеку необхідно ініціалізувати об’єктом класу Application, оскільки він використовується для того, щоб отримати доступ до такого важливого системного сервісу як ConnectivityManager (менеджер інтернет з’єднання). Даний сервіс використовується для того, щоб отримувати сповіщення про зміни стану мережі. Оскільки розроблена бібліотека використовує віддалені джерела інформації, а саме два публічних API, про

які було розказано раніше, важливо мати змогу контролювати зміни стану підключення до мережі та коректно на них реагувати.

Для цих цілей було створено окремий клас - NetworkManager. При ініціалізації даного класу, він отримує менеджер інтернет з'єднання (за допомогою об'єкта класу Application), створює інтернет запит для відслідковування змін стану підключення до мережі Wi-Fi та мобільного інтернету, та реєструє мережевий зворотний виклик (callback) для реагування на зміни стану мережі. Також даний клас має 6 корисних методів для добавлення та видалення слухачів змін стану мережі. Це стани доступності та недоступності мережі, а також стан, при якому найближчим часом втратиться з'єднання з мережею. NetworkManager використовується у застосунку для того, щоб максимально швидко реагувати на зміни стану інтернет з'єднання та за потреби виконувати певні операції, наприклад надіслати запит до публічного API, як тільки відновиться з'єднання.

```
object CryptoGraphsLib {
    private var application: Application? = null
    private var networkManager: NetworkManager? = null
    private var cryptocurrencyCoreApi: CryptocurrencyCoreApi? = null
    private var currencyCoreApi: CurrencyCoreApi? = null

    fun getCryptocurrencyCoreApi() : CryptocurrencyCoreApi {
        if(cryptocurrencyCoreApi == null) throw IllegalStateException("Library not initialized!")
        return cryptocurrencyCoreApi!!
    }

    fun getCurrencyCoreApi() : CurrencyCoreApi {
        if(currencyCoreApi == null) throw IllegalStateException("Library not initialized!")
        return currencyCoreApi!!
    }

    internal fun getNetworkManager() : NetworkManager {
        if(networkManager == null) throw IllegalStateException("Library not initialized!")
        return networkManager!!
    }

    fun initialize(application: Application) {
        this.application = application
        val networkManager = NetworkManagerImpl(application)
        this.networkManager = networkManager
        this.cryptocurrencyCoreApi = CryptocurrencyCoreApiImpl(networkManager)
        this.currencyCoreApi = CurrencyCoreApiImpl(networkManager)
    }
}
```

Рис. 8 Клас CryptoGraphsLib

Основним завданням практичної частини кваліфікаційної роботи було розроблення такої Android-бібліотеки, яка надаватиме графіки зміни курсів валют та криптовалют. Бібліотека надає користувачам три графіки для криптовалют та валют.

Бібліотека надає 3 варіанти графіків для криптовалют:

- CandlestickCryptoGraphWidget (Рис. 9а)- графік, який відображає інформацію у вигляді так званих “свічок”. Кожна така “свічка” відображає 4 ключові показники за певний період часу: ціну відкриття і закриття періоду та мінімальну і максимальну ціну криптовалюти за цей період часу.
- LineCryptoGraphWidget (Рис. 9б)- графік, що представляє собою послідовність точок, з'єднаних між собою лініями, де кожна точка відповідає ціні криптовалюти в певний час.
- VolumeCryptoGraphWidget (Рис. 9в)- графік-гістограма, яка відображає обсяг торгів криптовалютою за певний період часу, демонструючи динаміку ринку.

А також 2 варіанти графіків для валют:

- HistoryCurrencyGraphWidget (Рис. 9г)- лінійний графік, в якому кожна точка відповідає ціні валюти в певний час.
- FluctuationCurrencyGraphWidget (Рис. 9 г,д)- ще один лінійний графік, який відображає коливання курсу валюти протягом певного періоду часу. За допомогою цього графіку можна відобразити коливання курсу у двох виглядах:
    - Коливання ціни (Рис. 9г)
    - Коливання ціни у відсотках (Рис. 9д)

Рис. 9 (а-д) Типи графіків валют та криптовалют

Усі описані вище графіки мають певні спільні риси та функціонал:

- Фон
- Вісь значень
- Вісь часу
- Вказівник поточного значення

- Інтерактивність графіку:
    - Гортання
    - Приближення/віддалення
    - Повернення до початку графіку
- Логіка оновлення графіку (додавання нових періодів чи точок графіку)
- Логіка визначення моменту підвантаження більш старої частини графіку
- Параметри кастомізації

Особисті риси графіків:

- Логіка малювання основної частини графіку (оскільки залежить від типу даних та інформації, яку він в собі несе)
- Логіка визначення мінімального та максимального значення, для нормалізації висоти графіку

Для того, щоб не дублювати одну й ту саму логіку для усіх графіків, було прийнято рішення створити спільний базовий клас - GraphView, який буде її визначати.

GraphView - це абстрактний узагальнений клас, який наслідується від класу View, базового класу усіх елементів інтерфейсу користувача в Android. Типом цього “generic” класу може бути тип, який реалізує інтерфейс SymbolInTime, що визначає дату ціни чи іншої статистики символу, та інтервал, для якого буде побудований графік. Його реалізують усі класи, які є будівельними елементами різних графіків:

- CryptoCandlestickData для свічкового графіку криптовалют
- CryptoLineData для лінійного графіку цін криптовалют
- CryptoVolumeData для графіку-гістограми об’ємів торгів криптовалютою

- CurrencyPriceData для графіку зміни цін валюти
- CurrencyFluctuationData для графіку коливань ціни валюти

Для того, щоб визначити, як саме буде “малюватись” кастомний ui-елемент (View), а тобто клас GraphView, необхідно перевизначити відповідний метод класу View - onDraw(…). Налаштування зовнішнього вигляду графіку відбувається саме в цьому методі. Інтерактивна його частина визначається за допомогою перевизначення методу onTouchEvent(…), та опрацюванням різних видів подій: натискання на екран (Action Down), рухання одним або декількома пальцями по екрану (Action Move) та відпускання екрану (Action Up).

Стосовно оновлення графіку, це може відбутись трьома способами, за допомогою узагальненого “sealed” класу GraphUpdate:

- Очищення графіку (EmptyGraph) - усі елементи поточного списку видаляються та графік стає пустим.
- Добавлення списку елементів (HistoryUpdate) - в цьому випадку вони добавляються на початок списку, оскільки це вважається новою частиної підвантаженої історії курсу валюти чи криптовалюти.
- Добавлення одного елементу (LastUpdate) - оновлення останнього елементу списку (якщо елемент визначає період часу, і при цьому не завершений) або добавлення нового елементу в кінець списку (якщо попередній період було завершено, або елемент визначає єдину ціну в одну точку часу, як, наприклад, в лінійному графіку курсу цін валют чи криптовалют).

Як було сказано раніше, розроблена бібліотека надає 5 віджетів-графіків для валют та криптовалют, проте вони використовують три класи, для відображення самого графіку: CandlestickCryptoGraphView,

LineGraphView<T>, VolumeCryptoGraphView. Оскільки лінійних графіків є декілька в бібліотеці, а саме для відображення змін курсу криптовалют та коливання курсу валют (у форматі цін та відсотків), було прийнято рішення зробити один узагальнений клас LineGraphView<T>, який зможе працювати з різними типами даних, які реалізують інтерфейс SingleValueSymbol. Цей інтерфейс визначає одне поле, яке надає одне значення, що буде використовуватись для побудови лінійного графіку. Усі три класи для відображення різних типів графіків (CandlestickCryptoGraphView, LineGraphView<T>, VolumeCryptoGraphView) наслідують базовий клас GraphView, а також реалізують три його абстрактних методи:

- onDrawGraph(…) - для визначення специфічної логіки малювання самого графіку. Даний метод викликається в перевизначеному методі onDraw(…) базового класу GraphView.
- getMinMaxValues(List<T>) - для отримання мінімального та максимального значень із заданого списку об’єктів специфічного типу класу, оскільки для кожного типу даних, що відображає кожен графік, необхідна окрема логіка визначення цих значень.
- getValue(T) - для отримання значення об’єкту певного типу даних, визначеного для кожного класу графіку.

Усі графіки, які надає бібліотека, мають також меню, в якому можливо вибрати, для яких валют чи криптовалют та з яким інтервалом необхідно відобразити графік. Дане меню має три позиції:

- Основний символ - поле для вибору основного символу валюти чи криптовалюти, статистика для якого буде відображена на графіку.
- Ціновий символ - поле для вибору символу валюти чи криптовалюти, яким вимірюватиметься вартість базового символу.

- Інтервал - поле для вибору проміжку часу, для якого буде побудовано графік.

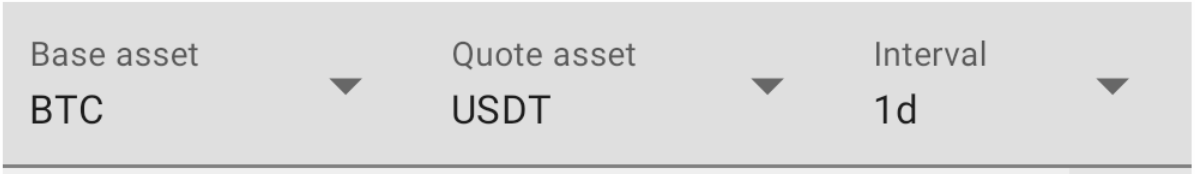


Рис. 10 Меню вибору символів та інтервалу графіку

Усі вищеописані поля повинні бути заповненими, щоб віджет зміг відобразити відповідний графік. Тобто, якщо одне з полей залишити незаповненим - графік не буде намальований. Також, якщо змінити значення одного з полей на інше, графік буде змінено для відповідної пари символів та вибраного інтервалу. Якщо ж змінити значення полей, коли з’єднання з мережею немає, графік відобразиться як тільки інтернет з’явиться.

Кожен з графіків також можна кастомізувати, змінювати їх зовнішній вигляд. Деякі налаштування притаманні усім графікам, в той час як інші більш спеціалізовані і залежать від конкретного типу графіку. Такі характеристики графіків можна задавати програмно, а також за допомогою параметрів графіку в xml.

Можливості кастомізації графіків:

- Можливість змінювати видимість різних елементів графіку:
  - Верхня панель управління символами та інтервалом
  - Вертикальна вісь значень графіку
  - Горизонтальна вісь часу графіку
  - Плаваючий вказівник поточного значення графіку
  - Тіло лінійного графіку (можливо тільки для лінійних графіків: LineCryptoGraphWidget, HistoryCurrencyGraphWidget та FluctuationCurrencyGraphWidget)

- Можливість змінювати спосіб відображення осі значень графіку:
    - Можливість зробити її прозорою (графік буде відображатись поверх осі значень)
    - Можливість зробити її непрозорою (графік відображатиметься під віссю значень)
- Можливість змінювати кольори графіку:
    - Колір вказівника поточного значення графіку
    - Колір елементів графіку (або свічкових даних, або стовбців гістограми), залежно від того, чи протягом періоду ціна криптовалюти збільшилась чи зменшилась (можливо тільки для графіків CandlestickCryptoGraphWidget та VolumeCryptoGraphWidget відповідно)
    - Колір лінії та тіла лінійного графіку (можливо тільки для лінійних графіків: LineCryptoGraphWidget, HistoryCurrencyGraphWidget та FluctuationCurrencyGraphWidget)
- Можливість змінювати кількість позначок значень графіку на вертикальній осі значень (мінімальна к-сть – 2 позначки)

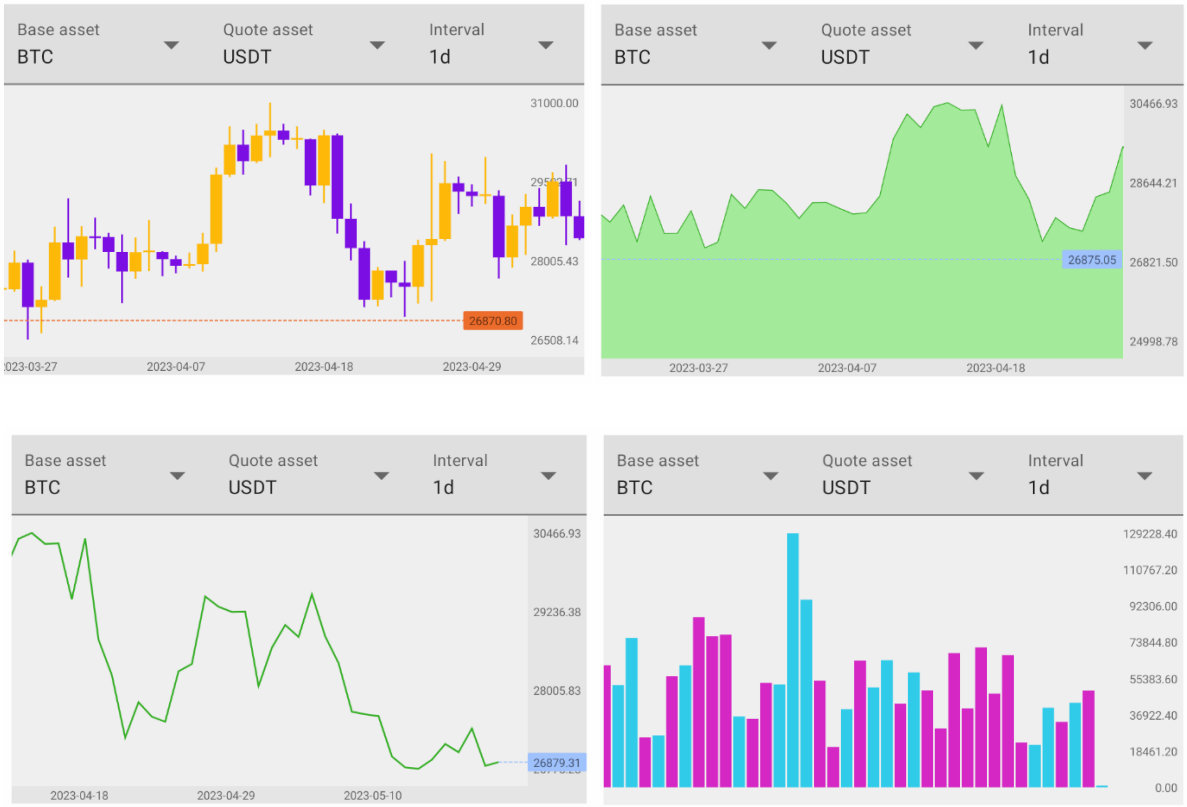


Рис. 11 Можливості кастомізації графіків

Усе, що було описано вище про графіки, стосувалось UI-шару, а саме логіка їх відображення як елементів інтерфейсу користувача. Проте, також був створений шар, який відповідає за бізнес логіку роботи відповідних віджетів. Даний шар також має певну ієрархію, яка складається з 4 шарів:

1. Основним класом, який є базовим для усіх інших класів цього шару - є абстрактний клас WidgetModel. Від відповідає за ініціалізацію та очистку ресурсів, які виділені для кожної віджет-моделі. Даний клас надає абстрактні методи для визначення специфічної логіки ініціалізації та очистки (inSetup() та inCleanup())), які викликаються в основний методах ініціалізації та очистки WidgetModel'і (setup() та cleanup()).
2. CurrencyWidgetModel та CryptoWidgetModel- класи, які наслідують клас WidgetModel, є абстрактними класами, які

відповідають за реалізацію стандартної бізнес логіки усіх віджет-моделей, пов’язаних з валютами та криптовалютами відповідно. Дані класи при ініціалізації роблять запит на отримання усіх символів валют чи криптовалют відповідно, та зберігають їх у списку, для можливості роботи з ними класів-нащадків. Також класи CryptoWidgetModel та CurrencyWidgetModel надають NetworkManager та API для роботи з валютами та криптовалютами відповідно для своїх нащадків, а також визначають абстрактний метод onSymbolsInit(), який викликається, коли символи були успішно ініціалізовані.

3. Наступний шар - це шар класів, які відповідають за бізнес логіку усіх віджет-моделей, пов’язаних з графіками для валют та криптовалют: CurrencyGraphWidgetModel та CryptoGraphWidgetModel відповідно. Дані класи зберігають інформацію та відповідають за роботу з даними про вибрані на даний момент пару символів та інтервал, а також про списки доступних символів та інтервалів для вибору в полях відповідного меню кожного графіку. Класи CurrencyGraphWidgetModel та CryptoGraphWidgetModel є узагальненими, а отже можуть працювати з різними типами, які реалізують інтерфейси CurrencyData та CryptoData відповідно. Ці класи є базовими для уже вищезгаданих класів, які є будівельними одиницями для різних графіків (CryptoCandlestickData, CryptoLineData, CryptoVolumeData, CurrencyPriceData та CurrencyFluctuationData). Також класи даного шару мають поле, яке визначає процесор даних для оновлень графіків даними певного типу: DataProcessor<GraphUpdate<T>>. Такий процесор даних є обгорткою для об’єкта класу бібліотеки RxJava -

BehaviorProcessor, для можливості визначення стандартного значення процесора та його перетворення в об’єкт класу Flowable. Крім цього, класи CurrencyGraphWidgetModel та CryptoGraphWidgetModel визначають абстрактний метод selectSymbolAndInterval() для специфічної реалізації логіки вибору нового символу та інтервалу, а також абстрактний метод updateHistory(Long) для можливості підвантаження старіших даних, під час гортання графіку.

4. Останній шар - це класи, які відповідають за специфічну бізнес логіку для кожного графіку. Це класи HistoryCurrencyGraphWidgetModel та FluctuationCurrencyGraphWidgetModel, які наслідують клас CurrencyGraphWidgetModel, а також класи CandlestickCryptoGraphWidgetModel, LineCryptoGraphWidgetModel та VolumeCryptoGraphWidgetModel, які наслідують клас CryptoGraphWidgetModel. Кожна така віджет-модель реалізує два основних методи selectSymbolAndInterval() та updateHistory(Long). В методі вибору символу та інтервалу відбувається запит на отримання історії зміни необхідної статистики валюти чи криптовалюти, після чого, у випадку віджет-моделі для графіків криптовалют, віджет-модель підписується на потік відповідного типу даних. В методі оновлення історії, відбувається запит на отримання старішої історії про валюту чи криптовалюту, яка закічується в заданий час, що є часом останньої статистики, до якої користувач прогортав графік.

Усі запити, які відправляються віджет-моделями, виконуються з використанням менеджера NetworkManager, а отже якщо на момент

виклику не було інтернет з’єднання, його буде виконано пізніше, коли він з’явиться.

Кожен з віджетів-графіків має свою специфічну віджет-модель. Вона ініціалізується та очищається у двох зворотних викликах класу View onAttachedToWindow() та onDetachedFromWindow() відповідно, для ефективного виділення та очищення ресурсів. Віджет-моделі використовуються віджетами для отримання оновлень для графіку, а також для отримання інших даних, таких як, наприклад, списків символів та інтервалів для відповідних полей верхньої панелі управління, а також для виконання операцій вибору символів та інтервалів.

Висновок

У результаті виконання кваліфікаційної роботи були виконані усі поставлені задачі. В ході роботи було проаналізовано як саме можуть представлятись різні статистичні дані, пов’язані зі зміною курсу валют та криптовалют, у вигляді графіків. Було розглянуто доступні публічні API для їх використання як джерел даних інформації про валюти та криптовалюти та обрано за визначеними критеріями ті, які найбільше підходять для поставленої задачі: Binance API та Exchangerate.host API.

Також було розглянуто інструменти, такі як мова програмування Kotlin, бібліотеки Retrofit та RxJava, а також технологія WebSocket, які були використані для створення Android-бібліотеки, а саме для роботи з обраними публічними API для отримання даних. Було описано їх особливості, та як саме вони були застосовані під час розробки.

Унаслідок виконання практичної частини даної роботи, було розроблено Android-бібліотеку для візуалізації статистичних даних про зміни курсів валют та криптовалют, а також API, в рамках даної бібліотеки, для роботи з цими статистичними даними.

Список використаних джерел

1. Exchangerate.host API – [Електронний ресурс]. – Доступно з: https://exchangerate.host/
2. Binance API – [Електронний ресурс]. – Доступно з: https://www.binance.com/en/binance-api
3. Binance – [Електронний ресурс]. – Доступно з: https://www.binance.com/
4. Gradle – [Електронний ресурс]. – Доступно з: https://gradle.org/
5. Android app modularization – [Електронний ресурс]. – Доступно з: https://developer.android.com/topic/modularization
6. Kotlin for Android – [Електронний ресурс]. – Доступно з: https://kotlinlang.org/docs/android-overview.html
7. Android’s Kotlin-first approach – [Електронний ресурс]. – Доступно з: https://developer.android.com/kotlin/first
8. JetBrains – [Електронний ресурс]. – Доступно з: https://www.jetbrains.com/
9. Google I/O – [Електронний ресурс]. – Доступно з: https://en.wikipedia.org/wiki/Google_I/O
10. Retrofit – [Електронний ресурс]. – Доступно з: https://square.github.io/retrofit/
11. Square – [Електронний ресурс]. – Доступно з: https://square.github.io/
12. Gson – [Електронний ресурс]. – Доступно з: https://github.com/google/gson
13. Moshi – [Електронний ресурс]. – Доступно з: https://github.com/square/moshi

14. RxJava – [Електронний ресурс]. – Доступно з: https://github.com/ReactiveX/RxJava
15. RxAndroid – [Електронний ресурс]. – Доступно з: https://github.com/ReactiveX/RxAndroid
16. View binding – [Електронний ресурс]. – Доступно з: https://developer.android.com/topic/libraries/view-binding
17. Backpressure handling – [Електронний ресурс]. – Доступно з: https://github.com/ReactiveX/RxJava/wiki/Backpressure
18. WebSocket – [Електронний ресурс]. – Доступно з: https://en.wikipedia.org/wiki/WebSocket
19. OkHttp – [Електронний ресурс]. – Доступно з: https://square.github.io/okhttp/