ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ АВТОМАТИЗОВАНИХ СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розробка мікросервісу для аналізу курсів криптовалют і відображення рахунків клієнтів у обмінних платформах»

на здобуття освітнього ступеня бакалавра
зі спеціальності 126 Інформаційні системи та технології
(код, найменування спеціальності)
освітньо-професійної програми Інформаційні системи та технології
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Владислав БЄЛЬСЬКИЙ
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав: здобувач вищої освіти гр. ІСД- 42
Владислав Бєльський
Ім'я, ПРІЗВИЩЕ

Керівник: Іван ШАХМАТОВ
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Рецензент: _____
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Інженерії програмного забезпечення автоматизованих систем
Ступінь вищої освіти бакалавр
Спеціальність Інформаційні системи та технології
Освітньо-професійна програма Інформаційні системи та технології

ЗАТВЕРДЖУЮ
Завідувач кафедрою ІПЗАС
_____Каміла СТОРЧАК
«____» _____2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ
Бєльського Владислав Олеговича

(прізвище, ім'я, по батькові здобувача)

1.Тема кваліфікаційної роботи: Розробка мікросервісу для аналізу курсів криптовалют і відображення рахунків клієнтів у обмінних платформах

керівник кваліфікаційної роботи Іван ШАХМАТОВ
(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» лютого 2024 р. № 36

2. Строк подання кваліфікаційної роботи «31» травня 2024 р.

3. Вихідні дані до кваліфікаційної роботи:
1. Якісний аналіз предметної області.
2. Принцип функціонування «мікросервісної» архітектури.
3. Науково-технічна література.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Аналіз предметної області
2. Обґрунтуання архітектури мікросервіса
3. Розробка мікросервіса
4. Висновки

5. Ілюстративний матеріал: *презентація*
6. Дата видачі завдання: «27» лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	27.02-05.03.2024	
2	Аналіз існуючих прототипів	06.03-11.03.2024	
3	Дослідження програмних засобів	12.03-27.03.2024	
4	Моделювання об’єкту проектування	28.03-10.04.2024	
5	Розробка функціоналу мікросервісу	11.04-15.05.2024	
7	Вступ, висновки, реферат	16.05-22.05.2024	
8	Розробка презентації застосунку	23.05-24.05.2024	

Здобувач(ка) вищої освіти _____ Владислав Бєльський

(підпис) *(Ім’я, ПРІЗВИЩЕ)*

Керівник

кваліфікаційної роботи _____ Іван ШАХМАТОВ

(підпис) *(Ім’я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавр: 55 стор., 8 табл., 39 рис., 33 джерела.

Мета роботи – аналіз особливостей розробки та практичної реалізації мікросервісу аналізу криптовалют.

Об’єкт дослідження – проектування автоматизованого мікросервісу аналізу криптовалют.

Предмет дослідження – програмне забезпечення для аналізу криптовалют.

Короткий зміст роботи – У роботі отримано зручний інструмент для аналізу фіатних та криптовалют за різні періоди часу, задля подальшого прогнозування їх курсів і отримання важливих повідомлень про їх зміни через Telegram бота.

СОЦІАЛЬНА МЕРЕЖА, МЕСЕНДЖЕР, РОЗРОБКА, ЧАТ-БОТ, ПОМІЧНИК, JAVA, ПРОЕКТУВАННЯ, АНАЛІЗ, ПОВІДОМЛЕННЯ, ПРОГНОЗУВАННЯ, МЕТОД, БАЗА ДАНИХ, КРИПТОВАЛЮТИ, ФІНАНСИ, МІКРОСРВІС, АРХІТЕКТУРА .

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

Blockchain	Це умовна база даних яка дозволяє відкрито ділитися даними серед користувачами
Мікросервіси	Архітектурний підхід у розробці програмного забезпечення
HTTP	Протокол передачі тексу у мережі інтернет
Майнинг	Діяльність по створенню нових блоків у системі Blockchain та отримання за це прибутку
JSON	Текстовий формат для зберігання структурованих даних
XML	Розширювана мова розмітки
Clean Code	Принципи розробки програмного коду
Фреймворк	Програмне середовище яке визначає архітектуру інформаційної системи

ЗМІСТ

ВСТУП

У час світової нестабільності та турбулентності великого попиту набирає криптовалюта та технологія Blockchain загалом. Власники криптовалютних активів обирають анонімність, безпеку та прозорість. Саме ці характеристики відрізняють криптовалюту від фіатних валют.

За допомогою цієї технології люди переказують кошти одне одному без участі третіх осіб, зберігають та примножують свої активи; тому неймовірно важливо бути в курсі актуальних станів валют та слідкувати за своїми рахунками.

Сьогодні Blockchain претендує зайняти велику частину у фінансовій системі та можливо замінить її або візьме домінуючу позицію у майбутньому. Навіть сьогодні існують країни, які зробили криптовалюту своєю основною валютою, і з цього можна зробити висновок, що люди, які керують країнами, готові ризикувати та інвестувати гроші у цю технологію, тому що бачать за нею майбутнє.

Як не крути, важливо цікавитися новинками у фінансовій сфері. Зараз технології з’являються швидше, ніж про них встигають дізнатися, але важливо відокремлювати дійсно цінні екземпляри, від тих, що даремно витрачають час та ресурси.

Коли йде мова про технології, можна згадати про різні типи архітектур інформаційних систем, завдяки яким будують надійні та ефективні додатки. Сьогодні стандартом стала мікросервісна архітектура; її використовують найчастіше, тому що вона надає неймовірну гнучкість, хоча, також може за собою тягнути фінансові втрати на підтримку її розгалуженої системи. Загалом мікросервісний підхід дає змогу розділити велику систему на окремі частини, які можуть працювати незалежно одне від одного. Ці частини можуть бути розроблені на різних мовах програмування, використовувати різні бібліотеки та загалом працювати як самостійні частини, не знаючи про існування інших елементів системи.

Після аналізу фінансових ринків та актуальності криптовалюти, було запропоновано розробити новий інструмент, а саме мікросервіс, який допоможе відслідковувати і аналізувати курс фіатних так криптовалют у реальному часі. На даний момент фіатні валюти актуальні, тому важливо також бути у курсі їх змін. Також у нас повинна бути можливість швидкого оповіщення про ситуацію у світі, щоб корегувати свої активи та бути у фінансовій безпеці.

У майбутньому цей мікросервіс може бути розбудований новими модулями та інтеграціями у велику систему для аналітики діджитал валют, також з швидким розвитком технології Blockchain, буде простіше знайти інвестиції та зацікавлених користувачів.

Було запропоновано план для розробки програмного рішення:

– проаналізувати обрану предметну область;

– порівняти існуючі аналоги аналітичних сервісів;

– обрати технології та середовище для розробки;

– розробити та протестувати мікросервіс.

Також планується розміщення мікросервіса на віддаленому сервері, щоб дати можливість іншим розробникам використовувати його для інтеграції в їх системи та інтерфейси, що дасть змогу допомогти якомога більшій кількості людей бути фінансово обізнаними.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Мікросервіси як архітектурне рішення

Мікросервіси стали новою стадією у розвитку архітектури інформаційних систем; вони прийшли на заміну монолітній архітектурі.

Загалом, монолітна архітектура - це спосіб побудувати інформаційну систему, не розбиваючи її кодову базу на дрібні частини. Тобто, в результаті виходить один блок, у якому всі елементи системи залежні одне від одного. У такому підході є свої плюси, і досить часто його недооцінюють, надаючи перевагу мікросервісній архітектурі, яка не завжди є доцільною.

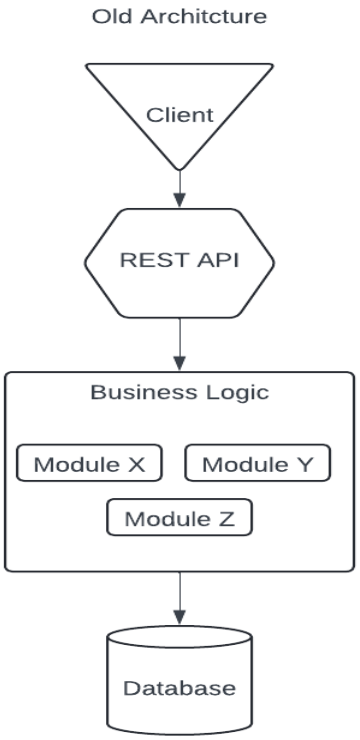


Рис.1.1. Приклад монолітної архітектури

На рисунку 1.1 можна побачити приклад звичайної монолітної архітектури. З основних плюсів можемо виділити те, що витрати на її підтримку набагато менші, ніж у випадку мікросервісної архітектури. Також, якщо проект не сильно великий, то у такій системі набагато простіше розібратися, ніж у десятках мікросервісів, які взаємодіють між собою. Загалом,

для невеликих додатків це чудове рішення, з яким буде дешево та легко працювати.

Основна проблема моноліта у тому, що з розвитком системи така архітектура накладає багато обмежень. Наприклад, така система стає сильно залежна від мови програмування, яка використовується у проекті, також з'являється проблема з бібліотеками, які використовуються. Коли проєкт продовжує збільшуватися, час збірки та тестування, а також загалом складність підтримки та розробки пропорціонально збільшується.

Зазвичай у таких випадках буде доцільно розділити цю велику неповоротку систему на менші логічні одиниці, де всі модулі будуть працювати незалежно та не впливати одне на одного. Їх можна буде розробляти паралельно та впроваджувати зміни, які не будуть впливати на одне одного. Це і є мікросервісною архітектурою.

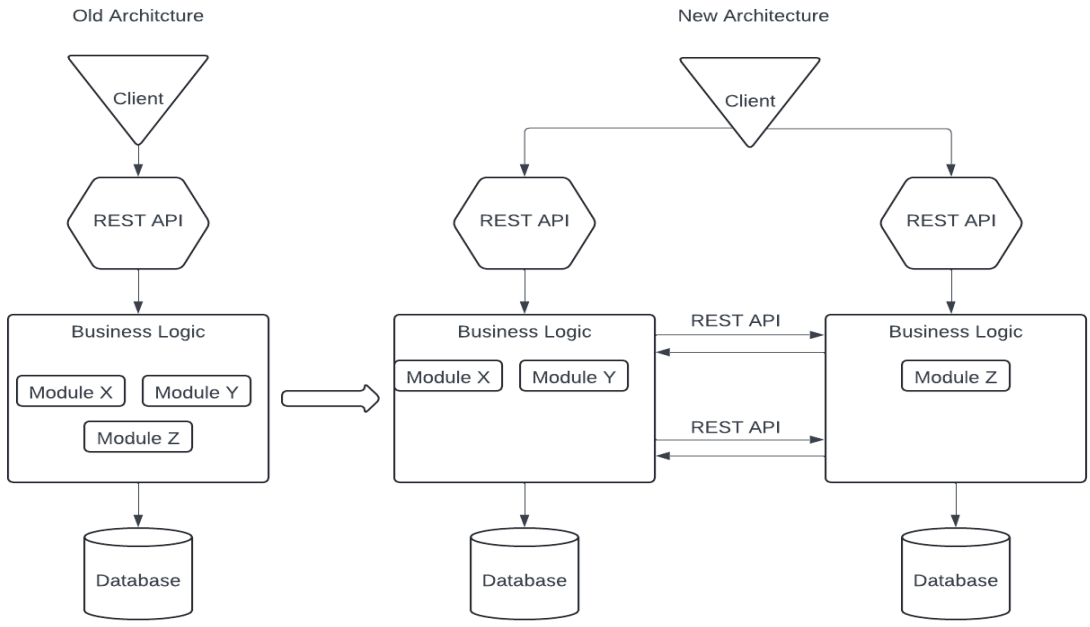


Рис.1.2. Приклад розділення монолітної архітектури на мікросервіси

На фото 1.2 можна побачити, як було розділено моноліт на 2 мікросервіси. Загалом, було винесено логіку з модуля Z в новий сервіс та додано до нього власну базу даних. Це дало можливість гнучкіше розвивати модулі системи

паралельно та збільшило відмовостійкість. Тепер можна дати гарантію, якщо один з сервісів припинить роботу, інший зможе продовжити обробляти запити від користувачів.

Тож з основних плюсів мікросервісної архітектури можна виділити:

1. Збільшує відмовостійкість за рахунок того, що сервіси працюють окремо і не впливають одне на одного.
2. Спрощує розробку та підтримку системи. Тепер, коли потрібно внести зміни до модуля Z, модулі X і Y не будуть задіяні.
3. Надає гнучкість і свободу у виборі нових технологій та мов програмування для різних сервісів.
4. Прибирає технічний борг та дає можливість для подальшого маштабування системи без зайвих проблем.

Щодо мінусів, можна зазначити значні затрати для підтримки такої системи та наявність кваліфікованих спеціалістів для її подальшої розробки. У неправильних руках мікросервіси можуть перетворитися на темний ліс, в якому розібратися та підтримувати його буде дуже складно.

1.2. Загальні відомості про фіатні та криптовалюти

Фіатні валюти - це валюти цінність яких підкріплюється не гарантією обміну цієї валюти на золото, а фіатним наказом, тобто це всім відомі долари, євро та інші валюти, які регулюються країною виробником. Вони є частиною валютної системи, яка залишалася без змін протягом довго часу. Така система виявилася зручним для нас способом обмінюватися валютами, зберігати їх та робити покупки, але у цій системі є серйозні недоліки.

Регулятори, які контролюють їх обіг та фінансовий стан власників таких валют, можуть у будь-який момент заблокувати ваш рахунок чи накласти інші обмеження на використання ваших коштів. Також ця система не завжди має

на увазі прозорість, багато валюти проходить в тіні, що не дає можливості прозоро бачити рух коштів у системі.

На допомогу цьому прийшла технологія Blockchain. Якщо коротко говорити, це бухгалтерська книга, у якій зберігаються записи абсолютно всіх переказів всередині системи. Кожний запис у цій книзі називається блоком, а саме цікаве, що ця книга відкрита для будь-якого користувача і всі можуть бачити рух коштів всередині, зберігаючи анонімність учасників угоди. Тому, ймовірність шахрайства зменшується до мінімального рівня.

Сама назва говорить, що Block це блок, а Chain - ланцюг. У нас виходить ланцюг блоків, які не можливо змінити. Перед кожним новим створенням блока відбувається підтвердження його додавання у всіх нодів системи, за що вони отримують прибуток. Завдяки цій схемі з'явився майнінг криптовалют.

У системі Blockchain, для обміну, замість коштів використовується криптовалюта, популярність якої у останні роки швидко зросла. Вона перекриває більшість проблем фіатних валют, але, звісно, має свої мінуси.
Серед них є:

1. Висока волатильність. Курс може різко змінюватися у короткі проміжки часу.
2. Ризик втратити доступ до свого гаманця, який не можливо повернути.
3. Хакери, які можуть отримати доступ до твого гаманця і викрасти твої кошти.

Зараз існує величезна кількість різноманітних криптовалют, і вона з кожним роком збільшується, тому важливо цікавитися цією технологією.

1.3. Огляд існуючих інструментів аналізу валют

Перед початком розробки сервісу аналітики криптовалюти, я провів попереднє дослідження, щоб зрозуміти цілі задля яких користувачі шукають аналітичні сервіси, їхню проблематику, а також, щоб не повторюватися і виявити нові функції які ще не було запроваджено. Варто зазначити, що ця ніша доволі сильно розвинути та має серйозну кількість конкурентів, що постійно стимулює проєкт до розвитку. З іншого боку велика кількість

конкурентів означає, що у цій ніші велика кількість клієнтів, та не потрібно проводити зайві дослідження на рахунок актуальності сервіси серед користувачів.

Було запропоновано декілька сильних конкурентів які досить відрізняються між собою але ціль мають одну – аналітика криптовалютного курсу та заощаджень. Вони використовуються нові технології та мають приємний інтерфейс, що покращує користувацький досвід, для проєкту було виписано дизайнерські рішення задля подальшого запровадження їх у свою інформаційну систему.

1.3.1. **Аналіз функціональних можливостей сервісу “Moralis Money”**

Moralis Money це один з найкращих крипто-моніторинг сервісів, він збирає blockchain інформацію про криптовалюту і показує її у зручному та зрозумілому форматі. Загалом у цьому сервісі можна отримати всю необхідну інформацію про світ криптовалюти та побачити графіки.

Основні функціональні можливості:

1. Відслідковування індивідуального токен контракту: Моніторинг індивідуального токен контракту з повідомленнями, коли щось цікаве трапляється on-chain.
2. Моніторинг ринку термінових монет: Можна налаштувати повідомлення для термінових токенів і нових крипто прєктів базуючись на своїх власних фільтрах.
3. Пошук токенів: Можна обрати серед 15 унікальних пошукових параметрів для пошуку токенів.
4. Захист токенів: Захист токенів запроваджує перевірку безпеки для всіх монет
5. Сторінка для токенів: Завдяки сторінки для токенів можна придбати монети і отримати майбутні прогнози на рахунок цієї валюти. Ця сторінка включає, поточну ціну, історії змін цін, альфа-метрики, деталі безпеки.

6. Відслідковування гаманця: Завдяки одному з найкращих трекерів, можна відслідковувати гаманці в своїх ланцюгах.
7. Підтримка мульти-ланцюгів: Сервіс підтримує багато мереж, серед яких Ethereum, BNB, Chain, Polygon, Optimism, Pulsechain

Переваги використання цього сервісу:

Money надає зручний та інтуїтивно зрозумілий інтерфейс, що дозволяє користувачам ефективно працювати з криптовалютними даними. Підтримка мульти-ланцюгів і широкий спектр інструментів для аналізу робить цей сервіс універсальним і корисним для різних типів користувачів.

1.3.2. Аналіз функціональних можливостей сервісу “TradingView”

TradingView була представлена в 2011 році це соціальна мережа та платформа для створення діаграм, використовується трейдерами та інвесторами. Сервіс підтримує більшість фінансових ринків, включаючи ринок криптовалют. Також цей сервіс допомагає дивитися різноманітні діаграми.

Основні функціональні можливості:

1. Діаграми: Це один з найбільших виробників діаграм, також діаграми підтримують багато ринків, включаючи криптовалютні
2. Технічні індикатори: Сервіс пропонує більше 100 індикаторів і 90 інструментів для прогнозування
3. Тестування трейдингових стратегій: Користувачі можуть імітувати їхні трейдингові стратегії прямо на діаграмах

Переваги використання цього сервісу:

TradingView забезпечує потужний інструментарій для аналізу ринків, надаючи користувачам можливість створювати та налаштовувати діаграми за власними потребами. Соціальна складова платформи дозволяє трейдерам обмінюватися ідеями та стратегіями, що сприяє кращому розумінню ринку.

1.3.3. **Аналіз функціональних можливостей сервісу “Cointracker”**

Цей інструмент сфокусований на відслідковуванні Blockchain гаманців. Це платформа допомагає користувачам відслідковувати їхні активи в мережі

Blockchain. Платформа має більше 1 мільйона користувачів та більше 500 підключених гаманців

Основні функціональні можливості:

1. Автоматичні портфоліо трекери: Сервіс допомагає користувачам відслідковувати їхню ринкову цінність, та дохідність інвестицій в реальному часі.
2. Податковий звіт: Платформа допомагає користувачам відслідковувати податкові квитанції на дохід від криптовалют
3. Мобільний додаток: Користувачі можуть відслідковувати свої гаманці прямо у мобільному додатку.

Переваги використання цього сервісу:

Cointracker пропонує зручний і ефективний спосіб відстеження криптовалютних активів. Автоматичні портфоліо трекери дозволяють користувачам бачити повну картину своїх інвестицій, а податковий звіт значно полегшує процес податкового обліку. Мобільний додаток додає зручності, забезпечуючи доступ до інформації у будь-який момент.

1.3.4. Аналіз функціональних можливостей сервісу “CryptoPanic”

CryptoPanic це легкий для користування агрегатор новин. Це зручний інструмент для відслідковування змін цін, та взагалі тримає користувачів в курсі останніх новин, більше того, користувачі можуть фільтрувати інформацію налаштовуючи повідомлення для моніторингу конкретних монет чи ключових слів.

Основні функціональні можливості:

1. Зручні налаштування оповіщень для користувачів: Користувачі можуть встановити сповіщення для будь яких ключових слів, монет чи фраз
2. Багатоджерельність: Сервіс поєднує комбінацію зручних медіа джерел, також сервіс включає форуми, блоги, і соціальні мережи.

Переваги використання цього сервісу:

CryptoPanic надає користувачам зручний спосіб відстежувати останні новини та зміни на ринку криптовалют. Налаштування сповіщень дозволяють персоналізувати потік інформації, що є дуже корисним для оперативного реагування на події. Багатоджерельність забезпечує широту і різноманітність інформаційного контенту

1.4. Призначення мікросервіса

Умовні користувачі мікросервісу – це люди, які будуть використовувати зручний WEB інтерфейс для отримання інформації про зміни криптовалютних та фіатних курсів та проводити аналітику для майбутніх угод та можливих змін у своїх рахунках. Також цей сервіс можуть використовувати юзери бажаючі бути вчасно оповіщеними про зміни у світі криптовалюти для прогнозування подальшої торгової стратегії. Іншими користувачами цього сервісу будуть схожі сервіси які захочуть реалізувати у себе функціонал який буде надаватися через відкритий API.

Також сервіс буде актуальним користувачам які хочуть познайомитися з світом криптовалюти та спробувати себе у новій сфері.

Сервіс може легко маштабуватися та обробляти великі потоки користувачів через механізми кешування.

Призначення мікросервісу:

1. Відстеження актуальних курсів криптовалют і фіатних валют
2. Відслідковування актуально стану рахунків у обмінних платформах
3. Система оповіщення про зміни у світі криптовалюти
4. Інтеграції зовнішніх сервісі до мікросервісу через відкрите API

Основна мета мікросервісу, це приносити користь досвідченним трейдерам та початківцям.

1.5. Опис вимог до мікросервісу

Мікросервіс повинен працювати швидко, зберігаючи актуальність даних, зберігати історію змін та використовувати механізми кешування. Також для комунікації з сервісом звичайних користувачів повинен бути запроваджений WEB інтерфейс з зрозуміли графіками, системою створення,

авторизації користувачів та перегляду рахунків клієнта, інтерфейс повинен бути не складний та зрозумілий для будь якому користувачу без додаткових інструкцій чи пояснень зі сторони служби технічної підтримки. Також користувачі повинні бути оповіщенні через телеграм бот про зміни курсів.

Сервіс повинен зберігати високу продуктивність, тому що потенційна кількість користувачів може бути доволі високою, також оновлення інформації про курси досить важка операція, потрібно оптимізувати всі процеси.

Для збереження актуальності даних потрібно запровадити механізми асинхронної роботи для періодичного оновлення актуальних курсів валют. Сервіс повинен інтегрувати інші системи для отримання актуальної інформації, також банківські системи для отримання інформації про рахунки.

WEB інтерфейс та телеграм бот повинні бути окремими мікросервісами щоб забезпечити відмовостійкість системи та простоту майбутньої підтримки та розробки. Також необхідно витримати принципи Clean Code для подальшого легко онбордингу нових розробників. Сервіс повинен використовувати реляційну базу даних Postgres, key-value сховище Redis, мову програмування Java там фреймоврки Spring для розробки Backend частини і Jmix для інтерфейсу користувача.

Основні функціональні вимоги:

1. Мікросервіс повинен надавати інформацію про курси криптовалют та фіатних валют для своїх користувачів
2. Web-інтерфейс повинен мати меню у якому будуть, графіки курсів фіатних валют, графіки курсів криптовалют
3. Мікросервіс повинен відправляти повідомлення у телеграм бот про актуальний стан курсів валют
4. Мікросервіси та інтегровані сервіси повинні спілкуватися між собою використовуючи протокол HTTP та архітектурний стиль REST

2 ОБҐРУНТУВАННЯ АРХІТЕКТУРИ МІКРОСЕРВІСА

2.1. Проектування функціональних можливостей мікросервіса

Мікросервіс , що розробляється у дипломному проєкті, має наступні функціональні характеристики:

- Мати точку доступу для отримання інформацію про актуальний стан курсів фіатних валют з 3 різних джерел: ПриватБанк, Монобанк, Open data
- Мати точку доступу для отримання інформації про актуальний стан курсів фіатних валют на конкретну дату
- Мати точку доступу для отримання інформації про курс вибраної криптовалюти за останній час
- Регулярно оновлювати та зберігати інформацію про стан курсів валют
- Мати WEB інтерфейс який в свою чергу повинен мати такий функціонал:
    1. Авторизація нового користувача.
    2. Зручний механізм управління користувачами: створення, видалення, обмеження прав доступу
    3. Меню “Fiat Analytics” Де клієнт зможе обрати з списку фіатну валюту та побачити на графіку як вона змінювалася з 3 різних джерелах. Також клієнт повинен бути у змозі обрати період за який він бажає отримати інформацію про курс валют
    4. Меню “Crypto Analytics” де клієнт зможе обрати пару криптовалют і побачити на графіку “Свічка” як змінювалася ціна іншої відносно до першої
- Бути спроможним підключити Telegram бота щоб отримувати інформацію про актуальний стан курсу валют

Сьогодні Telegram став просунутою платформою у світі криптовалюти та навіть має свою власну монету TON, тому важливо дати змогу користувачам мікросервісу бути оповіщенними про зміни курсу у Telegram боті.

Для того, щоб забезпечити надійну та відмовостійку архітектуру, мною було прийнято рішення розробити 3 окремих мікросервіса з власною зоною відповідальності.

Перший мікросервіс буде місцем де інші користувачі зможуть отримати звіти про зміни крусів фіатних та криптовалют, завдяки цьому рішенню можна відокремити основну бізнес логіку від не потрібних елементів.

Другий мікросервіс буде WEB інтерфейсом у якого буде своя власна база даних, він буде відображати клієнтам графіки. Інформацію він буде брати з першого мікросервісу використовуючи протокол HTTP та архітектурний стиль REST.

Третій мікросервіс, буде з періодичністю відправляти всім користувачам інформацію про зміни курсів, саму інформацію він також буде брати з першого мікросервісу через проткол HTTP та не буде мати власної бази даних.

За для збереження швидкодії, було вирішено, додати до проєкту механізми кешування використовуючи key-value сховище Redis

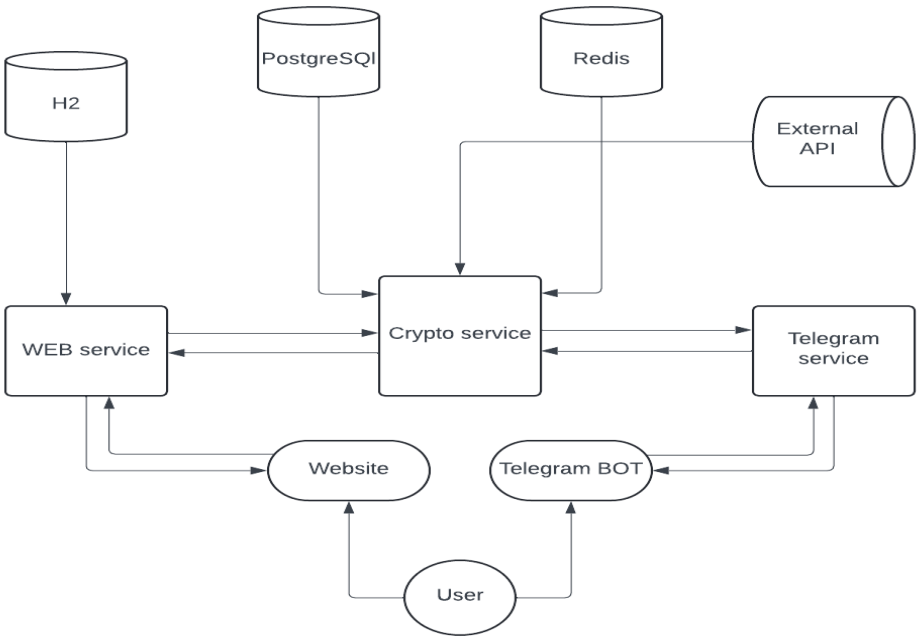


Рис.2.1. Загальна схема роботи системи та елементи з якими взаємодіє користувач

Можна побачити, що серцем системи є Crypto service, він відповідальний за отримання, передачу та збереження основних даних. Потім Web service відповідальний за обробку звернень користувача до Website, та Telegram service який відправляє повідомлення у телеграм бот. У цій схемі користувач взаємодіє виключно з елементами Website та Telegram bot

2.2. Вибір засобів розробки

2.2.1. Мова програмування Java

Під час вибору мови програмування, було звернено увагу на декілька показників. Насамперед мова повинна бути високорівнева, кросплатформова, об’єкто орієнтована, строготипізована та підтримувала багатопоточність. Також важливим фактором була популярність мови програмування та наявність уже готових рішень для вирішення типових задач. Ідеально підходила під ці критерії мова програмування Java.

Java – високорівнева, компільована мова програмування. Перед тим як вихідний код програми написаній на мові програмування java буде виконано, він буде скомпільований у byte code, компайлером javac та переданий на виконання у JVM. JVM (Java Virtual Machine) це технологія завдяки якій Java може виконуватися на різних платформах. Слоганом компанії “Sun Microsystem”, яка розробила мову програмування Java було “Write once, run everywhere”, це значить, що програма колись написана одного разу буде однакова працювати всюди, на різних платформах. Також цікаво, що технологію JVM використовує не тільки Java, а і доволі великий пул інших популярних мов програмування, що стимулює цю технологію до подальшого розвитку.

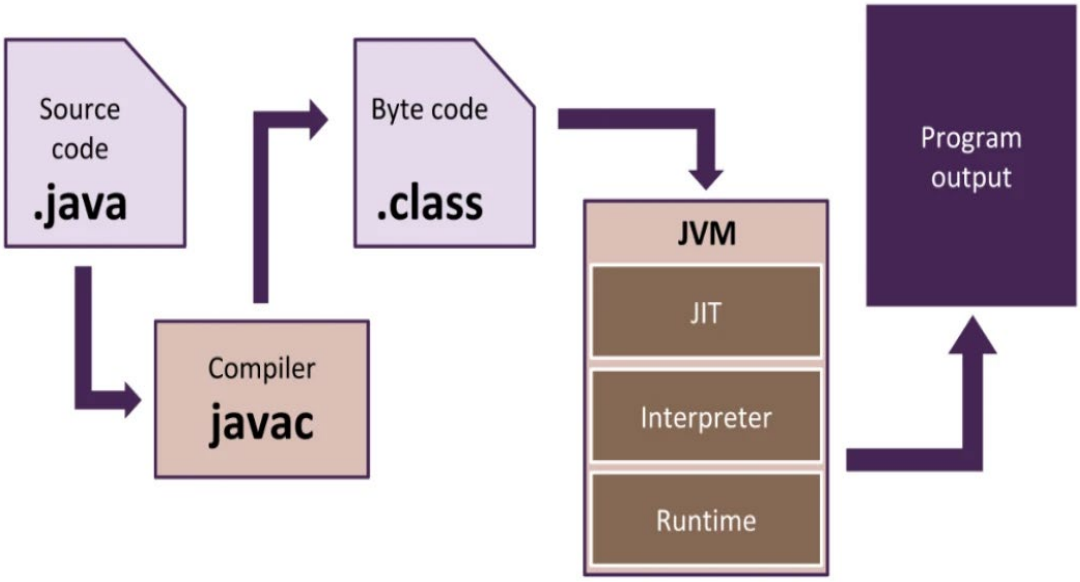


Рис.2.2. Схема роботи JVM (Навчальний ресурс Java Rush https://javarush.com/quests/lectures)

Рис.2.3. Логотип мови програмування Java

Історія мови програмування Java почалася у червні 1991 року. Тоді команда Джеймса Гослінга почала розробку мови програмування, яка б могла використовуватися для телебачення, але створений інструмент виявився занадто технологічним для цієї індустрії.

Перша назва цієї мови програмування була Oak (Дуб), цю назву придумав Джеймс через дерево яке стояв біля офісу його компанії, але потім

назва проєкту була змінена на Green і врешті-решт на Java, в честь назви індонезійської кави.

Цікавою особливістю було те, що Java розроблялася використовуючи синтаксичні конструкції тоді популярних мов програмування C і C++ для того, щоб адаптації нових розробників відбувалася просто і без зайвих проблем.

“Sun Microsystem” опублікували першу версію Java 1.0 у 1996 році. Технологія мала дуже багато переваг у ті часи, особливою її робив принцип WORA (Write once, run everywhere) та системи безпеки які працювали під час поширенню файлів, що зробило цю мову програмування неймовірно популярною с світі Аплетів.

У грудні 1998 років, компанія випустила нову версію мови програмування Java J2SE (Java 2 Standard Edition), з новим модулем J2EE (Java 2 Enterprise Edition), що повністю змінило сферу користування Java. J2EE дала багато зручних інструменті для WEB розробки і почала займати перші лідируючі у цьому напрямку. Навіть у наші часи Java, це сама поширена мова програмування для розробки серверних додатків. У 2010 році, компанія Oracle викупила цю технологію у компанії “Sun Microsystem” та займається її підтримкою та розробкою по наші дні.

Ключові переваги мови програмування Java:

- Легка для вивчення: Мова програмування Java досить дружелюбна до нових користувачів та немає великої кількості складних конструкцій. Розробники попіклувалися для того, щоб залишити тільки необхідні інструменти без зайвих надбудов ти синтаксичного цукру. Тому код написаний на мові програмування Java легко читається та піддається вивченню
- Має велике ком’юніті: Java дуже популярна мова програмування на різних форумах, тому для нас не буде проблемою знати вирішення проблеми або ж запитати у більш досвідчених розробників допомоги. Це величезний плюс для новачків які прагнуть оволодіти нею та почати свій шлях у розробці програмного забезпечення

- Має велику кількість готових рішень: Java має з коробки багато готових рішень, але через її популярність, ком’юніті створило ще цілі стеки фреймоворків, що є досить цікавою особливістю для неї
- Багатопоточна: Java підтримує багатопоточну розробку, щоб робить її ефективним інструментом для вирішення комплексних задач, завдяки діленню одного процесу на менші одиниці та виконуючи їх асинхронно
- Бази даних: Java з коробки підтримує готові рішення для роботи з більшістю комерційних баз даних.
- Створення GUI: Java має готові інструменти для розробки десктопних додатків які будуть працювати незалежно від операційної системи.
- Кросплатформовість: Java можна запускати на різних платформах без зміни коду програми та збереження однакового представлення

2.2.2. Вибір фреймворків Spring та Jmix

З розвитком мови програмування Java у сфері Web розробки, почала збільшуватися кількість бажаючих покращити наявні процеси розробки API. До серйозних нововведень, більшість розробників для створення API користувалися модулем Java EE (Enterprise Edition) який був досить громіздкий та складний у підтримці. У 2014 компанія Pivotal Software розробила фреймворк Spring, який набрав швидкої популярності та зараз використовується у більшості випадків використання мови програмування Java. Цей фреймворк вирішив більшість “болячок” модулю Java EE та сильно спрощує розробникам налаштування проєктів, роботу з базами даних, розробку нових модулі, та має дуже велику кількість нововведень.

Spring запровадив новий механізм конфігурації проєктів через Анотації, які сильно сподобалися користувачам. Після цього відпала необхідність у написанні значної частини boiler plate коду

Наразі проєкт Spring включає у себе величезну кількість модулів, серед яких є:

- Spring MVC: Цей модуль дає змогу працювати всередині додатку використовуючи патерн MVC (Model View Controller), цей патерн умовно розділяє систему на Рівень Моделі, Рівень Уявлення та рівень Контролера, що дає змогу гнучко розробляти додаток
- Spring Rest: Цей модуль дає змогу використовувати архітектурний стиль REST всередині додатку
- Spring Web: Модуль WEB дає можливість працювати з налаштованими серверами Tomcat або Jetty прямо з коробки
- Spring Data Jpa: Цей модель дуже сильно спрощує роботу з базою даних, додаючи такі інструмент як ORM, трансакційність та інші.
- Spring Core: Це основний модуль, який запроваджує механізми IoC (Inversion of Control) і DI(Dependency injection), це основні принципи фреймворку Spring. IoC дає змогу передати управління над створенням об’єктів фреймворку Spring, там DI дає можливість “вприскувати” залежності у необхідні місця
- Spring Boot: Цей модуль дуже сильно спрощує конфігурацію проєкту, та прибирає величезну кількість boiler plate коду
- Spring Transaction: Це модуль додає принцип трансакційності додаючи підтримку ACID

У проєкті було використано всі модулі з списку, але на справді їх існує на багато більше. Для дипломну проєкту було обрано Spring тому що це стало стандартом індустрії та він має надзручний інструментарій, який постійно оновлюється та пристосовується до теперішніх реалій. Також фреймворк має велике ком’юніті, що спростить процес пошуку помилок та вирішення проблем

Рис.2.4. Логотип проєкту Spring

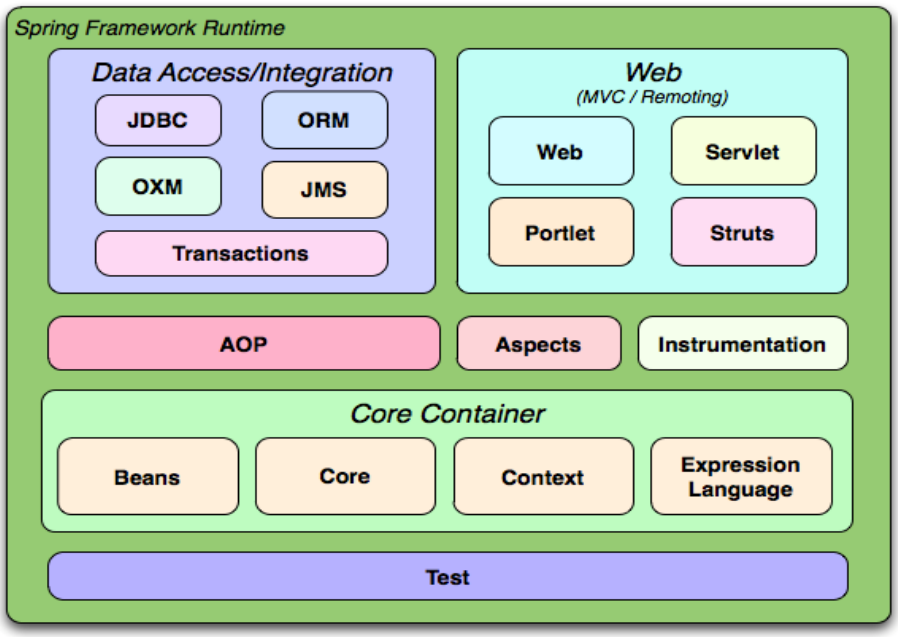


Рис.2.5. Spring Framework Runtime (Офіційна документація Spring framework https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/overview.html)

Фреймворк Jmix

Сьогодні розробка інтерфейсу користувача доволі складна та комплексна задача, яка вимагає кваліфікованого спеціаліста та достатній багаж часу.

Для того, щоб спростити цей процес, було проаналізовано існуючі рішення та обрано фреймворк Jmix, він дає змогу розробляти WEB інтерфейси користуючись Java кодом та XML конфігурацією. Це неймовірно зручний і ефективний інструмент який економить купу часу, будь який Java розробник, який до цього ніколи не мав досвід з розробкою користувацького інтерфейсу, може швидко і просто будувати web інтерфейси без зайвих клопотів.

Під капотом Jmix використовує бібліотеку Vaadin, особливість Jmix в тому що він заточений на вузькопрофільні потреби бізнесу там має великий перечень готових рішень для роботи з базою даних, пагінацією, фільтрами, користувачами, ролями доступу, меню, візуальними компонентами та багато чого іншого. Також в основі цього фреймоворку лежить Spring, що дає змогу користуватися його модулями та легко розробляти додатки.

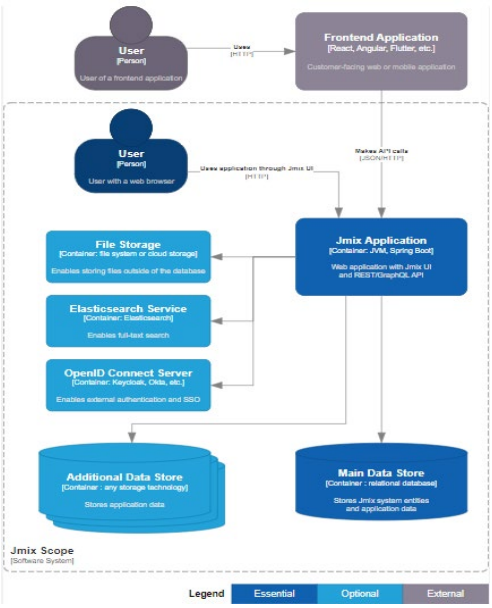


Рис.2.6. Принцип роботи Jmix (Офіційна документація Jmix https://docs.jmix.io/jmix/intro.html)

На рис 2.6. Можна побачити велику кількість інтеграцій з зовнішніми системи що дає змогу розробляти додатки без будь яких обмежень, також існує можливість додавання великої кількості баз даних у одному проєкті, що дає змогу опрацьовувати великі масиви інформації.

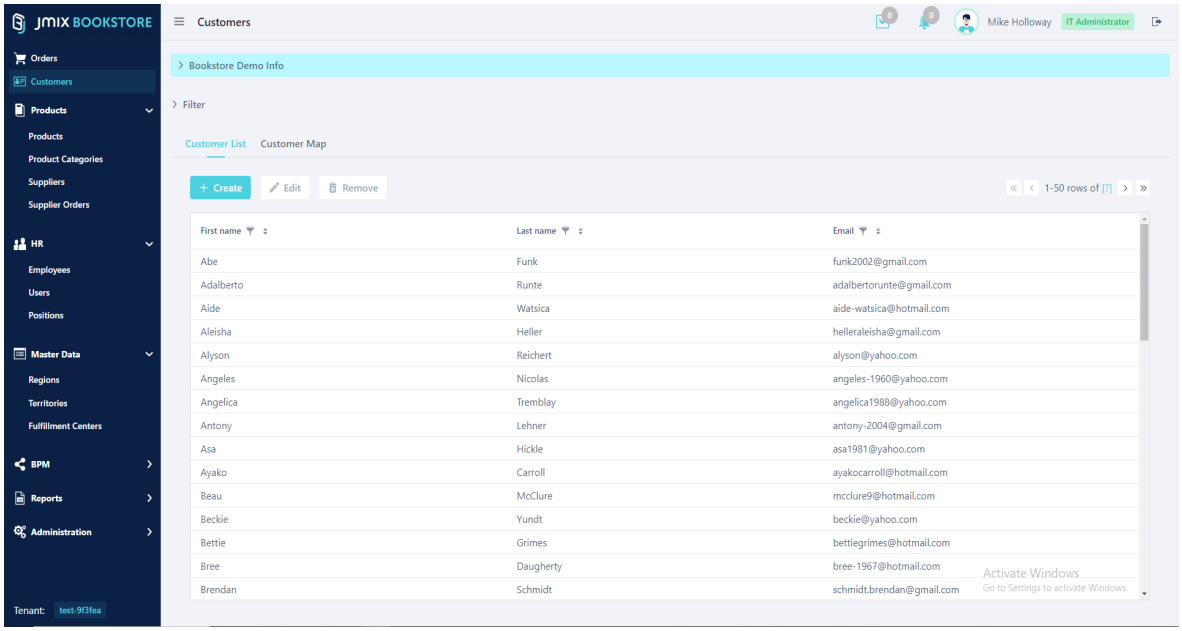


Рис.2.7. Приклад інтерфейсу розробленого за допомогою Jmix

В результаті виходить якісний і готовий до користування продукт з мінімальними зусиллями, саме із за цієї причини було запропоновано цей фреймворк.

2.2.3. Програмний інтерфейс Telegram Bot API

Telegram Bot API - це HTTP-інтерфейс для роботи з ботами у Telegram. По суті бот – це спеціальний обліковий запис, створений для автоматичної відправки та обробки повідомлень.

Для роботи з Telegram Bot API вивчено документацію, в якій описані всі методи і параметри, що передаються. Існує два протилежних за логікою способи отримання оновлень від бота.

1) getUpdates(використовуючи long polling) використовується для отримання оновлень через long polling. Логіка полягає у періодичному опитуванні серверів Telegram на предмет наявності нової інформації. З'єднfння відкривається на нетривалий час і всі оновлення одразу відправляються боту. Відповідь повертається у вигляді масиву об'єктів Update. Параметри методу наведені у таблиці 2.1.

Табл 2.1. Параметри методу getUpdates

Параметри	Тип	Обов’язковий	Опис
offset	Integer	Ні	Дозволяє переглядати нові повідомлення, ігноруючи старі.
limit	Integer	Ні	Обмежує кількість елементів масиву Update. Приймає значення від 1 до 100.
timeout	Integer	Ні	Таймаут в секундах для long polling. За замовчуванням 0, тобто короткий запит.

2) setWebhook необхідний для задання URL вебхука, на який бот відправлятиме оновлення. Mfється на увазі, що тепер якщо в чат приходить повідомлення, то Telegram сам говорить про це. Кожний раз при отриманні оновлення на цю адресу відправлятиметься HTTPS POST із JSON-об'єктом Update. При невдfлому запиті до сервера спроба повториться декілька разів. Для більшої безпеки рекомендується включити токен в URL вебхука, наприклад, наступним чином: https://yourwebhookserver.com/ <token>. Для того щоб отримати токен (власний ключ авторизації) необхідно написати спеціальному боту @BotFather. Оскільки ніхто сторонній не знає особистий токен користувача, запити до вебхуку посилає саме Telegram. Параметри методу наведені в таблиці 2.2.

Табл. 2.2. Параметри методу setWebhook

Параметри	Тип	Обов’язковий	Опис
url	String	Ні	HTTP url для відправки запитів. Щоб видалили вебхук, треба відправити порожній рядок.

certificate	InputFile	Hi	Завантаження публічного ключа для перевірки кореневого сертифікату.

Примітка:

- при підключеному та налfштованому вебхуці метод getUpdates не працюватиме;
- при використанні підписfного власноруч сертифікату, вам необхідно завантажити публічний ключ за допомогою параметра certificate;
- на сьогоднішній день відправка оновлень через вебхуки доступна тільки на ці порти: 443, 80, 88, 8443.

Всі запити до Telegram Bot API повинні здійснюватися через HTTPS в наступному вигляді: https://api.telegram.org/bot<токен>/НАЗВА_МЕТОДУ.

Принцип взаємодії чат-бота та користувача зображено на рисунку 2.4.

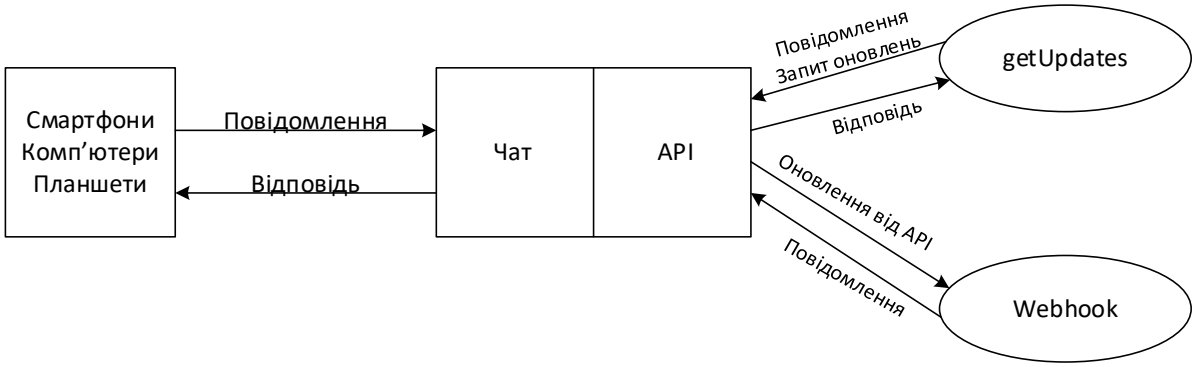


Рис.2.4.Принцип роботи чат-бота на платформі Telegram

Приклади інших доступних для Telegram Bot API методів описані нижче:

- метод для відправки текстових повідомленнях *sendMessage*;
- метод для відправки точки на карті *sendLocation*;
- метод для редагування текстових повідомлень, відправлених ботом чи за допомогою бота *editMessageText*;
- метод для відправки аудіофайлів, якщо бажаєте, щоб клієнти Telegram відображали їх в музичному плеєрі *sendAudio*;

- метод для відправки телефонних контактів *sendContact*.

Допускаються POST і GET запити. Існує 4 способи для передачі параметрів в Bot API:

- запит в URL;

- application / x-www-form-urlencoded;

- application / json (не прийнятний для завантаження файлів);

- multipart / form-data (для завантаження файлів).

2.3. База даних PostgreSql

Загалом база даних - це технологія яка дозволяє зберігати та керувати інформацією. У моєму проєкті використовується база даних для зберігання інформації про історію змін курсів валют, для цієї мети ідеально підходить реляційна база даних. Реляційна база даних – це база даних у якої данні зберігаються у вигляді таблиць та можуть з’єднуватися між собою за допомогою зовнішніх ключів. Ці дані повинні зберігатися по деяким принципам, які називаються “нормалізацією”. Це потрібно для правильного збереження даних без зайвого сміття. Серед усіх варіантів на ринку, було запропоновано обрати Postgres

PostgreSql це система управління базами даних виробничого рівня. Вона має відкритий код, що дає змогу її користувачам вносити зміни у кодову базу задля подальшого розвитку проєкту. Вона може підтримувати як і SQL так і JSON запити. Також ця система підтримує просунуті типи даних та можливість покращення швидкодії які зазвичай існують тільки у платних версіях комерційних баз даних. Сьогодні ця система керування базами даних одна з найпопулярніших у світі.

Ключові переваги, через які ця база даних є досить популярно це:

1. Легкий у використанні. Новим користувачам не потрібні спеціальні тренінги, все інтуїтивно зрозуміло.
2. Менше персоналу. Через простоту цієї бази даних, відпадає потреба у великій кількості працівників для її обслуговування.

3. Відкритий код. Користувачі можуть легко запропонувати свої зміни до кодової бази цього проєкту, також це допомагає вчасно помічати вразливості та виправляти їх.
4. Postgres підтримує географічні типи даних, що дає змогу зручно зберігати координати.
5. Високий рівень логування, що допомагає помітити найдрібніші помилки.

Але ця база даних має свої недоліки:

1. Postgres трішки повільніше за деяких її конкурентів по типу MySQL.
2. Багато додатків з відкритим вихідним кодом не підтримують цю базу даних.
3. Зміни які потрібно для збільшення швидкодії можуть зайняти більше часу ніж у конкурентів.

Загалом наявні плюси перекривають мінуси, тому було обрано саме цю базу для використання.

2.4. База данних Redis

Реляційні бази даних є досить зручними у користуванні і покривають більшість потреб, але є один великий мінус цієї технології - це швидкодія. У світі де мільярди людей користуються інтернетом дуже важливо швидко відповідати на запити користувачів, тому що від цього залежить твоя репутація та дохід.

Виконання запиту на отримання інформації з реляційної бази даних це досить складна операція, що потребує великої кількості ресурсів, а коли цих запитів сотні та тисячі, можна зустрітися з проблемами перенавантаження додатку або бази даних, що призводить до зменшення продуктивності та інших негативних ефектів. Складність цієї операції складає з того що у базі може зберігатися величезна кількість даних, і чим цих даних більше тим складніше знайти те що потрібно, також витрачаються ресурси на відкриття підключень, перетворення даних з таблиць у об’єкти і так далі.

Для вирішення цієї проблеми був придуманий механізм кешування. Цей механізм корисно запроваджувати тоді коли до нас приходить більша кількість запитів на отримання інформації ніж запитів на її оновлення. У випадку дипломного проєкту, до нього будуть приходити запити виключно на отримання інформації, тому буде доречно кешувати дані, щоб запобігти перенавантаженню системи.

Але може виникнути питання, якщо не зберігати інформацію у реляційній базі даних то де це потрібно робити ? На допомогу приходить key-value база даних Redis яка працює у оперативній пам’яті. Дані у цій базі зберігаються у вигляді ключ – значення, що забезпечує дуже швидкий доступ до інформації. Завдяки цьому можна кешувати окремі запити від клієнтів, для того щоб не використовувати кожний раз зайві ресурси.

Переваги використання Redis серед конкурентів:

1. Ця технологія швидша ніж інші сервіси кешування
2. Дуже простий у користуванні
3. Гнучкий да підтримує більшість типів даних
4. Просунуті функціонал кешування
5. Відкритий код
6. Легко маштабується .

Недоліки з якими можна зустрітися:

1. Використовує багато оперативної пам’яті.

Загалом Redis є ідеальним варіантом для мікросервісу.

3 РОЗРОБКА МІКРОСЕРВІСУ

3.1. Отримання інформації про курс фіатних та криптовалют через API

Після проектування мікросервісу час переходити до розробки. Насамперед потрібно знайти джерела з яких можна отримати інформацію про курси валют. З цими сервісами буде проходити комунікація через протокол HTTP та архітектурний стиль REST

Для фіатних валют було запропоновано 3 сервіса:

1. Monobank API (api.monobank.ua)
2. Privatbank API (api.privatbank.ua)
3. Open Data API (bank.gov.ua)

Ці сервіси досить популярні в Україні та мають багато функціоналу. Для дипломного проєкту був вибраний лише функціонал отримання інформації про курси валют і рахунків. Плюсом цих сервісів є те, що не потрібно отримувати API ключ для отримання інформації, але існують обмеження у кількості запитів на секунду.

Також у цих сервісах з різною частотою оновлюються дані, наприклад у Монобанку це 1 раз на 5 хвилин, тому варто кешувати запити до Монобанку мінімум на 5 хвилин, в іншому випадку буде вичерпано ліміт по запитам до цього сервісу даремно.

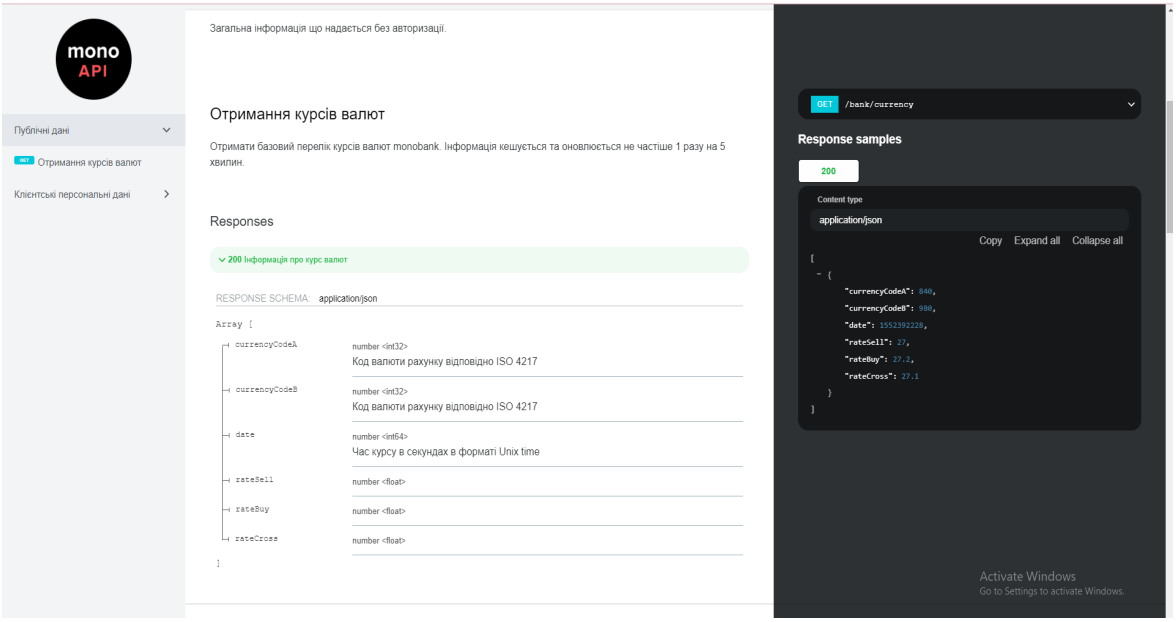


Рис.3.1. Документація по отриманню інформації курсів валют у сервісу Monobank

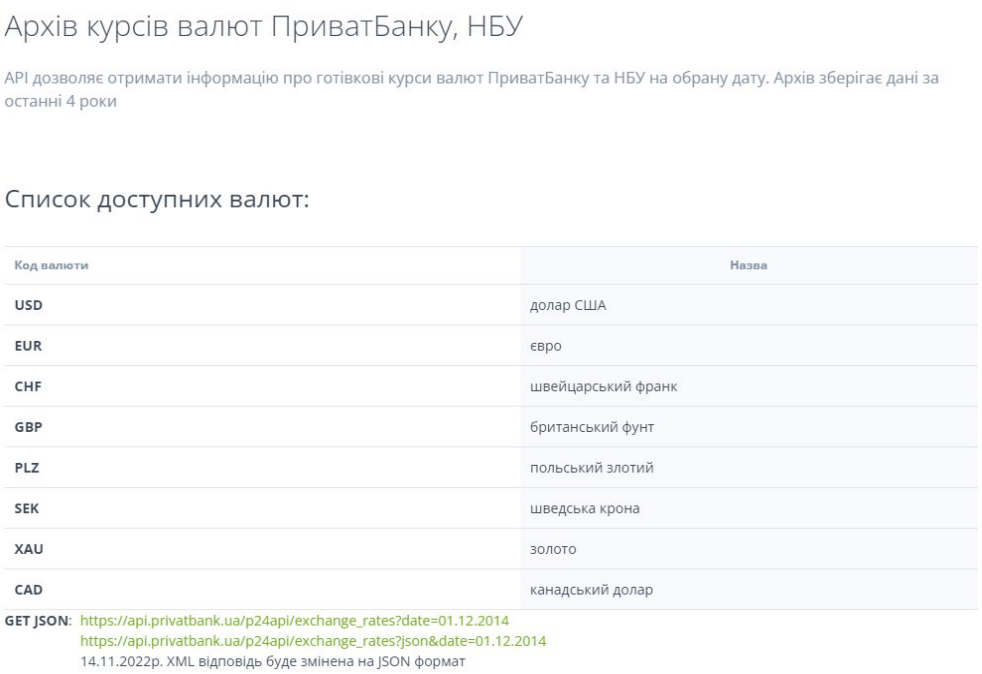


Рис.3.2. Документація по отриманню інформації курсів валют у сервісу Privatbank

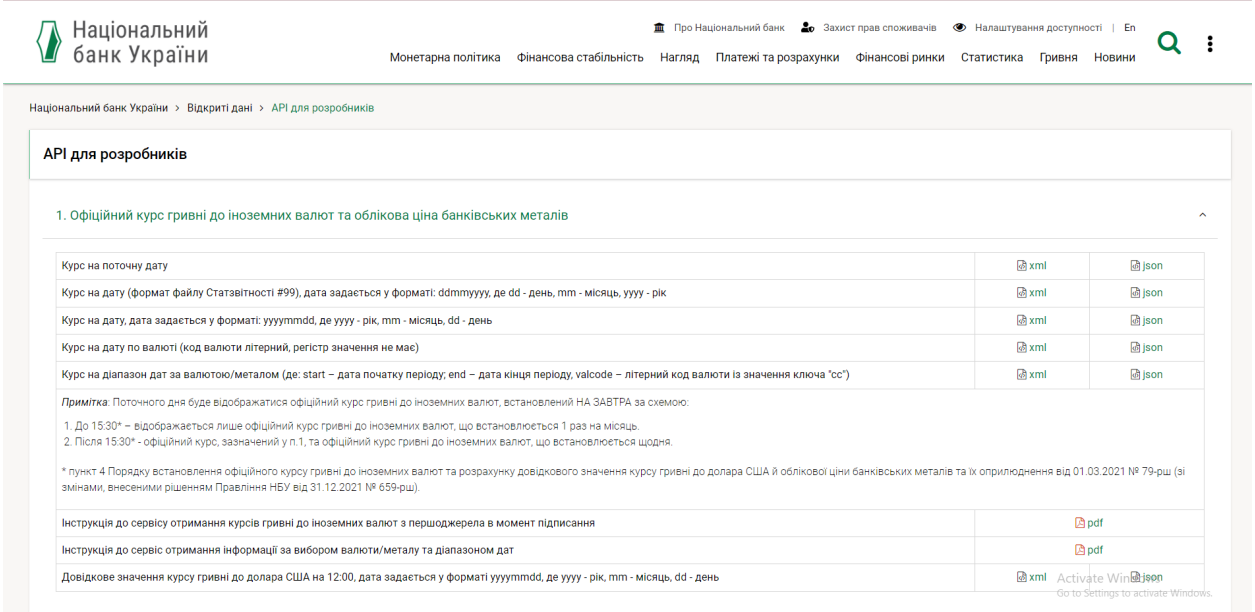


Рис.3.3. Документація по отриманню інформації курсів валют у сервісу Open Data

Ці сервіси підтримують архітектурний стиль REST, а саме використовують JSON для обміну повідомлень, це досить сильно спрощує комунікацію з ними та стандартизує формат обміну повідомлень у додатку. Також варто зазначити, що деякі з сервісів також підтримують інші формати, що може бути корисно для майбутнього функціоналу мікросервісу.

Для отримання інформації про стан курсів криптовалюти було використано сервіс:

1. CoinAPI (rest.coinapi.io)

Це сервіс з величезним функціоналом, що дає багато можливостей для розширення платформи у майбутньому. Щоб використовувати цей сервісу необхідно пройти реєстрацію, для отримання API ключа.

API ключ - це ключ за допомогою якого можна отримати доступ до ресурсів сервісу Coin API. Його потрібно передавати у HTTP запиті, додавши необхідний загаловок “Authorization”. Сам токен керується сервісом який його видає, та може бути видалений. Також, варто зазначити, що мінусом цього сервісу є кількість запитів на добу, а саме 100 запитів, для того, щоб цей функціонал розширити, потрібно купити підписку. На даному етапі 100

запитів цілком достатньо, тому варто частіше кешувати ці данні, та з розвитком сервісу підписка може бути розширена.

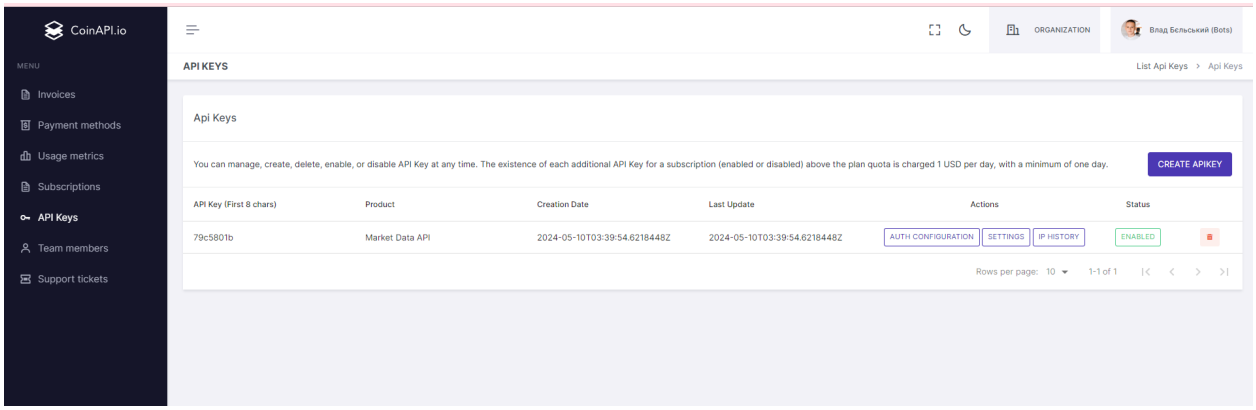


Рис.3.4. Приклад особистого кабінету у сервісі CoinAPI

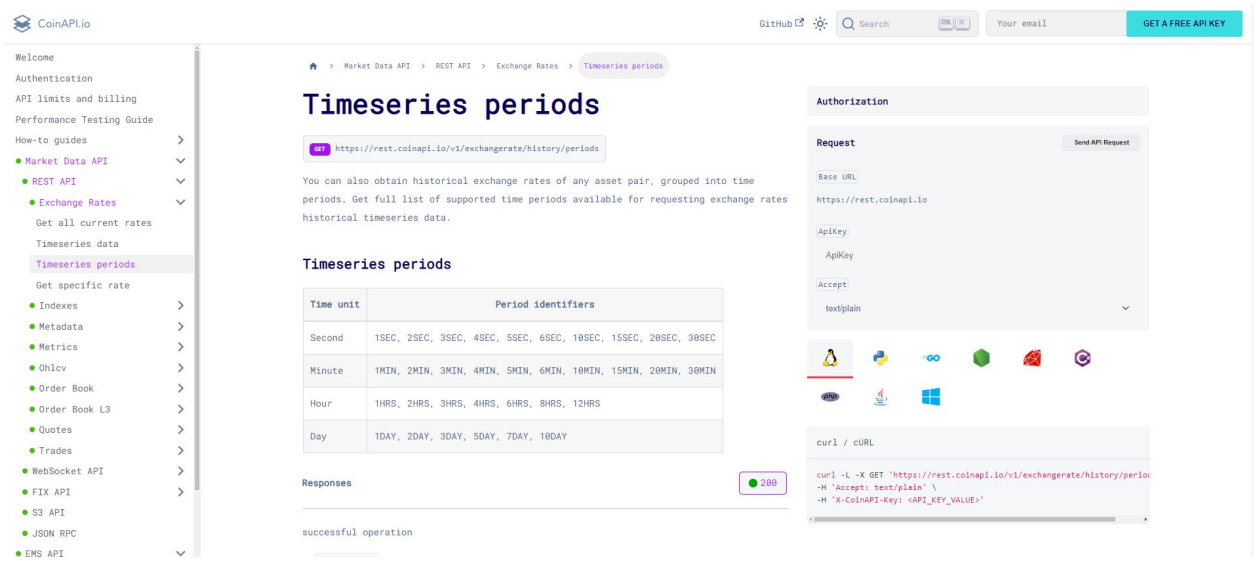


Рис.3.5. Документація по отриманню курсів криптовалюти Coin API

3.2. Реалізація програмної частини мікросервіса

3.2.1. Написання основної логіки мікосервісу

Проєкт складається з файлових структур використовуючи мову програмування Java. Самі пакети розділені за логічним призначенням елементів у них.

Поміж всього для збірки проєкту використовується Gradle, це технологія яка допомагає додавати залежності до проєкту та загалом запаковувати його у

повноцінний додаток. Сам додаток розроблявся у інтегрованій середі розробки Intellij Idea

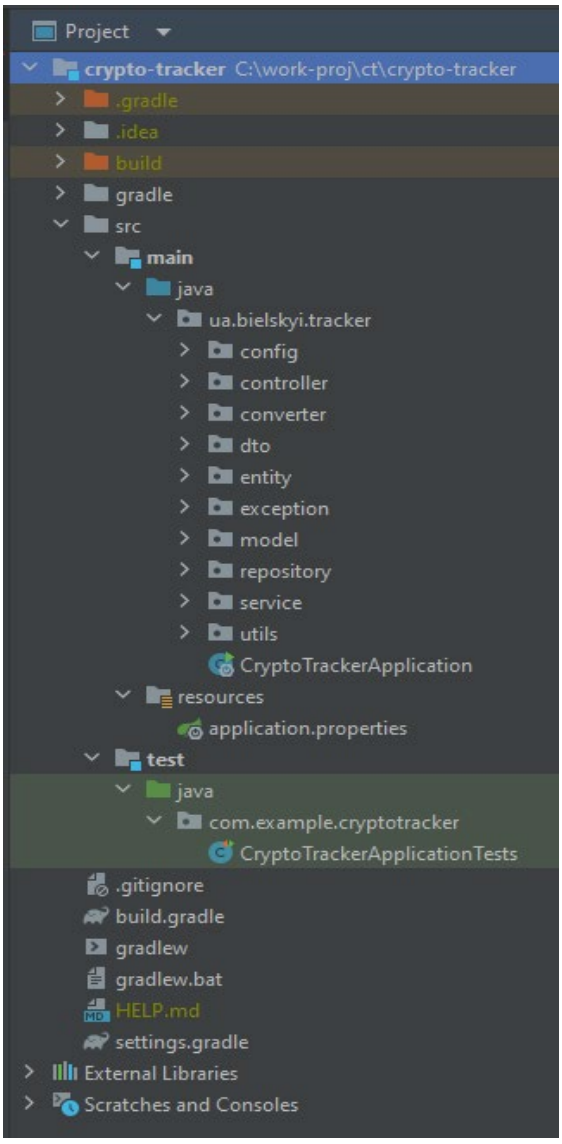


Рис.3.5. Файлова структура проєкту

Розглянемо структуру проєкту по пакетам:

1. config – тут зберігаються основні налаштування проєкту
2. controller – у цьому пакеті знаходяться точки доступу до мікросервісу для сторонніх користувачів
3. converter – у цьому пакеті знаходяться допоміжні клас які надають додатковий функціонал для роботи з датами

4. dto – проєкт використовує паттерн DTO (Data Transfer Object) тобто у системі використовуються окремі об’єкти для перенесення інформації всередині системи не торкаючись persistence рівня, у цьому пакеті зберігаються класи DTO
5. entity – тут зберігаються класи persistence рівня, тобто репрезентації таблиць баз даних у Java коді, це потрібно для роботи з ORM таких як Hibernate
6. exception – у цьому пакеті знаходяться власноруч створені помилки для опрацювання неординарних кейсів
7. repository – у цьому пакеті зберігаються інтерфейси для роботи з базою даних
8. service – тут серце системи, а саме, бізнес логіка по отриманню та опрацюванню курсів валют
9. utils – у цьому пакеті знаходяться допоміжні класи
10. resources – це папка у якій знаходяться значення по типу паролів, адресів баз даних та сторонніх сервісів, що спрощує їх конфігурацію.
11. test – у цій папкі знаходиться тести

Перед початком роботи потрібно додати необхідні залежності до проєкту.

```
dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    implementation 'org.springframework.boot:spring-boot-starter-data-redis'
    implementation 'redis.clients:jedis'
    implementation 'com.squareup.okhttp3:okhttp:4.12.0'
    compileOnly 'org.projectlombok:lombok'
    runtimeOnly 'org.postgresql:postgresql'
    annotationProcessor 'org.projectlombok:lombok'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
}
```

Рис.3.6. Залежності проєкту

Ми можемо побачити декілька нових бібліотек про які не була мова раніше:

1. Lombok – зменшує кількість boiler plate коду за допомогою анотацій які автогенерують код.
2. Jedis – Redis клієнт для зручної роботи з базою даних
3. OkHttp3 – бібліотека для виконання HTTP запитів

Далі необхідно створити папку у якій буде зберігатися бізнес логіка.

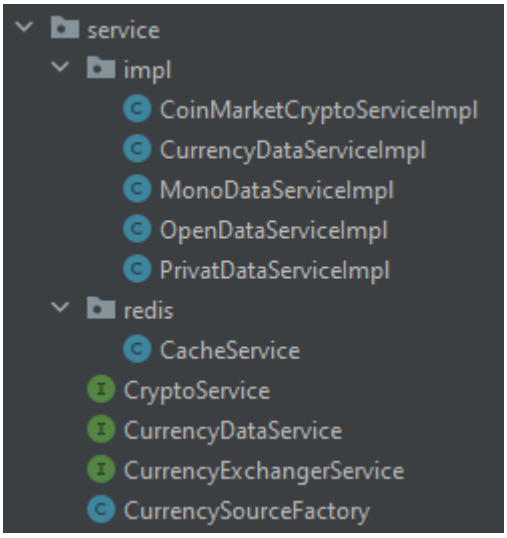


Рис.3.7. Структура пакету service

Можна підкреслити, що тут використовується GOF патерн Abstract Factory - це creational патерн, який робить процес створення схожих об’єктів - простіше.

Наприклад, раніше було визначено, що система буде мати 3 різних джерела з яких будемо брати курси валют. Взагалі ці 3 різні джерела незалежні одне від одного, але їх можна об’єднати за спільними критерієм, всі вони повертають інформацію про курс валют. Для цього потрібно створити об’єкт який буде містити в собі інформацію яка є у всіх 3 джерелах, назвемо його FetchedCurrencyDataDto.

```java
@Getter
@Setter
@AllArgsConstructor
@Builder
public class FetchedCurrencyDataDto {
    private final String baseCurrencyCode;
    private final String currencyCode;
    private final Float rate;
    private final Float buyRate;
    private final Float sellRate;
    private final LocalDateTime date;
    private final Integer source;
}
```

Рис.3.8. Об’єкт FetchedCurrencyDataDto

Після цього необхідно створити один спільний інтерфейс для всіх 3 джерел, який вони зможуть реалізувати однаково CurrencyExchangerService.

Інтерфейс – це контракт, тобто умовний договір між об’єктами у системі. Якщо клас реалізує інтерфейс то він має реалізувати і усі його методи. Зазвичай сам інтерфейс не виконує логічні операції, а виступає шаблоном, але у мові програмування Java існують способи додавання логіки у інтерфейс, це так звані static та default методи, вони були створенні для розширення уже давно написаного функціоналу, та не використовуються у цьому проєкті.

```java
14 usages  3 implementations  vlados1kkk
public interface CurrencyExchangerService {

    2 usages  3 implementations  vlados1kkk
    List<FetchedCurrencyDataDto> getCurrencyDataForThePeriod(LocalDateTime date);

    4 usages  3 implementations  vlados1kkk
    CurrencyData.SourceName getSourceName();
}
```

Рис.3.9. Інтерфейс CurrencyExchangerService

У інтерфейсі CurrencyExchangerService створено методи які повинні будуть реалізувати класи з яких буде братися інформацію про курси валют. getCurrencyDataForThePeriod(LocaDateTime date) - це метод який повертає інформацію про курс валют на конкретну дату. Другий метод getSourceName() повертає тип джерела який реалізує сервіс, це може бути Monobank, Privatbank або Open Data.

Далі потрібно створити конкретні реалізації сервісів та імплементувати новостворений інтерфейс

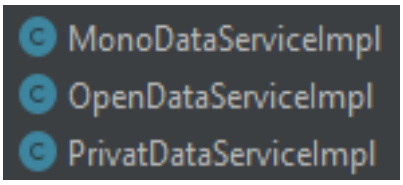
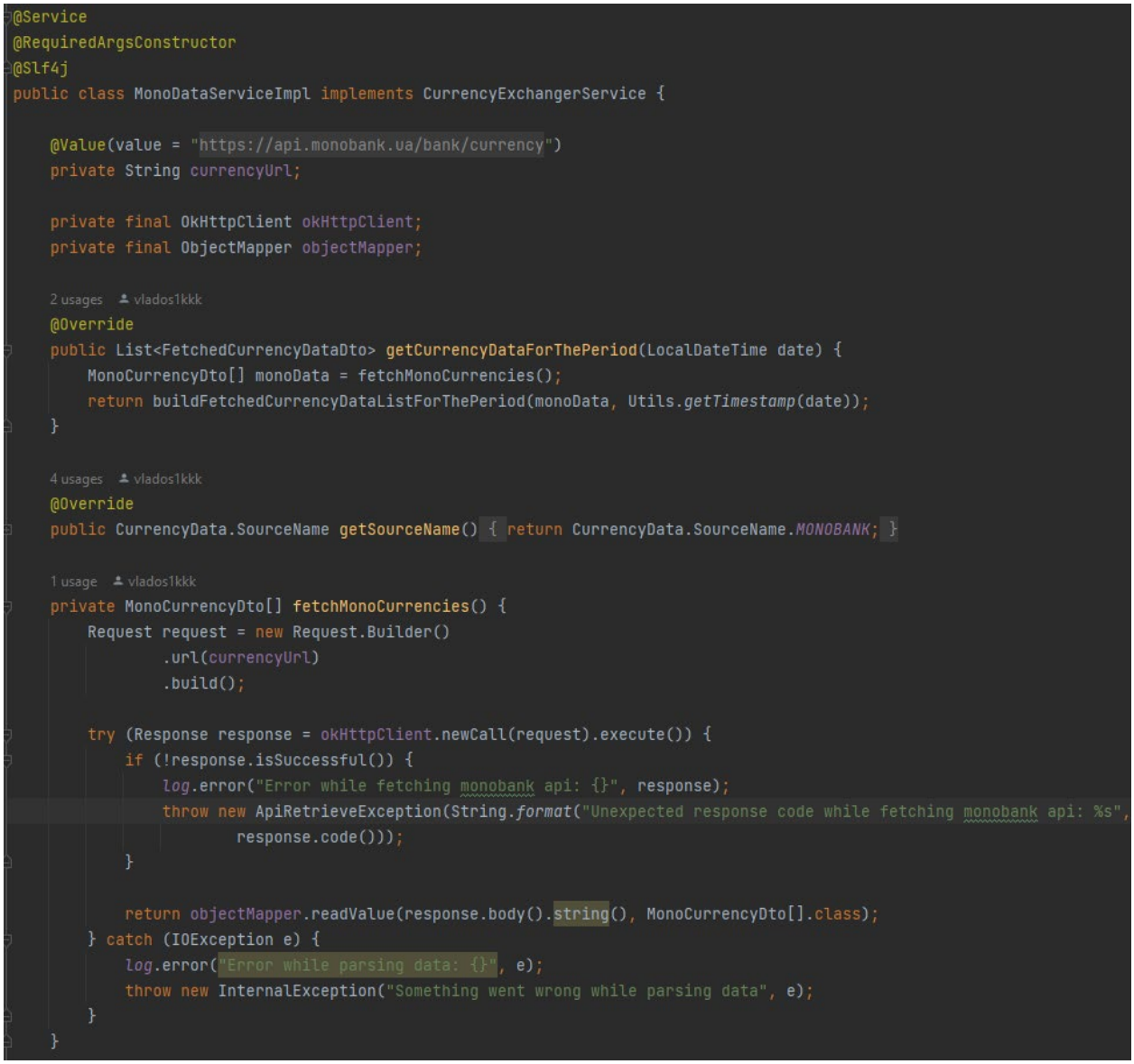


Рис.3.10 Імплементації інтерфейсу CurrencyExchangerService

Давайте розглянемо імплементації інтерфейсу на прикладі MonoDataServiceImpl



```java
@Service
@RequiredArgsConstructor
@Slf4j
public class MonoDataServiceImpl implements CurrencyExchangerService {

    @Value(value = "https://api.monobank.ua/bank/currency")
    private String currencyUrl;

    private final OkHttpClient okHttpClient;
    private final ObjectMapper objectMapper;

    2 usages  vlados1kkk
    @Override
    public List<FetchedCurrencyDataDto> getCurrencyDataForThePeriod(LocalDateTime date) {
        MonoCurrencyDto[] monoData = fetchMonoCurrencies();
        return buildFetchedCurrencyDataListForThePeriod(monoData, Utils.getTimestamp(date));
    }

    4 usages  vlados1kkk
    @Override
    public CurrencyData.SourceName getSourceName() { return CurrencyData.SourceName.MONOBANK; }

    1 usage  vlados1kkk
    private MonoCurrencyDto[] fetchMonoCurrencies() {
        Request request = new Request.Builder()
                .url(currencyUrl)
                .build();

        try (Response response = okHttpClient.newCall(request).execute()) {
            if (!response.isSuccessful()) {
                log.error("Error while fetching monobank api: {}", response);
                throw new ApiRetrieveException(String.format("Unexpected response code while fetching monobank api: %s",
                        response.code()));
            }

            return objectMapper.readValue(response.body().string(), MonoCurrencyDto[].class);
        } catch (IOException e) {
            log.error("Error while parsing data: {}", e);
            throw new InternalException("Something went wrong while parsing data", e);
        }
    }
```

Рис.3.11. Імплементацію інтерфейсу CurrencyExchangerService

На рис.3.11. Можна побачити, що клас реалізує всі методи які було раніше оголошено у інтерфейсі та виконує свою бізнес задачу, у методі getCurrencyDataForThePeriod він робить запит на отримання інформації про курс валюти у Monobank API і повертає необхідний список об’єктів, та у методі getSourceName() повертає свій тип це - Monobank. Хочу зазначити, що у цьому сервісі виконується опрацювання помилок які можуть виникнути у ході виконання програми.

Також існують 2 інші сервіси які реалізують цей інтерфейс OpenDataServiceImpl та PrivatDataServiceImpl. Сенс роботи у них такий самий, лише відрізняються деталі реалізації.

Після того як було створено 3 окремі реалізації інтерфейсу необхідно створити клас який допоможе зручно створювати імплементації об’єктів базуючись на методі getSourceName(). Він буде названий CurrencySourceFactory

```
@Component
public class CurrencySourceFactory {

    2 usages
    private Map<CurrencyData.SourceName, CurrencyExchangerService> sources;

    vlados1kkk
    @Autowired
    public CurrencySourceFactory(Set<CurrencyExchangerService> service) { createSource(service); }

    2 usages  vlados1kkk
    public CurrencyExchangerService findSource(CurrencyData.SourceName sourceName) { return sources.get(sourceName); }

    1 usage  vlados1kkk
    private void createSource(Set<CurrencyExchangerService> sourceSet) {
        sources = sourceSet.stream()
                .collect(Collectors.toMap(CurrencyExchangerService::getSourceName, Function.identity()));
    }
}
```

Рис.3.12. Фабрика для створення об’єктів CurrencySourceFactory

findSource(SourceName sourceName), у цей метод потрібно передати назву джерела, щоб отримати необхідний об’єкт з конкретною реалізацією.

Також можна бачити, що тепер буде легко розширювати функціонал додатку без змін старого коду, якщо у нас з’явиться нове джерело з якого буде братися інформацію про курс валют, необхідно буде створити лише один новий клас замість того що чіпати уже написаний код, тут відбувається притримування одного з принципів SOLID, а саме O (Open closed principle) це означає, що сутності повинні бути відкриті до розширення та закриті до змін

Для роботи з криптовалютою також потрібен інтерфейс, тут не буде створюватися фабрика, тому що зараз у системі лише 1 джерело звідки

береться інформація про курс криптовалют та це принесе непотрібну складність для системи – притримано принципу KISS (Keep it simple).

Рис.3.13. Інтерфейс для роботи з криптовалютою

У цьому інтерфейсі можна побачите лише один метод, який повертає масив об’єктів у яких записана статистика про стан курсів криптовалюти. Ну і для того, щоб отримати результат від цього інтерфейсу, потрібно створити його реалізацію.

Після того як було розроблено зручний спосіб створення об’єктів, необхідно створити сервіс який буде об’єднувати логіку та створювати звіти про стан курсів валют з усіх джерел в одному об’єктів. Також цей сервіс буде мати механізми кешування та асинхронної роботи. Для більшої гнучкості буде створено інтерфейс, щоб у майбутньому без проблем додавати нові реалізації цього сервісу.

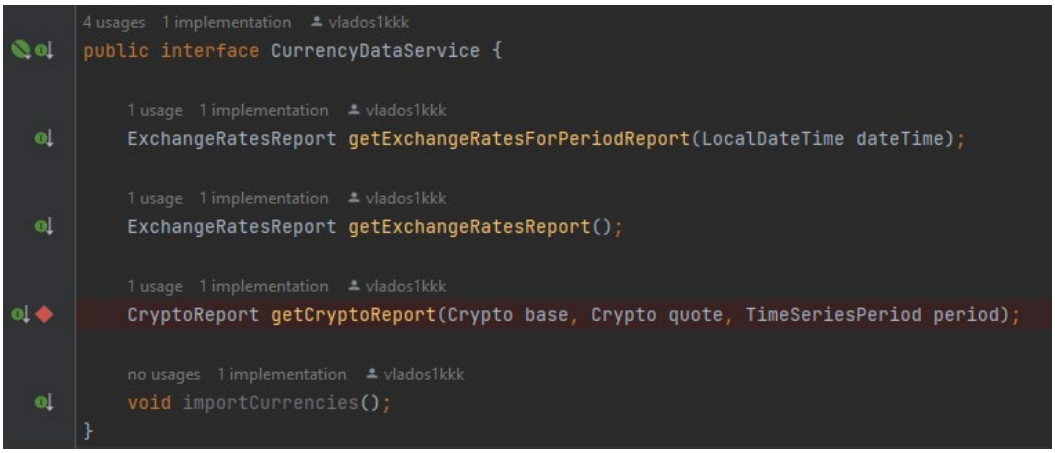


Рис.3.14. Інтерфейс CurrencyDataService

Інтерфейс CurrencyDataService буде мати такі методи як:

1. getExchangeRatesForPeriodReport – повертає звіт про стан курсів валют на конкретну дату
2. getExchangeRatesReport – повертає звіт про стан курсів валют на актуальну дату
3. getCryptoReport – повертає звіт про стан курсів криптовалют
4. importCurrencies - метод який асинхронно імпортуємо інформацію про стан курсів фіатних валют

Перейдемо до реалізації цього сервісу.

```
@Override
@Transactional
@Scheduled(cron = "0 */20 * * * *") // Run every 20 minutes
public void importCurrencies() {
    log.info("Currency data import started at {}", LocalDateTime.now());

    for (CurrencyData.SourceName source : CurrencyData.SourceName.values()) {
        CurrencyExchangerService exchangerService = currencySourceFactory.findSource(source);

        List<CurrencyData> currencyData = null;

        try {
            currencyData = exchangerService.getCurrencyDataForThePeriod(LocalDateTime.now()) List<Fetched
                    .stream() Stream<FetchedCurrencyDataDto>
                    .map(x -> buildCurrencyData(x, actual: true)) Stream<CurrencyData>
                    .collect(Collectors.toList());
        } catch (ApiRetrieveException e) {
            log.error("Error while fetching data {}", e);
        }

        if (currencyData != null && !currencyData.isEmpty()) {
            currencyDataRepository.updateActualToFalseBySource(source.getValue());
            currencyDataRepository.saveAll(currencyData);

            cacheService.delete(Collections.singletonList(RedisUtils.RATES_REPORT_KEY));
        } else {
            log.debug("Currency data for source {} unavailable", source);
        }
    }

    log.info("Currency data import finished at {}", LocalDateTime.now());
}
```

Рис.3.15. Метод асинхронного оновлення інформації про стан фіатних валют

Метод importCurrencies() раз на 20 хвилин імпортує у базу даних нову інформацію про стан курсів фіатних валют та робить стару інформацію неактуальною

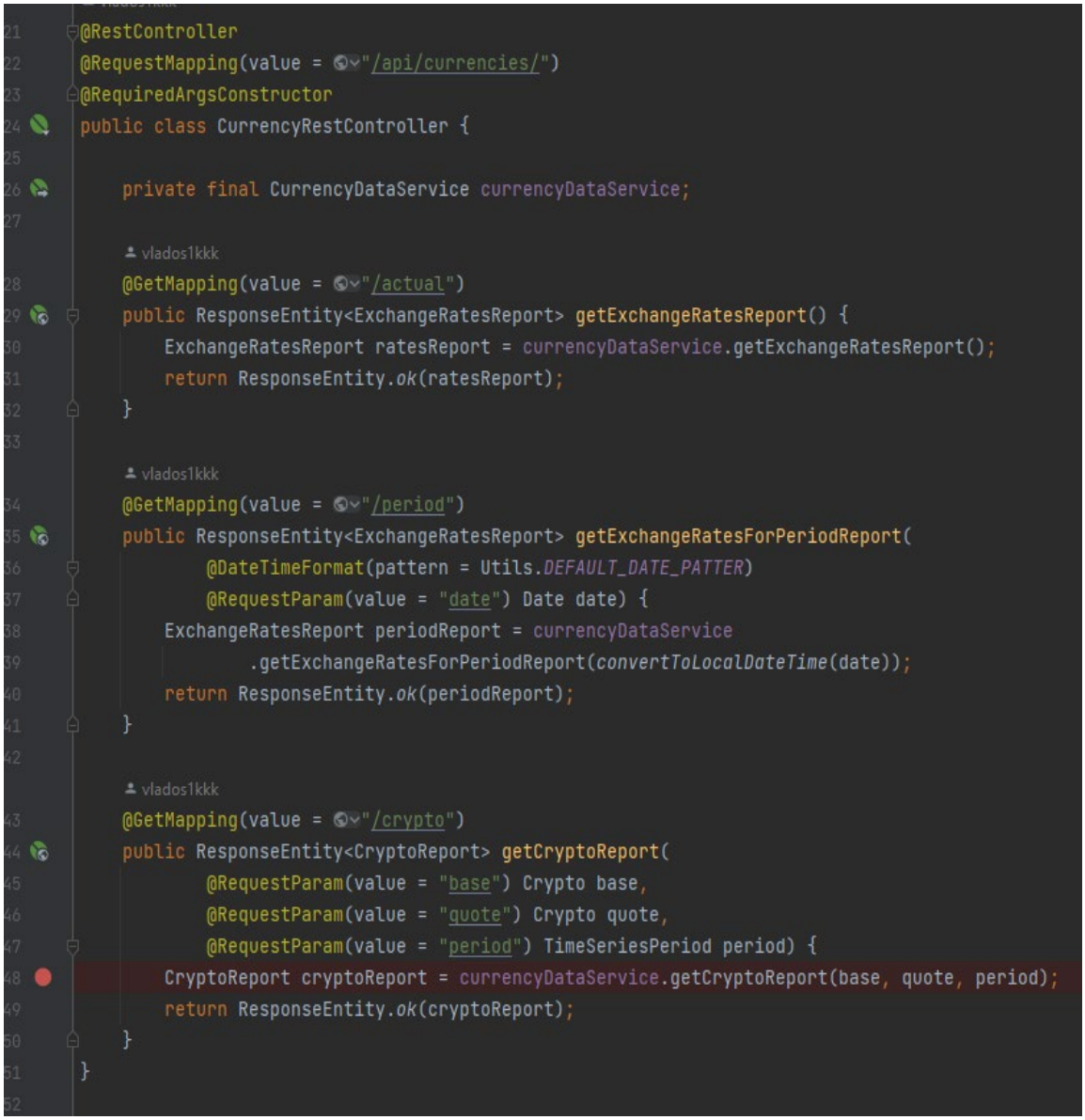
```
@Override
public ExchangeRatesReport getExchangeRatesForPeriodReport(LocalDateTime date) {
    Optional<ExchangeRatesReport> cacheReport = getDateReportFromCache(date);
    if (cacheReport.isPresent()) {
        return cacheReport.get();
    } else {
        ExchangeRatesReport report = generateExchangeRatesReport(date, Boolean.FALSE);
        if (report.getErrors().isEmpty()) {
            cacheService.set(RedisUtils.getRatesPeriodReportKey(Utils.getTimestamp(date)), report, RedisUtils.ONE_HOUR_MILLIS)
        }
        return report;
    }
}

1 usage  vlados1kkk
@Override
public ExchangeRatesReport getExchangeRatesReport() {
    Optional<ExchangeRatesReport> cacheReport = getReportFromCache();
    if (cacheReport.isPresent()) {
        return cacheReport.get();
    } else {
        ExchangeRatesReport report = generateExchangeRatesReport(LocalDateTime.now(), Boolean.TRUE);
        if (report.getErrors().isEmpty()) {
            cacheService.set(RedisUtils.RATES_REPORT_KEY, report, RedisUtils.TWENTY_MINUTES_MILLIS);
        }
        return report;
    }
}

1 usage  vlados1kkk
@Override
public CryptoReport getCryptoReport(Crypto base, Crypto quote, TimeSeriesPeriod period) {
    Optional<CryptoReport> cacheReport = getCryptoReportFromCache(base, quote, period);
    if (cacheReport.isPresent()) {
        return cacheReport.get();
    } else {
        CryptoReport report = generateCryptoReport(base, quote, period);
        return report;
    }
}
```

Рис.3.16. Методи отримання валютних звітів

Тут можна спостерігати методи отримання валютних звітів. Після кожної генерації успішного звіту він кешується на 20 хвилин та подалі повертаємо закешований варіант замість того щоб збирати інформацію з нуля.

На даному етапі готовий сервіс по отриманню звітів, далі, щоб мікросервісом могли користуватися інші користувачі, необхідно створити так звані “ендпоїнти”.

Систему було чітко розділено за патерном MVC, зараз вже розроблені Model layer, View layer залишилося реалізувати Controller layer. Для цього буде створено клас CurrencyRestController



```
@RestController
@RequestMapping(value = "/api/currencies/")
@RequiredArgsConstructor
public class CurrencyRestController {

    private final CurrencyDataService currencyDataService;

    vlados1kkk
    @GetMapping(value = "/actual")
    public ResponseEntity<ExchangeRatesReport> getExchangeRatesReport() {
        ExchangeRatesReport ratesReport = currencyDataService.getExchangeRatesReport();
        return ResponseEntity.ok(ratesReport);
    }

    vlados1kkk
    @GetMapping(value = "/period")
    public ResponseEntity<ExchangeRatesReport> getExchangeRatesForPeriodReport(
            @DateTimeFormat(pattern = Utils.DEFAULT_DATE_PATTER)
            @RequestParam(value = "date") Date date) {
        ExchangeRatesReport periodReport = currencyDataService
                .getExchangeRatesForPeriodReport(convertToLocalDateTime(date));
        return ResponseEntity.ok(periodReport);
    }

    vlados1kkk
    @GetMapping(value = "/crypto")
    public ResponseEntity<CryptoReport> getCryptoReport(
            @RequestParam(value = "base") Crypto base,
            @RequestParam(value = "quote") Crypto quote,
            @RequestParam(value = "period") TimeSeriesPeriod period) {
        CryptoReport cryptoReport = currencyDataService.getCryptoReport(base, quote, period);
        return ResponseEntity.ok(cryptoReport);
    }
}
```

Рис.3.17. контролер CurrencyRestController

Завдяки цьому класу буде можливість повертати результат роботи CurrencyDataService через протокол HTTP у вигляді JSON файлу, що дасть змогу зовнішнім користувачам отримувати валютні звіти.

В результаті було створено такі точки входу у інформаційну систему:

1. /api/currencies/actual – Звіт по всім джерелам на актуальну дату

2. /api/currencies/period – Звіт по всім джерелам за конкретний період
3. /api/currencies/crypto – Звіт по курсу криптовалюти

3.2.2. Підключення бази даних PostgreSql та Redis

Для того щоб зберігати інформацію про валютні курси було запропоновано базу даних Postgres. Щоб підключитися до неї через Java додаток, необхідно додати у проєкт залежність з необхідним драйвером та у файлі application.properties вказати дані для підключення до бази даних, далі відбудеться “магія” та Spring сконфігурує все за нас.

```
# DataSource Configuration
spring.datasource.url=jdbc:postgresql://localhost:5432/currency_db
spring.datasource.driverClassName=org.postgresql.Driver
spring.datasource.username=postgres
spring.datasource.password=fury
```

Рис.3.18. Налаштування підключення до бази даних у файлі application.properties

Також було створено базу даних яка називається currency_db та всередині неї таблицю currencies. Ця таблиця містить у собі інформацію про курс валюти за конкретний період.

```
CREATE TABLE public.currencies (

actual bool NULL,

buy_rate float4 NULL,

rate float4 NULL,

sell_rate float4 NULL,

"source" int4 NULL,

created_at int8 NOT NULL,

"date" int8 NULL,
```

```
id bigserial NOT NULL,

base_currency_code varchar(255) NULL,

currency_code varchar(255) NULL,

CONSTRAINT currencies_pkey PRIMARY KEY (id)

);
```

Для роботи з базою даних, було створено клас у додатку. За допомогою технології ORM він репрезентує таблицю currencies.

```
@Entity
@Table(name = "currencies")
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
public class CurrencyData extends BaseEntity {

    @Column(name = "base_currency_code")
    private String baseCurrencyCode;

    @Column(name = "currency_code")
    private String currencyCode;

    @Column(name = "buy_rate")
    private Float buyRate;

    @Column(name = "sell_rate")
    private Float sellRate;

    @Column(name = "rate")
    private Float rate;

    @Column(name = "date")
    @Convert(converter = LocalDateTimeConverter.class)
    private LocalDateTime date;

    @Column(name = "actual")
    private Boolean actual;

    @Column(name = "source")
    private Integer source;
```

Рис.3.19. Клас який репрезентує таблицю currencies

Далі, потрібно буде створити зручний інтерфейс для роботи з базою, більшість роботи у цьому напрямку виконує модуль фреймворку Spring – Spring Data Jpa він суттєво спрощує роботу з базою даних.

Для того щоб підключитися до бази даних Redis, необхідно також вказати конфігурацію у файлі.

```
# Redis Configuration
spring.data.redis.host=localhost
spring.data.redis.port=6379
spring.data.redis.password=
```

Рис.3.20. Налаштування підключення до Redis у файлі

Більшість роботи з налаштування виконає модуль Spring Data Redis він спрямований на автоматичне налаштування роботи з Redis, від нас потрібно лише створення клієнта для взаємодії з цією технологією.

3.2.3. Розробка WEB інтерфейсу та Telegram бота

Для того, щоб зручно відслідковувати інформацію про актуальні курси валют було запропоновано створити WEB інтерфейс.

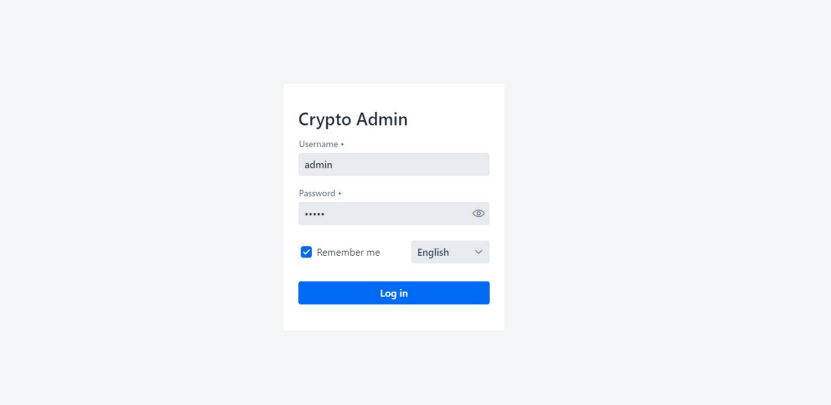


Рис.3.21. Зручна панель авторизації користувача

Інтерфейс має зручний спосіб авторизації користувача, також може бути розширена кількість доступних локалізацій, на разі підтримується тільки англійська. Користувачу доступна кнопка “Запам’ятати мене”, щоб не було потреби заново авторизуватися у систему.

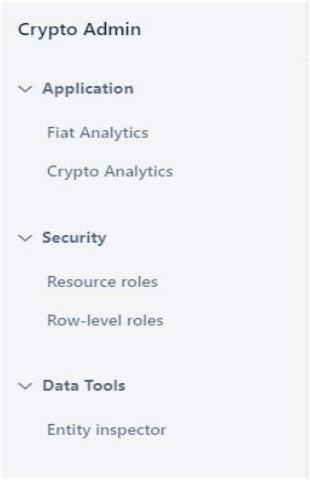


Рис.3.22. Меню WEB інтерфейсу

Додаток має зручне меню з основним функціоналом. Розглянемо декілька з них:

1. Fiat Analytics

   Зручний інструмент для відслідковування курсу обраної валюти за певний проміжок часу, цікавою особливістю є те, що можна бачити інформацію відразу від 3 різних джерел.

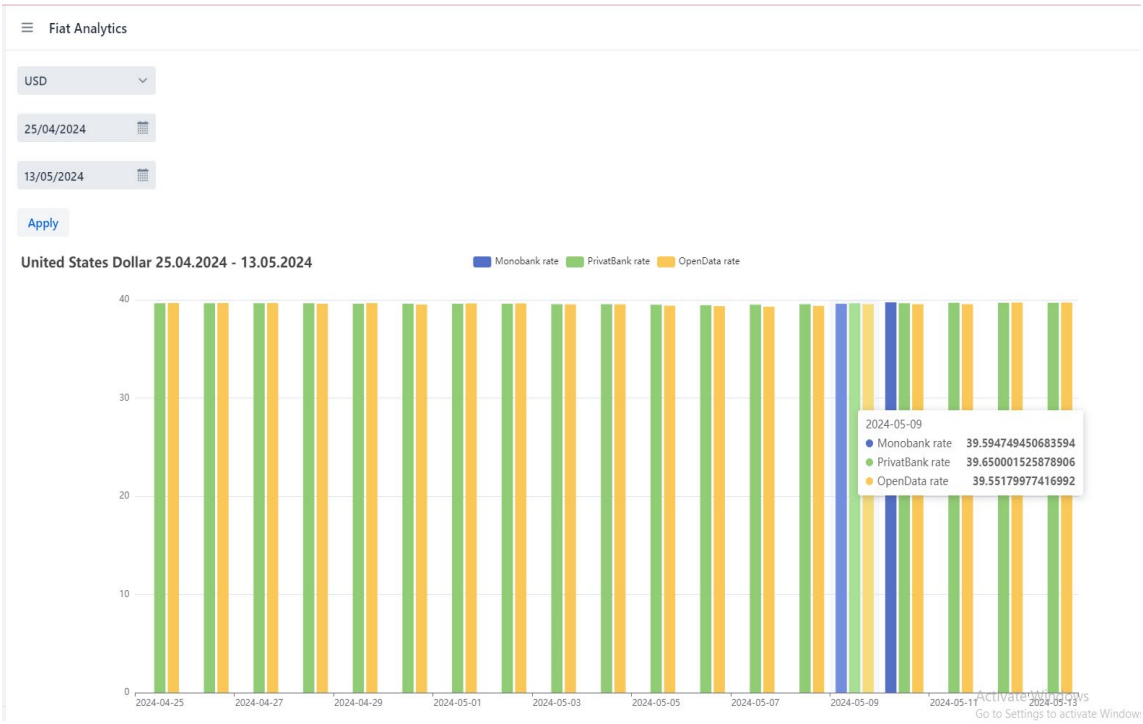


Рис.3.23. Інтерфейс Fiat Analytics

2. Crypto Analytics

Неймовірно зручний інструмент у якому можна побачити курс криптовалюти відносно іншої за певний проміжок часу. Використовується графік типу “Свічка”, що є стандартом для аналізу фінансових ринків

Рис.3.24. Інтерфейс Crypto Analytics

Для того, щоб скористатися Телеграм ботом достатньо запустити його виконавши команду /start, після цього з періодичністю в 1 день до нас будуть приходити актуальні курси фіатних та криптовалют.

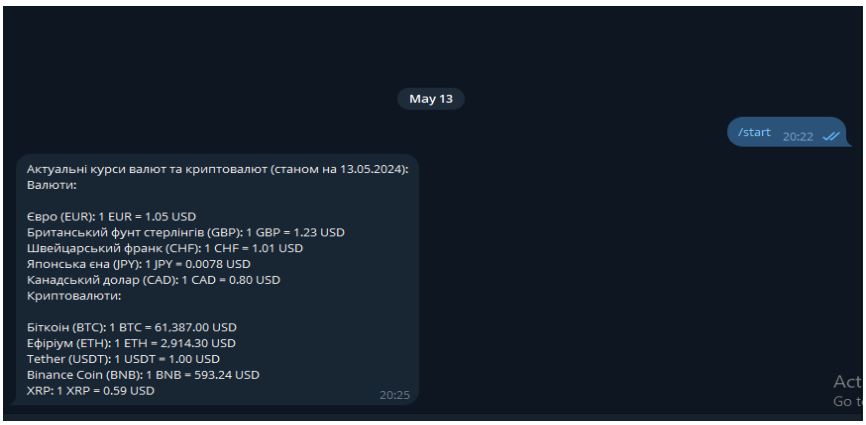


Рис.3.25. Приклад оповіщенням у телеграм

ВИСНОВКИ

Результатом виконання роботи є мікросервіс який повертає звіти про курс фіатних та криптовалют. Дані з цього сервісу можна використовувати у користувацьких інтерфейсах, телеграм ботах. Також інші системи можуть інтегрувати звіти у свої системи використовуючи протокол HTTP і архітектурний стиль REST.

В першому розділі було розглянуто мікросервісну архітектуру, також було порівняно її з монолітною архітектурою та знайдено плюси та мінуси її використання. Також було покращено знання у області фіатних та криптовалют, щоб знайти різницю між ними. Було проаналізовано функціональні можливості криптовалютних аналітичних сервісів. В результаті було визначено призначення та вимоги до майбутнього мікросервісу

В другом розділі було розроблено функціонально можливості мікросервісу, була обрана мова програмування і фреймворки та бази даних, що грають важливу роль у проєктуванні та забезпеченню надійної роботи інформаційної системи.

Третій розділ було присвячено безпосередньо розробці мікросервісу, було розібрано певні джерела з яких береться інформація про актуальний курс валют, також було показано кінцеву архітектуру мікросервісу та патерни які в ній використовуються. В результаті було презентовано результат робити інформаційної системи через WEB інтерфейс та телеграм бота.

Завдяки правильно підібраним фреймворкам, бібліотекам, базам даних та архітектурі, було побудовано надійну систему, яка зможе оброблювати велику кількість користувачі та забезпечувати їх актуальною інформацією про стан курсів валют.

Мікросервіси є унікальним архітектурним рішенням, яке допомагає забезпечити побудову гнучких та надійних інформаційних систем, які легкі у розробці та підтримці, також вони можуть легко маштабуватися та розширюватися новими модулями та інтеграціями.

Проєкт має багато перспектива до розвитку, та можна зайняти своє місце на ринку IT технології і аналітичних сервісів. Важливим кроком буде публікації мкросервісу на сервер задля подальшого його розповсюдження у мережі інтернет.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Spring Framework Documentation [Електронний ресурс] – https://docs.spring.io/spring-framework/reference/index.html
2. Jmix Documentation [Електронний ресурс] - https://docs.jmix.io/jmix/intro.html
3. Java (programming language) [Електронний ресурс] - https://en.wikipedia.org/wiki/Java_(programming_language)#:~:text=Java%20was%20originally%20developed%20by,by%20Sun%20under%20proprietary%20licenses.
4. Documentation PotgresSQL [Електронний ресурс] - https://www.postgresql.org/docs/
5. Redis Documentation [Електронний ресурс] - https://redis.io/docs/latest/
6. Microservice architecture style [Електронний ресурс] - https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices
7. Bitcoin, cryptocurrency, blockchain... So what does it all mean? [Електронний ресурс] - https://www.pwc.com/us/en/industries/financial-services/fintech/bitcoin-blockchain-cryptocurrency.html
8. Telegram Bot API [Електронний ресурс] – Режим доступу до ресурсу: https://core.telegram.org/bots/api.
9. Best Blockchain Analysis Tools [Електронний ресурс] – https://medium.com/coinmonks/12-best-blockchain-analysis-tools-2fb8bd62db5c
10. Telegram FAQ [Електронний ресурс] – Режим доступу до ресурсу: https://telegram.org/.

ДОДАТОК А. ТЕКСТ ПРОГРАМИ

Файл ‘’FetchedCurrencyDataDto.java

```
package ua.bielskyi.tracker.dto;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Getter;
import lombok.Setter;

import java.time.LocalDateTime;

@Getter
@Setter
@AllArgsConstructor
@Builder
public class FetchedCurrencyDataDto {
    private final String baseCurrencyCode;
    private final String currencyCode;
    private final Float rate;
    private final Float buyRate;
    private final Float sellRate;
    private final LocalDateTime date;
    private final Integer source;
}
```

Файл CurrencyExchangerService.java

```
package ua.bielskyi.tracker.service;

import ua.bielskyi.tracker.dto.FetchedCurrencyDataDto;
import ua.bielskyi.tracker.entity.CurrencyData;

import java.time.LocalDateTime;
import java.util.List;
```

```
public interface CurrencyExchangerService {

   List<FetchedCurrencyDataDto> getCurrencyDataForThePeriod(LocalDateTime date);

   CurrencyData.SourceName getSourceName();
}
```

Файл CurrencyDataService.java

```
package ua.bielskyi.tracker.service;

import ua.bielskyi.tracker.model.Crypto;
import ua.bielskyi.tracker.model.CryptoReport;
import ua.bielskyi.tracker.model.ExchangeRatesReport;
import ua.bielskyi.tracker.model.TimeSeriesPeriod;

import java.time.LocalDateTime;

public interface CurrencyDataService {

   ExchangeRatesReport getExchangeRatesForPeriodReport(LocalDateTime dateTime);

   ExchangeRatesReport getExchangeRatesReport();

   CryptoReport getCryptoReport(Crypto base, Crypto quote, TimeSeriesPeriod period);

   void importCurrencies();
}
```

Файл CryptoService.java

```
package ua.bielskyi.tracker.service;

import ua.bielskyi.tracker.dto.sources.CoinApiCryptoDto;
import ua.bielskyi.tracker.model.Crypto;
import ua.bielskyi.tracker.model.TimeSeriesPeriod;

public interface CryptoService {
```

```
    CoinApiCryptoDto[] fetchCryptoData(Crypto base, Crypto quote, TimeSeriesPeriod period);
}
```

Файл MonoDataServiceImpl.java

```
package ua.bielskyi.tracker.service.impl;

import com.fasterxml.jackson.databind.ObjectMapper;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import okhttp3.OkHttpClient;
import okhttp3.Request;
import okhttp3.Response;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;
import ua.bielskyi.tracker.dto.FetchedCurrencyDataDto;
import ua.bielskyi.tracker.dto.sources.MonoCurrencyDto;
import ua.bielskyi.tracker.entity.CurrencyData;
import ua.bielskyi.tracker.exception.ApiRetrieveException;
import ua.bielskyi.tracker.exception.InternalException;
import ua.bielskyi.tracker.service.CurrencyExchangerService;
import ua.bielskyi.tracker.utils.Utils;

import java.io.IOException;
import java.time.Instant;
import java.time.LocalDateTime;
import java.time.ZoneOffset;
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

@Service
@RequiredArgsConstructor
@Slf4j
public class MonoDataServiceImpl implements CurrencyExchangerService {

    @Value(value = "${ua.bielskyi.coin.market.mono.api}")
```

```
    private String currencyUrl;

    private final OkHttpClient okHttpClient;
    private final ObjectMapper objectMapper;

    @Override
    public List<FetchedCurrencyDataDto> getCurrencyDataForThePeriod(LocalDateTime date) {
        MonoCurrencyDto[] monoData = fetchMonoCurrencies();
        return buildFetchedCurrencyDataListForThePeriod(monoData, Utils.getTimestamp(date));
    }

    @Override
    public CurrencyData.SourceName getSourceName() {
        return CurrencyData.SourceName.MONOBANK;
    }

    private MonoCurrencyDto[] fetchMonoCurrencies() {
        Request request = new Request.Builder()
                .url(currencyUrl)
                .build();

        try (Response response = okHttpClient.newCall(request).execute()) {
            if (!response.isSuccessful()) {
                log.error("Error while fetching monobank api: {}", response);
                throw new ApiRetrieveException(String.format("Unexpected response code while fetching
monobank api: %s",
                        response.code()));
            }

            return objectMapper.readValue(response.body().string(), MonoCurrencyDto[].class);
        } catch (IOException e) {
            log.error("Error while parsing data: {}", e);
            throw new InternalException("Something went wrong while parsing data", e);
        }
    }
```

```
    private List<FetchedCurrencyDataDto>
buildFetchedCurrencyDataListForThePeriod(MonoCurrencyDto[] monoDataDto, long date) {
        return Arrays.stream(monoDataDto)
                .filter(x -> Instant.ofEpochSecond(x.getDate()).atZone(ZoneOffset.UTC).toLocalDate()
                        .isEqual(Instant.ofEpochSecond(date).atZone(ZoneOffset.UTC).toLocalDate()))
                .map(dataDto -> mapFetchedCurrencyDto(dataDto))
                .collect(Collectors.toList());
    }

    private FetchedCurrencyDataDto mapFetchedCurrencyDto(MonoCurrencyDto dataDto) {
        boolean hasRate = dataDto.getRateBuy() != 0 && dataDto.getRateSell() != 0;
        return FetchedCurrencyDataDto.builder()
                .baseCurrencyCode(dataDto.getCurrencyCodeB())
                .currencyCode(dataDto.getCurrencyCodeA())
                .date(LocalDateTime.ofEpochSecond(dataDto.getDate(), 0, ZoneOffset.UTC))
                .buyRate(hasRate ? dataDto.getRateBuy() : 0)
                .sellRate(hasRate ? dataDto.getRateSell() : 0)
                .rate(hasRate ? Utils.getRate(dataDto.getRateBuy(), dataDto.getRateSell()) :
dataDto.getRateCross())
                .source(getSourceName().getValue())
                .build();
    }
}
```

Файл CoinMarketCryptoServiceImpl.java

```
package ua.bielskyi.tracker.service.impl;

import com.fasterxml.jackson.databind.ObjectMapper;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import okhttp3.HttpUrl;
import okhttp3.OkHttpClient;
import okhttp3.Request;
import okhttp3.Response;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;
import ua.bielskyi.tracker.dto.sources.CoinApiCryptoDto;
```

```
import ua.bielskyi.tracker.exception.ApiRetrieveException;
import ua.bielskyi.tracker.exception.InternalException;
import ua.bielskyi.tracker.model.Crypto;
import ua.bielskyi.tracker.model.TimeSeriesPeriod;
import ua.bielskyi.tracker.service.CryptoService;

import java.io.IOException;

@Slf4j
@Service
@RequiredArgsConstructor
public class CoinMarketCryptoServiceImpl implements CryptoService {

    @Value(value = "${ua.bielskyi.coin.api.url}")
    private String currencyUrl;

    @Value(value = "${ua.bielskyi.coin.api.key}")
    private String currencyKey;

    private final OkHttpClient okHttpClient;
    private final ObjectMapper objectMapper;

    @Override
    public CoinApiCryptoDto[] fetchCryptoData(Crypto base, Crypto quote, TimeSeriesPeriod period) {
        HttpUrl url = HttpUrl.parse(currencyUrl + "/v1/exchangerate/"
                        + base.name() + "/"
                        + quote.name() + "/history")
                .newBuilder()
                .addQueryParameter("period_id", period.getPeriodIdentifier())
                .build();
        Request request = new Request.Builder()
                .url(url)
                .addHeader("Accept", "application/json")
                .addHeader("X-CoinAPI-Key", currencyKey)
                .build();
```

```
        try (Response response = okHttpClient.newCall(request).execute()) {
            if (!response.isSuccessful()) {
                log.error("Error while fetching coin api: {}", response);
                throw new ApiRetrieveException(String.format("Unexpected response code while fetching coin api:
%s",
                        response.code()));
            }

            return objectMapper.readValue(response.body().string(), CoinApiCryptoDto[].class);
        } catch (IOException e) {
            log.error("Error while parsing data: {}", e);
            throw new InternalException("Something went wrong while parsing data", e);
        }
    }
}
```

Файл CurrencyDataServiceImpl.java

```
package ua.bielskyi.tracker.service.impl;

import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import ua.bielskyi.tracker.dto.FetchedCurrencyDataDto;
import ua.bielskyi.tracker.dto.sources.CoinApiCryptoDto;
import ua.bielskyi.tracker.entity.CurrencyData;
import ua.bielskyi.tracker.exception.ApiRetrieveException;
import ua.bielskyi.tracker.model.Crypto;
import ua.bielskyi.tracker.model.CryptoReport;
import ua.bielskyi.tracker.model.ExchangeRatesReport;
import ua.bielskyi.tracker.model.Rate;
import ua.bielskyi.tracker.model.SourceItem;
import ua.bielskyi.tracker.model.TimeSeriesPeriod;
import ua.bielskyi.tracker.repository.CurrencyDataRepository;
import ua.bielskyi.tracker.service.CryptoService;
```

```
import ua.bielskyi.tracker.service.CurrencyDataService;
import ua.bielskyi.tracker.service.CurrencyExchangerService;
import ua.bielskyi.tracker.service.CurrencySourceFactory;
import ua.bielskyi.tracker.service.redis.CacheService;
import ua.bielskyi.tracker.utils.RedisUtils;
import ua.bielskyi.tracker.utils.Utils;

import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Optional;
import java.util.stream.Collectors;

import static java.util.Objects.nonNull;

@Service
@RequiredArgsConstructor
@Slf4j
public class CurrencyDataServiceImpl implements CurrencyDataService {

    private final CurrencyDataRepository currencyDataRepository;
    private final CurrencySourceFactory currencySourceFactory;
    private final CryptoService cryptoService;
    private final CacheService cacheService;

    @Override
    public ExchangeRatesReport getExchangeRatesForPeriodReport(LocalDateTime date) {
        Optional<ExchangeRatesReport> cacheReport = getDateReportFromCache(date);
        if (cacheReport.isPresent()) {
            return cacheReport.get();
        } else {
```

```
        ExchangeRatesReport report = generateExchangeRatesReport(date, Boolean.FALSE);
        if (report.getErrors().isEmpty()) {
          cacheService.set(RedisUtils.getRatesPeriodReportKey(Utils.getTimestamp(date)), report,
RedisUtils.ONE_HOUR_MILLIS);
        }
        return report;
      }
    }

    @Override
    public ExchangeRatesReport getExchangeRatesReport() {
      Optional<ExchangeRatesReport> cacheReport = getReportFromCache();
      if (cacheReport.isPresent()) {
        return cacheReport.get();
      } else {
        ExchangeRatesReport report = generateExchangeRatesReport(LocalDateTime.now(),
Boolean.TRUE);
        if (report.getErrors().isEmpty()) {
          cacheService.set(RedisUtils.RATES_REPORT_KEY, report,
RedisUtils.TWENTY_MINUTES_MILLIS);
        }
        return report;
      }
    }

    @Override
    public CryptoReport getCryptoReport(Crypto base, Crypto quote, TimeSeriesPeriod period) {
      Optional<CryptoReport> cacheReport = getCryptoReportFromCache(base, quote, period);
      if (cacheReport.isPresent()) {
        return cacheReport.get();
      } else {
        CryptoReport report = generateCryptoReport(base, quote, period);
        return report;
      }
    }

    @Override
```

```
@Transactional
@Scheduled(cron = "0 */20 * * * *") // Run every 20 minutes
public void importCurrencies() {
   log.info("Currency data import started at {}", LocalDateTime.now());

   for (CurrencyData.SourceName source : CurrencyData.SourceName.values()) {
      CurrencyExchangerService exchangerService = currencySourceFactory.findSource(source);

      List<CurrencyData> currencyData = null;

      try {
         currencyData = exchangerService.getCurrencyDataForThePeriod(LocalDateTime.now())
               .stream()
               .map(x -> buildCurrencyData(x, true))
               .collect(Collectors.toList());
      } catch (ApiRetrieveException e) {
         log.error("Error while fetching data {}", e);
      }

      if (currencyData != null && !currencyData.isEmpty()) {
         currencyDataRepository.updateActualToFalseBySource(source.getValue());
         currencyDataRepository.saveAll(currencyData);

         cacheService.delete(Collections.singletonList(RedisUtils.RATES_REPORT_KEY));
      } else {
         log.debug("Currency data for source {} unavailable", source);
      }
   }

   log.info("Currency data import finished at {}", LocalDateTime.now());
}

private Optional<ExchangeRatesReport> getReportFromCache() {
   return cacheService.get(RedisUtils.RATES_REPORT_KEY, ExchangeRatesReport.class);
}
```

```
    private Optional<CryptoReport> getCryptoReportFromCache(Crypto base, Crypto quote,
TimeSeriesPeriod period) {
        return cacheService.get(RedisUtils.getCryptoRatesPeriodReportKey(base.name(),
                quote.name(), period.getPeriodIdentifier()), CryptoReport.class);
    }

    private Optional<ExchangeRatesReport> getDateReportFromCache(LocalDateTime date) {
        return cacheService.get(RedisUtils.getRatesPeriodReportKey(Utils.getTimestamp(date)),
ExchangeRatesReport.class);
    }

    private ExchangeRatesReport generateExchangeRatesReport(LocalDateTime date, boolean actual) {
        List<CurrencyData> reportData = new ArrayList<>();
        List<ExchangeRatesReport.Error> errors = new ArrayList<>();

        for (CurrencyData.SourceName sourceName : CurrencyData.SourceName.values()) {
            List<CurrencyData> currencyData = fetchCurrencyData(sourceName, date, actual, errors);
            reportData.addAll(currencyData);
        }

        ExchangeRatesReport exchangeRatesReport =
                buildExchangeRatesReportFromCurrencyDataList(reportData,
OpenDataServiceImpl.UAH_CODE, date);

        exchangeRatesReport.setErrors(errors);

        return exchangeRatesReport;
    }

    private CryptoReport generateCryptoReport(Crypto base, Crypto quote, TimeSeriesPeriod period) {
        CoinApiCryptoDto[] cryptos = cryptoService.fetchCryptoData(base, quote, period);
        CryptoReport report = new CryptoReport(Arrays.asList(cryptos));
        if (nonNull(report.getCryptos()) && !report.getCryptos().isEmpty()) {
            cacheService.set(RedisUtils.CRYPTO_PERIOD_REPORT_KEY, report,
RedisUtils.FIVE_HOURS_MILLIS);
        }
```

```
        return report;
    }

    private List<CurrencyData> fetchCurrencyData(CurrencyData.SourceName sourceName,
                                                 LocalDateTime date,
                                                 boolean actual,
                                                 List<ExchangeRatesReport.Error> errors) {
        List<CurrencyData> currencyData;
        LocalDateTime startOfTheDay = date.toLocalDate().atStartOfDay();

        if (actual) {
            currencyData =
currencyDataRepository.findCurrencyDataByTimestampAndActual(sourceName.getValue(),
                    Boolean.TRUE,
                    Utils.getTimestamp(startOfTheDay),
                    Utils.getTimestamp(startOfTheDay.plusDays(1)));
        } else {
            currencyData = currencyDataRepository.findCurrencyDataByTimestamp(sourceName.getValue(),
                    Utils.getTimestamp(startOfTheDay),
                    Utils.getTimestamp(startOfTheDay.plusDays(1)));
        }

        if (currencyData.isEmpty()) {
            CurrencyExchangerService exchangerService = currencySourceFactory.findSource(sourceName);
            try {
                List<CurrencyData> currencyDataForTheDate =
exchangerService.getCurrencyDataForThePeriod(date).stream()
                        .map(x -> buildCurrencyData(x, actual))
                        .collect(Collectors.toList());
                if (!currencyDataForTheDate.isEmpty()) {
                    currencyDataRepository.saveAll(currencyDataForTheDate);
                    currencyData = currencyDataForTheDate;
                }
            } catch (ApiRetrieveException e) {
                log.error("Unexpected response from source: {}", sourceName);
                errors.add(ExchangeRatesReport.Error.builder()
```

```
                .sourceName(sourceName)
                .message(String.format("Unexpected response from source: %s", sourceName))
                .build());
        }
    }

    return currencyData;
  }

  private ExchangeRatesReport buildExchangeRatesReportFromCurrencyDataList(List<CurrencyData>
currencyDataList,
                                                                           String baseCurrency,
                                                                           LocalDateTime date) {
    Map<String, List<CurrencyData>> currencyInfoList = new HashMap<>();
    Map<String, Rate> ratesMap = new HashMap<>();
    currencyDataList.forEach(x -> currencyInfoList
        .computeIfAbsent(x.getCurrencyCode(), k -> new ArrayList<>()).add(x));

    currencyInfoList.forEach((x, y) -> {
      ratesMap.put(x, new Rate());

      for (CurrencyData currencyData : y) {
        if (currencyData.getBaseCurrencyCode().equals(baseCurrency)) {
          SourceItem sourceItem = new SourceItem();

sourceItem.setSourceName(CurrencyData.SourceName.getSourceName(currencyData.getSource()));
          sourceItem.setRate(currencyData.getRate());
          ratesMap.get(x).getSources().add(sourceItem);
        }
      }
    });

    ratesMap.forEach((x, y) -> {
      Float averageRate = 0F;
      for (SourceItem sourceItem : y.getSources()) {
        averageRate += sourceItem.getRate();
```

```
            }
            averageRate = averageRate / y.getSources().size();
            ratesMap.get(x).setAverageRate(averageRate);
        });

        return ExchangeRatesReport.builder()
                .baseCurrency(baseCurrency)
                .ratesMap(ratesMap)
                .date(date)
                .build();
    }

    private CurrencyData buildCurrencyData(FetchedCurrencyDataDto fetchedCurrencyDataDto, boolean
actual) {
        CurrencyData currencyData = new CurrencyData();
        currencyData.setCurrencyCode(fetchedCurrencyDataDto.getCurrencyCode());
        currencyData.setBaseCurrencyCode(fetchedCurrencyDataDto.getBaseCurrencyCode());
        currencyData.setRate(fetchedCurrencyDataDto.getRate());
        currencyData.setDate(fetchedCurrencyDataDto.getDate());
        currencyData.setSource(fetchedCurrencyDataDto.getSource());
        currencyData.setSellRate(fetchedCurrencyDataDto.getSellRate());
        currencyData.setBuyRate(fetchedCurrencyDataDto.getBuyRate());
        currencyData.setActual(actual);
        return currencyData;
    }
}
```

Файл CurrencyRestController.java

```
package ua.bielskyi.tracker.controller;

import lombok.RequiredArgsConstructor;
import org.springframework.format.annotation.DateTimeFormat;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
```

```
import ua.bielskyi.tracker.model.Crypto;
import ua.bielskyi.tracker.model.CryptoReport;
import ua.bielskyi.tracker.model.ExchangeRatesReport;
import ua.bielskyi.tracker.model.TimeSeriesPeriod;
import ua.bielskyi.tracker.service.CurrencyDataService;
import ua.bielskyi.tracker.utils.Utils;

import java.util.Date;

import static ua.bielskyi.tracker.utils.Utils.convertToLocalDateTime;

@RestController
@RequestMapping(value = "/api/currencies/")
@RequiredArgsConstructor
public class CurrencyRestController {

   private final CurrencyDataService currencyDataService;

   @GetMapping(value = "/actual")
   public ResponseEntity<ExchangeRatesReport> getExchangeRatesReport() {
      ExchangeRatesReport ratesReport = currencyDataService.getExchangeRatesReport();
      return ResponseEntity.ok(ratesReport);
   }

   @GetMapping(value = "/period")
   public ResponseEntity<ExchangeRatesReport> getExchangeRatesForPeriodReport(
         @DateTimeFormat(pattern = Utils.DEFAULT_DATE_PATTER)
         @RequestParam(value = "date") Date date) {
      ExchangeRatesReport periodReport = currencyDataService
            .getExchangeRatesForPeriodReport(convertToLocalDateTime(date));
      return ResponseEntity.ok(periodReport);
   }

   @GetMapping(value = "/crypto")
   public ResponseEntity<CryptoReport> getCryptoReport(
```

```
        @RequestParam(value = "base") Crypto base,
        @RequestParam(value = "quote") Crypto quote,
        @RequestParam(value = "period") TimeSeriesPeriod period) {
    CryptoReport cryptoReport = currencyDataService.getCryptoReport(base, quote, period);
    return ResponseEntity.ok(cryptoReport);
  }
}
```

Файл application.properties

```
# DataSource Configuration
spring.datasource.url=jdbc:postgresql://localhost:5432/currency_db
spring.datasource.driverClassName=org.postgresql.Driver
spring.datasource.username=postgres
spring.datasource.password=fury

# Spring Configuration
spring.mvc.pathmatch.matching-strategy = ANT_PATH_MATCHER

# JPA Configuration
spring.jpa.hibernate.ddl-auto=update

# Redis Configuration
spring.data.redis.host=localhost
spring.data.redis.port=6379
spring.data.redis.password=

# Currency-system Configuration
ua.bielskyi.coin.market.mono.api=https://api.monobank.ua/bank/currency
ua.bielskyi.coin.market.privat.api=https://api.privatbank.ua/p24api/exchange_rates
ua.bielskyi.coin.market.open_data.api=https://bank.gov.ua/NBUStatService/v1/statdirectory/exchange

ua.bielskyi.coin.api.key=
ua.bielskyi.coin.api.url=https://rest.coinapi.io

spring.application.name=crypto-tracker
```

ДОДАТОК В. ПРЕЗЕНТАЦІЯ

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи:** аналіз особливостей розробки та практичної реалізації мікросервісу аналізу криптовалют.
- **Об’єкт дослідження:** проектування автоматизованого мікросервісу аналізу криптовалют**.**
- **Предмет дослідження:** програмне забезпечення для аналізу криптовалют.

2

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Проаналізувати теоретичні дослідження в сфері аналізу криптовалют.
2. Визначити переваги та можливості сучасних програмних засобів.
3. Дослідити інструменти та підходи для розробки мікросервісу.
4. Спроектувати та розробити застосунок для автоматизації процесу аналізу криптовалютної пари та фіатних валют.

3

Аналіз конкурентів

Функції Сервісу	Переваги	Недоліки
Moralis Money	- Зручний та зрозумілий інтерфейс.	- Обмежений вибір криптовалют у порівнянні з іншими сервісами.
	- Підтримка мульти-ланцюгів.	
TradingView	- Можливість створення та налаштування діаграм за потребами користувача.	- Відсутність автоматизованих засобів для відслідковування індивідуальних токенів.
	- Соціальна складова для обміну ідеями та стратегіями.	
Cointracker	- Автоматичні портфоліо трекери для відслідковування ринкової цінності.	- Обмежений функціонал для аналізу та передбачення цінових змін.
	- Податковий звіт для спрощення процесу податкового обліку.	
CryptoPanic	- Зручні налаштування оповіщень для відстеження ключових слів та монет.	- Недостатнє покриття джерел інформації порівняно з іншими агрегаторами новин.
	- Багатоджерельність джерел інформації (форуми, блоги, соціальні мережі).	

4

ВИМОГИ ДО ДОДАТКУ

1. Мати точку доступу для отримання інформацію про актуальний стан курсів фіатних валют з 3 різних джерел: ПриватБанк, Монобанк, Open data
2. Мати точку доступу для отримання інформації про актуальний стан курсів фіатних валют на конкретну дату
3. Мати точку доступу для отримання інформації про курс вибраної криптовалюти за останній час
4. Регулярно оновлювати та зберігати інформацію про стан курсів валют
5. Мати WEB інтерфейс який в свою чергу повинен мати функціонал відображення графіків
6. Авторизація нового користувача.
7. Зручний механізм управління користувачами: створення, видалення, обмеження прав доступу
8. Бути спроможним підключити Telegram бота щоб отримувати інформацію про актуальний стан курсу валют

5

Загальні відомості про фіатні та криптовалюти

Рис 1.1 Фіатна валюта

Рис 1.2 Криптовалюта

6

Монолітна архітектура

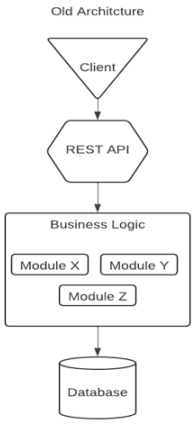


Рис 1.3 Приклад монолітної архітектури

7

Мікросервісна архітектура

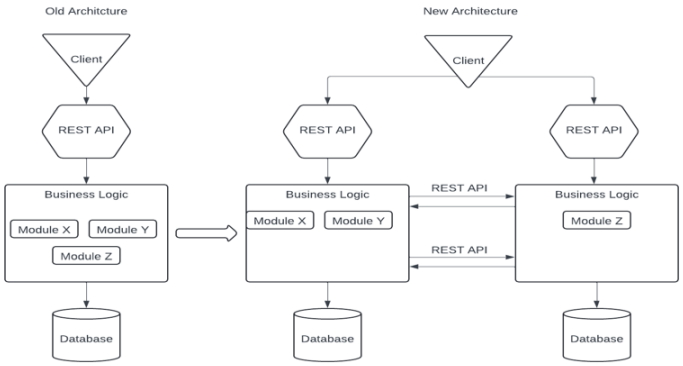


Рис 1.4 Приклад мікросервісної архітектури

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

Рис 1.5 Мова програмування Java

Рис 1.6 Фреймворк Spring

Рис 1.7 База даних Postgres

Рис 1.8 Фреймворк Jmix

Рис 1.9 База даних Redis

Схема роботи додатку

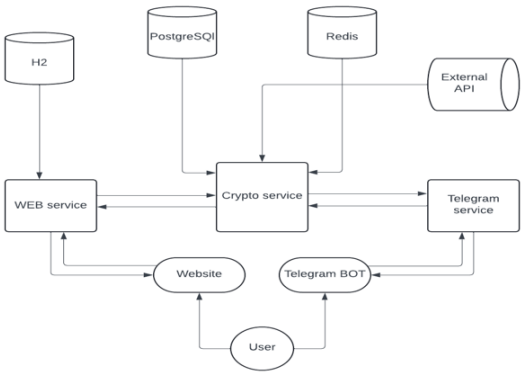


Рис 1.10 Загальна схема додатку

Сервіси для отримання інформації про стан курсів.

Рис 1.11 Monobank API

Рис 1.12 Privatbank API

Рис 1.14 Coin API

Рис 1.13 OpenData API

Схема бази.

Column Name	Data Type	Nullability	Description
actual	BOOLEAN	NULL	Indicates if the currency data is currently active or valid.
buy_rate	FLOAT4	NULL	The rate at which the currency can be bought.
rate	FLOAT4	NULL	The general rate or value of the currency.
sell_rate	FLOAT4	NULL	The rate at which the currency can be sold.
source	INT4	NULL	The source identifier of the currency data.
created_at	INT8	NOT NULL	The timestamp (in epoch format) when the record was created.
date	INT8	NULL	The date (in epoch format) associated with the currency data.
id	BIGSERIAL	NOT NULL	The unique identifier for each currency record.
base_currency_code	VARCHAR(255)	NULL	The base currency code (e.g., USD, EUR).
currency_code	VARCHAR(255)	NULL	The target currency code (e.g., GBP, JPY).

Рис 1.15 Таблиця currencies

Екранні форми.

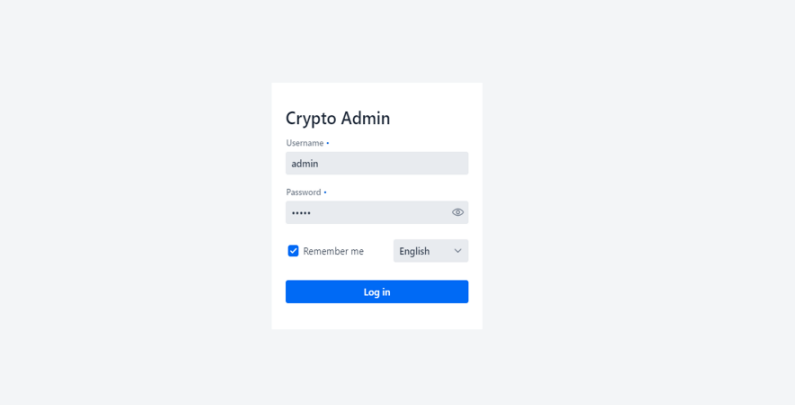


Рис 1.16 Авторизація користувача

Екранні форми.

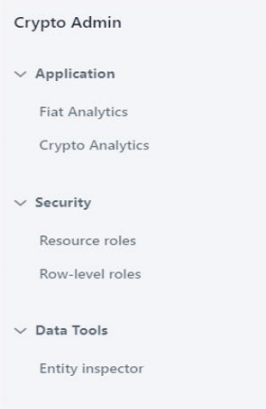


Рис 1.17 Меню додатку

Екранні форми.

Рис 1.18 Аналіз криптовалютної пари

15

Екранні форми.

Рис 1.19 Фіатних валют

16

Висновки

Було розроблено сильний інструмент який допоможе досвідченим трейдерам та початківцям проводити якісний аналіз криптовалютних трендів та знаходити зміни в курсі фіатних валют.

Простий інтерфейс допомагає на 50% швидше почати працювати з графіками.

Багато інтеграцій дає можливість відслідковувати зміни у курсі фіатних валют з 3 різних джерел

Інтеграція з телеграмом дає можливість тримати в курсі користувача про зміни курсів валют у найкоротші терміни

17

ДЯКУЮ ЗА УВАГУ!