АНОТАЦІЯ

В ході виконання курсової роботи було створено інформаційну систему, яка дає можливість відслідковувати в реальному часі курс крипто валют на різних торгівельних майданчиках (біржах). Система складається з двох компонентів: Telegram боту для взаємодії з користувачем, та API, яке використовується ботом для отримання необхідної інформації.

Під час створення інформаційної системи було розв’язанно наступні технічні проблеми:

- Можливість створення декількох ботів у різних системах миттєвого обміну повідомленнями була розв’язана за допомогою розділення системи на два компоненти (бот та API);
- Зменшення навантаження на API та пришвидшення роботи боту було досягнуто завдяки використанню кешування;
- Взаємодія між ботом та API була розв’язанна за допомогою використання архітектурного стилю взаємодії REST;

Для створення системи були використані такі технології: фреймворк .NET Core, web-фреймворк ASP .NET Core 2.1 та СУБД PostgreSQL.

ЗМІСТ

ВСТУП

На сьогоднішній день різні види цифрової валюти(далі крипто валюти) набирають стрімку популярність. Вони забезпечують прозору альтернативу фіатним грошам, а також дозволяють з легкістю проводити операції переводу, конвертації та зберігання валюти. В основі будь-якої валюти лежить так званий блокчейн, який відповідає за зберігання даних про транзакції користувачів. Стрімкий ріст популярності привів до того, що нові крипто валюти з’являються та зникають щодня. Також крипто валюти можна обмінювати на декількох торгових майданчиках, тому існує гостра потреба в інформаційному засобі для спостерігання за курсом різних валют на різних майданчиках.

Метою курсової роботи є розробка власного програмного продукту для спостерігання змін курсів крипто валют на різних торгових майданчиках. На початку курсової роботи було досліджено декілька існуючих рішень для спостерігання курсу крипто валют. В результаті дослідження були виявлені наступні риси гарного програмного продукту: простий і зрозумілий інтерфейс, крос-платформеність та відсутність потреби у встановлені додаткового програмного забезпечення на пристрій кінцевого користувача.

Загалом, розгляд доступного програмного забезпечення для спостеріганням за курсом крипто валют виявив головний недолік існуючих систем: всі вони прив’язані до певного торгівельного майданчику, тому основною метою цієї курсової роботи є створення сервісу для агрегації даних про курси крипто валют з існуючих торгівельних майданчиків. І сповіщення кінцевого користувача.

При створенні програми були використані основні принципи ООП: поліморфізм, інкапсуляція та наслідування – для полегшення процесу проектування програми та створення її функціоналу.

2 ТЕХНІЧНЕ ЗАВДАННЯ

Метою цієї курсової роботи є створення програмного рішення для моніторингу курсу крипто валют на різних торгівельних майданчиках(біржах). Користувач повинен мати можливість отримувати оновлення в реальному часі. Рішення має бути легко масштабованим. Для сповіщення користувачів рішення має використати систему миттєвого обміну повідомленнями “Telegram”. До того ж рішення має бути розрахованим на розробку та роботи з відсутністю початкового бюджету.

Рішення має надавати запроваджувати наступний функціонал для користувача:

- Отримання поточного курсу крипто валюти без прив’язки до певного торгівельного майданчику;
- Отримання поточного курсу крипто валюти для певного майданчику;
- Підписуватись на оновлення курсу певної крипто валюти з прив’язкою до певного торгівельного майданчика;
- Отримувати оновлення курсу крипто валют згідно з підписками у реальному часі;

Одним з головних обмежень, накладених на систему є відсутність бюджету, що унеможливлює використання багатьох технологій.

Виходячи з наведених до програмного рішення вимог, було розроблено декілька варіантів архітектури системи. Розглянемо перший з них:

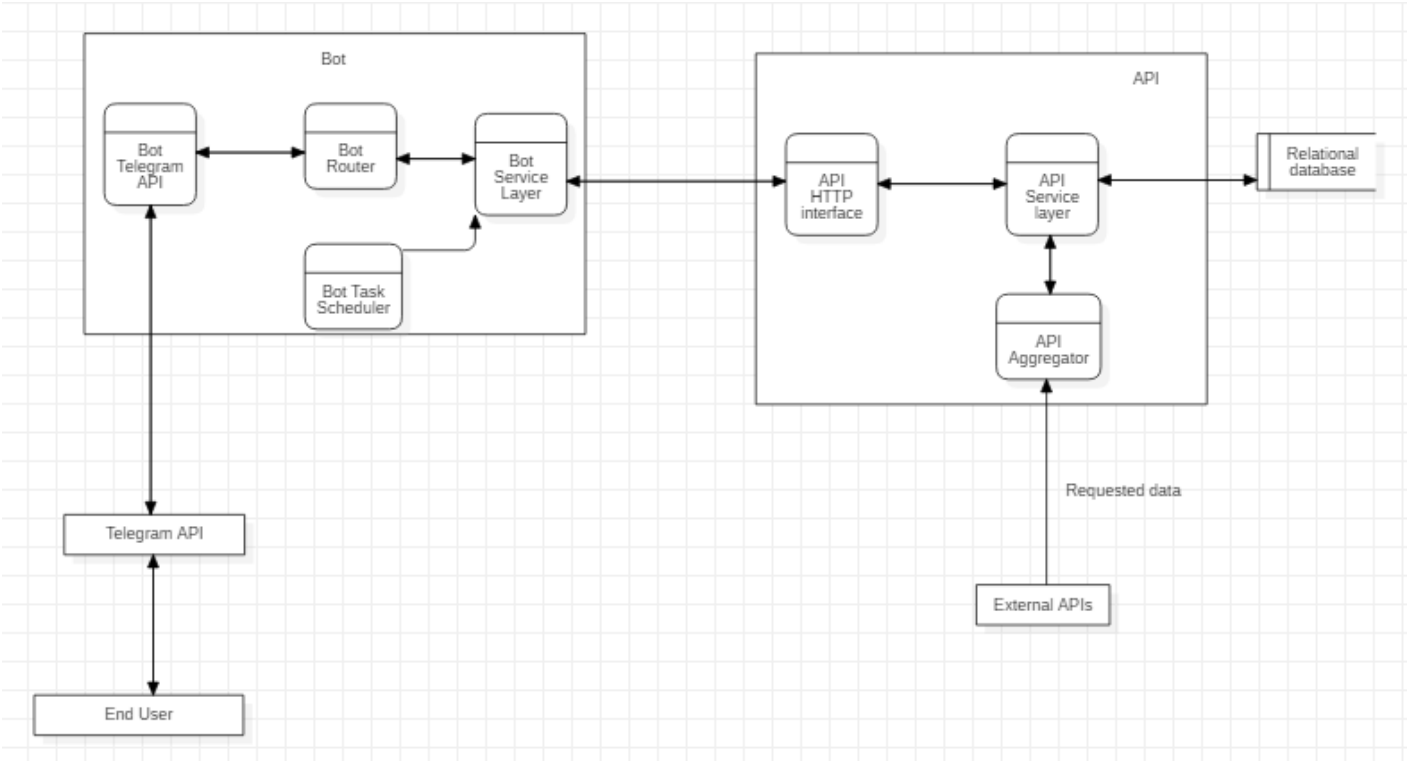

Рисунок 2.1 – Перший варінт архітектури системи

У цьому варіанті системи можна виділити такі компоненти:

1. End User – кінцевий користувач;
2. Telegram API – серевери телеграму, посередники між ботом та користувачем;
3. Бот – телеграм-бот. Обробляє запити від користувачів та передає запит далі на обробку API;
4. API – оообрбляє запити від боту. Необхідне для розширення у майбутньому.
5. External API – АПІ торгівельних майданчиків (бірж). З нього запитуються актуальні данні про курси валют;
6. Relational Database – база даних додатк;

Більш детальної уваги заслуговують Бот(3) та API(4), оскільки вони самі складаються з кількох підсистем.

Розглянемо детальніше архітектуру бота:

1. Bot Telegram API – посередник між ботом і TelegramAPI. У цьому компоненті моделі бізне логіки перетворюються до виду, необхідного для обробки серверами Telegram. Також він перетворює вхідні повідомлення до команд, які можить бути виконані маршрутизатором
2. Bot Router – маршрутизатор команд.За допомогою регулярних виразів вінпарсить команди та отримує аргументи, які потім предаються до сервісного шару.
3. Bot Task Scheduler – планувальник задач. Використовується для оновлення курсу криптовалют.
4. BotServiceLayer – шар сервісів. Взаємодіє з АПІ та перетворює результати запитів до АПІ на об’єкти бізнес логіки.

Тепер розглянемо детальніше структуру API:

1. API HTTP interface – http інтерфейс API. Передає забезпечує передачу даних між сервісним шаром та зовнішніми ресурсами.
2. API Service layer – шар сервісів. Взаємодіє з базою даних та представляє бізнес-логіку проектованої системи.
3. API Aggregator – шар агрегації зовнішніх API. Виконує запити до зовнішніх ресурсів та перетворює результат на моделі бізнес-логіки.

Другий варіант архітектури дуже схожий на перший, та є більш універсальним. Але його реалізація вимагає реалізацій паттерну SAGA для синхронізації та оркестрації між сервісами, тому його реалізацію варто відкласти до необхідності розширення системи. Використання цієї архітектури підвищить надійність системи, додавши можливість оркеструвати взаємодію сервісів та проводити розподілені транзакції. Схема цієї архітектури наведена на рисунку 2.2.

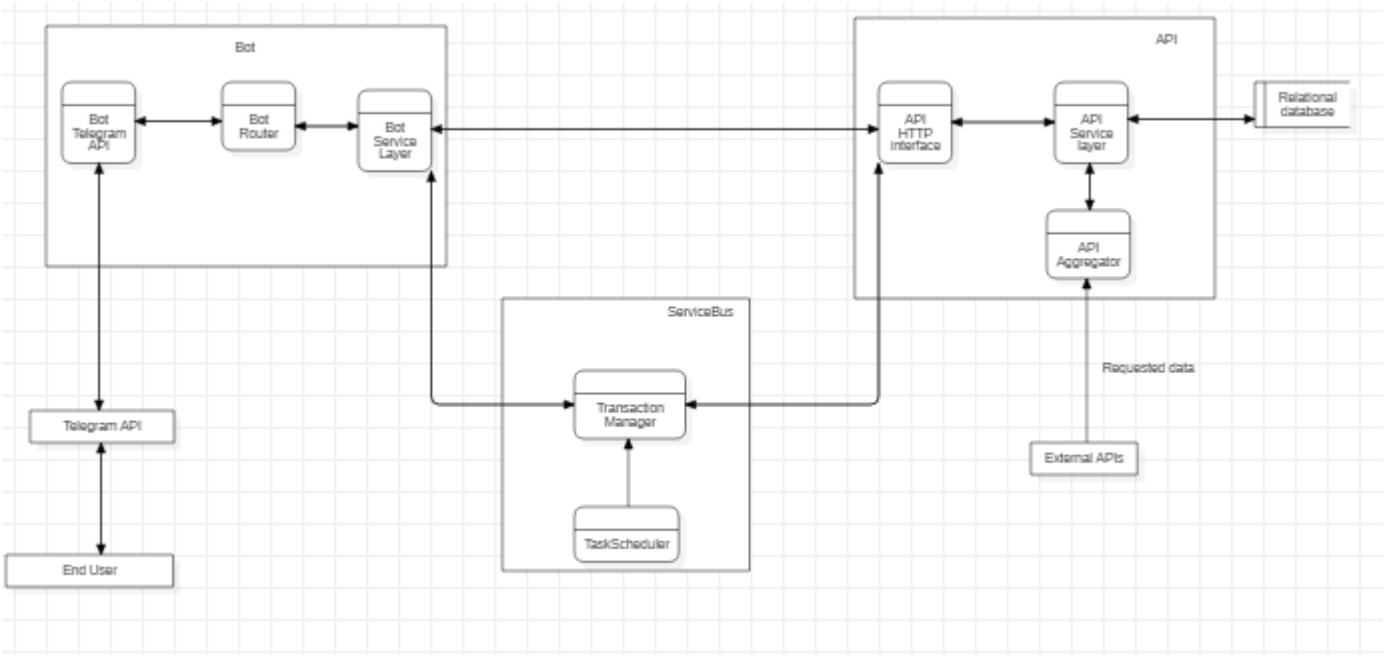

Рисунок 2.2 – Другий варіант архітектури додатку

Ця архітектура досить схожа на перший варіант. Компоненти 1,2,4,5,6 залишаються без змін у архітектурі. Додається новий компонент для оркестрації та синхронизації боту та АПІ. Він складається з TransactionManager-у, який дозваляє проводити транзакції, що розподілені між декількома сервісами та TaskScheduler-у, який замінює аналогічний компонент у боті(3).

Оскільки у рамках курсової роботи планується створити бот лише для Telegram, то варто використати перший варіант архітектури, який зображено на рисунку 2.1. Такий вибір обумовлено складністю реалізації паттерну SAGA та значним підвищенням загальної складності системи.

3 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ТА ОПИС МОДЕЛІ

3.1 Огляд та аналіз існуючих рішень

В ході аналізу існуючих рішень, були виявлені умовно безкоштовні аналоги реалізації функціоналу програмного рішення в мережі Інтернет. Нажаль, ці рішення, хоч і мали багато функціональних можливостей, були зосередженні на одному або декількох торгівельних майданчиках. Основною метою таких рішень є полегшення доступу користувача до особистого кабінету таких майданчиків.

З оглядом на важливість миттєвого сповіщення користувачів та дуже лімітований(або зовсім відсутній) бюджет системи, було обрано найбільш підходящий варіант – бот у системі миттєвого обміну повідомленнями. Такий підхід дає велику кількість переваг:

- зручність інтерфейсу забезпечується програмою для обміну повідомлень;
- гнучкість у системних вимогах;
- відсутність необхідності у встановленні додаткового програмного забезпечення на пристрій користувача;
- простота у користуванні;
- відсутність додаткових витрат на розробку або використання існуючої системи миттєвих сповіщень;
- Можливість створення ботів для різних систем обміну повідомленнями, не змінюючи серверну частину;

В якості кінцевої платформи для миттєвого обміну повідомленнями було обрано відносно нову – Telegram. Для знаходження найважливіших рис у телеграм боті було розглянуто декілька існуючих рішень. Нажаль, рішень з подібним функціоналом в ході пошуку не було знайдено, тому для аналізу було обрано декілька ботів з непов’язанним з крипто валютами функціоналом.

3.1.1 Бот для розкладу роботи терапевту “SobkoBot”

Цей бот створений для студентів декількох факультетів КПІ, у яких штатним терапевтом є Собко І.І. . Цей бот дозволяє дізнатись розклад прийому терапевту на сьогодні. Також він повідомляє загальний розклад роботи та телефони реєстратури. Бот містить лише одну команду та одну відповідь, але така простота досить приваблива і дозволяє боту легко доносити необхідну інформацію до користувача, і не потребує часу для того, щоб навитись їм користуватись. Проаналізувавши рисунок 3.1.1, на якому зображено графічний інтерфейс боту, можемо винести головні переваги цього боту:

- Простота інтерфейсу та використання;
- Немає потреби запам’ятовувати багато команд;
- Вся важлива інформація виділена за допомогою різних способів форматування шрифтів;

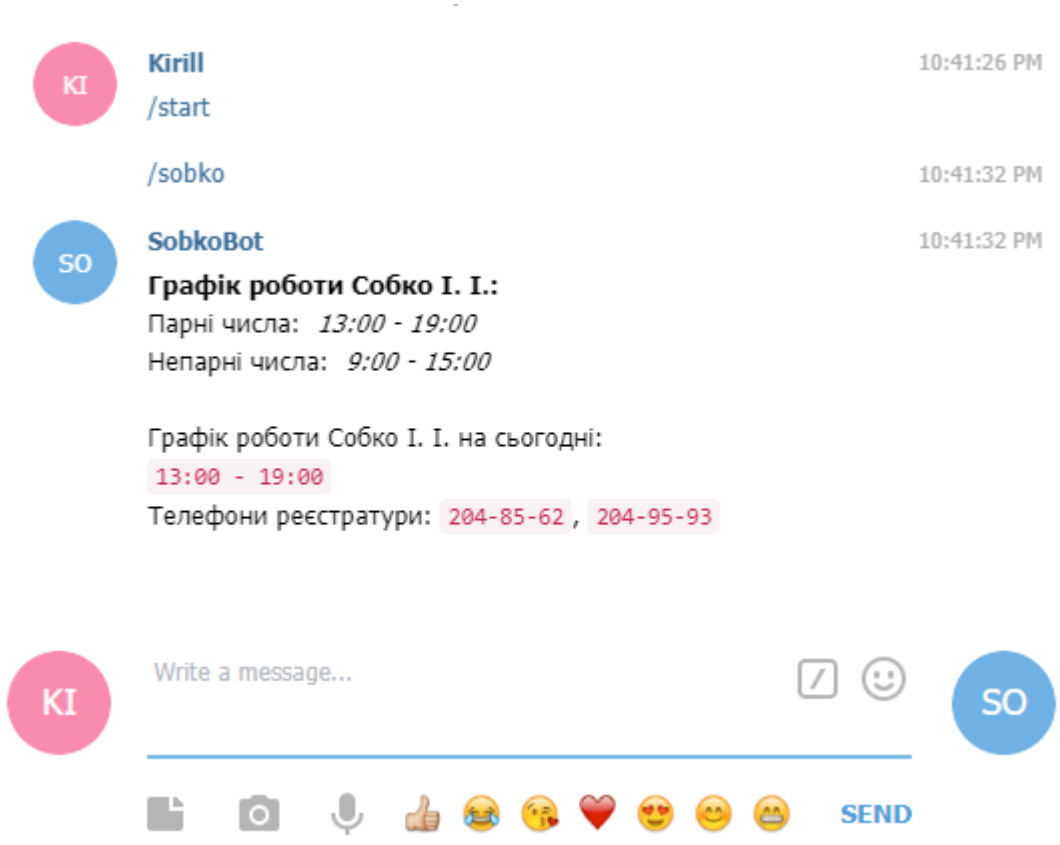

Рисунок 3.1.1 – графічний інтерфейс “SobkoBot”

3.1.2 Бот онлайн-видавництва Meduza “Meduzaprobot”

Цей бот було спроектовано для користування для зручного користування інформаційним ресурсом “Meduza”. При відкритті сторінки боти відображається основна інформація про призначення боту, як зображено на рисунку 3.1.2. Це є безперечною перевагою.

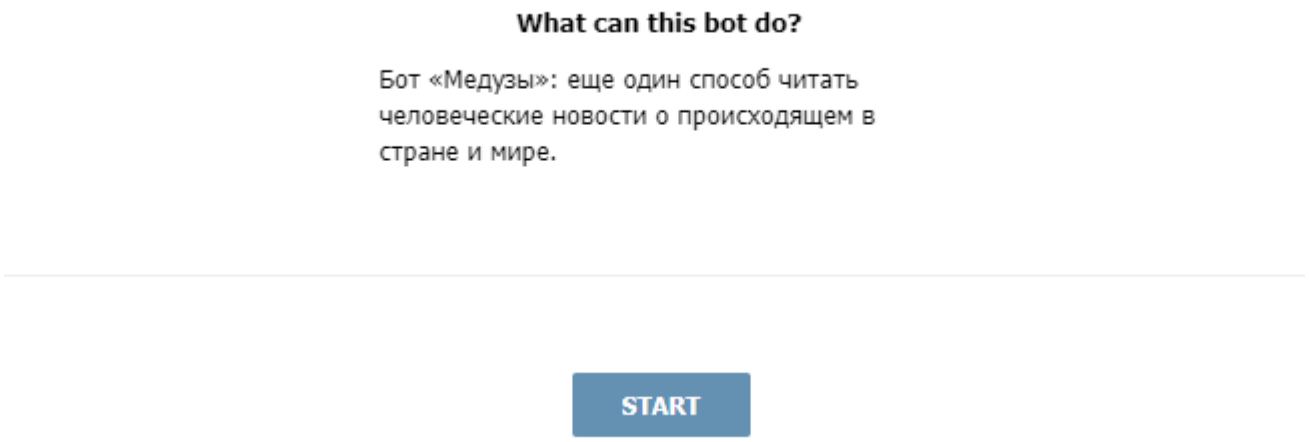

Рисунок 3.1.2 – Інформаційний екран “Meduzaprobot”

Після початку роботи з ботом стає доступним перелік команд боту, що є дуже зручно для користувача, але з годом цей список підніметься у переліку повідомлень, і щоб подивитись команди доведеться шукати перше повідомлення. Нажаль, більшість функцій боту не працювала, тому аналіз було завершено на початковому переліку команд, який зображено на рисунку 3.1.3

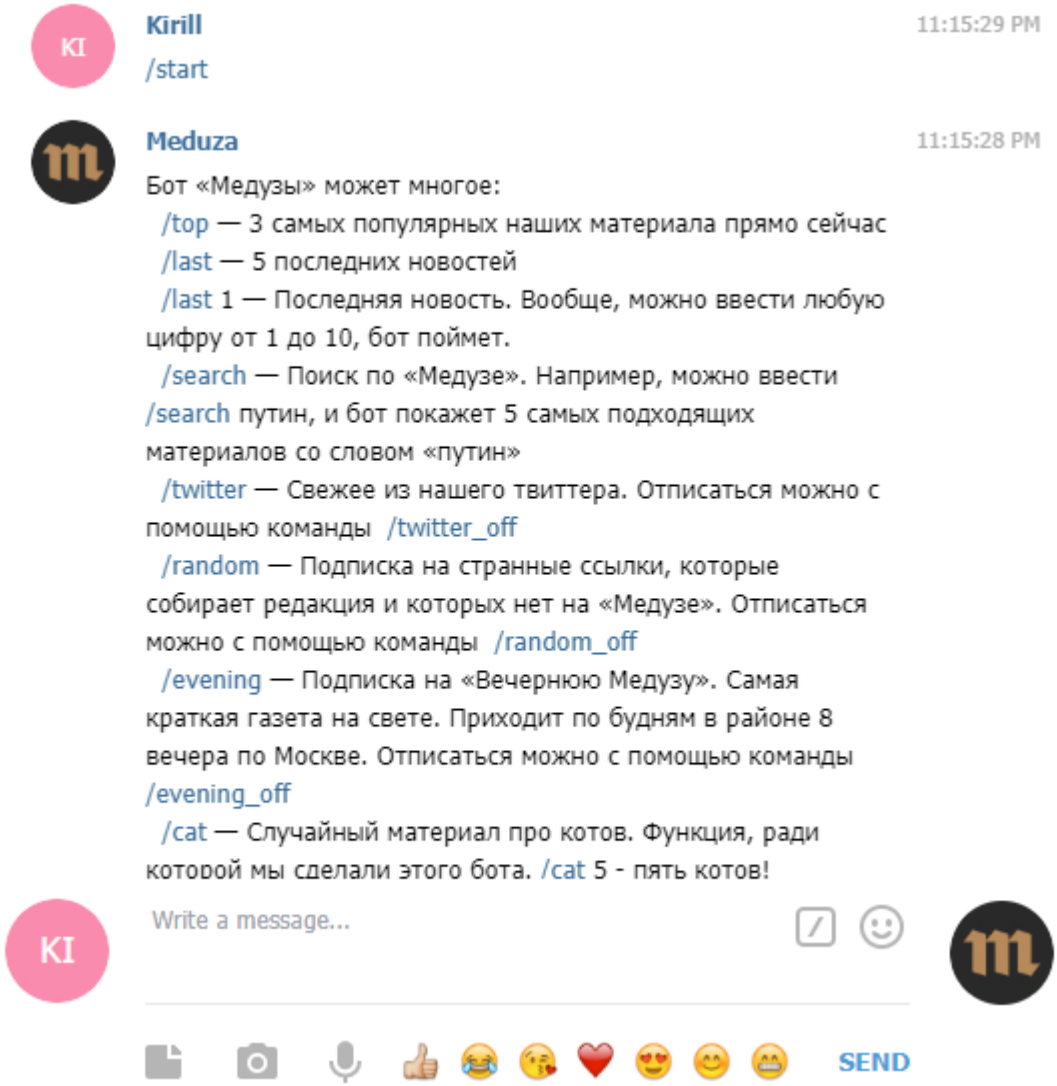

Рисунок 3.1.3 – Перелік команд у “Meduzaprobot”

3.1.3 Бот прогнозу погоди “Weatherman_bot”

Досить популярний бот для отримання актуального прогнозу погоди. Як і у випадку з “Meduzaprobot”, при першому відкритті відображається загальна інформація про бота, як зображено на рисунку 3.1.4. Нажаль, бот виявився не працездатним, і не відповів на жодне надіслане повідомлення. Але в цього боту вдалось знайти декілька переваг: по-

перше, відображається список команд, а по друге цих команд небагато, що дозволяє їх легко запам’ятати.

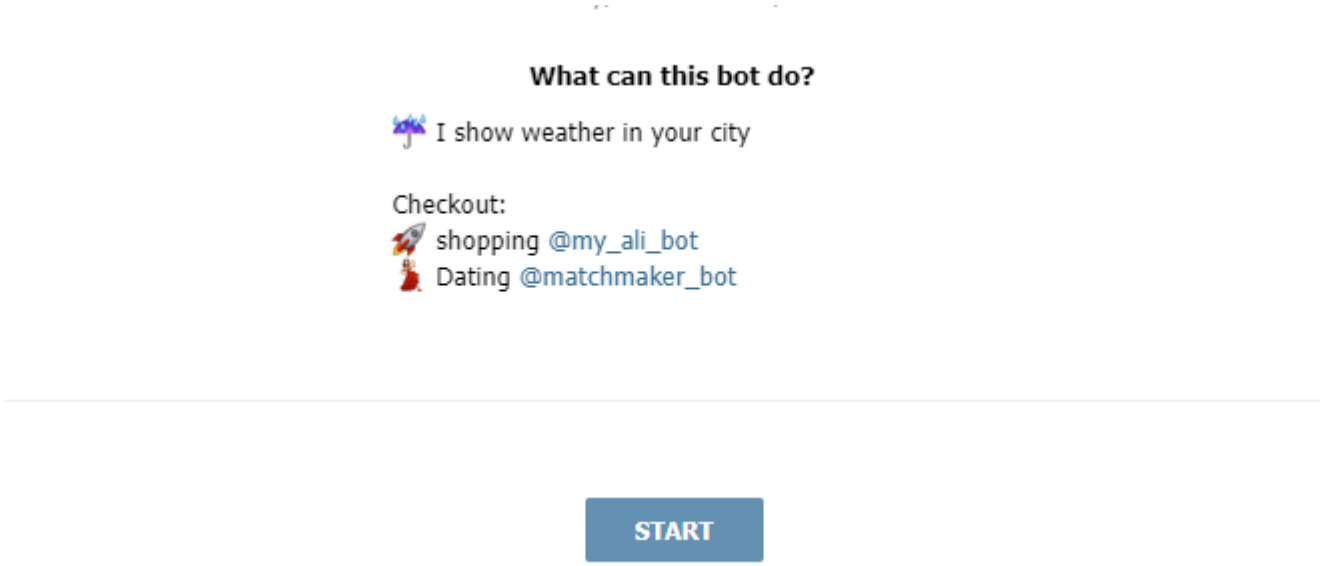

Рисунок 3.1.4 – Інформаційна сторінка “Weatherman_bot”

3.1.4 Порівняння існуючих рішень

В ході аналізу існуючих рішень, було виявлено їх переваги та недоліки. У таблиці 3.1.1 наведено їх порівняння.

Таблиця 3.1.1 – Порівняння існуючих рішень

Характеристика	“SobkoBot”	“Meduzaprobot”	“Weatherman_bot”
Велика кількість команд	-	+	-
Автодоповнення команд	+	+	+
Перелік всіх команд боту	-	+	-
Інформаційний екран	-	+	+

Альтернативні способи роботи з ботом	-	-	-
Повна працездатність	+	-	-
Наявність аватару	-	+	+

Проаналізувавши таблицю 3.1.1, можемо зробити декілька висновків:

- Наявність інформаційного екрану надає користувачам можливість дізнатись, що робить бот і чому варто його використовувати;
- Невелика кількість команд дозволить користувачу легше користуватись ботом;
- Якщо ж команд більше п’яти, то слід додати можливість вивести прелік всіх команд;
- Автодоповнення є обов’язковою умовою для зручності використання бота;
- Бажаним є використання екранних клавіатур, що полегшить процес роботи з ботом;
- Важливим є прецездатність боту. Всі команди повинні працювати, або хоча б повертати повідомлення про помилку. Команди, на які бот не реагує може спричинити враження, що бот взагалі не працю;
- Наявність аватару в бота робить його більш привабливим і більш помітним у боковій панелі;

3.2 Аналіз предметної області

Для аналізу предметної області скористаємося Entity-Relation моделлю. Entity-Relation модель – модель даних, що дозволяє описати концептуальні схеми предметної області, такі як моделі та зв’язки між ними для відображення в базі них. Ця модель відображається за допомогою простої і наочної Entity-Relation діаграми. Побудова цієї моделі дозволяє наочно зобразити предметну область, що полегшує подальше

проектування. На рисунку 2.1 зображена Entity-Relation модель для предметної області курсової роботи.

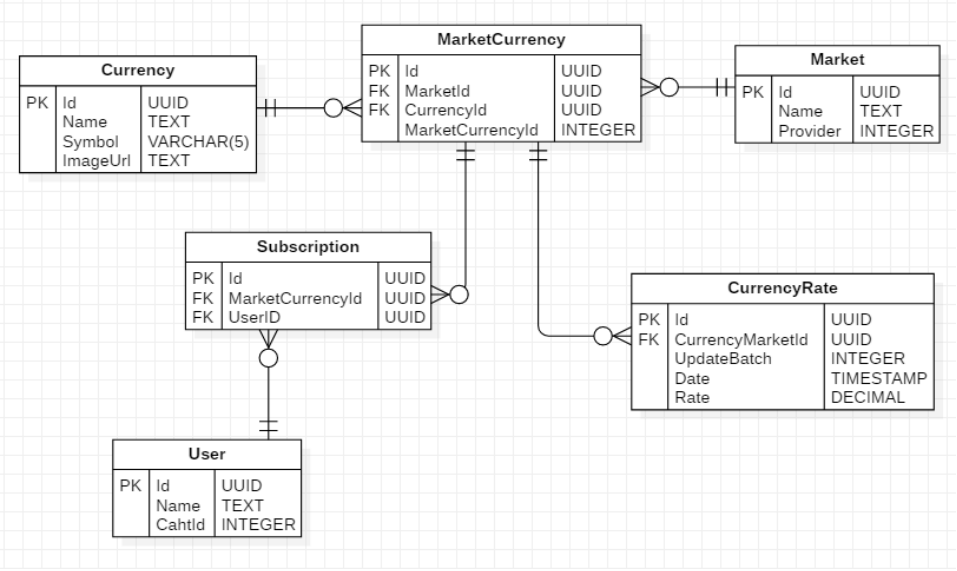

Рисунок 3.1 – Entity-Relation модель

В предметній області існують такі сутності:

- User – представлення користувача у системі;
- Currency – представлення крипто валюти у системі;
- Market – представлення торгівельного майданчику у системі;
- CurrencyRate – оновлення курсу певної крипто валюти на певному торгівельному майданчику**;**
- CurrencyMarket – розв’язочна сутність для відношення багато до багатьох між Currency та Market. Це важлива сутність, оскільки вона також вказує на id валюти на зовнішньому ресурсі(торгівельному майданчику);
- Subscription - розв’язочна сутність для відношення багато до багатьох між **User** та CurrencyMarke**t**. Цю сутність розглянуто, оскільки вона є важливою частиною бізнес логіки додатку(наприклад, не можна да рази підписатись на певну валюту на певному торгівельному майданчику).

Детальніше розглянемо атрибути кожної сутності:

Таблиця 3.1 – Список атрибутів сутності User

Назва атрибуту	*Тип атрибуту*	*Опис*
Id	UUID	Унікальний ідентифікатор
Name	TEXT	Ім’я користувача
ChatId	Integer	Ідентифікатор чату у Telegram

Таблиця 3.2 – Список атрибутів сутності Currency

Назва атрибуту	*Тип атрибуту*	*Опис*
Id	UUID	Унікальний ідентифікатор
Name	TEXT	Назва крипто валюти
Symbol	VARCHAR(5)	Коротка назва крипто валюти
ImageUrl	TEXT	Посилання на логотип валюти

Таблиця 3.3 – Список атрибутів сутності Market

Назва атрибуту	*Тип атрибуту*	*Опис*
Id	UUID	Унікальний ідентифікатор
Name	TEXT	Назва торгівельного майданчику
Provider	INTEGER	За допомогою цього поля визначається, який провайдер для агрегації буде використано

Таблиця 3.4 – Список атрибутів сутності CurrencyMarket

Назва атрибуту	*Тип атрибуту*	*Опис*
Id	UUID	Унікальний ідентифікатор
CurrencyId	UUID	Ідентифікатор валюти
MarketId	UUID	Ідентифікатор ринку

MarketCurrencyId	INTEGER	Ідентифікатор відповідної валюти у зовнішньому представленні торгівельному майданчику

Таблиця 3.5 – Список атрибутів сутності CurrencyRate

Назва атрибуту	*Тип атрибуту*	*Опис*
Id	UUID	Унікальний ідентифікатор
CurrencyMarketId	UUID	Ідентифікатор валюти
UpdateBatch	INTEGER	Номер агрегації валюти
Date	TIMESTAMP	Час проведення агрегації
Rate	DECIMAL	Курс валюти на час агрегації в USD

Таблиця 3.6 – Список атрибутів сутності Market

Назва атрибуту	*Тип атрибуту*	*Опис*
Id	UUID	Унікальний ідентифікатор
UserId	UUID	Ідентифікатор користувача
MarketCurrencyId	UUID	Ідентифікатор валюти на торгівельному майданчику (CurrencyMarket)

Тепер розглянемо відношення, які існують в предметній області:

- Відношення багато до багатьох між Currency та Market через розв’язочну сутність CurrencyMarket. Це відношення означає, що одна та сама валюта може бути на багатьох торгівельних майданчиках, та на одному торгівельному майданчику може бути представлено багато різних валют.
- Відношення багато до багатьох між User та CurrencyMarket через розв’язочну сутність Subscription. Це відношення означає, що один користувач може бути підписаний на декілька різних валют на декількох торгівельних майданчиках,

та за оновленням певної валюти на певному майданчику може слідкувати декілька користувачів.

- Відношення один до багатьох між CurrencyMarket та CurrencyRate. Це відношення означає, що курс певної валюти на певному торгівельному майданчику буде оновлюватись багато разів(при кожному агрегувані).

Підводячи підсумок аналізу предметної області, можна сказати, що побудована модель повністю задовольняє вимогам, поставленим до програмного продукту. Також ця модель є простою для розуміння і модифікації, і може бути представлена за допомогою засобів ORM Entity Framework Core. Але, при розширені можливостей додатку варто врахувати наступні нюанси:

По-перше, при додаванні нових кінцевих платформ(наприклад, Facebook), варто створити нову сутність UserProviderInfo, яка буде відповідати за представлення користувача на різних кінцевих платформах. Це слід зробити для того, щоб надати можливість користувача прив’язати до одного облікового запису декілька засобів миттєвого обміну повідомлення. Також слід додати зв’язок один до багатьох між **User** та UserInfoProvider.

По-друге, не всі торгівельні майданчики будуть використовувати послідовності цілих чисел для унікального ідентифікатору валюти. Можливим рішенням є використання типу RAW або VARCHAR для атрибуту MarketCurrencyId в сутності CurrencyMarket.

3.3 Аналіз існуючих технологій

Проаналізувавши предметну область, та врахувавши її специфіку, для побудови систему було обрано наступний стек технологій:

- .Net Core за зручність, наявність великої кількості бібліотек, фреймворків та крос-платформеність.;

- Об’єктно-реляційну систему управління базами даних PostgreSQL для збереження даних за безкоштовність для комерційного використання та гарну інтеграцію з EF core (на відміну від MySql);
- MVC web-фреймворк використано ASP .NET Core 2.1 для написання API для зменшення boilerplate-коду;
- ORM Entity Framework Core для підвищення рівня захищеності додатку та зменшення кількості boilerplate-коду;
- Декілька NuGet-пакетів, таких як Telegram.Bot та Quartz для зменшення кількості boilerplate-коду;
- Архітектурний стиль взаємодії компонентів REST для передачі даних між сервісами за простоту реалізації та надійність;
- Кешування для зменшення серверних затримок;

Розглянемо використані технології більш детально.

3.3.1 Фреймворк .Net Core

.NET Core - це безкоштовний крос-платформний фреймворк з керованим кодом підтримуваний на Windows, Linux і Mac OSX. На відміну від .NET Framework вихідний код .NET Core є повністю відкритим. Він містить CoreCLR - повністю крос-платформну реалізацію CLR, віртуальну машину, яка керує виконанням програм в .NET середовищі. CoreCLR поставляється з оптимізованим «just-in-time» компілятором RyuJIT. .NET Core також включає в себе CoreFX, яка представляє собою часткове відгалуження FCL (стандартна бібліотека класів .NET фреймворку). Реалізації усіх класів також відкриті.

У той час як .NET Core розділяє підмножину API .NET Framework, він містить також власний API, який не є частиною .NET Framework. Крім того .NET Core містить CoreRT, оптимізований під інтеграцію в AOT(компіляція перед виконанням) бінарні файли. Варіант бібліотеки .NET Core використовується для UWP (універсальна

платформа Windows). UWP платформа, створена Microsoft і вперше представлена в Windows 10. Метою даної платформи є допомога у створенні універсальних додатків Windows, що запускаються як на Windows 10, так і на Windows 10 Mobile без зміни в коді. Інтерфейс командного рядка .NET Core пропонує точку входу для операційних систем і надає послуги для розробників, такі як компіляція і пакети управління.

.NET Core підтримує чотири крос-платформних сценарії: ASP.NET Core веб-аплікації, консольні додатки, бібліотеки і UWP (універсальна платформа Windows) додатки. Він не реалізує Windows Forms або WPF, які створюють стандартний графічний інтерфейс для настільних ПК на Windows. .NET Core також модульна, а це означає, що замість збірок, розробники працюють з пакетами NuGet. На відміну від .NET Framework, який обслуговується за допомогою служби Windows Update, .NET Core залежить від його менеджера пакетів при отриманні оновлень.

3.3.2 MVC web-фреймворк ASP .NET core

ASP.NET Core - вільне та відкрите програмне забезпечення каркасу веб застосунків, з продуктивністю вищою ніж у ASP.NET, розроблене корпорацією Microsoft. Це модульна структура, яка працює як на повній платформі .NET Framework, так і на платформі .NET Core.

Фреймворк являє собою повний перепис, який об'єднує раніше окремі ASP.NET MVC та ASP.NET Web API у єдину програмувальну модель.

Не зважаючи на те, що це є новим фреймворком, побудованим на новому веб-стеку, ASP.NET Core має високу ступінь сумісності концепцій з ASP.NET MVC, який об'єднує функціональність MVC, Web API та Web Pages. В попередніх версіях платформи дані технології реалізовані окремо і тому містять багато дубльованої функціональності. Тепер це об’єднано в одну програмну модель ASP.NET Core MVC. Веб-форми повністю вийшли в минуле. Програми ASP.NET Core підтримують

програмні версії, в якій різні програми, що працюють на одному комп'ютері, можуть орієнтуватися на різні версії ASP.NET Core. Це не можливо з попередніми версіями ASP.NET Core.

3.3.3 СУБД PostgreSQL

PostgreSQL - об'єктно-реляційна система керування базами даних(СКБД). Є альтернативою як комерційним СКБД (Oracle Database, Microsoft SQL Server, IBM DB2 та інші), так і СКБД з відкритим кодом (MySQL, Firebird, SQLite).

Порівняно з іншими проектами з відкритим кодом, такими як Apache, FreeBSD або MySQL, PostgreSQL не контролюється якоюсь однією компанією, її розробка можлива завдяки співпраці багатьох людей та компаній, які хочуть використовувати цю СКБД та впроваджувати у неї найновіші досягнення.

Сервер PostgreSQL написаний на мові C. Зазвичай розповсюджується у вигляді набору текстових файлів із сирцевим кодом. Для інсталяції необхідно відкомпілювати файли на своєму комп'ютері і скопіювати в деякий каталог. Весь процес детально описаний в документації.

Головними перевагами PostgreSQL є:

- Функції

  Функції дозволяють виконувати деякий код безпосередньо сервером бази даних. Ці функції можуть бути написані на SQL, який має деякі примітивні програмні оператори, такі як галуження та цикли. Але гнучкішою буде функція написана на одній із мов програмування, з якими PostgreSQL може працювати. Функції можуть виконуватись із привілеями користувача, який її викликав, або із привілеями користувача, який її написав;

- Індекси

У PostgreSQL є підтримка індексів наступних типів: B-дерево, хеш, R-дерево, GiST, GIN. При необхідності можна створити нові типи індексів;

- Багатоверсійність (MVCC)

PostgreSQL підтримує одночасну модифікацію БД декількома користувачами за допомогою механізму Multiversion Concurrency Control (MVCC). Завдяки цьому виконуються вимоги ACID, і практично відпадає потреба в блокуванні зчитування;

- Типи даних

PostgreSQL підтримує великий набір вбудованих типів даних. Крім того, користувач може самостійно створювати нові необхідні йому типи та програмувати для них механізми індексування за допомогою GiST;

- Об'єкти користувача

PostgreSQL може бути розширено користувачем для власних потреб практично в будь-якому аспекті;

- Успадкування

Таблиці можуть успадковувати характеристики та набори полів від інших таблиць (батьківських). При цьому дані, які додаються до породженої таблиці, автоматично будуть брати участь (якщо це не вказано окремо) в запитах до батьківської таблиці. Цей функціонал в поточний час не є повністю завершеним. Однак він достатній для практичного використання;

- Тригери

Тригери визначаються як функції, що ініціюються DML-операціями. Наприклад, операція INSERT може запускати тригер, що перевіряє доданий запис на відповідність певним умовам. Тригери можна писати різними мовами програмування. Вони пов'язані з визначеною таблицею. Множинні тригери виконуються в алфавітному порядку;

3.3.4 ORM Entity Framework Core

Entity Framework (EF) Core - це легка, розширювана та крос-платформенна версія популярної технології доступу до даних Entity Framework.

EF Core може слугувати об'єктно-реляційним маппером (O / RM), що дозволяє розробникам .NET працювати з базою даних, що використовує об'єкти .NET, і виключає необхідність більшості коду доступу до даних, який вони, як правило, повинні писати. EF Core підтримує багато СУБД.

3.3.6 Архітектурний стиль взаємодії REST

REST – підхід до архітектури мережевих протоколів, які забезпечують доступ до інформаційних ресурсів. В основі REST закладено принципи функціонування Всесвітньої павутини і, зокрема, можливості HTTP. REST було розроблено паралельно з HTTP 1.1 базуючись на попередньому протоколі HTTP 1.0.

Дані повинні передаватися у вигляді невеликої кількості стандартних форматів (наприклад, HTML, XML, JSON). Будь-який REST протокол (HTTP в тому числі) повинен підтримувати кешування, не повинен залежати від мережевого прошарку, не повинен зберігати інформації про стан між парами «запит-відповідь». Стверджується, що такий підхід забезпечує масштабовність системи і дозволяє їй еволюціонувати з новими вимогами.

3.3.7 Система керування пакунками NuGet

NuGet - це вільна система керування пакунками, розроблена для Microsoft development platform (що раніше називалась NuPack) З моменту запуску у 2010-му, NuGet було включено до широкої системи інструментів та сервісів.

NuGet спочатку було презентовано як додаток для Visual Studio. У 2012 році NuGet був інстальований в Visual Studio за замовчуванням. NuGet також інтегровано з SharpDevelop. NuGet можна використовувати як з командної стрjки, так і автоматизовано, за допомогою скриптів.

Він підтримує багато з мов програмування, таких як:

- Пакети для .NET Framework;
- Нативні пакети, написані мовою C++, з можливістю створення пакетів на CoApp або OneGet

3.3.8 Кешування

Кеш — інформаційна технологія для тимчасового зберігання (кешування) даних(наприклад зображень) задля зменшення серверних затримок. Система кешу зберігає дані, що проходять через неї; подальші запити можуть бути виконані з кешу за певних умов.

4 ДЕТАЛЬНИЙ ОПИС АРХІТЕКТУРИ

4.1 Вимоги до архітектури системи

В умовах розробки open-source продукту до архітектуру проекти з’являються наступні вимоги:

- Простота та прозорість архітектури;
- Простота та повнота інтерфейсів програмних компонентів;
- Простота модифікації існуючого коду;
- Легкість розгортання проекту на локальній машині;
- Наявність гарної документації, або слідування принципу само-документованого коду;

Також важливим був вибір парадигми програмування. На сьогоднішній день домінуючими є функціональна та об’єктно-орієнтована парадигми. Через специфіку обраних технологій та більшу популярність, було застосовано об’єктно-орієнтовану парадигму.

4.2 Архітектура програмного продукту

Оскільки програмний продукт повинен мати змогу працювати з декількома кінцевими платформами(наприклад, бот у Telegram, чат-бот у Facebook або Twitter),то було прийняте рішення розділити бота та АПІ для агрегації та роботи з курсом крито валют. Також на етапі планування, була прийнята ідею використовувати Service Bus для оркестрування та синхронізації сервісів, але в процесі розробки цю ідею було відкинуто через високу складність реалізації цієї частини. Замість неї було використано простий task scheduler. Яка вже було сказано, система складається з двох компонентів: бекенду

боту та бекенду API. Бекенд боту та API не мають стану та є REST-ful сервісами. Для зберігання даних використовується реляційна база даних.

4.3 Огляд структури бота

При попередньому проектуванні, планувалась наявність додаткового компоненту системи: Service Bus-у. Цей компонент відповідав за орекестрацію та синхронизацію роботи бота та API. Але при більш детальному вивчені предметної області через складність реалізації двоетапних транзакцій було вирішено відкласти цей компонент на наступну ітерацію розробки, і замість нього використати простіший у реалізації Task Scheduler.

Бот є прошарком між API та користувачем. Він перетворює команди користувача на запити до API, обробляє відповідь та подає у зручному для користувача представлені. Також бот віддає команду на агрегацію зовнішніх ресурсів та розсилає оновлення курсу валют.

Для більш глибшого розуміння структури надалі буде розглянута структура найважливіших компонентів боту.

4.3.1 Компонент Bot.Routers

Цей компонент є пов’язочную ланкою між ботом і бізнес логікою у боті. У ньому знаходиться все необхідне для маршрутизації команд, отримання вхідних даних та виконання дій в залежності від маршруту. Головними елементами у цьому компоненті є:

- Інтерфейс IRouter

Цей інтерфейс є кінцевою абстракцію маршрутизатора. Він відповідає за ініціалізацію та оброблення команд користувача. Містить лише одну імплементацію, яка приймає маршрути виду “/command_name arg={arg1} {arg2}”;

- Інтерфейс IRouterExpressionParser

Абстракція, яка відповідає за перетворення маршруту до кінцевого регулярного виразу. Містить лише метод string ParseExpression(string command), який на вхід приймає строкове представлення маршруту, а повертає регулярний вираз;

- Клас Route – представляє абстракцію над маршрутом.
- Клас ParameterBag – представляє собою абстракцію, яка зберігає параметри команди;

4.3.2 Компонент Bot.Exceptions

У цьому модулі знаходяться всі виключення, що можуть відбутись в ході виконання програми. Перелік класів:

- DomainException – базове виключення, яке представляє помилку предметної області;
- RouteException – виключення, яке представляє те, що маршрут не було знайдено в маршрутизаторі;
- RouteParamsException – виключення, яке виникає при спробі використати невірний формат для строки маршруту;
- ApiException – виключення, яке відібражає, що при обробці запиту на АПІ сталася помилка

4.3.3 Компонент Bot.Entities

Представляє з собою необхідні сутності предметної області. Містить наступні класи:

- Currencies – представлення сутності крипто валюти;

4.3.4 Компонент Bot.APIs

Представляє собою абстракції над бекендом API. Містить в собі інтерфейс IAPI. Цей інтерфейс запроваджує необхідні для роботи бізнес логіки методи, які потім використовуються у сервісах. Цей модуль містить дві реалізації: proxy, яке здійснює доступ до API за допомогою http client-у та декоратор, який здійснює кешування. Кешування допомагає значно прискорити роботу боту, оскільки для часто виконуваних запитів, результати зберігаються, і не потрібно робити нове звернення до API. Кожний запис в кеші зберігається певний час(за замовчуванням 20 хвилин), тому користувач може не отримати данні реального часу. Також дані кешу скидуються при агрегації зовнішніх API.

4.3.5 Компонент Bot.Services

Цей компонент слугує для групування обробників для маршрутів по класам. У цьому модулі зібрана вся бізнес логіка боту. Цей модуль містить наступні класи:

- SubscriptionService – сервіс, який відповідає за підписку та відписку від оновлень певної крипто валюти.
- StartService – сервіс, який відповідає за ініціалізацію бота (реєстрацію користувача у АПІ)
- RateService – сервіс, який відповідає за оновлення курсу валют

4.3.6 Компонент Bot.Bot

Цей компонент є абстракцією над ботом. Він містить в собі всі залежності, необхідні для маршрутизації, обробки вхідних та вихідних даних. Також містить в собі компонент Bot.Bot.Replies, який реалізує паттерн Visitor і відповідає за перетворення об’єктів відображення додатку до формату, які відповідають вимогам Telegram API.

4.4 Огляд структури API

Оскільки API побудовано за допомогою фреймворку ASP .NET Core, який реалізує архітектурний паттерн MVC, у архітектурі API не так багато особливостей, на які варто було б звернути увагу.

4.4.1 Компонент BotApi.Services

Цей компонент містить у собі основну бізнес логіку системи. Складається з трьох інтерфейсів та їх реалізацій:

- ISubscriptionService – виконує операції, пов’язані з підпискою на оновлення курсу;
- IRateService – виконує операції, пов’язанні з отриманням актуального курсу валют;
- IUserService – виконує операції, пов’язанні з користувачами;

4.4.2 Компонент BotApi.Data

Цей компонент представляє шар доступу до даних(DAL). Містить в собі контекст даних ApplicationDBContext, який використовується EF Core. Також містить дав підкомпоненти:

- BotApi.Data.DAL – містить власне об’єкти доступу до даних, такі як IUnitOfWork та IRepository<T>;
- BotApi.Data.Models – містить в собі моделі, що представляють об’єкти предметної області та зберігаються в базі даних.

5 КЕРІВНИЦТВО АДМІНІСТРАТОРА

Оскільки система написана з використанням технології .net core, її можна розгорнути на MacOS, Linux та Windows.

Для розгортання додатку рекомендується використовувати технологію контейнеризації Docker. Обидва проекти налаштовані для роботи з докером. Для розгортання проекту в докері необхідно виконат наступні кроки:

1. Встановити Docker;
2. Встановити Docker-compose;
3. Встановити git;
4. Виконати команду “git clone https://github.com/klesogor/CryptoKursach.git” . Це склонує проект з гітхабу у поточну директорію
5. У разі, якщо сирцевий код знаходиться на локальній машині або іншому електронному носії, пропустити пункти 3 та 4, та просто скопіювати файли проекту у зручну для Вас директорію.
6. У разі, якщо Ви хочете змінити токен авторизації для боту, необхідно встановити нові значення у BotApi/appsettings.json та Bot/APIs/AspNetApi.cs (дивіться зображення 5.1 та 5.2)
7. Виконати команду docker-compose up. Ця команда скомпілює проект та зробить доступним API на порті 8080.

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=localhost;Port=5432;Database=bot;User Id=bot;Password=bot;"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Warning"
    }
  },
  "AllowedHosts": "*",
  "Secrets": {
    "BotToken": "c2VjcmV0IGtleQ=="
  }
}
```

Рисунок 5.1 – Змінна BotToken у файлі appsettings.json

```
namespace Bot.APIs
{
    public class AspNetApi : IAPI
    {
        private const string key = "c2VjcmV0IGtleQ==";
        //incapsulate dependencies. No real need for DI, since this is .NET objects
        private readonly HttpClient _http;

        public AspNetApi()
        {
            _http = new HttpClient() {BaseAddress = new Uri("http://localhost:64132/api/v1/") };
            _http.DefaultRequestHeaders.Add("X-AUTH-TOKEN", key);
        }

        public async Task<List<Currency>> GetAvailableCurrencies()
```

Рисунок 5.2 – Змінна key у файлі AspNetApi.cs