МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Кваліфікаційна робота магістра

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ АГРЕГУВАННЯ ДАНИХ КРИПТОВАЛЮТНИХ БІРЖ

Здобувач вищої освіти гр. ІН.м-13 Артем МЕНЯК

Науковий керівник:
кандидат фізико-математичних наук,
асистент кафедри комп'ютерних наук Ольга ШУТИЛЄВА

В.о. завідувача кафедри
кандидат технічних наук, доцент,
доцент кафедри комп'ютерних наук Ігор ШЕЛЕХОВ

Суми 2022

Сумський державний університет
(назва вузу)

Факультет *ЕлІТ* Кафедра *Комп'ютерних наук*
Спеціальність *«122 - Комп'ютерні науки»*

Затверджую:
зав.кафедрою _____
_____
"_____" _____ 20___р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТОВІ

Меняку Артему Володимировичу
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Інформаційна технологія агрегування даних криптовалютних бірж*
затверджую наказом по інституту від "____" _____ 20 ___ р. №_____

2. Термін здачі студентом закінченого проекту (роботи)_____

3. Вхідні данні до проекту (роботи)_____
_____
_____
_____
_____
_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз проблеми та постановка задачі 2) Інформаційна технологія агрегування даних криптовалютних бірж 3) Проєктування системи агрегатора криптовалютних бірж*

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
_____
_____
_____
_____
_____
_____

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка
	Аналіз проблеми та постановка задачі		
	Постановка задачі		
	Інформаційна технологія агрегування даних криптовалютних бірж		
	Програмна реалізація		
	Оформлення пояснювальної записки до кваліфікаційної магістерської роботи		

Студент – дипломник _____
(підпис)

Керівник проекту _____
(підпис)

РЕФЕРАТ

Записка: 86 сторінок, 31 рисунок, 2 додатки, 39 джерел.

Об’єкт дослідження — тенденції та стан сучасних систем для роботи з криптовалютами, агрегатори даних з криптобірж, паттерни програмування, використання API, об’єктно-орієнтовані технології роботи з даними.

Мета роботи — розробка системи агрегатора даних криптовалютних бірж використовуючи сторонні API.

Результати — у середовищі Microsoft Visual Studio, використовуючи фреймворк ASP.NET MVC було розроблено систему агрегатор даних з криптовалютних бірж, яка використовує API для отримання актуальних даних по всіх доступних токенах. Дана система є багатошаровою та використовує наступні паттерни: MVC, UnitOfWork, Repository.

КРИПТОВАЛЮТА, КРИПТОБІРЖА, АГРЕГАТОР КРИПТОВАЛЮТ, C#, ASP.NET, ENTITYFRAMEWORK, БАГАТОШАРОВА АРХІТЕКТУРА, MVC, UNITOFWORK, REPOSITORY PATTERN, MICROSOFT VISUAL STUDIO

ЗМІСТ

ВСТУП

Валютні біржі, цінні папери, різноманітні коштовні метали та інші фінансові інструменти вже давно стали невід'ємною частиною життя інвесторів. Але час невпинно йде і з ним неодмінно змінюються різноманітні речі, в тому числі й вищесказані. Наше століття можна назвати часом цифрових технологій та віртуального світу, тому виникнення такого явища, як криптовалюти та відповідно криптовалютні біржі є цілком очікуваним.

Сучасні тенденції віртуального світу ведуть до того, що сфера криптовалют перетворюється не тільки в місце заробітку. Багато фінансових експертів заявляють, що це лише чергова повітряна куля, яка найближчим часом лопне та призведе до банкрутства мільйонів людей. Тим не менш, у цю сферу залучені величезні фінансові вливання. Маючи достатньо досвіду, навички та специфічні знання – можна працювати та заробляти кошти настільки ж ефективно, як і на традиційних фінансових ринках.

Ринки криптовалют останнім часом зазнали значних трансформацій, як виключно статистично, так і публічно. Капіталізація виросла з 327.3 мільярдів до 2.14 трильйона доларів у період з квітня 2020 року до квітня 2022 року [1]. Якщо виключити Bitcoin як найвідомішу криптовалюту з найбільшою капіталізацією, капіталізація виключно альтернативних монет, тобто альткоїнів, виросла з 90.45 мільярдів до 1.204 трильйона в той самий період. Більше того, серед різноманітних міжнародних компаній зростає тренд інвестування в різноманітні рішення, пов'язані саме з криптовалютами. Наприклад, всесвітньо відома компанія Tesla дає можливість своїм клієнтам придбати власну продукцію викоистовуючи монету Dogecoin, яка була видумана та створена виключно як жарт. Таким чином, популярність даної сфери з кожним днем стає все більшою та стає зрозуміло, що незабаром така інтеграція буде невід'ємною частиною будь-яких фінансових операцій.

Криптовалютні біржі стали дуже потужним інструментом для інвестування. Дуже багато малих інвесторів заробили статок спекулюючи криптовалютою слідкучючи за новинами в трендах та хвилями популярності різних монет. З іншого боку, звісно ж, існує значний ризик втратити всі накопичення через природну волатильність (коливання) та нестабільність ринку. Наприклад, проаналізувавши дані з ресурсу coinmarketcap.com (станом на 1 жовтня), капіталізація монети Bitcoin впала з 901.61 мільярдів до 372.3 мільярдів в період з квітня 2022 року до жовтня 2022 року [2]. Таким чином, навіть існуючі інструменти для аналізу торгів, які постійно оновлюються та додаються, не завжди домагають інвесторам зберегти свої гроші. Саме через волатильність та той факт, що криптовалюти поводять себе зовсім не так як фіатні (тобто паперові) робить їх дуже мінливим інструментом для інвесторів та вимагає створення власних торгових стратегій та методик [3].

Особливим гравцем на ринку криптовалют стають різноманітні агрегатори криптобірж. Це швидкий та зручний інструмент, який дозволяє знайти найкращі умови та найвигідніші курси монет. До найвідоміших можна віднести CoinGecko або CoinMarketCap.

Саме тому, розробка додатку, який зможе агрегувати монети з різних бірж в одному місці, щоб інвестор надалі міг обирати для себе найкращі умови є актуальною задачею, виконання якої передбачає використання різноманітних сучасних технологій, паттернів та методик.

1. АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Основні ідеї та положення технології Blockchain

Зазвичай, фінансові системи використовують сторонні фінансові установи, наприклад банки, з метою обробки різноманітних операцій та платежів. Всі ці установи можна розглядати як посередників між сторонами, та звісно ж, вони мають повний контроль всіх транзакцій. Незважаючи на те, що такі системи вже давно існували і вправно виконували поставлені задачі, існували в них і власні недоліки. Як приклад таких недоліків можна назвати обмеження об'єму транзакції, безпосередньо її вартість, відсутність довіри до фінансових установ, безпекові нюанси, прозорість та гнучкість. Було багато спроб розробки децентралізованої системи нерегульованих віртуальних валют, щоб прибрати всі вищеназвані проблеми, проте лише одна з них закінчилась успішно.

У 1991 році Стюарт Ґабер та У. Скотт Сторнетт розробили технологію Блокчейн (англ. Blockchain), яка використовується і сьогодні [4]. Блокчейн по своїй суті – це розподілена база даних, певний список публічних записів, які підтверджені вузлами учасниками. Вся інформація про будь-яку транзакцію записується в публічну книгу, яка повністю доступна всім її учасникам. Таким чином, технологія блокчейн більш прозора ніж централізована мережа транзакцій, яка включає в себе сторонніх осіб. Більше того, оскільки все вузли-учасники анонімні, таким чином це значно безпечніше для перевірки операцій інших вузлів-учасників [5]. Термін блокчейн походить від того факту, що він інкрементно збільшується за рахунок блоків даних, або ланцюжків даних [6], кожний з яких містить наступні дані:

- Дані блоку (Block-data) – набір повідомлень або транзакцій.
- Хеш ланцюга (Chaining-hash) – копія хеш значення безпосередньо попереднього блоку.
- Хеш блоку (Block-hash) – хеш значення самого блоку.

На рисунку 1.1 можна побачити схематичний вигляд блокчейну.

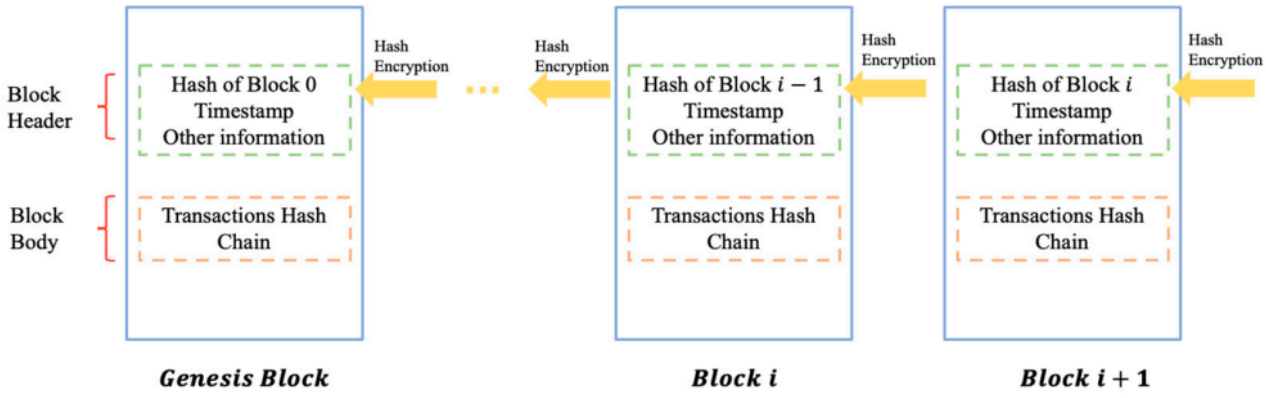

Рисунок 1.1 – Схематичний вигляд ланцюга блокчейну

Транзакції – це атоми мережі блокчейн. Зазвичай, транзакції створюються користувачами або автономними об’єктами (смарт контрактами) з метою позначення передачу токенів адресату відправником. В транзакції вказується, цілком можливо, пустий список вхідних даних, що асоційовані з токеном, що мають ідентифікатори (адреси) відправників [7]. Також в транзакції обов’язково вказується непорожній список виходів, що позначає результат перерозподілу вхідних токенів між пов’язаними ідентифікаторами отримувачів. Транзакцію можна позначати як статичний запис, що показує ідентифікаційні дані адресатів/адресантів, значення токена, яке потрібно перерозподілити, та стан отримання токена. З метою захисту автентичності запису транзакції використовуються функції криптографічного хешування та асиметричного шифрування [7]:

- Хеш-функція: криптографічна хеш-функція рандомно прив’язує бінарний вхід довільної довжини до унікального виходу (зображення) фіксованої довжини. Використовуючи криптографічно безпечну хеш-функцію (наприклад SHA-256), обчислювально неможливо відновити вхідні дані з вхідного зображення. Також варто зазначити, що згенерувати однаковий вихід для двох різних входів неможливо.

- Асиметричний ключ: Кожен вузол у мережі блокчейн генерує пару приватних та публічних ключів. Приватний ключ пов’язаний з функцією цифрового підпису, на виході якої отримуємо підпис фіксованої довжини для будь-якого вхідного повідомлення. Публічний ключ, в свою чергу, пов’язаний з функцією верифікації, яка на вхід бере аналогічне повідомлення та вже відомий нам підпис для цього повідомлення. Функція верифікації на виході повертає булеве значення *true* лише коли підпис був згенерований функцією цифрового підпису з відповідним приватним ключем та вхідним повідомленням.

Вузли або різні автономні об’єкти в мережі ідентифікують себе власними публічними ключами у вигляді перманентних (постійних) адрес (також відомих як псевдо-ідентифікаторів) у блокчейні. Виходячи з того, що кожний вхідний кортеж у транзакції підписується відповідним обліковим записом відправника, мережа може публічно підтверджувати автентичність вхідних даних шляхом верифікації підпису, який згенерований з публічним ключем відправника.

В системі блокчейн для криптовалют заголовок блоку складається з метаданих, таких як попередній блок, версія блоку, хеш, час операції та її деталі. Всі ці дані потрібні для того, щоб було можливо ідентифікувати конкретний блок у мережі, оскільки кожний блок має власний унікальний заголовок. Таблиця 1.1 відображає інформацію про атрибути, які включає в себе кожна ланка блокчейну.

Таблиця 1.1 – Інформація про атрибути системи блокчейн в сфері криптовалют

Заголовки	**Атрибути та їх опис**
Статистична інформація	**Загальна кількість монет:** загальна кількість монет в мережі, які були видобуті шляхом майнінгу **Вартість на момент операції:** вартість монети **Капіталізація ринку:** загальна ціна даної криптовалюти в обігу

Інформація про блок	**Розмір блокчейна:** загальний розмір блокчейна **Середній розмір:** середній розмір блоку за останні 24 години **Середня кількість транзакцій:** середня кількість транзакцій в блоці за останні 24 години **Загальна кількість транзакцій:** загальна кількість транзакцій в блокчейні **Медіана часу підтвердження:** середній час очікування для обчисленого блоку перед додаванням в публічну книгу
Інформація про майнінг	**Загальний хеш-рейт:** очікуваний номер терахешів за секунду **Розподіл хеш-рейту:** очікування розподілу хеш-рейту серед всіх майнінгових пулів **Складність мережі:** складність майнінгу (обчислення) нового блоку **Винагорода за обчислення:** Загальна інформація про винагороди та комісії виплачені майнерам за даний блок **Загальна комісія на транзакцію** **Комісія за транзакцію:** середнє комісії за кожну транзакцію

З моменту винайдення технології було представлено безліч проєктів які на ній базуються в найрізноманітниших секторах, починаючи з міжнародних фінансових систем і закінчуючи системи контролю здоров’я в медичних закладах. Більше того, почались спроби створення віртуальних валют, наприклад Bit Gold у 1998 році.

У 2008 сталась історична подія для світу криптовалют та технології блокчейн. Саме тоді Сатоші Накамото опублікував наукову працю [8], яка описує концепт електронної платіжної мережі, а потім у 2009 вже реалізував її.

Згідно публікації Накамото, криптовалюти базуються на децентралізованій мережі peer-to-peer, використовуючи технологію блокчейн щоб зберігати всі транзакції у децентралізованій публічній книзі. Окрім цього, технологія блокчейн відповідає за підтвердження транзакцій та синхронізацію вузлів у мережі учасників.

З метою безшовного вирішення проблем псевдонімів, масштабування та жахливої синхронізації, Накамото запропонував бездозвольний прокол [8], який базується на фреймворку криптографічної “гри” з відкриття блоків. Цей протокол також відомий як Proof of Work (PoW) [9]. З точки зору єдиного вузла, протокол Накамото виділяє три головних процедури, як зазначено в назві, процедура підтвердження ланцюга, процедура порівняння ланцюга та розширення, і процедура пошуку вирішення PoW [10]. Процедура підтвердження ланцюга надає булеве значення, чи має наданий ланцюг, що складається з блоків, правильну структуру. Перевіряється, чи кожний блок в ланцюгу має коректне вирішення PoW задачі, відсутність конфліктів та записів в історії блоку. Процедура порівняння ланцюга та розширення порівнює довжину набору ланцюгів, що можуть бути або отримані від однорангових вузлів, або, можливо, надійшли з одного місця. Це гарантує, що доброчесному вузлу належить лише найдовший ланцюг з всіх запропонованих. Процедура пошуку рішення PoW головна робоча сила протоколу і визначає вона криптографічне вирішення головоломки у інтенсивно-обчислювальній манері.

Якщо коротко, то процедура пошуку рішення PoW вимагає безперервного посилання запитів до криптографічної хеш-функції для отримання часткового прообразу, який і генерується блоком-кандидатом. Очікується, що хеш-код цього блоку повинен задовільняти заздалегідь визначені умови. Для простоти викладення, можна позначити хеш-функцію як $H(\cdot)$, а змінною x – вхідний двійковий рядок, який зібраний на основі даних блоку-кандидата, включаючи в себе набір транзакцій, та вказівники на хеш-посилання. Формально визначення виглядає так:

Враховуючи регульований параметр умови твердості h, процес вирішення проблеми PoW спрямований на пошук рядка рішення, криптографічного нонса ($nonce$), такого, що для кожного вхідного рядка x, який зібраний на основі даних блоку-кандидата, хешкод (тобто заголовок цільового блоку) конкатенації x та $nonce$ є меншим за цільове значення $D(h)$:

$$bh = \mathrm{H}(x \| nonce \leq D(h), \tag{1.1}$$

де для певної фіксованої довжини бітів L, $D(h) = 2^{L-h}$, $H(\cdot)$ – хеш функція, x – вхідний рядок, $nonce$ – рішення [7].

Протокол Накамото вимагає надзвичайно великих обчислювальних ресурсів, оскільки вузлу для перемоги у цьому “змаганні” з вирішення головоломок, треба отримати найвищу можливу частоту запитів хеш-функції. З іншого боку, фінансова сторона проблеми (рахунки за електрику) також роблять добровільну участь у всьому процесі максимально незручною та невигідною для будь-якого вузла. З метою коректного функціонування мережі блокчейн, протокол PoW запроваджує певні стимули для учасників процесу на основі вбудованого механізму постачання токенів та заохочення транзакцій [8]. З точки зору теорії ігор, неявне припущення, прийняте протоколом консенсусу Накамото, полягає в тому, що всі вузли-учасники є індивідуально раціональними [11]. У свою чергу очікується, що механізм консенсусу буде сумісним із системою винагород для вузлів. Іншими словами, протокол консенсусу повинен гарантувати, що будь-який вузол консенсусу зазнає фінансових втрат щоразу, коли він відхиляється від дотримання доброчесності використання протоколу.

Однак сумісність із системою винагород протоколу Накамото була піддана відкритому сумніву [12]. Оскільки протокол Накамото дозволяє вузлам пропонувати довільні блоки зі свого локального набору транзакцій, які очікують розгляду, в мережі неминуче відбудеться розширення з (тимчасовим) розділенням, тобто форком, з локальної точки зору вигляду блокчейну [13] (див.

рис. 1.2). Щоб гарантувати правильність консенсусу та, таким чином, конвергенцію до одного канонічного стану блокчейну, протокол PoW спирається на припущення, що більшість вузлів консенсусу будуть дотримуватися правила найдовшого ланцюга та будуть альтруїстичними в пересиланні інформації. У роботі [12] було виявлено, що раціональні консенсусні вузли можуть не мати винагороди за розповсюдження транзакцій/блоків. Як наслідок, проблема розгалуджень не може бути вирішена простим шляхом у поточній версії протоколу PoW. В подальшому, при модернізації протоколу варто вжити деяких змін у дизайні та запровадити певні принципи, які були запропоновані з метою покращення механізму системи винагород та спрямування його на безпечні та стійкі блокчейн-мережі [14, 15]:

- Механізм консенсусу повинен переконати, що поширення інформації та розширення найдовшого ланцюжка блоків є монотонними стратегіями вузлів консенсусу [16].
- Механізм консенсусу повинен заохочувати децентралізацію та справедливість. А саме, він повинен не просто перешкоджати ботнетам та майнінг-пулам [17, 18], а і робити сам процес незручним для учасників з накопиченою обчислювальною потужністю.
- Механізм консенсусу повинен знайти належний баланс між пропускною здатністю та масштабованістю мережі [18].

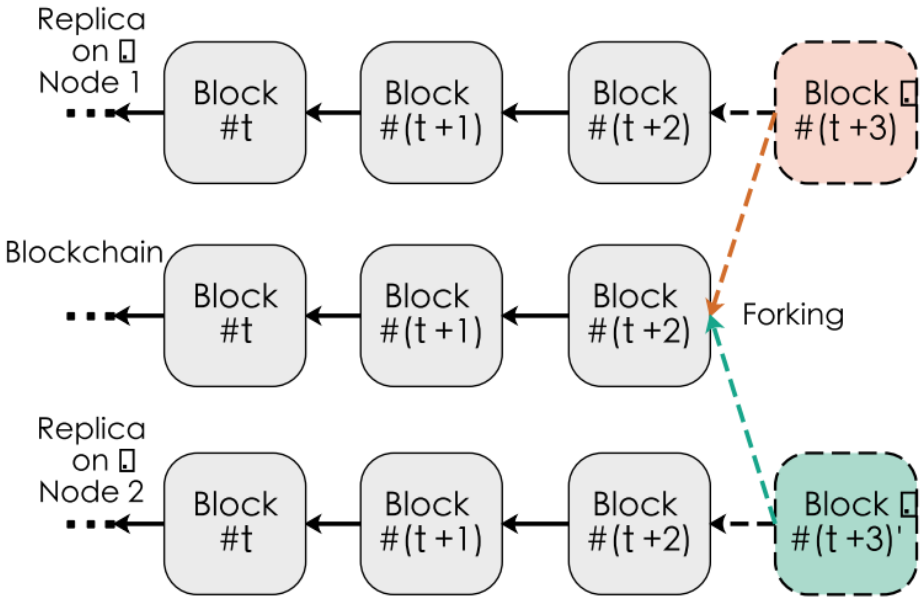

Рисунок 1.2 – Тимчасовий форк, який з'являється на вузлах 1 та 2 коли результати їх локальних процесів PoW призводять до різних можливих варіантів нового заголовку блокчейна, тобто (t + 3) та (t+3)' одночасно. Обидва результати (t + 3) та (t + 3)' задовільняють формулу (1.1)

1.2 Основні ідеї та положення криптовалюти

Традиційні фінансові системи та криптовалюти можуть порівнюватись у багатьох відношеннях, таких як власне фінансові, регуляторні та децентралізовані. З фінансової точки зору, у грошей є три функції, а саме міра вартості (unit of accounting), засіб обігу (medium of exchange) та засіб заощадження (store of value). Функція міри вартості значить те, що відносна вартість товарів та різноманітних послуг забезпечена грошима, таким чином утворюючи ціну. Функція засобу обігу буквально відображує сутність у назві, тобто за послуги та товари можна розплатитися певною валютою. Засіб заодщадження, очевидно, значить те, що дана фінансова одиниця може бути збережена та в майбутньому використана для оплати послуг чи товарів. Наприклад, золото та серебро це два найпопулярніші товари, які зберегли свою вартість протягом багатьох століть, завдяки своїй властивості зберігати вартість.

Хоча криптовалюти цілком виконують функцію засобу обігу на даний час вже в багатьох країнах, проте вони не здатні виконувати дві інші функції [19]. На відміну від справжніх фіатних грошей, обіг яких держава має змогу централізовано регулювати відповідно до економічної ситуації в країні, криптовалюти централізовано регулювати не ефективно. Саме через це більш правильним вважається практика регуляції взаємодії криптовалют з цілком реальними фінансовими установами [20]. Щодо децентралізації, фінансові установи виступають в ролі основних посередників, які безпосередньо контролюють та полегшують фінансові операції в централізованій економічній системі. Однак, ці установи можуть домінувати у фінансових операціях коли вони зростають та призводять до непрорційності в ринкових відносинах. З іншого боку, децентралізовані платформи знижують вартість транзакцій та усувають монополію будь-якої інституції шляхом створення ефективної децентралізованої однорангової мережі [21].

Унікальність криптовалют базується на трьох принципах, а саме анонімності, децентралізації та подвійній витраті (double-spending), що являє собою надійний захист від кібератак [22]. Однак, навіть з цими характеристиками досі існує певна невизначеність між криптовалютами та електронними грошима, яка показана в таблиці 1.2, на основі даних з визначення Центрального Європейського Банку [23]. Віртуальні гроші відрізняються від цифрових всім, окрім формату.

Таблиця 1.2 – Порівняння електронних грошей та віртуальних, згідно визначення Центрального Європейського Банку.

	Електронні	**Віртуальні**
Формат грошей	Цифровий	Цифровий
Визнання	Загально визнані	Зазвичай всередині певної віртуальної спільноти

Статус в державі	Регульовані	Нерегульовані
Видавець	Легально видані офіційною установою	Не фінансова приватна компанія
Надходження коштів	Фіксоване	Не фіксоване (залежить від рішень видавця)
Державний контроль	Так	Ні
Види ризиків	Операційний	Легальний, кредитний, ліквідний та операційний

На даний час Bitcoin можна вважати найпопулярнішою та найбільш поширеною криптовалютою та де-факто стандартом серед світових криптовалют. Тим не менш, на момент листопаду 2022 року існує приблизно 17000 інших криптовалют, себто альткоїнів, які займають приблизно половину від загальної капіталізації ринку. На рисинку 1.3 зображено ринкову капіталізацію найпопулярніших 10 криптовалют криптовалют станом на вересень 2022 року за версією агрегатора CoinMarketCap [2]. На даному рисунку зображено наступні монети: BTC, ETH, USDT, BNB, USDC, SOL, ADA, XRP, LUNA, DOT, повні назви яких Біткоїн, Етеріум, Тезер, Бінанс коїн, USD коїн, Солана, Кардано, Ріпл, Терра та Полькадот.

Суто технічно, альткоїни використовують приблизно таку або ідентичну технологію блокчейн як і Біткоїн. Але головна суть впровадження альткоїнів на ринок – представити користувачу щось нове в порівнянні з флагманським Біткоїном з метою підвищення вартості власної розробки, таким чином, відповідно, підвищуючи ціну альткоїна на ринку. Для прикладу можна назвати Етеріум, другу за популярністю монету після Біткоїн. Використовує вона майже всі переваги технології вищеназваного флагмана, але ще вводить додаткові інновації, такі як податкові обмеження та платформу смарт-контрактів (Smart Contracts) [24]. А якщо брати як приклад іншу монету, Лайткоїн (Litecoin), то він

розроблений для того, щоб під час майнінгу інших монет виконувались різноманітні вимоги обчислень. Більше того, головна мета монети Деш (Dash) – зробити найшвидший процес транзакцій та більш досконалий захист приватності користувачів [25].

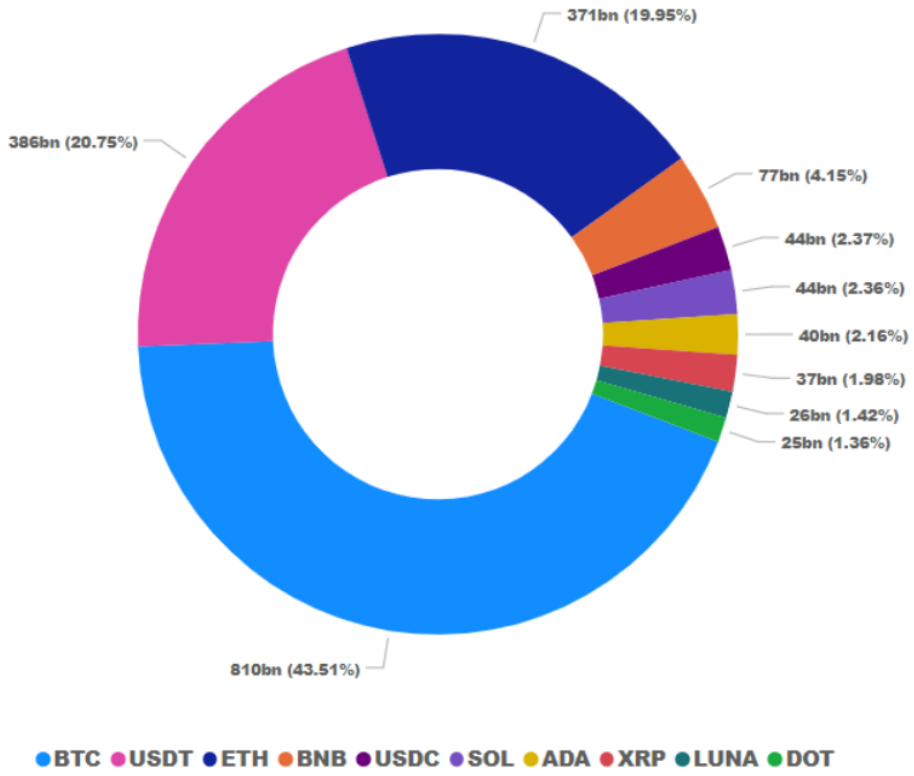

Рисунок 1.3 – Ринкова капіталізація різних криптовалют

Окрім очевидних технічних вдосконалень, криптовалюти можуть відрізнятися більш ґрунтовно, а саме розділені на наступні категорії:

- Базовані на майнінгу (mining-based). Мають схожі характеристики до Біткоїна, проте як зазначено в назві, використовують процес майнінгу для генерації нових монет. Статистично таких монет найбільше. Найвідоміші з них – це Біткоїн (Bitcoin), Верткоїн (Vertcoin), Етеріум Класік (Ethereum Classic), Монеро (Monero), Рейвенкоїн (Ravencoin), Хейвен Протокол (Haven Protocol), Біткоїн Ґолд (Bitcoin Gold), Догкоїн (Dogecoin).
- Стейблкоїни (stablecoins). Найбільша проблема монет, що базуються на майнінгу – висока волатильність та флуктуація цін, що робить їх надзвичайно

ризикованим та складним інструментом для інвесторів. Головна мета стеблкоїнів – прибрати ризики та як вказано в назві, зробити монету стабільною. Якщо точніше, стабільною як справжні валюти, наприклад фіатні. Звісно ж, ці монети як і будь-які інші підкріплені широким стеком безпекових технологій, що забезпечує їх вартість та ціну. Найвідоміший стейблкоїн це Тезер (Tether або USDT), який прикріплений до американського долара та Дієм (Diem, раніше відомий як Libra), розроблений корпорацією Meta.

- Утилітарні токени (utility tokens). Даний тип криптовалюти становить вартість для інвесторів пропонуючи доступ до майбутнього або існуючого продукту або сервісу. До прикладів можна віднести Arweave (AR), де децентралізована мережа для зберігання даних використовує токен AR щоб стимулювати майнерів до допомоги в збереженні величезних кількостей мережевих даних. Або Binance Coin (BNB), нативний коїн криптобіржі Binance, який одночасно коїн та токен. Дозволяє користувачам отримувати знижку в 25% на торгові податки, коли вони торгують BNB. Також не можна не згадати про Chainlink та токен LINK. Сервіс Chainlink це “оракул”, який в реальному часі надає дані в блокчейни із зовнішнього джерела, може бути інтегрований в різноманітні блокчейни. Він видає токени ERC-20 (LINK) користувачам, які надають актуальні дані для передачі в різні блокчейни. Це може бути особливо корисним для додатків із смарт-контрактами щоб відслідковувати поточні ціни. Ще можна назвати платформу Zilliqa та їх нативний токен ZIL. Ця платформа використовується для створення доступних та безпечних DApp для розробників, як для фінансових систем, так і для NFT маркетплейсів. ZIL же використовується для транзакцій в іграх та цифровій рекламі [26].

1.3 Сучасний стан та тенденції розвитку криптобірж

Вся екосистема криптовалют та всього пов’язаного з ними в більшості своїй покладається на криптобіржі. Саме вони дозволяють значно полегшити

життя інвесторам та трейдерам, роблячи можливим обмін, купівлю та продаж криптовалют використовуючи інші активи але здебільшого фіатні валюти. Всього можна виділити три основних типи криптобірж:

- Централізовані (CEX). Даний різновид криптобірж регулюється корпорацією, як приклади можна назвати найвідоміші біржу Binance та Coinbase.
- Децентралізовані (DEX). Такі біржі пропонують автоматизований процес для однорангових (peer-to-peer) торгів між учасниками. Як приклади можна назвати біржі Uniswap та DODO.
- Гібридні. Як можна здогадатися з назви, це комбінація двох попередніх типів, тобто централізованих та децентралізованих [27].

Взагалі, у світі існує просто безліч різноманітних криптобірж. Для прикладу, Біткоїн може торгуватись в парі з іншими активами в кроспарах як мінімум в 387 криптобіржах, станом на жовтень 2022 року, за даними агрегатора CoinMarketCap [2]. В таблиці 1.3 наведені дані про декілька найвпливовіших та найвідоміших криптобірж у світі. Кількість монет що підтримується у другій колонці означає кількість криптовалют, які доступні для торгівлі. Більше того, дані про комісії за транзакції, локація де вони зареєстровані та рік заснування наведені в колонках три, чотири та п’ять відповідно. Згідно даних, наведених в таблиці 2, можна спостерігати, що значення комісій дуже сильно відрізняється, від безкоштовних до 5% за кожну транзакцію.

Таблиця 1.3 – Інформація про найпопулярніші криптовалютні біржі.

Назва біржі	**Кількість монет**	**Вартість транзакції (%)**	**Головний офіс**	**Засновано**
Binance	387	0.100	Мальта	2017
Huobi	623	0.200	Сейшели	2013
Bitfinex	191	0.200	Гонконг	2012
Coinbase Exchange	235	0.500	Сан-Франциско, США	2011

BitMex	17	0.075	Сейшели	2014
Kraken	217	0.260	Сан-Франциско, США	2011
Bittrex	378	0.350	Сіетл, США	2014
BitStamp	71	0.500	Люксембург	2011
KuCoin	773	0.100	Сейшели	2017
OKX	352	0.150	Мальта	2017

Але якими б чудовими та зручними не були технології, у всьому завжди є свої нюанси. Криптовалюти, звісно ж, не є виключенням з правил. Завдяки децентралізації криптовалют, та зокрема технології блокчейна, державні установи та правоохоронні органи не мають змоги належним чином контролювати фінансові операції, як легальні, так і не дуже. Більше того, майже всі нелегальні операції з грошима, наприклад купівля зброї, заборонених речовин та навіть людей в даркнеті можуть функціонувати лише завдяки анонімності блокчейна [28, 29]. На рисунку 1.4 зображено топ 10 спотових криптобірж за версією сайту-агрегатора CoinMarketCap [2].

#	Exchange	Score	Trading volume(24h)	Avg. Liquidity	Weekly Visits	# Markets	# Coins	Fiat Supported	Volume Graph
1	Binance	9.9	$11,428,545,531 ▲ 10.73%	947	15,046,498	1684	387	AED, ARS, AUD and +43 more	
2	Coinbase Exchange	8.1	$1,302,369,679 ▲ 0.74%	782	959,236	601	235	USD, EUR, GBP	
3	Kraken	7.6	$503,402,271 ▼ 12.48%	753	990,352	714	217	USD, EUR, GBP and +4 more	
4	KuCoin	6.7	$333,587,814 ▼ 2.35%	544	1,916,603	1393	773	USD, AED, ARS and +45 more	
5	Bitfinex	6.6	$135,397,057 ▲ 26.78%	576	446,991	422	191	USD, EUR, GBP and +1 more	
6	Bitstamp	6.5	$106,645,546 ▼ 8.54%	595	239,341	159	71	USD, EUR, GBP	
7	OKX	6.2	$812,837,453 ▲ 1.66%	532	1,460,509	784	352	AED, ARS, AUD and +43 more	
8	Bybit	6.1	$296,141,771 ▲ 17.75%	561	3,620,694	499	332	USD, EUR, GBP and +3 more	
9	Huobi	6.1	$441,731,018 ▲ 35.53%	541	1,011,390	990	623	ALL, AUD, BRL and +47 more	
10	Gemini	6.1	$32,174,803 ▲ 7.13%	629	247,069	140	113	USD, GBP, EUR and +4 more	

Рисунок 1.2 – Кращі спотові біржі та інформація про них від CoinMarketCap

1.4 Аналіз існуючих агрегаторів

Агрегатори криптовалютних бірж являють собою потужний інструмент для інвесторів, які хочуть дізнатись актуальні тренди криптовалютного ринку, проаналізувати актуальні події та зробити відповідні рішення.

За останні кілька років, декілька агрегаторів даних бірж криптовалют стали золотим стандартом в даній сфері. Інвестори довіряють кожній з цих платформ та постійно використовують їх повний функціонал для суттєвого збільшення вартості власних інвестиційних портфоліо.

Кожна з описаних нижче платформ уникільна та фокусується на певній конкретній задачі та баченні ринку власниками сервісу.

Декілька найважливіших факторів у виборі постачальника даних включають в себе:

- Точність історичних ринкових даних (агрегованих та окремих)
- Повнота асортименту представлених даних (біржі, торгові пари, активи)
- Протяжність історичних даних (роки, місяці)
- Доступність різних типів даних (книги ордерів, торгів, свічки)
- Доступність API (Websocket API, Rest API)

1.4.1 Shrimpy

Всесвітньо відомий криптоагрегатор Shrimpy здобув собі ім’я за накращий в світі функціонал з виконання торгових ордерів. Звісно ж, звичайний функціонал для доступу для історичних даних по активам теж присутній.

На відміну від інших агрегаторів даних, Shrimpy єдиний, який спеціально був розроблений для розробників. Замість того, щоб фокусуватися на інвесторах чи фінансових установах, ця платформа здебільшого фокусується на забезпеченні найкращого сервісу для розробників [30].

Shrimpy Developer API надає доступ до 16 різних криптобірж. Функціонал онлайн вебсокетів та Rest API з даними про ринок надають розробникам доступ до актуальних торгових книг, цін на певні активи та дані про останні угоди.

Даний сервіс може бути корисний тим розробникам, які бажають створити торгових ботів, сервіси для керування портфоліо або системи для арбітражу. Головні переваги Shrimpy:

- Достовірність історичних даних – сервіс пропонує точні дані для кожної окремої криптобіржі та торгової пари, без агрегацій та передбачень
- Широкий вибір підримуваних активів
- Історичні дані з 2014 року
- Широкий вибір доступних типів даних
- API: підтримка різноманітних API та WebSocket

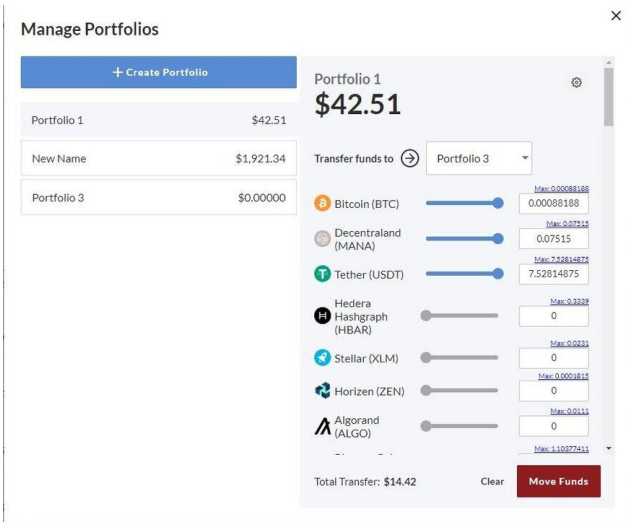

Рисунок 1.3 – Інтерфейс платформи Shrimpy

1.4.2 Kaiko

Даний криптоагрегатор позиціонує себе як найкращий в сфері історичних даних з 2014 року. Різноманітні фінансові установи та підприємці використовують саме Kaiko, оскільки саме він пропонує найкращу та найповнішу вибірку даних OHLCV (акронім від Open, High, Low, Close та

Volume), снепшоти торгової книги та дуже багато інших різноманітних даних з близько 100 різних криптобірж [31].

З останніми оновленнями Kaiko почав пропонувати глибоку аналітику крипторинку, таким чином приваблюючи нових користувачів.

Коли мова заходить про точності історичних даних, підприємці не помиляться, якщо оберуть саме цей сервіс. Розлогий каталог різноманітних типів даних, які пропонує сервіс, легко може обійти будь-якого конкурента.

Головні переваги Kaiko:

- Точність історичних даних;
- Величезна кількість даних: 35000+ торгових пар, 100+ криптобірж, 3000+ активів;
- Історичні дані з 2014 року;
- API: широка доступність різних типів даних: снепшоти торгових книг, торгові дані по тіках, свічки OHLCV, VWAP та деривативи.

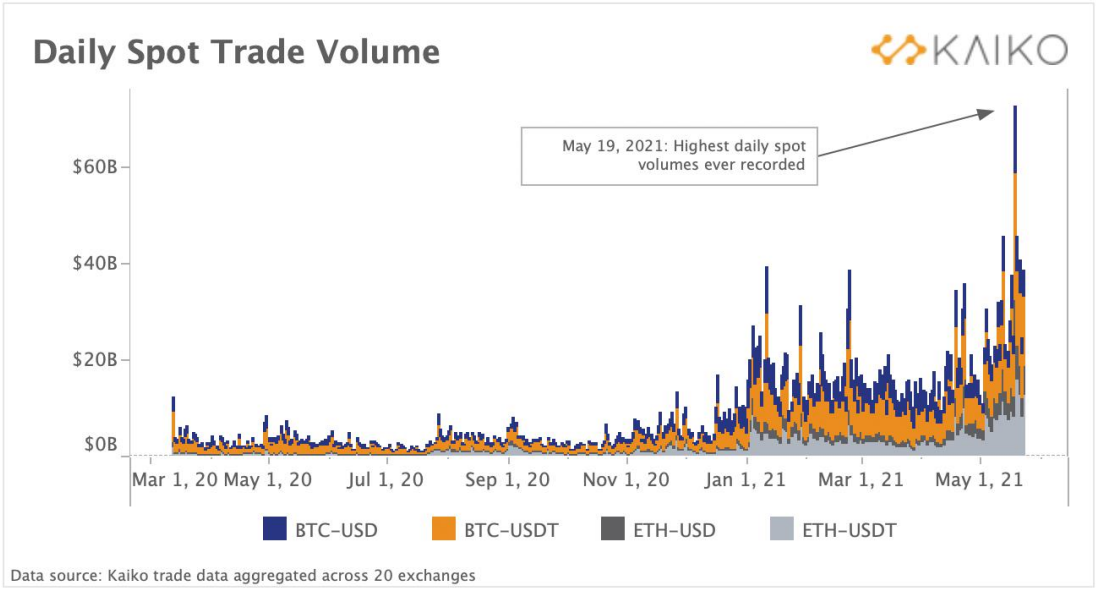

Рисунок 1.4 – Приклад даних аналітики, які пропонує сервіс Kaiko

1.4.3 CoinMarketCap

Дана платформа була запущена в 2013 році, та з того часу де-факто стала сайтом, де будь-хто може дізнатися щось нове про торгові активи, огляд цін на

активи від досвідчених трейдерів, проаналізувати дані з різних криптобірж та, звісно ж, як вказано в назві дізнатися ринкову капіталізацію як бірж, так і активів.

Незважаючи на те, що історичні дані від CoinMarketCap є доволі посередніми, якщо порівнювати з попередньо названими сервісами, проте є одна річ, яку дана платформа робить значно краще, а саме відслідковування активів на біржах та відстежування ринкової капіталізації кожного активу.

Коли мова йде про історичні та актуальні ринкові дані, CoinMarketCap в багатьох аспектах просто не вистачає. По-перше, ці дані доступні лише в агрегованому вигляді, тобто абсолютно не підходять для будь-якого технічного аналізу. По-друге, сервіс не підтримує такі значущі типи даних, як книги угод та торгові дані. Як висновок вищесказаного, багато інвесторів не можуть бути задоволеними функціоналом, що пропонує API даної платформи.

Головні переваги CoinMarketCap:

- Агреговані історичні дані
- Величезна кількість даних: 32000+ торгових пар, 300+ криптобірж, 7500+ активів
- Історичні дані з 2014 року
- Доступні дані: агреговані свічки OHLCV та дані про ринкову капіталізацію
- API: Rest API з історичними даними.

	#	Name	Price	1h %	24h %	7d %	Market Cap	Volume(24h)
☆	1	Bitcoin BTC	$16,914.56	▲0.37%	▲0.48%	▼0.83%	$325,231,405,404	$17,212,844,261 1,017,630 BTC
☆	2	Ethereum ETH	$1,248.88	▲0.47%	▲1.27%	▼1.95%	$152,830,679,013	$4,919,566,911 3,940,144 ETH
☆	3	Tether USDT	$1.00	▼0.00%	▼0.00%	▲0.00%	$65,689,580,872	$22,607,757,751 22,605,989,588 USDT
☆	4	BNB BNB	$286.73	▲0.23%	▲1.04%	▼2.03%	$45,867,118,611	$606,029,426 2,114,175 BNB
☆	5	USD Coin USDC	$1.00	▲0.01%	▲0.02%	▲0.03%	$42,861,466,618	$1,880,147,020 1,879,969,058 USDC
☆	6	Binance USD BUSD	$1.00	▲0.01%	▲0.03%	▼0.01%	$22,100,562,995	$5,023,897,540 5,023,366,457 BUSD
☆	7	XRP XRP	$0.3886	▲0.41%	▲1.74%	▼2.35%	$19,529,138,314	$701,690,369 1,803,544,015 XRP

Рисунок 1.5 – Приклад інтерфейсу сервісу CoinMarketCap

1.4.4 CoinGecko

Платформа CoinGecko в якомусь сенсі є схожою на попереднью CoinMarketCap. Проте, контрастуючи з ним, надає інші метрики, такі як соціальні рейтинги для активів та активність розробників.

Взагалі, можна назвати CoinGecko найкращим сервісом для даних, які не будуть дуже тісно пов’язані з ринком, такі як різноманітні новини, рейтинги та тому подібне. Також пропонують власний API зі списком індексів, різні фінансові інструменти, проводять живі заходи та багато іншого.

Основний ґеймченджер для даного сервісу – це унікальні дані, які дуже корисні для дослідження соціальних трендів та настроїв трейдерів [32]. Головні переваги CoinGecko:

- Точність історичних даних: тільки агреговані дані
- Кількість даних: 25000+ торгових пар, 400+ криптобірж, 6000+ активів
- Історичні дані з 2014 року
- Доступні дані: агреговані свічки OHLCV та дані про ринкову капіталізацію, соціальні дані

- API: Rest API з історичними даними

Top Asset-backed Tokens Coins by Market Cap Show Stats

The Asset-backed Tokens market cap today is $3.55 Billion, a -0.3% change in the last 24 hours.

Asset-backed Tokens | Show Fully Diluted Valuation

#	Coin	Price	1h	24h	7d	24h Volume	Mkt Cap
65	cUSDC CUSDC	$0.02269163	-0.0%	-0.1%	-0.4%	$3.05	$582,252,468
77	PAX Gold PAXG	$1,788.09	0.1%	0.2%	1.2%	$6,737,956	$481,581,950
83	cETH CETH	$25.08	0.0%	0.9%	-4.5%	$271	$438,479,127
85	Tether Gold XAUT	$1,745.30	-0.1%	1.6%	2.1%	$775,549	$431,049,901
89	cDAI CDAI	$0.02214718	-0.0%	0.1%	-0.5%	$1,186	$418,152,807
119	Olympus OHM	$9.15	0.1%	0.9%	6.6%	$259,019	$266,205,200
138	Tenset 10SET	$1.20	-1.9%	3.5%	2.2%	$157,493	$220,009,319

Рисунок 1.6 – Приклад інтерфейсу CoinGecko

1.4.5 Messari

Даний криптоагрегатор надзвичайно відомий в колах фінансових установ та професійних трейдерів. Взагалі, вся платформа була створена з розумінням, що фокусуватись треба не на безпосередньо даних з криптобірж, а скоріше на результатах аналітики, які будуть з них отримані.

Оскільки найпопулярнішим аспектом Messari є саме програма перевірки, основною пропозицією їх API є легкий доступ до цих даних. Клієнти можуть використовувати Messari для доступу до опису активів, різноманітних показників активів, ринкових пар, новин та статистики блокчейну [33].

Хоча даний сервіс не можна назвати саме постачальником ринкових даних, клієнтам пропонується унікальна вибірка даних у API, за допомогою якого можна поглянути на ринок з нової точки зору.

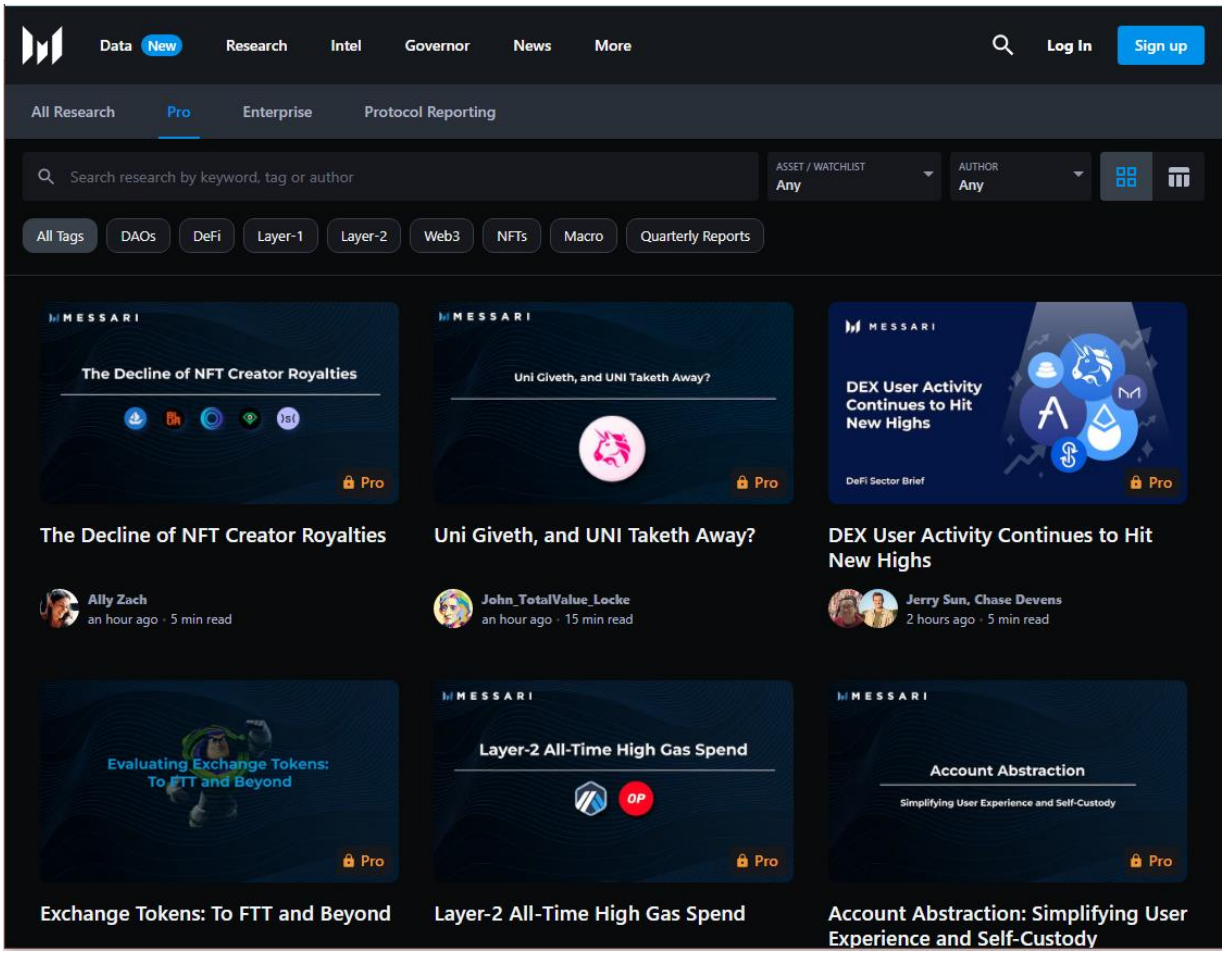

Рисунок 1.7 – Інтерфейс платформи Messari

Головні переваги Messari:

- Точність історичних даних: неточні агреговані дані
- Кількість даних: 35000+ торгових пар, 100+ криптобірж, 3000+ активів
- Історичні дані починаючи з 2014 року.
- Доступні дані: профілі активів, метрики активів, новини
- API: Rest API

1.5 Формалізована постановка задачі

Станом на 2022 рік криптовалюти є надзвичайно актуальним, популярним та корисним інструментом як для великих фінансових установ, так і для приватних трейдерів та інвесторів. Криптобіржі відіграють значну роль у обігу

криптовалют, але для того щоб обрати біржу з найвигіднішими умовами та найбільшим асортиментом треба витратити багато часу, сил та, цілком можливо, зайвих нервів.

Вирішенням вищеназваних проблем можна без перебільшень назвати криптоагрегатори. Криптоагрегатори дозволяють комбінувати, тобто агрегувати дані з різноманітних бірж у єдиний інформаційний потік в реальному часі. Деякі з них дозволяють не тільки переглядати ці дані, а й напряму проводити різні операції з ними на єдиній платформі.

Агрегатори використовують dApps, смарт контракти, оракулів та, звісно ж, API щоб збирати дані з різноманітних DEX та CEX бірж на єдиній платформі. Таким чином, трейдерам не треба робити зайвих кроків у вигляді переключень між біржами, щоб знайти накращу ціну актива.

Агрегатори криптовалют можуть бути корисні не тільки звичайним трейдерам та інвесторам, але ще й великим фінансовим установам. Користувачі можуть отримати надзвичайно цінну інформацію в єдиному місці з єдиним зручним інтерфейсом. Також деякі агрегатори можуть напряму зв’язувати користувачів з біржами, dApps, інвестиційними платформами і т.д.

До переваг використання агрегаторів можна віднести наступні:

- Значне збереження часу: агрегатор дозволяє дослідити бажані активи з різних криптобірж в одному місці
- Допомагають прийняти правильне рішення: завдяки наявності інформації від різних провайдерів, можна знайти найвигідніші пропозиції
- Широкий вибір активів: завдяки агрегації даних з різних майданчиків, можна знайти новий проєкт, який з часом може стати популярним та значно зрости в ціні.

Проаналізувавши ринок та сферу, було прийняте рішення про розробку власної інформаційної системи для агрегування даних з криптобірж із наступним функціоналом:

- Зручний та інтуїтивний інтерфейс.

- Підтримка декількох криптобірж.
- Відображення історичних графіків для кожної валюти.
- Про кожний актив має бути відома його: ціна, капіталізація, об’єм та інші відомості.
- Система повинна напряму брати дані з бірж шляхом використання API від криптобірж.
- Можливість реєстрації користувача в системі.
- Можливість додавання активу в список обраних для подальшого відслідковування.

2. ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ АГРЕГУВАННЯ ДАНИХ КРИПТОВАЛЮТНИХ БІРЖ

2.1 Фреймворк ASP.NET Core та патерн MVC

Фреймворк ASP.NET MVC був випущений всесвітньо відомою корпорацією Майкрософт у 2007 році. Даний фреймворк базується на паттерні MVC (Model-View-Controller) та розв'язує руки розробникам, які були невдоволені використанням WebForms, авторства Майкрософт. WebForms було створено з метою спростити роботу для клієнт-серверних розробників, які займались веб-розробкою, та варто зазначити, що дана технологія була доволі успішна та користувалась популярністю. Тим не менш, з часом розробники стали більш вибагливими до веб-розробки, деякі хотіли дещо більше контролю над фінальним виглядом отриманого продукту, інші бажали щоб фреймворк підтримував певний сталий паттерн для веб розробки. Саме щоб вирішити ці потреби тогочасних розробників був створений фреймворк ASP.NET MVC [34].

Патерн Model-View-Controller (Модель-Вигляд-Контроллер) є одним з найбільш відомих та використовуваних патернів у світі. Його історія почалась у вигляді фреймворку, за авторством Трюгве Реенскауга (Trygve Reenskaug) для платформи Smalltalk в кінці 1970х. З того часу даний патерн відіграє надзвичайно важливу роль у UI фреймворках та у сприйнятті UI вцілому.

Model (Модель) – це об'єкт, який відображає інформацію домену. Це невізуальний об'єкт, який зберігає всю інформацію, яка використовується для UI [35]. В ідеальному випадку модель повинна бути ідеально чистою, тобто не містити ніякої валідації чи будь-якої іншої бізнес логіки. Насправді, містить модель бізнес логіку чи ні, залежить виключно від обраної мови програмування та фреймворку [34]. Наприклад, Entity Framework Core містить багато анотацій даних, які є механізмом формування таблиць бази даних та засобом перевірки у веб-додатках ASP.NET Core.

View (Вигляд) – відповідає за відображення моделі в користувацькому інтерфейсі. Якщо наша модель це користувацький об’єкт, то наш view може виглядати як сторінка повна віджетів, або HTML сторінка, на якій відображені відомості з моделі [34]. Вигляд, це, як не важко здогадатись, лише візуальна репрезентація даних. Всі зміни, які можуть бути внесені через view до моделі повинні оброблятись іншою інстанцією – контролером.

Controller (Контролер) – це розум додатку побудованого на даному фреймворку. Контролер обробляє всі запити від користувача (через view) або клієнта (через API) використовуючи різноманітні методи та в подальшому коректно їх опрацьовує, відповідно до бізнес логіки додатка. Результати операцій потім можуть бути повернуті клієнту, знову ж, використовуючи контролер. В ідеальному випадку контролер повинен бути легким та делегувати запити іншим компонентам або сервісам [34]. Це сприяє підвищенню обслуговуваності та зручності тестування .

Так само як Entity Framework Core є повним переосмисленням Entity Framework 6, ASP.NET Core – це переосмислення надзвичайно популярного ASP.NET Framework. Це переосмислення було необхідним, оскільки потрібно було прибрати залежність від System.Web. Саме завдяки цьому, стало можливим використовувати додатки, написані на новому фреймворку, на системах відмінних від Windows та різноманітних веб-серверах окрім Internet Information Services (IIS). Саме це зробило можливим використання нового, кросплатформного, простого, швидкого веб-сервера Kestrel. Саме Kestrel дозволяє уніфікувати розробку на всіх платформах.

Фреймворк ASP.NET Core дозволяє створювати додатки на Razor Pages, з паттерном MVC, RESTful сервіси, SPA додатки, використовувати різноманітні JavaScript фреймфорки такі як Angular або React. В той же час, при тому що в нас є такий широкий вибір для створення додатку серед різноманітних технологій, базовий фреймворк, який лежить в основі розробки залишається незмінним незалежно від вибору решти технологій.

2.2 Dependency Injection

Механізм Dependency Injection (DI або ін’єкція залежностей) – це механізм, вбудований в ASP.NET Core, який був розроблений для підтримки слабкого зв’язку між об’єктами. Замість безпосереднього створення залежних об’єктів або передачі конкретних реалізацій у класи та/або у методи, параметри визначені інтерфейсами. Таким чином, будь-яка реалізація інтерфейсу може бути передана в класи або методи та класи, значно підвищуючи гнучкість програми.

Підтримка DI є одним з основних принципів у новій переписаній версії ASP.NET Core. Не тільки Startup приймає всі служби конфігурації та проміжного програмного забезпечення через ін’єкції залежностей, а і будь-які користувацькі класи можуть (та повинні) додаватися до контейнера сервісу для подальшої ін’єкції в інші частини програми [34]. Коли елемент налаштований в ASP.NET Core DI контейнері, існують 3 варіанти його життєвого циклу, які продемонстровані в таблиці 2.1.

Таблиця 2.1 – Варіанти життєвого циклу сервісів

Тип життєвого циклу	**Надає функціональність**
Transient	Створюється кожного разу при необхідності.
Scoped	Створюється кожного разу при запиті. Рекомендується використовувати з об’єктами Entity Framework.
Singleton	Створюється при першому запиті і потім повторно використовується протягом усього життя об’єкта.

2.3 ADO.NET

ADO.NET – це набір класів, які значно спрощують роботу з даними для розробників .NET Core. ADO.NET надає багатий набір компонентів для

створення розподілених програм для роботи з даними. Це невід'ємна частина .NET Core, яка забезпечує доступ до реляційних даних, XML та даних додатків. ADO.NET дозволяє вирішувати різноманітні проблеми розробників, включаючи створення клієнтів для баз даних та бізнес-об'єктів середнього рівня, які використовуються додатками, інструментами, мовами та браузерами [36].

ADO.NET не надає єдиного набору об'єктів, які взаємодіють із кількома СУБД. Навпаки, ADO.NET підтримує кілька постачальників даних, кожен з яких оптимізований для взаємодії з конкретною СУБД. Перша перевага такого підходу полягає в тому, що можна запрограмувати певного постачальника даних для доступу до будь-яких унікальних функцій певної СУБД. Друга перевага полягає в тому, що конкретний постачальник даних може підключитися до базового механізму СУБД без проміжного репрезентаційного рівня [34].

Постачальник даних – це набір типів, що визначені в заданому просторі імен, що розуміють як комунікувати з конкретним типом джерала даних. В залежності від того, який провайдер використовується, кожний визначає набір класів для базового функціоналу [34]. Таблиця 2.2 відображає декілька базових класів та ключові інтерфейси.

Таблиця 2.2 – Базові класи ADO.NET

Базовий клас	**Інтерфейс**	**Опис**
DbConnection	IDbConnection	Дозволяє підключатись та відключатись від бази даних. Об'єкт підключення також дає доступ до пов'язаного з транзакцією об'єкта.
DbCommand	IDbCommand	Відображує SQL запит або збережену процедуру.
DbDataReader	IDataReader, IDataRecord	Дозволяє доступ до перегляду даних.

DbDataAdapter	IDataAdapter, IDbDataAdapter	Передає об’єкти типу DataSet між БД та користувачем.
DbParameter	IDataParameter, IDbDataParameter	Іменований параметр всередині параметризованого запиту.
DbTransaction	IDbTransaction	Інкапсулює транзакцію в БД.

Незважаючи на те, що назви цих класів різні, залежно від постачальників даних (SqlConnection, OdbcConnection), кожен з них наслідується від одного базового класу (DbConnection у випадку об’єктів підключення), і також реалізують однакові інтерфейси [34].

На рис. 2.1 схематично зображено принцип роботи постачальників даних в ADO.NET.

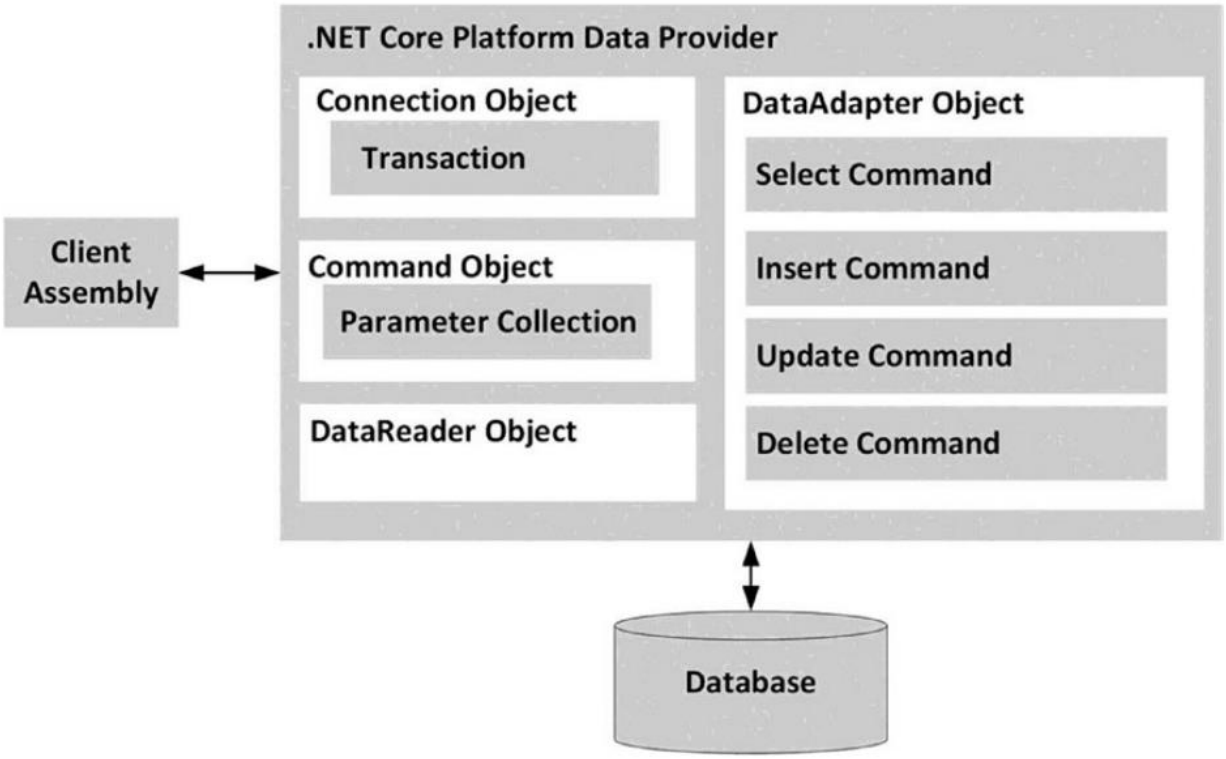

Рисунок 2.1 – Постачальники даних ADO.NET надають доступ до обраної СУБД

2.4 Entity Framework Core та концепція ORM

Як відомо, ADO.NET дозволяє виділяти, вставляти, оновлювати та видаляти дані використовуючи підключення, команди та зчитувачі даних. Для коректної роботи будь-якого додатку цього більш ніж достатньо, але ці аспекти ADO.NET примушують розробника поводитись з оброблюваними даними так, ніби вони дуже тісно пов’язані з фізичною таблицею бази даних [34].

Використовуючи ADO.NET необхідно постійно пам’ятати фізичні типи з серверної бази даних, необхідно знати схему кожної таблиці, створювати потенційно складні SQL запити для взаємодії з таблицею (таблицями), відслідковувати зміни в отриманих (чи змінених) даних, тощо. Таким чином, розробник може написати надзвичайно багато рядків коду виключно для елементарної взаємодії з БД, оскільки зазвичай бази даних проєктуються з метою швидкодії та фокусуються на зовнішніх ключах, збережених процедурах, нормалізації даних.

.NET підтримує спеціальні об'єктно-реляційні маппери (object-relational mapping framework), інакше кажучи ORM, які значно спрощують стандартні операції видалення, читання, оновлення та виділення даних (CRUD) для розробників. Розробник створює маппінг між об’єктами .NET та реляційною базою даних, а вже безпосередньо сама ORM відповідає за підключення, генерацію запитів та збереження даних [34]. Завдяки цьому, розробник може сфокусуватись безпосередньо на бізнес-вимогах додатку.

Всередині Entity Framework Core використовує інфраструктуру ADO.NET. Так само як ADO.NET взаємодіє зі сховищем даних, точно так само Entity Framework Core неявно використовує постачальників даних ADO.NET для всіх взаємодій з базою даних. Головна перевага всієї даної схеми в тому, що можна поєднувати обидві технології в одному проєкті, значно збільшуючи можливості при розробці.

Entity Framework Core найкраще підходить для процесу розробки в ситуаціях forms-over-data (або API-over-data). Операції з невеликою кількістю сутностей, та використання патерну unit of work для забезпечення послідовності – це ідеальний варіант використання даної технології. Однак, EF Core не дуже підходить для операцій з великою кількістю даних.

2.5 Багаторівнева архітектура додатку

Коли різні компоненти системи систематично організовані, це називається системною архітектурою. Архітектура – це розподіл системи в масштабі підприємства на різні рівні, кожен з яких відповідає за певну частину системи та має мінімальний прямий вплив на інші рівні [37].

Існує багато способів ділення системи на такі логічні рівні.

2.5.1 Монолітні додатки

Монолотні програми – це здебільшого доволі прості програми. Для них немає необхідності враховувати мережеві підключення та різноманітні мережеві збої, оскільки скоріше за все, вони не вимагають підключення до мережі.

Оскільки всі дані зберігаються в одній програмі, розробники не фокусуються на синхронізації даних. При фізичному розділенні рівнів продуктивність додатку суттєво знижується, оскільки мережевий зв’язок завжди вимагає додаткових часових витрат [37]. Саме через це однорівневі додаткі, безумовно, мають найкращу продуктивність з усіх можливих типів.

2.5.2 N-рівневі додатки

Зазвичай, базові рівні будь-якого n-рівневого додатку включають з наступні: UI (User Interface або користувацький інтерфейс), BLL (Business Logic Layer або рівень бізнес-логіки) та DAL (Data Access Layer або рівень доступу до даних). Схематично це можна побачити на рисунку 2.2. Використовуючи таку

архітектуру, користувач взаємодіє лише з UI, який в свою чергу взаємодіє лише з BLL, який теж напряму взаємодіє тільки з DAL. BLL звісно ж, може робити запити до DAL для отримання даних. Рівень UI не може робити прямих запитів до DAL. Кожний рівень повинен взаємодіяти тільки з найближчим до себе, не порушуючи ієрархії. Таким чином, кожен рівень має лише власну зону відповідальності [38].

До недоліків такої архітектури можна віднести те, що залежності компіляційної послідовності будуть йти знизу нагору. Тобто рівень UI залежить від BLL, який в свою чергу залежить від рівня DAL. Це значить, що BLL, який зазвичай включає в себе всю найважливішу логіку додатку, залежить від деталей реалізації рівня доступу до даних (а також на існуванні бази даних в принципі). Тестування бізнес логіки в додатку з такою архітектурою зазвичай доволі складне та потребує тестової бази даних [38].

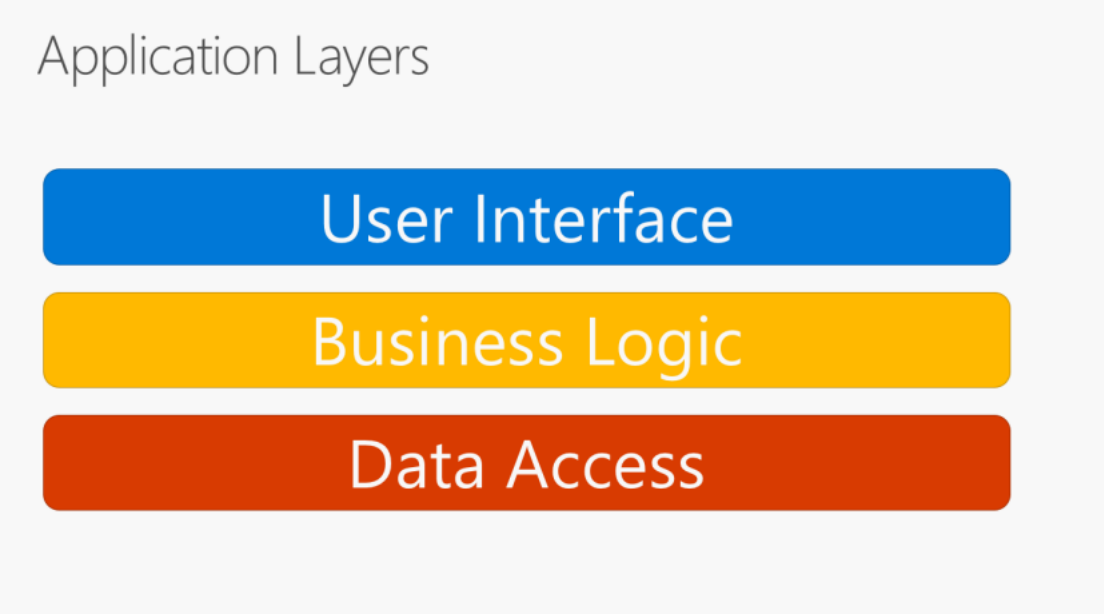

Рисунок 2.2 – Схематичне зображення трирівневої архітектури додатків

Ця сепарація дозволяє суттєво полегшити можливості для обслуговування програми. Переваги такої архітектури полягають в тому, що всі бізнес-правила централізовані, що значно полегшує їх створення, використання та повторне

використання. Доступ до даних також централізований, що має такі ж переваги, як і централізовані бізнес-правила. Централізація процедур доступу до даних також дуже корисна, коли діло доходить до обслуговування, оскільки зміни треба вносити лише в одному місці. Недоліків у такої архітектури досить мало, однак можна назвати дещо більший час розробки, оскільки треба написати багато окремих компонентів, які будуть взаємодіяти разом [37]. Також можна сказати, що така архітектура може дещо заплутати менш досвідчених розробників.

2.6 Патерн “Репозиторій”

Зазвичай, складним системам необхідний додатковий рівень, який ізолював би об’єкти рівня бізнес логіки від деталей рівня доступу до бази даних. В таких системах варто створити варто створити додатковий рівень абсторакії поверх рівня відображення, де і буде зосереджено код запитів до БД. Дана концепціє стає актуальнішою при наявності великої кількості класів у рівні бізнес-логіки та дуже багато різноманітних запитів до БД. Конкретно в такому випадку, додавання такого рівня може допомогти мінімізувати дублювання логіки запитів.

Патерн Репозиторій є посередником між рівнями бізнес логіки та відображення даних, діючи як вбудована колекція об’єктів бізнес логіки. Об’єкти з клієнтського боку можуть декларативно створювати особливості запитів та передавати їх в Репозиторій для підтвердження. Об’єкти можуть буди додані або видалені з Репозиторію, так само, як і з простої колекції об’єктів, а код рівня презентації, який теж інкапсульований Репозиторієм, виконуватиме відповідні операції за лаштунками. Взагалі, концептуально Репозиторій інкапсулює набір об’єктів, що зберігаються в сховищі даних, та операції що над ними виконуються, надаючи ґрунтовніше об’єктно-орієнтоване уявлення про рівень доступу до даних. Також даний патерн дозволяє здійснити чисту сепарацію

створити односторонню залежність між рівнем бізнес логіки та рівнем доступу до даних [35].

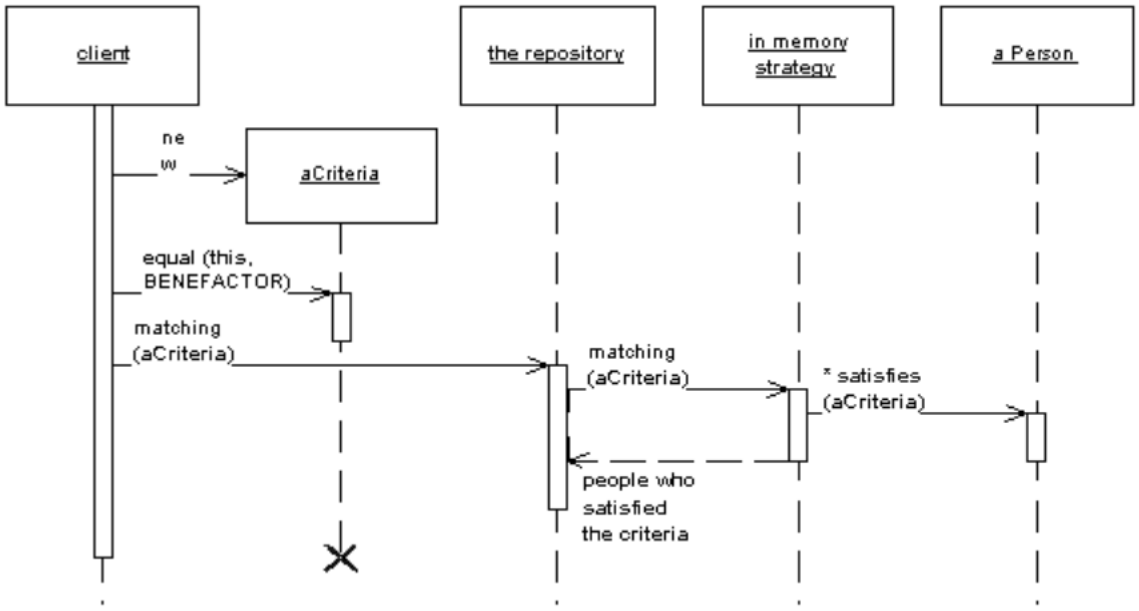

Рисунок 2.3 – Візуальна репрезентація паттерну Репозиторій

У великій системі з великою кількістю об’єктів рівня бізнес логіки та, відповідно, великою кількістю запитів, Репозиторій зменшує кількість коду, який необхідний для обробки всіх запитів. Даний патерн підтримує патерн Специфікації, який інкапсулює запит, що виконується суто об’єктно-орієнтованим способом. Саме тому, завдяки цьому, можна прибрати весь код для налаштування об’єкту запита в специфічних випадках. Розробникам більше не треба думати про SQL, а треба писати запити виключно з думкою про об’єкти.

2.7 Патерн “Unit of Work”

Коли постійно витягуються та змінюються сутності в базі даних, надзвичайно важливим є постійний контроль зробленого, інакше ці зміни не будуть збережені. Так само під час додавання чи видалення сутностей обов’язково треба контролювати цей процес.

Звісно, можна змінювати базу даних постійно, кожного разу коли об’єкт змінюється, але це може призвести до величезної кількості маленьких запитів, які потім призведуть до значних втрат швидкодії додатку. Крім того, це вимагає тримати процес транзакції постійно відкритим, що стає дуже непрактичним у випадку, якщо у розробника є бізнес-транзакція, що охоплює декілька запитів. Ситуація може стати ще гірше, якщо необхідно відстежувати нещодавно зчитані об’єкти, для уникнення непослідовності зчитування.

Саме тут і приходить на допомогу патерн Unit of Work, тобто одиниця роботи. Даний патерн відстежує все що відбувається під час бізнес-транзакції, що може вплинути на базу даних. Коли все закінчено – він сам відстежує все що треба зробити для запиту на зміну бази даних.

Очевидно, що треба відслідковувати створення, видалення та зміну об’єктів в базі даних. Unit of Work стає об’єктом, який саме це і робить. Кожного разу коли виконується щось, що може змінити сутність в базі даних, необхідно повідомити про це Unit of Work. Також можна повідомляти йому про зчитування даних, щоб в подальшому контролювати послідовність зчитування.

Ключовий фактор Unit of Work в тому, що коли справа йде до коміта в базу даних, то саме він вирішує що робити далі. Він відкриває транзакцію, виконує перевірки паралельності виконання та записує зміни до БД. Програмісту не потрібно явно викликати методи для оновлення бази даних. Таким чином, їм не потрібно стежити або хвилюватися за зміни, або як саме вони повинні змінювати базу даних.

Під час реєстрації абонента (рис. 2.4) користувач повинен не забути зареєструвати об’єкт Unit of Work для подальших змін. Будь-які незареєстровані об’єкти не будуть записані до коміта. Хоча через це можуть бути проблеми, таким чином існує певна гнучкість, що дозволяє вносити зміни, які не треба записувати до бази даних.

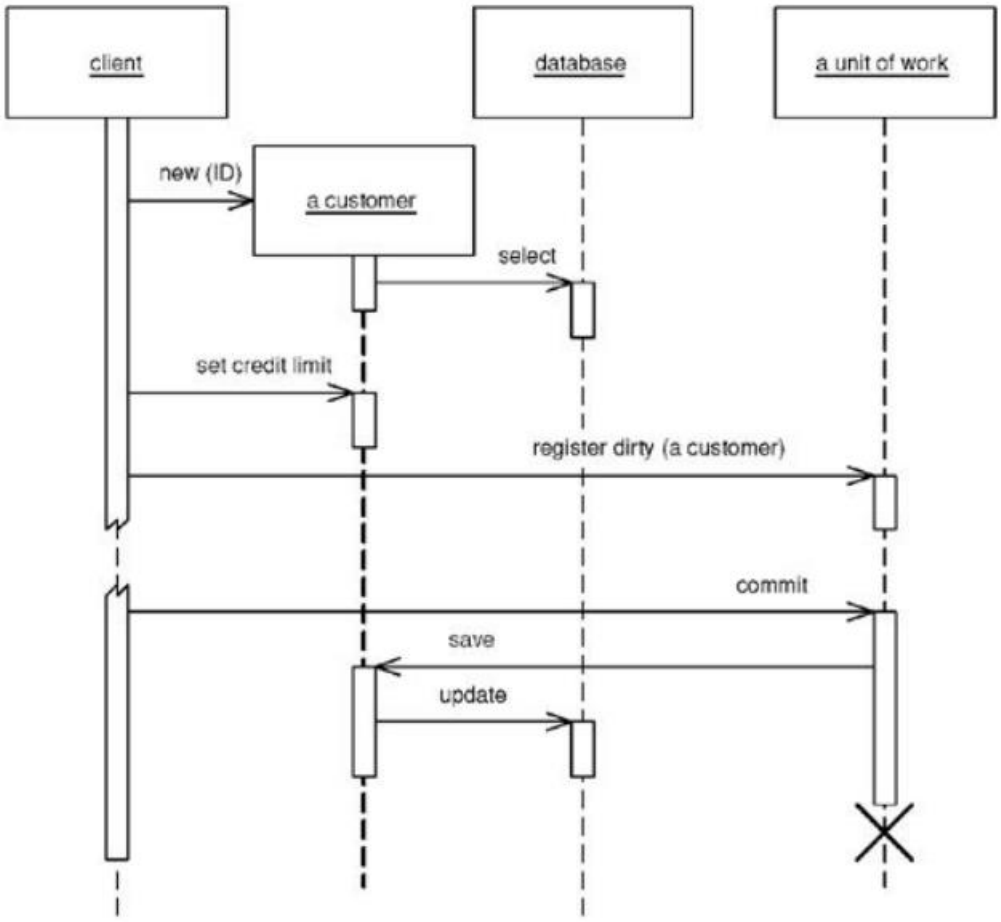

Рисунок 2.4 – Реєстрація зміненого об'єкту абонентом

Для зручності реєстрації об’єкта Unit of Work доцільно розмістити методи реєстрації прямо в методах об’єкта, який буде змінюватись.

Також можна використовувати даний патерн в якості контролера для бази даних. На рисунку 2.5 можна побачити схему, згідно якої Unit of Work контролює всі зчитування з бази даних. Під час зчитування створюється копія, з якою потім порівнюється об’єкт перед комітом. Незважаючи на те, що таким чином дещо ускладнюється процес коміта, це дозволяє вибірково оновлювати лише необхідні нам поля. Таким чином значно знижується навантаження на базу даних.

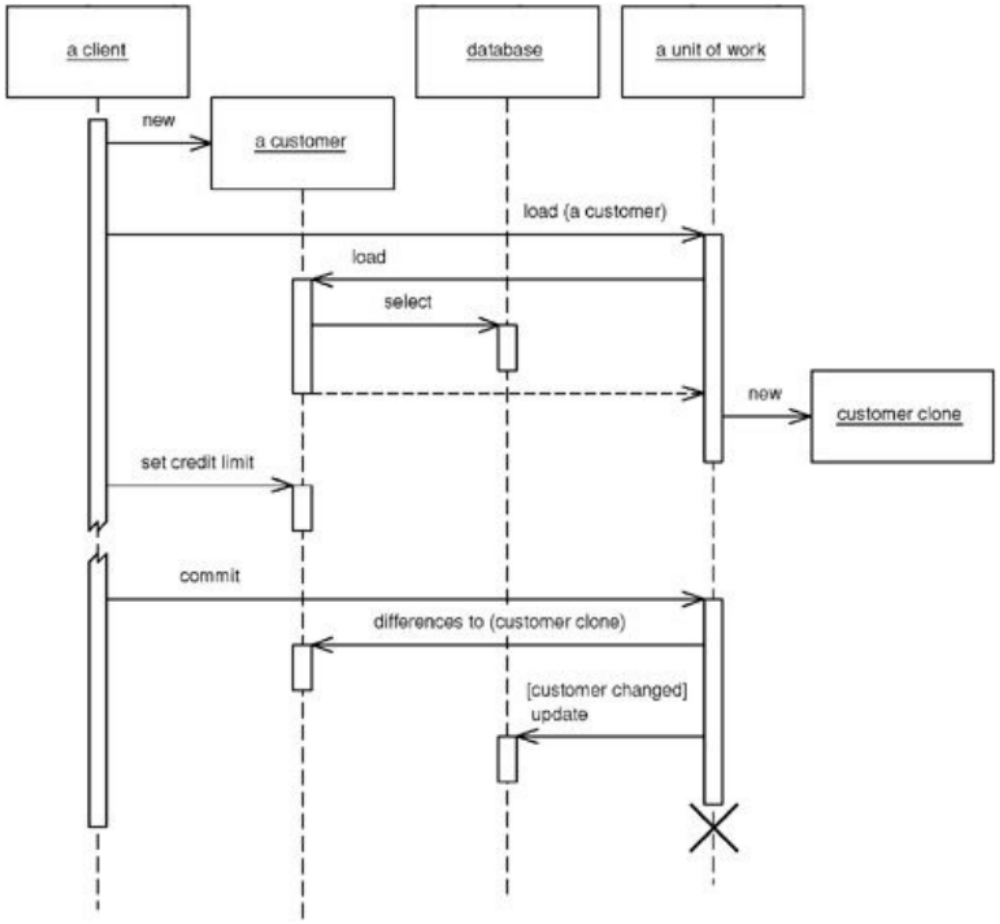

Рисунок 2.5 – Використання Unit of Work в якості контролера для бази даних

У фреймворку .NET реалізація Unit of Work виконана через Disconnected DataSet з вищезгаданого ADO.NET. Через це вигляд паттерну трішки відрізняється від класичних варіантів. Якщо в інших мовах паттерн буде просто реєструвати зміни в об’єктах та відслідковувати їх, то в .NET він зчитує дані з бази даних в тип DataSet. Цей набір даних є внутрішнім відображенням результату одного або декількох SQL запитів. Кожна сутність має такі значення, як версія (поточна, оригінальна та пропонована) та стан (незмінено, додано, видалено, модифіковано), які разом з тим фактом, що DataSet повторює фізичну структуру бази даних, робить внесення змін до БД досить простим та прямолінійним.

3. ПРОЄКТУВАННЯ СИСТЕМИ АГРЕГАТОРА КРИПТОВАЛЮТНИХ МАЙДАНЧИКІВ

3.1 Формування вхідних даних для інформаційної системи

Оскільки головна мета інформаційної системи для агрегування даних – це, очевидно, саме агрегування даних, то для початку ці дані треба отримати. Як вже було згадано, різноманітні криптобіржі надають власні API розробникам, за допомогою яких вже безпосередньо можна отримати відомості про активи.

Проаналізувавши велику кількість API від різних криптобірж, було прийняте рішення обрати 3 наступні: FTX, Kucoin, Bitfinex. Такий вибір було зроблено, оскільки вищезгадані біржі надають дані через публічні API у зручному вигляді саме для подальшої агрегації. На рисунках 3.1-3.4 зображено приклад даних, що надходять від API Kucoin та Bitfinex.

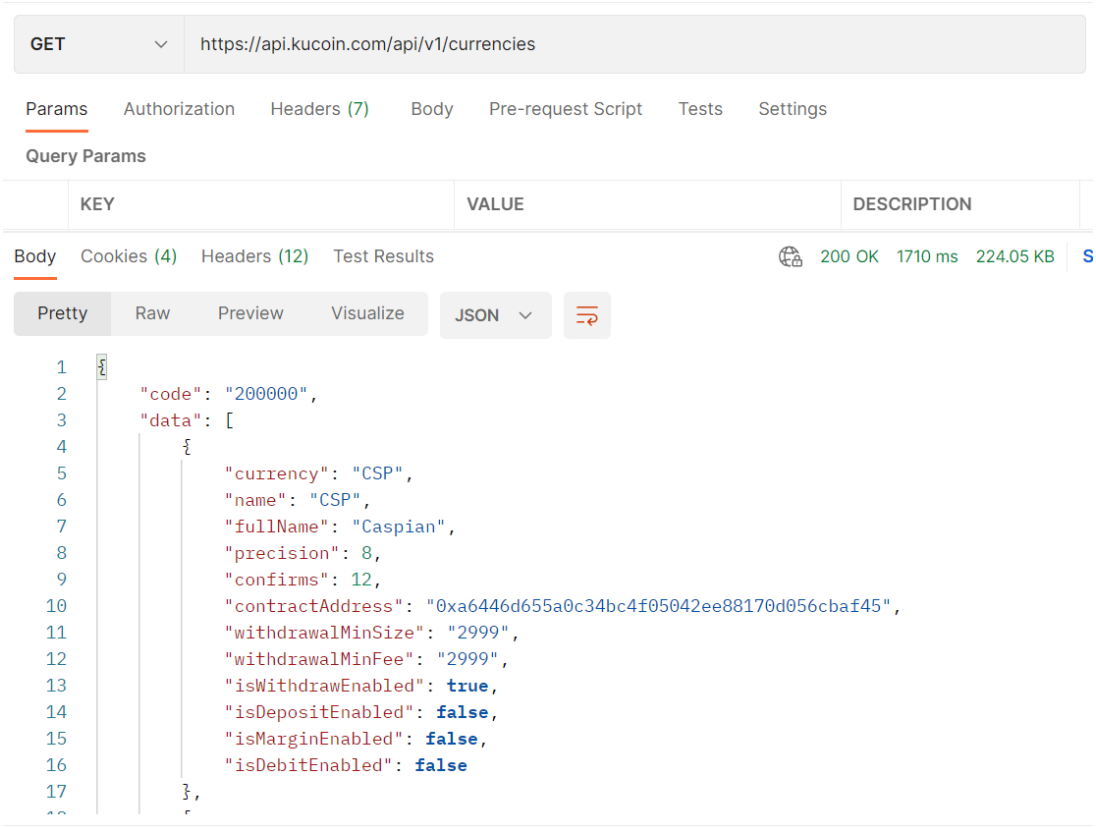

Рисунок 3.1 – Запит для отримання всіх доступних активів від біржі Kucoin

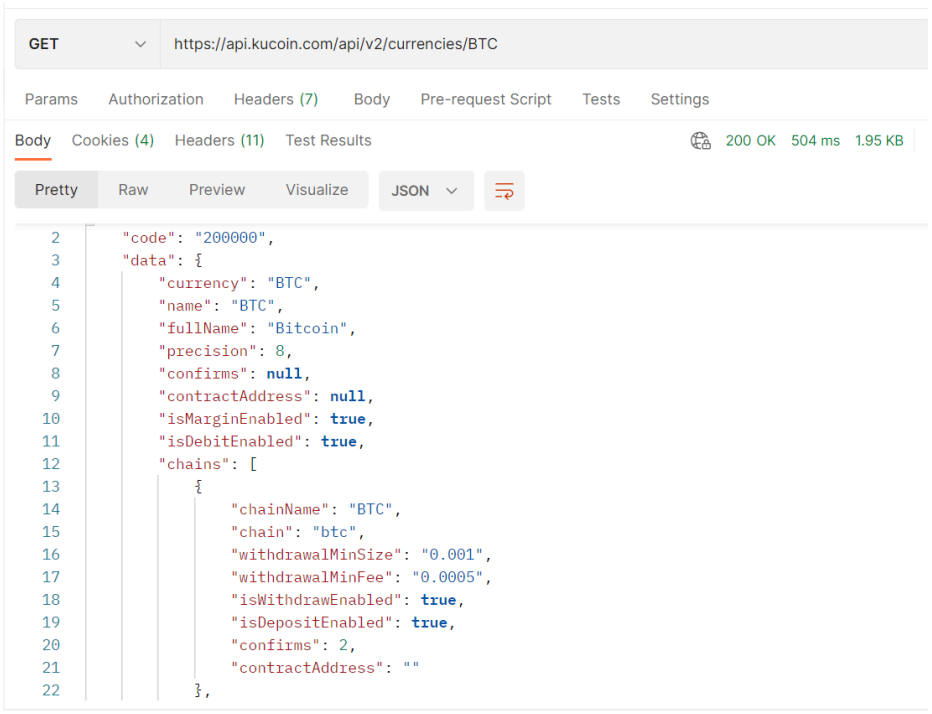

Рисунок 3.2 – Запит для отримання інформації про конкретний актив (BTC) від біржі Kucoin

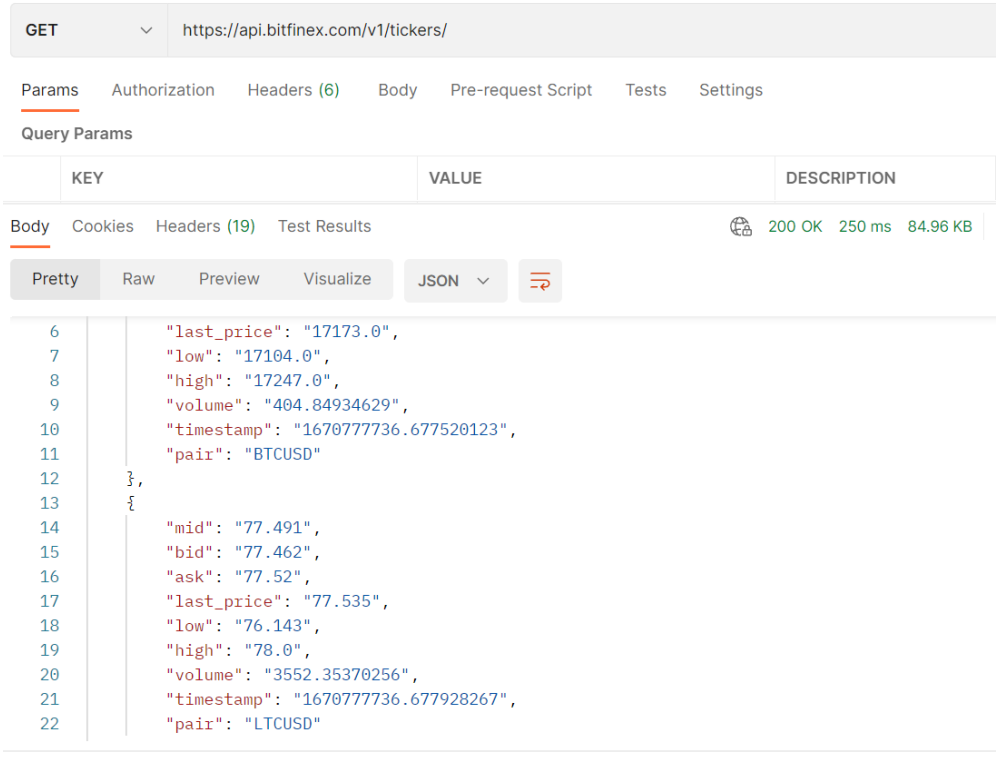

Рисунок 3.3 – Запит для отримання інформації про всі торгові пари від біржі Kucoin

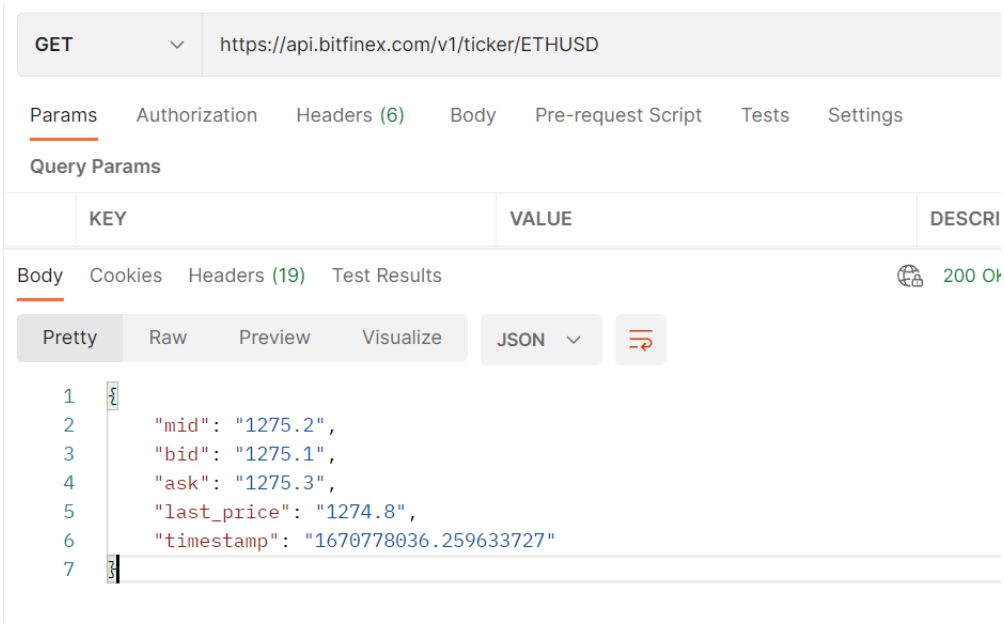

Рисунок 3.4 – Запит для отримання даних про конкретну торгову пару (ETHUSD) від біржі Kucoin

Після отримання даних з API, необхідно обрати спільну інформацію з усіх бірж, яка буде найкорисніша для користувача. В нашому випадку під ці критерії підходять: назва активу, символьне позначення, актуальна ціна, торговий об’єм за останні 24 години та зміна ціни у відсотках за останні 24 години.

3.2 Опис програмних засобів та бібліотек

Для виконання роботи використовується програмне середовище Microsoft Visual Studio 2022 разом з деякими бібліотеками та модулями (таблиця 3.1), Postman, Microsoft SQL Server Management Studio 2018. В якості мови програмування був обраний C# та фреймворк ASP.NET Core MVC.

Microsoft Visual Studio – інтегроване середовище розробки від корпорації Майкрософт. Підтримує велику кількість мов програмування та дозволяє створювати різноманітні типи додатків, такі як звичайні консольні програми, веб-додатки, програми з графічним інтерфейсом, вебслужби, серверні додатки, додатки з нейромережами.

Дане середовище розробки має безліч функцій, які здатні значно полегшити життя розробнику, а особливо .NET розробнику, наприклад Intellicode (який дописує код на основі контексту, використовуючи нейромережі), інструменти для дебагу, зручний інтуїтивний інтерфейс та функція Hot Reload, яка дозволяє змінювати код під час активного процесу дебагінгу.

Postman – інструмент, який дозволяє розробникам тестувати свої API. Тестувальник може надсилати свої запити та опрацьовувати відповіді на них. Дана утиліта має простий інтерфейс, який дозволяє легко оперувати запитами. Під час використання додатку, якщо розробник хоче створити власні тести, йому не треба власний код для HTTP клієнта. Для цієї мети були створені колекції, в які вже і треба додавати тести для подальшої роботи з API. Взагалі, практично будь-яка потреба, яка може з'явитись у розробника під час тестування інтерфейсів, з великою долею ймовірності вже реалізована в даній програмі.

Microsoft SQL Server Management Studio – інтегроване середовище для розробки та керування будь-якою SQL інфраструктурою, починаючи з SQL Server та закінчуючи базами даних Azure. Дозволяє конфігурувати, відслідковувати та адмініструвати будь-які інстанції SQL Server та бази даних. Надає функціонал для розгортки, моніторингу та оновлення компонентів які пов'язані з даними, створювати запити та збережені скрипти. Дозволяє обробляти запити, проєктувати та керувати базами даних як з власного ПК, так і в хмарі Azure [39].

Таблиця 3.1 – Основні бібліотеки та фреймфорки проєкту

Назва	**Опис**
ASP.NET Core MVC	Фреймворк для створення веб-додатків, в основі якого лежить паттерн MVC.
EntityFramework Core	ORM система для роботи з базою даних у об'єктно-орієнтованому стилі.

ASP.NET Core Identity 2.0	Система для авторизації та аутентифікації користувачів у додатках побудованих на ASP.NET Core, Web Forms, Web Pages. Дозволяє створити кастомізовану функціональність для входу/виходу та кастомізувати інформацію про зареєстрованого користувача. Зберігає дані про користувачів у базі даних, яку створює автоматично. Базується на системі ролей, наприклад “Адміністратор”, “Клієнт”. Таким чином визначається рівень доступу серед користувачів додатку. У оновленні Identity 2.0 отримав можливість двохфакторної аутентифікації з використанням сторонніх сервісів. Також є можливість реєстрації та авторизації користувача з використанням інформації від сторонніх постачальників (Google, Facebook)
MimeKit	Open Source бібліотека для парсингу та надсилання електронних листів. В додатку використовується для підтвердження електронної пошти нових користувачів.
LINQ	Language-Integrated Query – бібліотека, яка дозволяє працювати з колекціями шляхом запитів. Візуально і функціонально синтаксис схожий на SQL, дозволяє фільтрувати, сортувати та групувати різноманітні типи даних використовуючи мінімальну кількість коду.
NSubstitute	Бібліотека для створення моків, тобто несправжніх даних, які можна використовувати для тестування.

xUnit	Бібліотека для тестування з відкритим кодом, яка підтримується широкою спільнотою розробників на добровільних умовах для фреймворку .NET. Була написана розробником відомої nUnit. Може допомагати в написанні тестів до C#, F# VB.NET та інших мов .NET.
FluentAssertions	Бібліотека з набором методів розширення, які дозволяють більш природньо описати очікуваний результат виконання тестів.

Таким чином, наведені вище програмні інструменти дають можливість для розробки додатку для агрегації даних, що буде виконувати описані раніше вимоги. Лістинг коду наведений в додатку А.

3.3 Результати програмної реалізації

У якості архітектури системи була обрана описана раніше багатошарова архітектура, яка дозволить в подальшому якісно краще обслуговувати додаток та додавати нові модулі. Графічно структура додатку зображена на рис. 19. Всього 4 рівні: Presentation Layer, Business Logic Layer, Data Access Layer та Utility Layer.

Структуру безпосередньо проєктів всередині рішення можна побачити на рис. 20. Як можемо спостерігати, всього 13 різних проєктів. Для кожного рівня окремо існують проєкти з інтерфейсами та безпосередньо проєкти, з класами, які ці інтерфейси реалізують. Зроблено це для можливості використання вбудованого механізму Dependency Injection (ін’єкції залежностей). Таким чином значно збільшується гнучкість додатку.

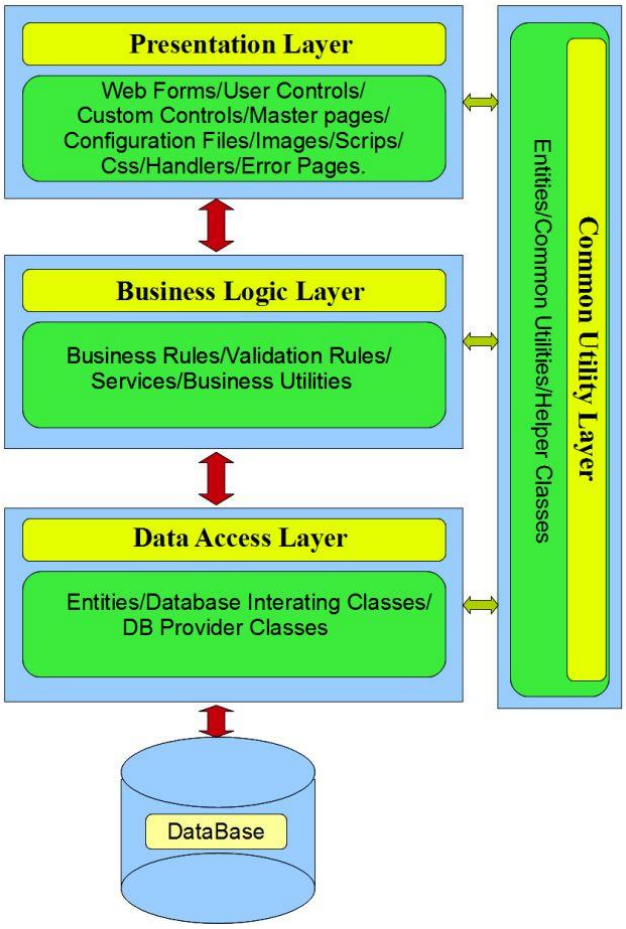

Рисунок 3.5 – Архітектура системи для агрегації даних

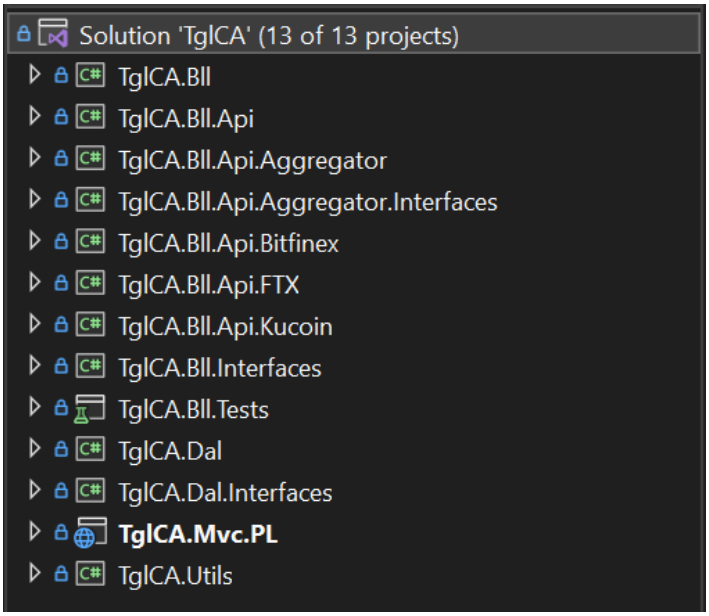

Рисунок 3.6 – Список проєктів всередині рішення

Таблиця 3.2 – Короткий опис кожного проєкту.

Назва	**Опис**
TglCA.Utils	Містить класи з різноманітними корисними функціями, що можуть бути використані в будь-якому рівні додатку (наприклад функція для отримання часу в форматі Unix)
TglCA.Dal.Interfaces	Містить інтерфейси для рівня доступу до даних. Включає в себе інтерфейси, які позначають сутності в базі даних (користувачі, активи), інтерфейси для патернів Репозиторій та Unit of Work.
TglCA.Dal	Описує реалізації інтерфейсів для роботи з об’єктами БД, також містить класи міграцій EntityFramework.
TglCA.Bll.Interfaces	Описує інтерфейси для сутностей рівня бізнес-логіки (наприклад UserModel або BllCurrency).
TglCA.Bll.Api	Містить інтерфейси та класи що їх реалізують для класів, що є базовими для майбутніх інстанцій постачальників API.
TglCA.Bll.Api.Aggregator.Interfaces	Містить інтерфейс для реалізації агрегатора даних.
TglCA.Bll.Api.FTX	Містить реалізацію класу-нащадка TglCA.Bll.Api для криптобіржі FTX.
TglCA.Bll.Api.Kucoin	Містить реалізацію класу-нащадка TglCA.Bll.Api для криптобіржі Kucoin.
TglCA.Bll.Api.Bitfinex	Містить реалізацію класу-нащадка TglCA.Bll.Api для криптобіржі Bitfinex.
TglCA.Bll.Api.Aggregator	Реалізує функціонал описаний в TglCA.Bll.Api.Aggregator.Interfaces та реалізує

	методи, що є базовими для даного додатку, зокрема для отримання агрегованої колекції активів.
TglCA.Bll	Містить реалізації різноманітних сервісів та маппери.
TglCA.Mvc.Pl	Рівень презентації, всередині містить структуру MVC, яка і використовує всі попередньо описані проєкти для відображення користувачу.
TglCA.Bll.Tests	Набір Unit-тестів.

Вигляд головної сторінки додатку можна спостерігати на рис. 3.7. Тут можна побачити список криптовалют, та дві кнопки для авторизації та реєстрації.

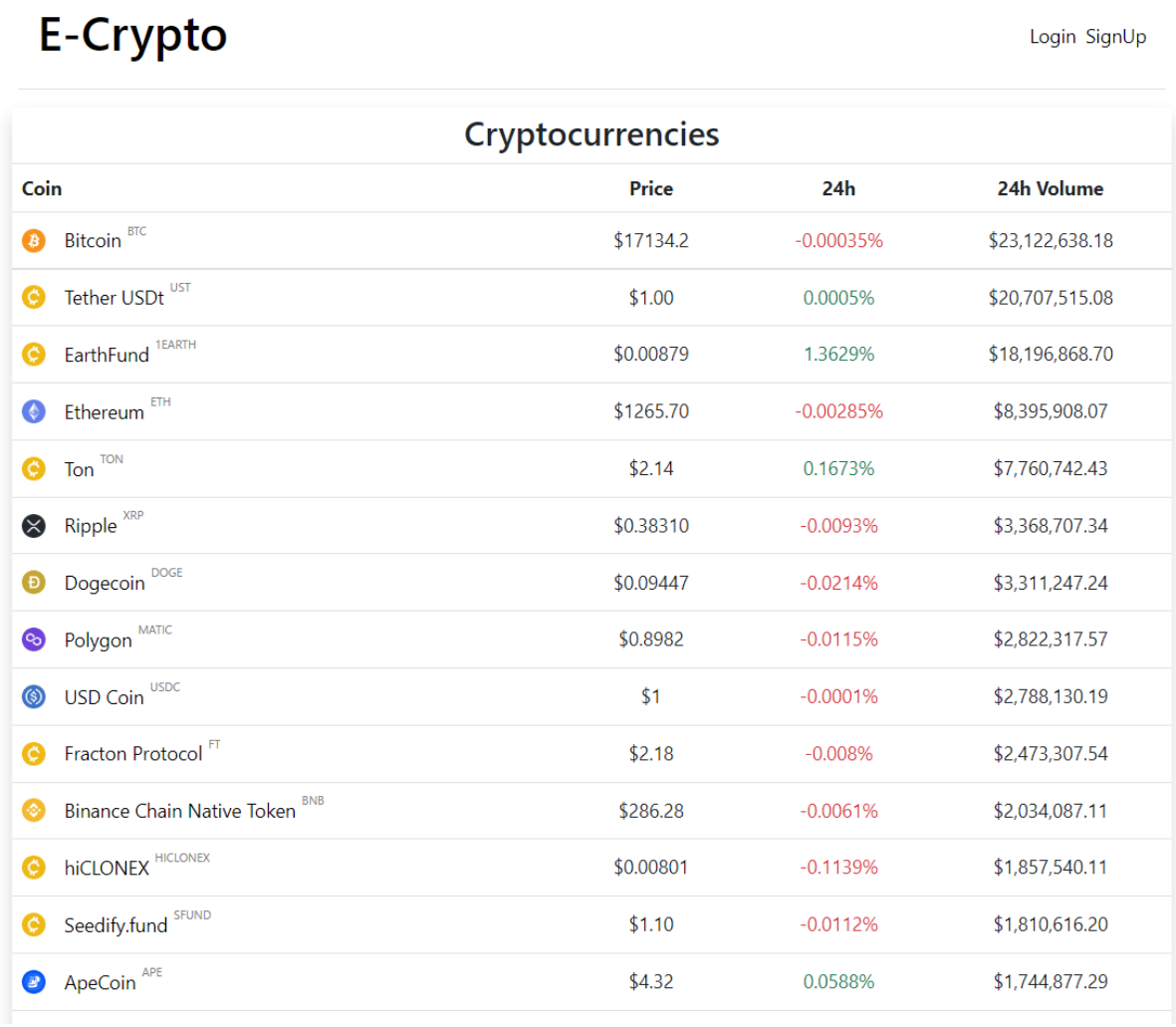

Рисунок 3.7 – Головна сторінка додатку

За виведення списку всіх криптовалют у агрегованому вигляді відповідає метод GetAggregatedCurrencies() з класу CoinAggregator проєкту TglCA.Bll.Api.Aggregator (рис. 3.8). Він викликає метод GetCurrenciesAsync() (який в свою чергу повертає колекцію типу BllCurrency з активами) для кожної біржі, потім обчислює середні значення однакових отриманих активів, які вже і вважаються агрегованими та додаються до результуючої колекції, яка далі передається безпосередньо з контролера у View.

```
public async Task<IEnumerable<BllCurrency>> GetAggregatedCurrencies()
{
    var tasks = new List<Task<IEnumerable<BllCurrency>>>();

    foreach (var p in providers)
    {
        tasks.Add(p.GetCurrenciesAsync());
    }

    var lists = await Task.WhenAll(tasks);

    var currencies = lists.SelectMany(l => l).GroupBy(x => x.Symbol);

    var result = new List<BllCurrency>();

    foreach (var currency in currencies)
    {
        result.Add(new BllCurrency()
        {
            Symbol = currency.Key,
            AssetName = currency.First().AssetName,
            Price = currency.Average(x => x.Price),
            PercentChange24h = currency.Average(x => x.PercentChange24h),
            Volume24hUsd = currency.Average(x => x.Volume24hUsd),
        });
    }
    return result;
}
```

Рисунок 3.8 – Метод для отримання списку всіх криптовалют у агрегованому вигляді

```
public override async Task<IEnumerable<BllCurrency>> GetCurrenciesAsync()
{
    var task1 = bitfinexClient.SpotApi.ExchangeData.GetAssetsAsync();
    var task2 = bitfinexClient.SpotApi.ExchangeData.GetTickersAsync();

    await Task.WhenAll(task1, task2);

    var currenciesNames = task1.Result;
    var currenciesStats = task2.Result;

    var filteredCurrenciesStats = currenciesStats.Data
        .Where(x => x.Symbol.EndsWith(QuoteAsset))
        .DistinctBy(x => x.Symbol).ToList();

    filteredCurrenciesStats.ForEach(x =>
    {
        var index = x.Symbol.LastIndexOf(QuoteAsset);
        if (index >= 2)
        {
            x.Symbol = x.Symbol.Substring(1, index - 1);
        }
    });

    var result = from x in currenciesNames.Data
                 join y in filteredCurrenciesStats
                 on x.Name equals y.Symbol
                 select new BllCurrency()
                 {
                     AssetName = x.FullName,
                     Symbol = x.Name,
                     Price = y.LastPrice,
                     Volume24hUsd = y.LastPrice >= 1 ? y.Volume / y.LastPrice : y.Volume * y.LastPrice,
                     PercentChange24h = y.DailyChangePercentage

                 };

    return result;
}
```

Рисунок 3.9 – Метод для отримання списку всіх активів від біржі Bitfinex

```
public override async Task<IEnumerable<BllCurrency>> GetCurrenciesAsync()
{
    var task1 = ftxClient.TradeApi.ExchangeData.GetAssetsAsync();
    var task2 = ftxClient.TradeApi.ExchangeData.GetSymbolsAsync();

    await Task.WhenAll(task1, task2);

    var currenciesNames = task1.Result;
    var currenciesStats = task2.Result;

    var filteredCurrenciesStats = currenciesStats.Data
        .Where(x => x.Name.EndsWith(QuoteAsset))
        .DistinctBy(x => x.BaseAsset).ToList();

    // Removing USD or USDT symbols.
    filteredCurrenciesStats.ForEach(x =>
    {
        var symbolSpan = new Span<char>(x.Name.ToCharArray());
        x.Name = symbolSpan.Slice(0, symbolSpan.IndexOf('/')).ToString();
    });

    var result = from x in currenciesNames.Data
                 join y in filteredCurrenciesStats
                 on x.Name equals y.Name
                 select new BllCurrency()
                 {
                     AssetName = x.FullName,
                     Symbol = x.Name,
                     Price = y.CurrentPrice.HasValue ? y.CurrentPrice.Value : decimal.MinValue,
                     Volume24hUsd = y.QuoteVolume24H,
                     PercentChange24h = y.Change24Hour,
                 };
    Console.WriteLine(currenciesNames.Data.Count());
    return result;
}
```

Рисунок 3.10 – Метод для отримання списку всіх активів від біржі FTX

```
public override async Task<IEnumerable<BllCurrency>> GetCurrenciesAsync()
{
    var task1 = kucoinClient.SpotApi.ExchangeData.GetAssetsAsync();
    var task2 = kucoinClient.SpotApi.ExchangeData.GetTickersAsync();

    await Task.WhenAll(task1, task2);

    var currenciesNames = task1.Result;
    var currenciesStats = task2.Result;

    var filteredCurrenciesStats = currenciesStats.Data.Data
        .Where(x => x.Symbol.EndsWith(QuoteAsset))
        .DistinctBy(x => x.Symbol).ToList();

    // Removing USD or USDT symbols.
    filteredCurrenciesStats.ForEach(x =>
    {
        var symbolSpan = new Span<char>(x.Symbol.ToCharArray());
        x.Symbol = symbolSpan.Slice(0, symbolSpan.IndexOf('-')).ToString();
    });

    var result = from x in currenciesNames.Data
                 join y in filteredCurrenciesStats
                 on x.Asset equals y.Symbol
                 select new BllCurrency()
                 {
                     AssetName = x.FullName,
                     Symbol = x.Asset,
                     Price = (decimal)y.LastPrice,
                     Volume24hUsd = (decimal)y.QuoteVolume,
                     PercentChange24h = (decimal)y.ChangePercentage
                 };

    return result;
}
```

Рисунок 3.11 – Метод для отримання списку всіх активів від біржі Kucoin

Наступним кроком, який можливо захоче зробити користувач, буде реєстрація. Виконати її можна двома способами: використавши існуючий обліковий запис Google, або створивши новий обліковий запис.

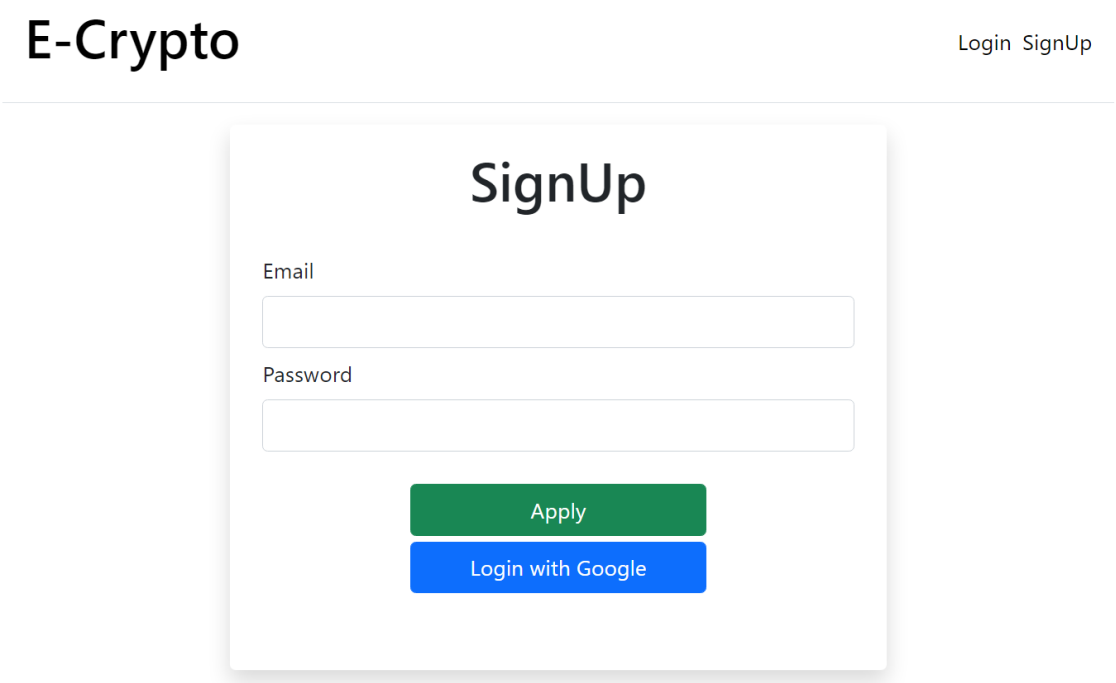

Рисунок 3.12 – Форма реєстрації

Після вводу даних в полі, на пошту буде надісланий лист з проханням підтвердити реєстрацію.

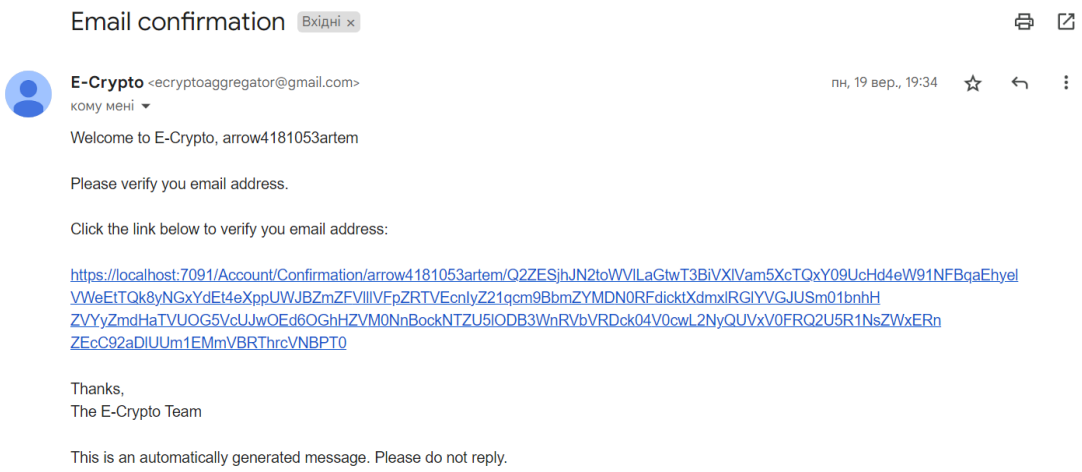

Рисунок 3.13 – Лист з підтвердженням реєстрації

Підтверджуємо реєстрацію, переходимо знову на сайт для авторизації, після цього вводимо дані в полі для авторизації та авторизуємось.

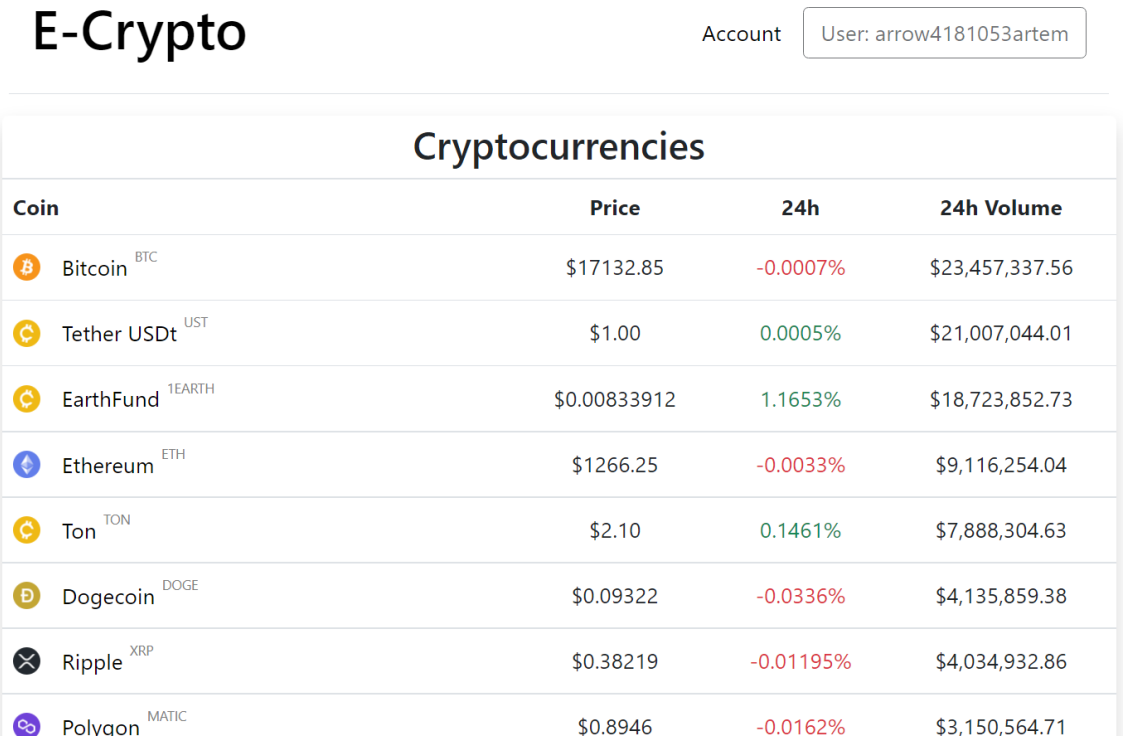

Рисунок 3.14 – Вигляд головної сторінки при авторизованому користувачі

Для того, щоб дізнатись більше про будь-яку представлену на сайті криптовалюту, достатньо натиснути на її назву. Також користувач може додавати активи до списку обраних, натиснувши на кнопку “Subscribe” на її сторінці (рис. 3.15). Також на даній сторінці ми можемо спостерігати графіки криптовалюти з бірж де вона наявна, та загальні відомості про неї окремо з кожної біржі, тобто не агреговані дані. Метод з CoinAggregator під назвою GetAggregatedChart(string symbol) повертає словник, який містить назву біржі та колекцію типу ChartPoint, яка позначає точку на графіку. У свою чергу, метод GetAggregatedCurrency повертає так само словник, проте в ньому в ролі ключа виступає назва біржі, а значення – об’єкт типу BllCurrency, який відображає криптовалюту. Функції, що виконують це представлені в додатку А.

Рисунок 3.15 – Сторінка криптовалюти, на яку ще не підписався користувач

Після підписки на декілька цікавих для користувача активів, можна перейти до списку з ними для подальшої взаємодії та моніторингу.

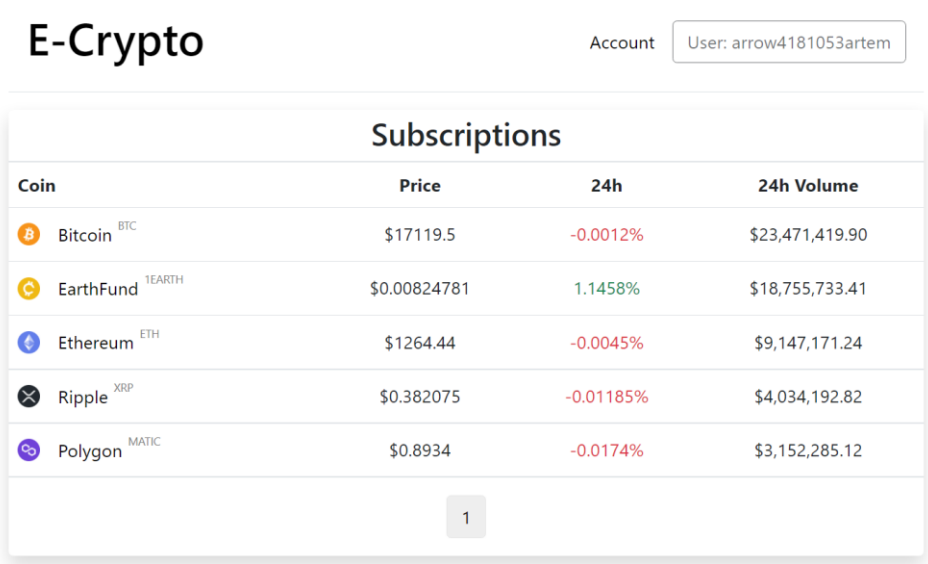

Рисунок 3.16 – Список обраних активів користувача

3.4 Unit-тестування

Оскільки кожна система завжди повинна працювати коректно, доцільно було написати unit тести, для перевірки коректності даних, які ми отримуємо через API від бірж. Дані тести перевіряють тип отримуваних даних, чи надходять певні дані взагалі, порядок їх надходження, чи правильно працюють написані функції для конвертування даних та агрегації. Лістинг всіх тестів наведений в додатку Б. Результати виконання тестів можна спостерігати на рис. 3.17.

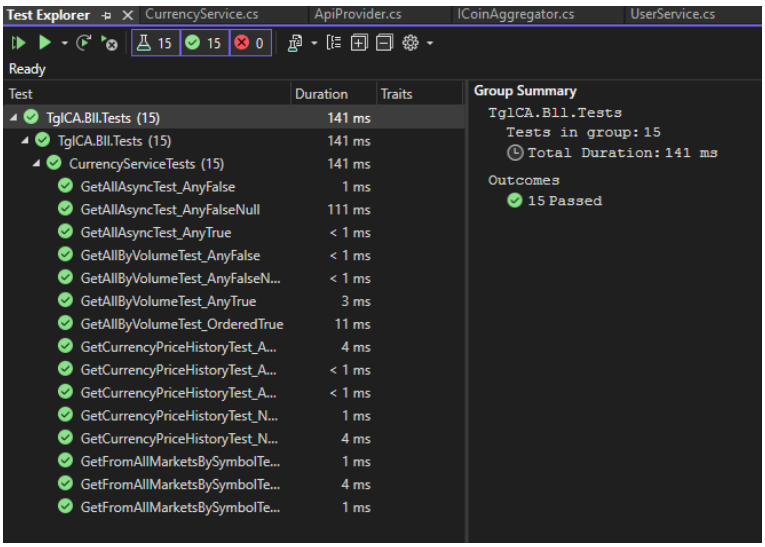

Рисунок 3.17 – Результати виконання unit тестів

ВИСНОВКИ

Отже, в рамках даної кваліфікаційної роботи було розглянуто сферу криптовалют та велику кількість пов’язаних тем. Зокрема були розглянуті криптобіржі, завдяки яким і здійснюється переважна частина фінансових операцій з криптовалютою та криптоагрегатори, які упорядковують дані з них в одному місці. Також було ґрунтовно розглянуте питання безпосередньо криптовалют, а саме деякі загальні відомості про них та подробиці реалізації технології блокчейн, на якій і базується більшість криптовалют.

Даний проєкт було реалізовано використовуючи мову програмування C# та фреймворк ASP.NET Core MVC. Також реалізована багаторівнева архітектура з рівнем презентації, рівнем бізнес-задач та рівнем доступу до даних. Також були використані патерни Репозиторій та Unit of Work. Дана архітектура дозволяє розділити ділянки коду для створення гнучких додатків. Коли ми розділяємо наш код на частини – таким чином в майбутньому буде набагато легше модифікувати код або додати нову частину, ніж переробляти весь додаток.

Як результат, можна сказати, що було розроблено першу в Україні інформаційну технологію, яка може агрегувати дані з криптовалютних бірж.

СПИСОК ЛІТЕРАТУРИ

1. TradingView [Електронний ресурс] – Режим доступу до ресурсу: https://tradingview.com/

2. CoinMarketCap [Електронний ресурс] – Режим доступу до ресурсу: https://coinmarketcap.com/

3. Abraham J., Higdon D., Nelson J., Ibarra J. Cryptocurrency price prediction using tweet volumes and sentiment analysis / SMU Data Science. – 2018.

4. Al-Jaroodi J., Mohamed N. Blockchain in industries: A survey / IEEE Access. – 2019.

5. Yli-Huumo J., Ko D., Choi S., Park S., Smolander K. Where is curreny research on blockchain technology? – A systematic review / PloS ONE. – 2016.

6. De Leon D., Stalick A., Jillepalli A., Haney M., Sheldon F. Blockchain: Properties and misconceptions / Asic Pacific Journal of Innovation and Enterpreneurship. – 2017. – 11 с.

7. Wang W., Hoang D., Hu P., Xiong Z., Niyato D., Wang P., Wen Y., Kim D. A Survey on Consensus Mechanisms and Mining Strategy Management in Blockchain Networks / IEEE Access – 2019.

8. Nakamoto S. Bitcoin, A peer-to-peer electronic cash system [Електронний ресурс] / – 2008. – Режим доступу до ресурсу: https://bitcoin.org/bitcoin.pdf

9. Yeow K., Gani A., Ahmad R., Rodrigues J., Ko K. Decentralized consensus from edge-centric Internet of Things: A review, taxonomy, and research issues / IEEE Access – 2018. – 1513-1524 с.

10. Garay J., Kiayias A., Leonardos N. The Bitcoin backbone protocol: Analysis and applications / Advances in Cryptology (EUROCRYPT) – 2015. – 281-310 с.

11. Nisan N., Roughgarden T., Tardos E., Vazirani V. Algorithmic Game Theory, vol. 1 / Cambridge University Press – 2007.

12. Athey S., Parashkevov I., Sarukkai., Xia J. Bitcoin pricing, adoption, and usage: Theory and evidence / Stanford Institute for Economic Policy Research – 2016.

13. Conti M., Kumar E., Lal C., Ruj S. A survey on security and privacy issues of bitcoin / IEEE Communications Surveys & Tutorials vol.20 – 2018. – 3416-3452 с.

14. Bonneau J., Miller A., Clark J., Narayanan A., Kroll J., Felten E. SoK: Research perspectives and challenges for Bitcoin and cryptocurrencies. / IEEE Symposium on Security and Privacy – 2015. – 104-121 с.

15. Debus J. Consensus methods in blockchain systems / Frankfurt School Finance & Management – 2017.

16. Kroll A., Davey I., Felten E. The economics of Bitcoin mining, or Bitcoin in the presence of advesaries / WEIS – 2013. – 1-21 с.

17. Garay J., Kiayias A., Leonardos N. The Bitcoin backbone protocol: Analysis and applications / Advances in Cryptology (EUROCRYPT) – 2015. – 281-310 с.

18. Rosenfield M. Analisys of Bitcoin pooled mining reward systems [Електронний ресурс] / – 2011. – Режим доступу до ресурсу: https://arxiv.org/abs/1112.4980

19. Ammous S. Can cryptocurrencies fulfil the functions of money? / Review of Economics and Finance – 2018. – 286-300 с.

20. Nabilou H. How to regulate bitcoin? Decentralized regulation for a decentalized cryptocurrency / International Journal of Law and Information Technology – 2019, – 266-291 с.

21. Chen Y., Bellavitis C. Decentralized finance: Blockchain technology and the quest for an open financial system / SSRN – 2019.

22. Lansky J. Possible state approaches to cryptocurrencies / Journal of Integrated Circuits and Systems – 2018. – 19-31 с.

23. European Central Bank Virtual Currency Schemes / European Central Bank: Frankfurt, Germany – 2012.

24. Monrat A., Schelen O., Andersson K. A survey of blockchain from the perspectives of applications, challenges, and opportunities / IEEE Access – 2019.

25. Ciaian P., Raicaniova M., Virtual relationships: short and long run evidence from Bitcoin and altcoin markets / Journal of International Financial Markets, Institutions & Money – 2018. – 173-195 с.

26. Bybit Learn: Utility Token [Електронний ресурс] – Режим доступу до ресурсу: https://learn.bybit.com/glossary/definition-utility-token/

27. Xia P., Wang H., Zhang B., Ji R., Gao B., Wu L., Luo X., Xu G., Characterizing cryptocurrency exchange scams / Computers & Security Journal – 2020.

28. Sabry F., Labda W., Erbad A., Malluhi Q., Cryptocurrencies and Artificial Intelligence: Challenges and Opportunities / IEEE Access – 2020.

29. DeVries P. An analysis of cryptocurrency, bitcoin, and the future / International Journal of Business Marketing and Management – 2016. – 1293-1302 с.

30. Shrimpy [Електронний ресурс] – Режим доступу до ресурсу: https://www.shrimpy.io/about-us

31. Kaiko [Електронний ресурс] – Режим доступу до ресурсу: https://www.kaiko.com/pages/about-kaiko

32. CoinGecko [Електронний ресурс] – Режим доступу до ресурсу: https://www.coingecko.com/

33. Messari [Електронний ресурс] – Режим доступу до ресурсу: https://messari.io/

34. Troelsen A., Japikse P. Pro C# 9 with .NET 5. Foundational principles and practices in programming / – 2021.

35. Fowler M., Rice D., Foemmel M., Hieatt E., Mee R., Stafford R. Patterns of Enterprice Application Architecture / – 2002.

36. ADO.NET Tutorial [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/dotnet/framework/data/adonet/

37. Understanding Multilayered Architecture in .NET [Електронний ресурс] – Режим доступу до ресурсу: https://www.c-sharpcorner.com/UploadFile/1492b1/understanding-multilayered-architecture-in-net/

38. Smith S. Architecting Modern Web Applications with ASP.NET Core and Microsoft Asure / – 2022.

39. Microsoft SQL Server Management Studio [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver16

ДОДАТОК А

BaseEntity.cs

```
namespace TglCA.Dal.Interfaces.Entities.Base;

public abstract class BaseEntity
{
    public int Id { get; set; }
    public bool IsDeleted { get; set; } = false;
}
```

Currency.cs

```
using TglCA.Dal.Interfaces.Entities.Base;
using TglCA.Dal.Interfaces.Entities.Identity;

namespace TglCA.Dal.Interfaces.Entities;

public class Currency : BaseEntity
{
    public string Symbol { get; set; }

    public IEnumerable<User> Users { get; set; } = new List<User>();
}
```

IRepository.cs

```
using TglCA.Dal.Interfaces.Entities.Base;

namespace TglCA.Dal.Interfaces.IRepositories.IBase;

public interface IRepository<T> where T : BaseEntity, new()
{
    Task CreateAsync(T entity);
    Task CreateRange(IEnumerable<T> entity);
    Task UpdateAsync(T entity);
    Task DeleteAsync(T entity);
    T? GetById(int id);
    IEnumerable<T> GetAll();
    IEnumerable<T> GetAllWithoutQueryFilters();
    Task SafeDeleteAsync(T entity);
}
```

ICurrencyRepository.cs

```
using TglCA.Dal.Interfaces.Entities;
using TglCA.Dal.Interfaces.Entities.Identity;
using TglCA.Dal.Interfaces.IRepositories.IBase;

namespace TglCA.Dal.Interfaces.IRepositories;

public interface ICurrencyRepository : IRepository<Currency>
{
    Task CreateOrUpdateAsync(Currency entity);
    Currency? GetBySymbol(string currencyId);
}
```

IUnitOfWork.cs

```
using Microsoft.EntityFrameworkCore;

namespace TglCA.Dal.Interfaces.IUnitOfWork;

public interface IUnitOfWork : IDisposable
{
    DbContext Context { get; }

    void Commit();
}
```

DateTimeHelper.cs

```
namespace TglCA.Utils
{
    public static class DateTimeHelper
    {
        /// <summary>
        /// Converts a given DateTime into a Unix timestamp
        /// </summary>
        /// <param name="value">Any DateTime</param>
        /// <returns>The given DateTime in Unix timestamp format</returns>
        public static long ToUnixTimestamp(this DateTime value)
        {
            return (long)Math.Truncate(value.ToUniversalTime().Subtract(new DateTime(1970, 1, 1)).TotalMilliseconds);
        }

        /// <summary>
        /// Gets a Unix timestamp representing the current moment
        /// </summary>
        /// <param name="ignored">Parameter ignored</param>
        /// <returns>Now expressed as a Unix timestamp</returns>
        public static long UnixTimestampNow()
        {
            return (long)Math.Truncate(DateTime.UtcNow.Subtract(new DateTime(1970, 1, 1)).TotalMilliseconds);
        }
    }
}
```

RepositoryBase.cs

```
using Microsoft.EntityFrameworkCore;
using TglCA.Dal.Data.DbContextData;
using TglCA.Dal.Interfaces.Entities.Base;
using TglCA.Dal.Interfaces.IRepositories.IBase;

namespace TglCA.Dal.Repositories.Base
{
    public abstract class RepositoryBase<T> : IRepository<T> where T : BaseEntity, new()
    {
        protected readonly MainDbContext _context;
        protected readonly DbSet<T> _dbSet;

        protected RepositoryBase(MainDbContext context)
        {
            _context = context;
            _dbSet = _context.Set<T>();
        }
        public async Task CreateAsync(T entity)
        {
            await _dbSet.AddAsync(entity);
            await _context.SaveChangesAsync();
        }
```

```
    public async Task CreateRange(IEnumerable<T> entities)
    {
        await _dbSet.AddRangeAsync(entities);
        await _context.SaveChangesAsync();
    }

    public async Task DeleteAsync(T entity)
    {
        _dbSet.Remove(entity);
        await _context.SaveChangesAsync();
    }

    public IEnumerable<T> GetAll()
    {
        return _dbSet;
    }

    public IEnumerable<T> GetAllWithoutQueryFilters()
    {
        return _dbSet.IgnoreQueryFilters();
    }

    public async Task SafeDeleteAsync(T entity)
    {
        T? tEntity = GetById(entity.Id);
        if (tEntity == null)
        {
            return;
        }
        tEntity.IsDeleted = true;
        await UpdateAsync(tEntity);
    }

    public T? GetById(int id)
    {
        return _dbSet.Find(id);
    }

    public async Task UpdateAsync(T entity)
    {
        _dbSet.Update(entity);
        await _context.SaveChangesAsync();
    }
}
```

CurrencyRepository.cs

```
using TglCA.Dal.Data.DbContextData;
using TglCA.Dal.Interfaces.Entities;
using TglCA.Dal.Interfaces.Entities.Identity;
using TglCA.Dal.Interfaces.IRepositories;
using TglCA.Dal.Repositories.Base;

namespace TglCA.Dal.Repositories;

public class CurrencyRepository : RepositoryBase<Currency>, ICurrencyRepository
{
    public CurrencyRepository(MainDbContext context) : base(context)
    {
    }
    public async Task CreateOrUpdateAsync(Currency entity)
    {
        Currency? currency = _dbSet.FirstOrDefault(c => c.Symbol == entity.Symbol);
        if (currency == null)
        {
            await CreateAsync(entity);
            return;
        }
    }

    public Currency? GetBySymbol(string symbol)
    {
        return _dbSet.FirstOrDefault(c => c.Symbol == symbol);
    }

}
```

UnitOfWork.cs

```
using Microsoft.EntityFrameworkCore;
using TglCA.Dal.Interfaces.IUnitOfWork;

namespace TglCA.Dal.UnitOfWork;

public class UnitOfWork : IUnitOfWork
{
    public UnitOfWork(DbContext context)
    {
        Context = context;
    }

    public DbContext Context { get; private set; }

    public void Commit()
    {
        if (Context == null)
            throw new NullReferenceException(nameof(Context));

        Context.SaveChanges();
    }

    public void Dispose()
    {
        if (Context == null)
            throw new NullReferenceException(nameof(Context));

        Context.Dispose();

        Context = null;
    }
}
```

MainDbContext.cs

```
public class MainDbContext : IdentityDbContext<User, Role, int, UserClaim, UserRole, UserLogin, RoleClaim, UserToken>
{
    public MainDbContext(DbContextOptions<MainDbContext> options) : base(options)
    {
    }
    #region DbSets
        DbSet<Currency> Currencies { get; set; }
    #endregion
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);

        modelBuilder.Entity<User>()
            .ToTable("Users");

        modelBuilder.Entity<UserRole>()
            .ToTable("UserRoles");

        modelBuilder.Entity<UserClaim>()
            .ToTable("UserClaims");

        modelBuilder.Entity<UserLogin>()
            .ToTable("UserLogins");

        modelBuilder.Entity<UserToken>()
            .ToTable("UserToken");

        modelBuilder.Entity<Role>()
            .ToTable("Roles");

        modelBuilder.Entity<RoleClaim>()
            .ToTable("RoleClaims");

        modelBuilder.Entity<Currency>()
            .ToTable("Currencies")
            .HasQueryFilter(c => !c.IsDeleted);
        modelBuilder.Entity<User>()
            .HasMany(t => t.Currencies)
            .WithMany(p => p.Users);
    }
}
```

CurrencyMapper.cs

```
using TglCA.Bll.Interfaces.Entities;
using TglCA.Bll.Interfaces.Interfaces;
using TglCA.Dal.Interfaces.Entities;

namespace TglCA.Bll.Mappers;

public class CurrencyMapper : ICurrencyMapper
{
    public BllCurrency ToBllCurrency(Currency currency)
    {
        return new BllCurrency
        {
            Id = currency.Id,
            Symbol = currency.Symbol
        };
    }

    public Currency ToCurrency(BllCurrency bllCurrency)
    {
        return new Currency
        {
            Id = bllCurrency.Id,
            Symbol = bllCurrency.Symbol
        };
    }
}
```

CurrencyService.cs

```
using TglCA.Bll.Api.Aggregator.Interfaces;
using TglCA.Bll.Interfaces.Entities;
using TglCA.Bll.Interfaces.Entities.Chart;
using TglCA.Bll.Interfaces.Interfaces;
using TglCA.Dal.Interfaces.Entities;
using TglCA.Dal.Interfaces.IRepositories;

namespace TglCA.Bll.Services;

public class CurrencyService : ICurrencyService
{
    private readonly ICurrencyRepository _currencyRepository;
    private readonly ICurrencyMapper _currencyMapper;
    private readonly ICoinAggregator _coinAggregator;

    public CurrencyService(ICurrencyRepository repository, ICoinAggregator aggregator, ICurrencyMapper currencyMapper)
    {
        _currencyRepository = repository;
        _currencyMapper = currencyMapper;
        _coinAggregator = aggregator;
    }

    public async Task CreateAsync(BllCurrency entity)
    {
        Currency currency = _currencyMapper.ToCurrency(entity);
        await _currencyRepository.CreateAsync(currency);
    }

    public async Task UpdateAsync(BllCurrency entity)
    {
        Currency currency = _currencyMapper.ToCurrency(entity);
        await _currencyRepository.UpdateAsync(currency);
    }

    public async Task DeleteAsync(BllCurrency entity)
    {
        Currency currency = _currencyMapper.ToCurrency(entity);
        await _currencyRepository.SafeDeleteAsync(currency);
    }

    public Currency GetBySymbol(string symbol)
    {
        Currency? currency = _currencyRepository.GetBySymbol(symbol);
        if (currency == null)
        {
            return null;
        }
        return currency;
    }
```

```
    public async Task<IEnumerable<BllCurrency>> GetAllAsync()
    {
        return await _coinAggregator.GetAggregatedCurrencies();
    }

    public async Task<BllCurrency> GetAverageByMarketId(string currencyId)
    {
        var result = await _coinAggregator.GetBllCurrency(currencyId);
        if (result == null)
        {
            return null;
        }

        return result;
    }

    public async Task<IEnumerable<BllCurrency>> GetAllByVolume()
    {
        var result = await _coinAggregator.GetAggregatedCurrencies();
        if (result == null)
        {
            return null;
        }

        //await UpdateDatabase();

        return result.OrderByDescending(x => x.Volume24hUsd);
    }

    public async Task<Dictionary<string, BllCurrency>> GetFromAllMarketsBySymbol(string symbol)
    {
        var result = await _coinAggregator.GetAggregatedCurrency(symbol);

        return result;
    }

    public async Task<Dictionary<string, IEnumerable<ChartPoint<long, decimal>>>> GetCurrencyPriceHistory(string currencyId)
    {
        var result = await _coinAggregator.GetAggregatedChart(currencyId);
        if (result == null)
        {
            return null;
        }
        return result.Where(r => r.Value != null && r.Value.Any())
            .ToDictionary(r => r.Key, r => r.Value);
    }
```

```
    public async Task<Dictionary<string, ChartPoint<long, decimal>>> GetLatestPriceHistoryPoint(string currencyId)
    {
        var result = await _coinAggregator.GetCurrentChartPoint(currencyId);
        if (result == null)
        {
            return null;
        }
        return result;
    }

    public async Task InitialiseDB()
    {
        var result = await _coinAggregator.GetAggregatedCurrencies();
        var coinList = result.Select(x => _currencyMapper.ToCurrency(x));
        await _currencyRepository.CreateRange(coinList);
    }

    public async Task UpdateDatabase()
    {
        var coins = await _coinAggregator.GetAggregatedCurrencies();
        var currencies = coins.Select(x => _currencyMapper.ToCurrency(x));
        foreach (var c in currencies)
        {
            await _currencyRepository.CreateOrUpdateAsync(c);
        }
    }
}
```

UserService.cs

```
namespace TglCA.Bll.Services
{
    public class UserService : IUserService
    {
        private readonly SignInManager<User> _signInManager;
        private readonly UserManager<User> _userManager;
        private readonly IEmailService _emailService;
        private readonly ICurrencyService _currencyService;
        private readonly MainDbContext _context;


        public UserService(UserManager<User> userManager,
            SignInManager<User> signInManager,
            ICurrencyService service,
            MainDbContext context)
        {
            _userManager = userManager;
            _signInManager = signInManager;
            _currencyService = service;
            _context = context;
        }

        public async ValueTask<ErrorModel> CreateAsync(BllUserModel userModel)
        {
            var user = new User
            {
                Email = userModel.Email,
                UserName = userModel.UserName
            };

            var result = await _userManager.CreateAsync(user, userModel.Password);

            if (!result.Succeeded)
            {
                var errorModel = new ErrorModel()
                {
                    ErrorDetails = result.Errors.Select(er => new ErrorDetail()
                    {
                        ErrorMessage = er.Description
                    })
                };
                return errorModel;
            }

            return new ErrorModel();
        }

        public async ValueTask<ErrorModel> LoginAsync(BllUserModel userModel)
        {
            User user = await _userManager.FindByEmailAsync(userModel.Email);

            if (user == null)
            {
                var errorModel = new ErrorModel();
                errorModel.ErrorDetails = new List<ErrorDetail>()
                {
                     new ErrorDetail()
                     {
                         ErrorMessage = "SignIn failure: Wrong login info!"
                     }
                };
                return errorModel;
            }

            await _signInManager.SignOutAsync();

            var result =
                await _signInManager.PasswordSignInAsync(user, userModel.Password, false, true);

            if (!result.Succeeded)
            {
                var errorModel = new ErrorModel();
                if (result.IsNotAllowed)
                {
                    errorModel.ErrorDetails = new List<ErrorDetail>()
                    {
                        new ErrorDetail()
                        {
                            ErrorMessage = "SignIn failure: Email is not confirmed"
                        }
                    };
                    return errorModel;
                }
                errorModel.ErrorDetails = new List<ErrorDetail>()
                {
                     new ErrorDetail()
                     {
                         ErrorMessage = "SignIn failure: Wrong login info!"
                     }
                };
                return errorModel;
            }

            return ErrorModel.CreateSuccess();
        }
```

```
public AuthenticationProperties GetAuthenticationProperties(string provider, string redirectUrl)
{
    return _signInManager.ConfigureExternalAuthenticationProperties(provider, redirectUrl);
}

public async ValueTask<ErrorModel> GoogleResponse()
{
    var info = await _signInManager.GetExternalLoginInfoAsync();

    if (info == null)
    {
        var errorModel = new ErrorModel();
        errorModel.ErrorDetails = new List<ErrorDetail>()
        {
            new ErrorDetail()
            {
                ErrorMessage = "SignIn failure: Wrong login info!"
            }
        };
        return errorModel;
    }

    var result = await _signInManager.ExternalLoginSignInAsync(
        info.LoginProvider, info.ProviderKey, false);

    if (result.Succeeded)
    {
        return ErrorModel.CreateSuccess();
    }

    var bllUser = new BllUserModel()
    {
        Email = info.Principal.FindFirst(ClaimTypes.Email).Value
    };

    var user = new User
    {
        Email = bllUser.Email,
        UserName = bllUser.UserName
    };

    var identityResult = await _userManager.CreateAsync(user);


        var errorModel = new ErrorModel();

        errorModel.ErrorDetails = identityResult.Errors.Select(er => new ErrorDetail()
        {
            ErrorMessage = er.Description
        });

        return errorModel;
    }

    identityResult = await _userManager.AddLoginAsync(user, info);

    string token = await _userManager.GenerateEmailConfirmationTokenAsync(user);
    await _userManager.ConfirmEmailAsync(user, token);

    await _signInManager.SignInAsync(user, isPersistent: false);

    return ErrorModel.CreateSuccess();
}
```

```
public async ValueTask<ErrorModel> CoinSubscribe(string userId, string symbol)
{
    var user = await _context.Users
        .Include(x => x.Currencies)
        .SingleOrDefaultAsync(x => x.Id == int.Parse(userId)); ;

    if (user == null)
    {
        var errorModel = new ErrorModel();
        errorModel.ErrorDetails = new List<ErrorDetail>()
        {
            new ErrorDetail()
            {
                ErrorMessage = "Something went wrong!"
            }
        };
        return errorModel;
    }

    user.Currencies = user.Currencies.Any(x => x.Symbol == symbol)
        ? user.Currencies.Where(c => c.Symbol != symbol).ToList()
        : user.Currencies.Append(_currencyService.GetBySymbol(symbol));

    await _userManager.UpdateAsync(user);

    return ErrorModel.CreateSuccess();
}
```

```
        public async ValueTask<IEnumerable<BllCurrency>> GetSubscriptions(string userId)
        {
            var user = await _context.Users
                .Include(x => x.Currencies)
                .SingleOrDefaultAsync(x => x.Id == int.Parse(userId));

            if (user == null || user.Currencies == null)
            {
                return null;
            }

            var coins = await _currencyService.GetAllByVolume();

            coins = from x in coins
                    join y in user.Currencies on x.Symbol equals y.Symbol
                    select x;

            return coins;
        }

        public async ValueTask<bool> IsSubscribed(string userId, string symbol)
        {
            var user = await _context.Users
                .Include(x => x.Currencies)
                .SingleOrDefaultAsync(x => x.Id == int.Parse(userId));

            if (user == null || user.Currencies == null)
            {
                return false;
            }

            return user.Currencies.Any(x => x.Symbol == symbol);
        }

        public async ValueTask<string> CreateConfirmationTokenAsync(BllUserModel user)
        {
            User userToConfirm = await _userManager.FindByEmailAsync(user.Email);
            string token = await _userManager.GenerateEmailConfirmationTokenAsync(userToConfirm);
            return Base64EncodeHelper.Base64Encode(token);
        }
```

```
        public async ValueTask<IdentityResult> ConfirmEmailByUserNameAsync(string userName, string token)
        {
            User user = await _userManager.FindByNameAsync(userName);
            string decodedToken = Base64EncodeHelper.Base64Decode(token);
            return await _userManager.ConfirmEmailAsync(user, decodedToken);
        }

        public async ValueTask<BllUserModel> GetUserByName(string name)
        {
            var user = await _userManager.FindByNameAsync(name);
            BllUserModel bllUser = new()
            {
                Email = user.Email
            };
            return bllUser;
        }
    }
}
```

MailKitEmailService.cs

```
using System.Net.Mail;
using MailKit.Security;
using MimeKit;
using TglCA.Bll.Interfaces.Interfaces.EmailService;
using SmtpClient = MailKit.Net.Smtp.SmtpClient;

namespace TglCA.Bll.Services.Mail
{
    public class MailKitEmailService : IEmailService
    {
        private readonly string _senderEmail;
        private readonly string _senderPassword;

        public MailKitEmailService(string sender, string password)
        {
            _senderEmail = sender;
            _senderPassword = password;
        }
        public async Task<string> SendMailAsync(MailAddress to, string message, string subject)
        {
            MimeMessage mimeMessage = CreateMessage(to, message, subject);
            using (SmtpClient smtpClient = new SmtpClient())
            {
                await smtpClient.ConnectAsync("smtp.gmail.com", 587, SecureSocketOptions.StartTls);
                await smtpClient.AuthenticateAsync(_senderEmail,_senderPassword);
                var result = await smtpClient.SendAsync(mimeMessage);
                await smtpClient.DisconnectAsync(true);
                return result;
            }
        }
```

```
        public async Task<string> SendConfirmationMessage(MailAddress to, string confirmationEndpoint)
        {
            string messageText =
            $@"Welcome to E-Crypto, {to.DisplayName}

Please verify you email address.

Click the link below to verify you email address:

{confirmationEndpoint}

Thanks,
The E-Crypto Team

This is an automatically generated message. Please do not reply.";

            return await SendMailAsync(to, messageText, "Email confirmation");
        }

        private MimeMessage CreateMessage(MailAddress to, string message, string subject)
        {
            MimeMessage mimeMessage = new MimeMessage();
            mimeMessage.From.Add(new MailboxAddress("E-Crypto", _senderEmail));
            mimeMessage.To.Add(new MailboxAddress(to.DisplayName, to.Address));
            mimeMessage.Subject = subject;
            mimeMessage.Body = new TextPart("plain")
            {
                Text = message
            };
            return mimeMessage;
        }
    }
}
```

ApiProvider.cs

```
using TglCA.Bll.Api.IProviders.Base;
using TglCA.Bll.Interfaces.Entities;
using TglCA.Bll.Interfaces.Entities.Chart;

namespace TglCA.Bll.Api.Providers
{
    public abstract class ApiProvider : IApiProvider
    {
        public abstract Task<ChartPoint<long, decimal>> GetChartPointAsync(string symbol);

        public abstract Task<IEnumerable<BllCurrency>> GetCurrenciesAsync();

        public abstract Task<BllCurrency> GetCurrencyAsync(string symbol);

        public abstract Task<IEnumerable<ChartPoint<long, decimal>>> GetHistoricalDataAsync(string symbol);

        public abstract string GetProviderName();
    }
}
```

CoinAggregator.cs

```
using TglCA.Bll.Api.Aggregator.Interfaces;
using TglCA.Bll.Api.Providers;
using TglCA.Bll.Interfaces.Entities;
using TglCA.Bll.Api.Bitfinex.Provider;
using TglCA.Bll.Api.FTX.Provider;
using TglCA.Bll.Api.Kucoin.Provider;
using TglCA.Bll.Interfaces.Entities.Chart;

namespace TglCA.Bll.Api.Aggregator
{
    public class CoinAggregator : ICoinAggregator
    {
        private readonly List<ApiProvider> providers;

        public CoinAggregator()
        {
            providers = new List<ApiProvider>()
            {
                new BitfinexProvider(),
                new KucoinProvider(),
                new FTXProvider(),
            };
        }
```

```
public async Task<IEnumerable<BllCurrency>> GetAggregatedCurrencies()
{
    var tasks = new List<Task<IEnumerable<BllCurrency>>>();

    foreach (var p in providers)
    {
        tasks.Add(p.GetCurrenciesAsync());
    }

    var lists = await Task.WhenAll(tasks);

    var currencies = lists.SelectMany(l => l).GroupBy(x => x.Symbol);

    var result = new List<BllCurrency>();

    foreach (var currency in currencies)
    {
        result.Add(new BllCurrency()
        {
            Symbol = currency.Key,
            AssetName = currency.First().AssetName,
            Price = currency.Average(x => x.Price),
            PercentChange24h = currency.Average(x => x.PercentChange24h),
            Volume24hUsd = currency.Average(x => x.Volume24hUsd),
        });
    }
    return result;
}
```

```
public async Task<Dictionary<string, BllCurrency>> GetAggregatedCurrency(string symbol)
{
    var tasks = new List<Task<BllCurrency>>();

    foreach (var p in providers)
    {
        tasks.Add(p.GetCurrencyAsync(symbol));
    }

    var list = await Task.WhenAll(tasks);

    var result = new Dictionary<string, BllCurrency>();

    for (int i = 0; i < list.Length; i++)
    {
        if (list[i] != null)
            result.Add(providers[i].GetProviderName(), list[i]);
    }
    return result;
}

public async Task<BllCurrency> GetBllCurrency(string symbol)
{
    var tasks = new List<Task<BllCurrency>>();

    foreach (var p in providers)
    {
        tasks.Add(p.GetCurrencyAsync(symbol));
    }

    var list = await Task.WhenAll(tasks);

    var notNullList = list.Where(x => x != null);

    return new BllCurrency()
    {
        Symbol = notNullList.First().Symbol,
        AssetName = notNullList.First().AssetName,
        Price = notNullList.Average(x => x.Price),
        PercentChange24h = notNullList.Average(x => x.PercentChange24h),
        Volume24hUsd = notNullList.Average(x => x.Volume24hUsd)
    };
}
```

```
        public async Task<Dictionary<string, IEnumerable<ChartPoint<long, decimal>>>> GetAggregatedChart(string symbol)
        {
            var tasks = new List<Task<IEnumerable<ChartPoint<long, decimal>>>>();

            foreach (var p in providers)
            {
                tasks.Add(p.GetHistoricalDataAsync(symbol));
            }

            var lists = await Task.WhenAll(tasks);

            var result = new Dictionary<string, IEnumerable<ChartPoint<long, decimal>>>();

            for (int i = 0; i < lists.Length; i++)
            {
                result.Add(providers[i].GetProviderName(), lists[i]);
            }
            return result;
        }

        public async Task<Dictionary<string, ChartPoint<long, decimal>>> GetCurrentChartPoint(string symbol)
        {
            var tasks = new List<Task<ChartPoint<long, decimal>>>();

            foreach (var p in providers)
            {
                tasks.Add(p.GetChartPointAsync(symbol));
            }

            var list = await Task.WhenAll(tasks);

            var result = new Dictionary<string, ChartPoint<long, decimal>>();

            for (int i = 0; i < list.Length; i++)
            {
                if (list[i] != null)
                {
                    result.Add(providers[i].GetProviderName(), list[i]);
                }
            }
            return result;
        }
    }
}
```

BitfinexProvider.cs

```
using Bitfinex.Net.Clients;
using Bitfinex.Net.Enums;
using TglCA.Bll.Api.Providers;
using TglCA.Bll.Interfaces.Entities;
using TglCA.Bll.Interfaces.Entities.Chart;
using TglCA.Utils;

namespace TglCA.Bll.Api.Bitfinex.Provider
{
    public class BitfinexProvider : ApiProvider
    {
        private readonly string QuoteAsset = "USD";

        public readonly BitfinexClient bitfinexClient;

        public BitfinexProvider()
        {
            bitfinexClient = new BitfinexClient();
        }

        public override async Task<IEnumerable<BllCurrency>> GetCurrenciesAsync()
        {
            var task1 = bitfinexClient.SpotApi.ExchangeData.GetAssetsAsync();
            var task2 = bitfinexClient.SpotApi.ExchangeData.GetTickersAsync();

            await Task.WhenAll(task1, task2);

            var currenciesNames = task1.Result;
            var currenciesStats = task2.Result;

            var filteredCurrenciesStats = currenciesStats.Data
                .Where(x => x.Symbol.EndsWith(QuoteAsset))
                .DistinctBy(x => x.Symbol).ToList();

            filteredCurrenciesStats.ForEach(x =>
            {
                var index = x.Symbol.LastIndexOf(QuoteAsset);
                if (index >= 2)
                {
                    x.Symbol = x.Symbol.Substring(1, index - 1);
                }
            });
```

```
            var result = from x in currenciesNames.Data
                         join y in filteredCurrenciesStats
                         on x.Name equals y.Symbol
                         select new BllCurrency()
                         {
                             AssetName = x.FullName,
                             Symbol = x.Name,
                             Price = y.LastPrice,
                             Volume24hUsd = y.LastPrice >= 1 ? y.Volume / y.LastPrice : y.Volume * y.LastPrice,
                             PercentChange24h = y.DailyChangePercentage

                         };

            return result;
        }

        public override async Task<BllCurrency> GetCurrencyAsync(string symbol)
        {
            var task1 = bitfinexClient.SpotApi.ExchangeData.GetAssetsAsync();
            var task2 = bitfinexClient.SpotApi.ExchangeData.GetTickerAsync("t" + symbol + QuoteAsset);

            await Task.WhenAll(task1, task2);

            var currencyName = task1.Result.Data.FirstOrDefault(x => x.Name.Equals(symbol));
            var currencyStats = task2.Result;

            if (currencyName == null ||
                currencyStats.Data == null ||
                currencyStats.Data.LastPrice == 0)
            {
                return null;
            }

            return new BllCurrency()
            {
                AssetName = currencyName.FullName,
                Symbol = currencyName.Name,
                Price = currencyStats.Data.LastPrice,
                Volume24hUsd = currencyStats.Data.Volume,
                PercentChange24h = currencyStats.Data.DailyChangePercentage,
            };
        }

        public override async Task<IEnumerable<ChartPoint<long, decimal>>> GetHistoricalDataAsync(string symbol)
        {
            var chart = await bitfinexClient.SpotApi.ExchangeData.GetKlinesAsync("t" + symbol + QuoteAsset,
                KlineInterval.OneMinute,
                null,
                260,
                DateTime.Now.AddHours(-5), DateTime.Now);

            if (chart.Data == null)
            {
                return null;
            }

            var result = chart.Data
                .Select(x => new ChartPoint<long, decimal>(
                    DateTimeHelper.ToUnixTimestamp(x.OpenTime),
                    x.ClosePrice));

            return result;
        }

        public override string GetProviderName()
        {
            return bitfinexClient.SpotApi.CommonSpotClient.ExchangeName;
        }

        public override async Task<ChartPoint<long, decimal>> GetChartPointAsync(string symbol)
        {
            var currencyStats = await bitfinexClient.SpotApi.ExchangeData.GetTickerAsync("t" + symbol + QuoteAsset);

            if (currencyStats.Data.LastPrice <= 0)
            {
                return null;
            }

            return new ChartPoint<long, decimal>(DateTimeHelper.UnixTimestampNow(), currencyStats.Data.LastPrice);
        }
    }
}
```

AccountController.cs

```
using System.Net.Mail;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using System.Security.Claims;
using TglCA.Bll.Interfaces.Entities;
using TglCA.Bll.Interfaces.Entities.BllModels;
using TglCA.Bll.Interfaces.Interfaces;
using TglCA.Bll.Interfaces.Interfaces.EmailService;
using TglCA.Mvc.PL.Models;
using TglCA.Mvc.PL.Models.Mappers;
using X.PagedList;

namespace TglCA.Mvc.PL.Controllers;

[Route("/[controller]/[action]")]
public class AccountController : Controller
{
    private readonly IUserService _userService;
    private readonly IConfiguration _configuration;
    private readonly IEmailService _emailService;

    public AccountController(IUserService userUservice, IEmailService emailService, IConfiguration configuration)
    {
        _userService = userUservice;
        _emailService = emailService;
        _configuration = configuration;
    }

    [HttpGet]
    public IActionResult Successful(BllUserModel model)
    {
        return View(model);
    }

    [HttpGet]
    public IActionResult Error()
    {
        ViewBag.Errors = TempData["Errors"];
        return View();
    }

    [HttpGet]
    public IActionResult Register()
    {
        return View();
    }
```

```
    [HttpPost]
    public async ValueTask<IActionResult> Register(UserInputModel userInputModel)
    {
        if (!ModelState.IsValid)
        {
            return View(userInputModel);
        }

        BllUserModel userModel = new BllUserModel()
        {
            Email = userInputModel.Email,
            Password = userInputModel.Password
        };

        var errModel = await _userService.CreateAsync(userModel);
        if (!errModel.IsSuccess)
        {
            AddModelStateErrors(errModel);
            return View(userInputModel);
        }

        var token = await _userService.CreateConfirmationTokenAsync(userModel);

        var confirmationEndpoint = Url.Action(
            "Confirmation",
            "Account",
            values: new { userName = userModel.UserName, token = token },
            protocol: Request.Scheme
        );

        await _emailService.SendConfirmationMessage(new MailAddress(userModel.Email, userModel.UserName), confirmationEndpoint);

        return RedirectToAction(nameof(Successful), userModel);
    }
```

```
[HttpGet("{userName}/{token}")]
public async ValueTask<IActionResult> Confirmation(string userName, string token)
{
    var result = await _userService.ConfirmEmailByUserNameAsync(userName, token);
    var user = await _userService.GetUserByName(userName);

    if (result.Succeeded && user != null)
    {
        return View(user);
    }

    return View("Error");
}

[HttpGet]
public IActionResult Login()
{
    return View();
}

[HttpPost]
public async ValueTask<IActionResult> Login(UserInputModel userInputModel)
{
    if (!ModelState.IsValid) return View(userInputModel);

    var errModel = await _userService.LoginAsync(new BllUserModel()
    {
        Email = userInputModel.Email,
        Password = userInputModel.Password
    });

    if (!errModel.IsSuccess)
    {
        AddModelStateErrors(errModel);
        return View(userInputModel);
    }

    return RedirectToAction("All", "Main");
}

public async ValueTask<IActionResult> LogOut()
{
    await _userService.SignOutAsync();
    return RedirectToAction("All", "Main");
}
```

```
[AllowAnonymous]
public IActionResult GoogleLogin()
{
    string redirectUrl = Url.Action("GoogleResponse", "Account");
    var properties = _userService.GetAuthenticationProperties("Google", redirectUrl);
    return new ChallengeResult("Google", properties);
}

[AllowAnonymous]
public async ValueTask<IActionResult> GoogleResponse()
{
    var errModel = await _userService.GoogleResponse();

    if (!errModel.IsSuccess)
    {
        TempData["Errors"] = errModel.ErrorDetails
            .Select(e => e.ErrorMessage)
            .ToList();
        return RedirectToAction(nameof(Error));
    }

    return RedirectToAction("All", "Main");
}

public async Task<IActionResult> Subscribe(string symbol)
{
    var userId = User.FindFirst(ClaimTypes.NameIdentifier).Value;
    var errModel = await _userService.CoinSubscribe(userId, symbol);

    if (!errModel.IsSuccess)
    {
        TempData["Errors"] = errModel.ErrorDetails
            .Select(e => e.ErrorMessage)
            .ToList();
        return RedirectToAction(nameof(Error));
    }

    return RedirectToAction("CoinInfo", "Coin", new {symbol = symbol});
}
```

```
    private IPagedList<CurrencyViewModel> GetPagedViewModel(
        IEnumerable<BllCurrency> currencies,
        int pageNumber,
        int pageSize)
    {
        return currencies
            .ToViewModels()
            .ToPagedList(pageNumber, pageSize);
    }

    private int GetPageSize()
    {
        return _configuration.GetSection("PageSettings:PageSize").Get<int>();
    }

    [HttpGet("{symbol}")]
    public async Task<IActionResult> IsSubscribed(string symbol)
    {
        if (!User.Claims.Any())
        {
            return Json(new { status = "Unauthorized" });
        }
        var userId = User.FindFirst(ClaimTypes.NameIdentifier).Value;

        var result = await _userService.IsSubscribed(userId, symbol);

        return Json(new {status = result});
    }

    private void AddModelStateErrors(ErrorModel errorModel)
    {
        foreach (var item in errorModel.ErrorDetails)
        {
            ModelState.AddModelError(string.Empty, item.ErrorMessage);
        }
    }
}
```

CoinController.cs

```
using Microsoft.AspNetCore.Mvc;
using Newtonsoft.Json;
using TglCA.Bll.Interfaces.Entities;
using TglCA.Bll.Interfaces.Interfaces;
using TglCA.Mvc.PL.Models;
using TglCA.Mvc.PL.Models.Mappers;

namespace TglCA.Mvc.PL.Controllers
{
    [Route("/[controller]/[action]")]
    public class CoinController : Controller
    {
        private readonly ICurrencyService _currencyService;
        public CoinController(ICurrencyService currencyService)
        {
            _currencyService = currencyService;
        }

        [HttpGet("{symbol}")]
        public async ValueTask<IActionResult> CoinInfo(string symbol)
        {
            BllCurrency bllCurrency = await _currencyService.GetAverageByMarketId(symbol);
            if (bllCurrency == null)
            {
                return NotFound();
            }

            var currencyByMarket = await _currencyService.GetFromAllMarketsBySymbol(symbol);
            List<MarketViewModel> markets = new List<MarketViewModel>();
            foreach (var marketCurrency in currencyByMarket)
            {
                if (marketCurrency.Value != null)
                {
                    markets.Add(new MarketViewModel
                    {
                        Currency = marketCurrency.Value,
                        Name = marketCurrency.Key
                    });
                }
            }
            CoinViewModel viewModel = new CoinViewModel()
            {
                Currency = bllCurrency.ToViewModel(),
                Markets = markets.OrderByDescending(m => m.Currency.Volume24hUsd).ToList()
            };
            return View(viewModel);
```

```
        }
        [HttpGet("{id}")]
        public async Task<IActionResult> CoinChartInitialValues(string id)
        {
            var points = await _currencyService.GetCurrencyPriceHistory(id);

            return Content(JsonConvert.SerializeObject(points));
        }

        [HttpGet("{id}")]
        public async Task<IActionResult> CoinChartGetCurrentValue(string id)
        {
            var currentPoints = await _currencyService.GetLatestPriceHistoryPoint(id);
            return Content(JsonConvert.SerializeObject(currentPoints));
        }
    }
}
```

MainController.cs

```
using Microsoft.AspNetCore.Mvc;
using System.Diagnostics;
using TglCA.Bll.Interfaces.Entities;
using TglCA.Bll.Interfaces.Interfaces;
using TglCA.Mvc.PL.Models;
using TglCA.Mvc.PL.Models.Mappers;
using X.PagedList;

namespace TglCA.Mvc.PL.Controllers;

[Route("/[controller]")]
[Route("/[controller]/[action]")]
public class MainController : Controller
{
    private readonly IConfiguration _configuration;
    private readonly ICurrencyService _currencyService;

    public MainController(ICurrencyService service, IConfiguration configuration)
    {
        _currencyService = service;
        _configuration = configuration;
    }

    [Route("/")]
    [HttpGet("{pageNumber}")]
    public async Task<IActionResult> All(int? page)
    {
        var currencies = await _currencyService.GetAllByVolume();
        var pageSize = page ?? 1;
        return View(GetPagedViewModel(currencies, pageSize, GetPageSize()));
    }

    private IPagedList<CurrencyViewModel> GetPagedViewModel(
        IEnumerable<BllCurrency> currencies,
        int pageNumber,
        int pageSize)
    {
        return currencies
            .ToViewModels()
            .ToPagedList(pageNumber, pageSize);
    }

    private int GetPageSize()
    {
        return _configuration.GetSection("PageSettings:PageSize").Get<int>();
    }
    [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
    public IActionResult Error()
    {
        return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });
    }
}
```

CoinViewModel.cs

```
using System.ComponentModel.DataAnnotations;
using TglCA.Bll.Interfaces.Entities.Chart;

namespace TglCA.Mvc.PL.Models
{
    public record CoinViewModel
    {
        public CurrencyViewModel Currency { get; init; }
        public List<MarketViewModel> Markets { get; init; }

    }
}
```

CurrencyViewModel.cs

```
using System.ComponentModel.DataAnnotations;

namespace TglCA.Mvc.PL.Models;

public record CurrencyViewModel
{
    public string Img { get; init; }
    public string AssetName { get; init; }
    public string Symbol { get; init; }
    public decimal Price { get; init; }
    public decimal PercentChange24h { get; init; }
    [Display(Name = "24 Hour Trading Vol")]
    public decimal Volume24hUsd { get; init; }
}
```

ErrorViewModel.cs

```
namespace TglCA.Mvc.PL.Models
{
    public class ErrorViewModel
    {
        public string? RequestId { get; set; }
        public bool ShowRequestId => !string.IsNullOrEmpty(RequestId);
    }
}
```

MarketViewModel.cs

```
using TglCA.Bll.Interfaces.Entities;

namespace TglCA.Mvc.PL.Models
{
    public record MarketViewModel
    {
        public string Name { get; init; }
        public BllCurrency Currency { get; init; }
    }
}
```

UserInputModel.cs

```
using System.ComponentModel.DataAnnotations;

namespace TglCA.Mvc.PL.Models
{
    public record UserInputModel
    {
        [Required]
        [DataType(DataType.EmailAddress)]
        [Display(Name = "Email")]
        public string Email { get; init; } = default!;

        [Required]
        [DataType(DataType.Password)]
        [Display(Name = "Password")]
        public string Password { get; init; } = default!;
    }
}
```

Program.cs

```
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using TglCA.Bll.Api.Aggregator;
using TglCA.Bll.Api.Aggregator.Interfaces;
using TglCA.Bll.Interfaces.Interfaces;
using TglCA.Bll.Interfaces.Interfaces.EmailService;
using TglCA.Bll.Mappers;
using TglCA.Bll.Services;
using TglCA.Bll.Services.Mail;
using TglCA.Dal.Data.DbContextData;
using TglCA.Dal.Interfaces.Entities.Identity;
using TglCA.Dal.Interfaces.IRepositories;
using TglCA.Dal.Repositories;

var builder = WebApplication.CreateBuilder(args);

var connectionString = builder.Configuration.GetConnectionString("Default");

#region Services

// Add services to the container.
builder.Services.AddControllersWithViews();
builder.Services.AddDbContext<MainDbContext>(options
    => options.UseSqlServer(connectionString));
builder.Services.AddIdentity<User, Role>(options =>
    {
        #region Password options

        options.Password = builder.Configuration
            .GetSection("AppIdentitySettings:Password")
            .Get<PasswordOptions>();

        #endregion

        #region Lockout options

        options.Lockout = builder.Configuration
            .GetSection("AppIdentitySettings:Lockout")
            .Get<LockoutOptions>();

        #endregion

        #region User options

        options.User = builder.Configuration
            .GetSection("AppIdentitySettings:User")
            .Get<UserOptions>();

        #endregion

        #region SignIn options

        options.SignIn = builder.Configuration
            .GetSection("AppIdentitySettings:SignIn")
            .Get<SignInOptions>();

        #endregion
    })
    .AddEntityFrameworkStores<MainDbContext>()
    .AddDefaultTokenProviders();
builder.Services.AddAuthentication().AddGoogle(googleOptions =>
    {
        googleOptions.ClientId = builder.Configuration["Authentication:Google:ClientId"];
        googleOptions.ClientSecret = builder.Configuration["Authentication:Google:ClientSecret"];
    }
);
builder.Services.AddTransient<ICurrencyRepository, CurrencyRepository>();
builder.Services.AddTransient<ICoinAggregator, CoinAggregator>();
builder.Services.AddTransient<ICurrencyService, CurrencyService>();
builder.Services.AddTransient<ICurrencyMapper, CurrencyMapper>();
builder.Services.AddTransient<IUserService, UserService>();
builder.Services.AddTransient<IEmailService>(s =>
    new MailKitEmailService(
        builder.Configuration["Authentication:Google:MailService:Email"],
        builder.Configuration["Authentication:Google:MailService:Password"]));


#endregion
```

```
#region HTTP pipeline

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Main/Error");
    app.UseHsts();
}


app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();
app.UseAuthentication();
app.UseAuthorization();

app.UseEndpoints(endpoints => { endpoints.MapControllers(); });

#endregion


app.Run();
```

ДОДАТОК Б

CurrencyServiceTests.cs

```
using System.Collections.Generic;
using FluentAssertions;
using NSubstitute;
using NSubstitute.ReturnsExtensions;
using TglCA.Bll.Api.Aggregator.Interfaces;
using TglCA.Bll.Interfaces.Entities;
using TglCA.Bll.Interfaces.Entities.Chart;
using TglCA.Bll.Interfaces.Interfaces;
using TglCA.Bll.Mappers;
using TglCA.Bll.Services;
using Xunit;

namespace TglCA.Bll.Tests
{
    public class CurrencyServiceTests
    {
        [Fact]
        public async void GetAllAsyncTest_AnyTrue()
        {
            //Arrange

            ICoinAggregator coinAggregator = Substitute.For<ICoinAggregator>();
            coinAggregator.GetAggregatedCurrencies().Returns(new List<BllCurrency>
            {
                new()
                {
                    Id = 1,
                    AssetName = "AssetName1",
                    PercentChange24h = 1.0m,
                    Price = 1.1m,
                    Symbol = "AST1",
                    Volume24hUsd = 1.2m
                },
                new()
                {
                    Id = 2,
                    AssetName = "AssetName2",
                    PercentChange24h = 1.0m,
                    Price = 1.1m,
                    Symbol = "AST2",
                    Volume24hUsd = 1.2m
                },
                new()
                {
                    Id = 3,
                    AssetName = "AssetName3",
                    PercentChange24h = 1.0m,
                    Price = 1.1m,
                    Symbol = "AST3",
                    Volume24hUsd = 1.2m
                },
                new()
                {
                    Id = 4,
                    AssetName = "AssetName4",
                    PercentChange24h = 1.0m,
                    Price = 1.1m,
                    Symbol = "AST4",
                    Volume24hUsd = 1.2m
                }
            });
            ICurrencyService currencyService = new CurrencyService(null, coinAggregator, new CurrencyMapper());

            //Act

            var result = await currencyService.GetAllAsync();

            //Assert

            result.Should()
                .NotBeEmpty();
        }
```

```
[Fact]
public async void GetAllAsyncTest_AnyFalseNull()
{
    //Arrange

    ICoinAggregator coinAggregator = Substitute.For<ICoinAggregator>();
    coinAggregator.GetAggregatedCurrencies().ReturnsNull();
    ICurrencyService currencyService = new CurrencyService(null, coinAggregator, new CurrencyMapper());

    //Act

    var result = await currencyService.GetAllAsync();

    //Assert

    result.Should()
        .BeNull();
}
[Fact]
public async void GetAllAsyncTest_AnyFalse()
{
    //Arrange

    ICoinAggregator coinAggregator = Substitute.For<ICoinAggregator>();
    coinAggregator.GetAggregatedCurrencies().Returns(new List<BllCurrency>());
    ICurrencyService currencyService = new CurrencyService(null, coinAggregator, new CurrencyMapper());

    //Act

    var result = await currencyService.GetAllAsync();

    //Assert

    result.Should()
        .BeEmpty();
}
```

```
[Fact]
public async void GetAllByVolumeTest_AnyFalseNull()
{
    //Arrange

    ICoinAggregator coinAggregator = Substitute.For<ICoinAggregator>();
    coinAggregator.GetAggregatedCurrencies().ReturnsNull();
    ICurrencyService currencyService = new CurrencyService(null, coinAggregator, new CurrencyMapper());

    //Act

    var result = await currencyService.GetAllByVolume();

    //Assert

    result.Should()
        .BeNull();
}
[Fact]
public async void GetAllByVolumeTest_AnyFalse()
{
    //Arrange

    ICoinAggregator coinAggregator = Substitute.For<ICoinAggregator>();
    coinAggregator.GetAggregatedCurrencies().Returns(new List<BllCurrency>());
    ICurrencyService currencyService = new CurrencyService(null, coinAggregator, new CurrencyMapper());

    //Act

    var result = await currencyService.GetAllByVolume();

    //Assert

    result.Should()
        .BeEmpty();
}
```

```
[Fact]
public async void GetFromAllMarketsBySymbolTest_AnyFalseNull()
{
    //Arrange

    ICoinAggregator coinAggregator = Substitute.For<ICoinAggregator>();
    coinAggregator.GetAggregatedCurrency(Arg.Any<string>())
        .ReturnsNull();
    ICurrencyService currencyService = new CurrencyService(null, coinAggregator, new CurrencyMapper());

    //Act

    var result = await currencyService.GetFromAllMarketsBySymbol("SMB1");

    //Assert

    result.Should()
        .BeNull();
}
```

```
[Fact]
public async void GetCurrencyPriceHistoryTest_AnyTrue()
{
    //Arrange

    ICoinAggregator coinAggregator = Substitute.For<ICoinAggregator>();
    coinAggregator.GetAggregatedChart(Arg.Any<string>())
        .Returns(new Dictionary<string, IEnumerable<ChartPoint<long, decimal>>>
        {
            {
                "Market1",
                new List<ChartPoint<long,decimal>>
                {
                    new (1341234342L, 1.0m),
                    new (1341234342L, 1.0m),
                    new (1341234342L, 1.0m),
                }
            },
            {
                "Market2",
                new List<ChartPoint<long,decimal>>
                {
                    new (1341234342L, 1.0m),
                    new (1341234342L, 1.0m),
                    new (1341234342L, 1.0m),
                }
            },
            {
                "Market3",
                new List<ChartPoint<long,decimal>>
                {
                    new (1341234342L, 1.0m),
                    new (1341234342L, 1.0m),
                    new (1341234342L, 1.0m),
                }
            }

        });
    ICurrencyService currencyService = new CurrencyService(null, coinAggregator, new CurrencyMapper());

    //Act

    var result = await currencyService.GetCurrencyPriceHistory("SMB1");

    //Assert

    result.Should()
        .NotBeEmpty();
}
```

```
[Fact]
public async void GetCurrencyPriceHistoryTest_AnyFalseNull()
{
    //Arrange

    ICoinAggregator coinAggregator = Substitute.For<ICoinAggregator>();
    coinAggregator.GetAggregatedChart(Arg.Any<string>())
        .ReturnsNull();
    ICurrencyService currencyService = new CurrencyService(null, coinAggregator, new CurrencyMapper());

    //Act

    var result = await currencyService.GetCurrencyPriceHistory("SMB1");

    //Assert

    result.Should()
        .BeNull();
}
[Fact]
public async void GetCurrencyPriceHistoryTest_AnyFalse()
{
    //Arrange

    ICoinAggregator coinAggregator = Substitute.For<ICoinAggregator>();
    coinAggregator.GetAggregatedChart(Arg.Any<string>())
        .Returns(new Dictionary<string, IEnumerable<ChartPoint<long, decimal>>>());
    ICurrencyService currencyService = new CurrencyService(null, coinAggregator, new CurrencyMapper());

    //Act

    var result = await currencyService.GetCurrencyPriceHistory("SMB1");

    //Assert

    result.Should()
        .BeEmpty();
}
```