Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка веб-додатку для управління портфелями акцій з використанням API IEX

Виконав: студент IV курсу, групи СП-42 спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

_____ Сучков С.С.
(підпис) (прізвище та ініціали)

Керівник _____ Бревус В.М.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Стоянов Ю.М.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Петрик М.Р.
(підпис) (прізвище та ініціали)

Рецензент _____ _____
(підпис) (прізвище та ініціали)

Тернопіль
2023

РЕФЕРАТ

Кваліфікаційна робота бакалавра. Тернопільський національний технічний університет імені Івана Пулюя, кафедра прорамної інженерії, спеціальність 121 «Інженерія програмного забезпечення». ТНТУ, 2023, Сторінок 75 , рисунків 20, таблиць 1, презентація.

Тема: Розробка веб додатку для управління портфелями акцій з використанням API IEX.

Мета дослідження: Створити веб-додаток, який дозволить ефективно управляти та вести облік персонального портфеля акцій та забезпечити детальну оцінку ефективності інвестицій.

Предмет дослідження: Автоматизація управління та обліку персонального портфеля цінних паперів з використанням платформи .NET та Entity Framework.

Методологія дослідження:Методологія дослідження базується на аналітичному огляді наявних додатків та систем для управління персональним портфелем цінних паперів, вивченні інструментів аналізу фінансових показників.

Результати проекту: Це є розроблений веб-додаток, який дозволяє ефективно управляти та обліковувати персональний портфель акцій, сприяючи розширенню фінансової грамотності серед інвесторів та трейдерів. Архітектура та технічна реалізація веб-додатку відрізняються високими показниками розширюваності, переносимості, гнучкості та масштабованості, забезпечуючи його гнучку інфраструктуру для подальшого розвитку, модифікацій та розширень. Одним з ключових елементів розробки веб-додатку є використання API IEX (Investors Exchange) для отримання актуальної та достовірної інформації про акції. Цей API надає доступ до спектру фінансових даних, включаючи цінові показники, іта багато іншого, що дозволяє користувачам нашого веб-додатку отримувати інформацію для прийняття обґрунтованих рішень щодо управління своїм портфелем акцій.

ANNOTATION

Bachelor's qualifying work. Ternopil National Technical University named after Ivan Pulyu, department of software engineering, specialty 121 "Software engineering". TNTU, 2023, Pages 75 , drawings 20,charts 1, presentation.

Topic: Development of a web application for managing stock portfolios using the IEX API.

The purpose of the study: to create a web application that will allow you to effectively manage and keep records of your personal stock portfolio and provide a detailed assessment of investment performance.

Research subject: Automation of management and accounting of a personal portfolio of securities using the .NET platform and Entity Framework.

Research methodology: The research methodology is based on an analytical review of available applications and systems for managing a personal portfolio of securities, a study of tools for analyzing financial indicators and a comparative analysis of software tools for calculating and analyzing investment and financial indicators.

Project results: This is a developed web application that allows you to effectively manage and account for a personal stock portfolio, contributing to the expansion of financial literacy among investors and traders. The application also provides an opportunity to observe the effect of different approaches to the investment process in practice. The architecture and technical implementation of the web application are characterized by high extensibility, portability, flexibility and scalability, providing it with a flexible infrastructure for further development, modifications and extensions. One of the key elements of web application development is the use of the IEX (Investors Exchange) API to obtain up-to-date and reliable stock information. This API provides access to a wide range of financial data, including price indicators, historical data, news and more, enabling users of our web application to obtain information to make informed decisions about managing their stock portfolio.

ЗМІСТ

ВСТУП

Інвестиції є важливою складовою економічного розвитку і мають значний вплив на економіку. Однак, недостатня фінансова грамотність та навички управління інвестиціями у багатьох людей можуть стати перешкодою для досягнення ефективності та успіху у цій сфері. Саме тому розробка веб-додатку для управління та обліку персонального портфеля цінних паперів стає актуальною задачею для дипломного проекту.

Один із ключових елементів розробки цього веб-додатку - використання API IEX (Investors Exchange) для забезпечення достовірної та актуальної інформації про акції. API IEX надає доступ до широкого спектру фінансових даних, включаючи цінові показники, історичні дані та новини, що дозволяє користувачам отримувати актуальну інформацію для зроблення обґрунтованих рішень щодо управління своїм портфелем акцій.

Метою цього дипломного проекту є розробка веб-додатку, який надасть інвесторам вичерпну оцінку ефективності їх інвестицій. Для досягнення цієї мети, веб-додаток має об'єднувати переваги існуючих аналогів, але при цьому бути зручнішим у використанні та надавати більш повну інформацію для успішного управління та обліку персонального інвестиційного портфеля.

У процесі розробки цього веб-додатку потрібно провести огляд та аналіз наявних рішень у сфері управління інвестиціями, що дозволяють зберігати та корегувати дані та надавати користувачам необхідну статистичну інформацію. Також необхідно вивчити різні підходи та патерни архітектур, а також програмні засоби для створення інформаційних систем.

Окрім того, важливим кроком є проектування та розробка власного програмного засобу, який буде реалізовувати функціонал аналітики та автоматизації процесів управління персональним портфелем цінних паперів. Цей веб-додаток має надавати інвесторам та трейдерам зручність у обліку їх портфеля акцій, а його інтерфейс та інфографіка повинні бути простими, інформативними та інтуїтивно зрозумілими для користувача.

Отримані результати цього дипломного проекту мають практичне значення. Розроблений веб-додаток допоможе інвесторам та трейдерам полегшити процес обліку їх персонального портфеля акцій. Він надасть користувачам зручний інтерфейс та інфографіку, які допоможуть їм зрозуміти та аналізувати їх інвестиційні рішення.

.

РОЗДІЛ 1. ЗАСОБИ ДЛЯ АНАЛІЗУ ПОКАЗНИКІВ ФІНАНСОВОГО РИНКУ ІНВЕСТИЦІЙ

1.1 Аналітичний огляд предметної області

Невід'ємною складовою процесу інвестування є формування веб-додатку для управління цінними паперами, їх регулярний перегляд та оцінка ефективності.

Формування портфеля цінних паперів включає вибір конкретних активів для інвестування коштів та розподіл капіталу між цими активами у відповідних пропорціях. Інвестор стикається з викликами селективності, вибору моменту проведення операцій та диверсифікації. Селективність, також відома як мікропрогнозування, відноситься до аналізу цінних паперів і пов'язана з прогнозуванням динаміки цін на окремі види цінних паперів. Вибір моменту проведення операцій, або макропрогнозування, включає прогнозування змін рівня цін на акції порівняно з цінами на фондові інструменти з фіксованим доходом, наприклад, корпоративні облігації. Диверсифікація полягає у створенні інвестиційного портфеля таким чином, щоб мінімізувати ризик при обмежених умовах.

Процес перегляду портфеля, який є необхідною складовою інвестиційного процесу, є важливим, оскільки з часом цілі інвестування можуть змінитися, що призводить до непридатності поточного портфеля для досягнення мети та цілей його власника. В такому випадку інвестору необхідно створити новий портфель шляхом продажу частини цінних паперів та придбання нових. Іншою причиною для перегляду портфеля є зміна курсу цінних паперів з плином часу.

У зв'язку з цим деякі цінні папери, які спочатку були менш привабливими для інвестора, можуть стати привабливими для веб додатка, а навпаки. Таким чином, інвестор може зацікавитися придбанням нових цінних паперів, одночасно продавши ті, що вже є в його портфелі. Рішення про перегляд портфеля залежить від різних факторів, включаючи транзакційні витрати та очікуване зростання прибутковості переглянутого портфеля.

Оцінка ефективності портфеля веб додатка включає періодичну оцінку значень та показників прибутковості інвестиційного портфеля, часто разом з оцінкою показників ризику, з якими стикається інвестор [4]. Для цього необхідно використовувати прийнятні показники прибутковості та ризику, а також відповідні стандарти для порівняння.

Ще одним типом учасників на фондовому ринку є трейдери. Інвестори та трейдери працюють на одному ринку, використовують подібні напрямки та методи аналізу, але мають різні стратегії. Головна різниця між інвестиціями та трейдингом полягає у часовому факторі. Угоди, що закриваються протягом короткого періоду, вважаються трейдингом, тоді як інвестиції передбачають більш тривалі інвестиційні періоди. Трейдери зазвичай не базують свій вибір активів на фундаментальному аналізі компаній, а зосереджуються на стані ринку та короткостроковій волатильності для отримання прибутку. Трейдери можуть здобувати прибуток як на зростаючих, так і на падаючих ринках.

Оскільки трейдери мають обмежений час, якість фінансових інструментів не є основним чинником для їх аналізу, вони зосереджуються на поточній позиції цих інструментів на ринку, використовуючи технічний аналіз.

Трейдери виконують важливу роль у фондовому ринку, забезпечуючи ліквідність для інвесторів і діючи як протилежна сторона угоди. Це означає, що трейдери готові купувати і продавати акції, що дозволяє інвесторам легко реалізовувати свої інвестиційні позиції. Взаємодія між трейдерами і інвесторами формує сучасні фінансові ринки, і обидва ці учасники є необхідними для функціонування ринку.

Варто зазначити, що хоча інвестування спрямоване на отримання прибутку,

воно не є гарантованим шляхом досягнення цієї мети. Різні інвестиційні підходи надають різні рівні гарантій щодо отримання доходу, але завжди існує ризик збитку замість прибутку. Важливо приймати обґрунтовані інвестиційні рішення, які ґрунтуються на ретельному фінансовому аналізі і виконанні грамотного інвестиційного менеджменту.

Один із методів оцінки активів є фундаментальний аналіз, який зосереджений на оцінці фінансових показників активів та їх фундаментальних характеристик. Його мета полягає вибрати цінні папери, які мають потенціал зростання вартості або стабільні виплати дивідендів. Інший метод - технічний аналіз, ґрунтується на вивченні цінових змін та моделей поведінки цін на ринку. Технічні аналітики використовують широкий спектр моделей, від простих до складних, для прогнозування цінових тенденцій.

Узагалі, існує два основних підходи до оцінки в рамках технічного аналізу. Перший підхід - оцінка дисконтованих грошових потоків (DCF) - порівнює вартість активу з поточною вартістю очікуваних майбутніх грошових потоків, що припадають на цей актив. Згідно з другим підходом, відомим як порівняльна оцінка, вартість активу визначається шляхом аналізу цінової динаміки аналогічних активів та їхнього зв'язку з різними змінними (наприклад, доходами, грошовими потоками, балансовою вартістю або обсягами продажів).

Результати оцінки можуть суттєво варіюватися залежно від застосовуваного підходу. Вибір конкретної моделі та підходу до оцінки здійснюється на основі конкретних цілей та завдань інвестора.

Процес обчислення загальної оцінки активу включає розрахунок ряду фінансових показників, таких як рентабельність власного капіталу (ROE), чиста приведена вартість (NPV), індекс рентабельності (PI), дисконтований період окупності (DPP), коефіцієнт ліквідності (current ratio), коефіцієнт швидкого покриття (quick ratio), коефіцієнт оборотності (turnover ratio), коефіцієнт покриття відсотків (interest coverage ratio), коефіцієнт покриття постійних витрат (fixed charges coverage ratio) та багато інших.

1.2 Види систем керування цифровими активами

У зв'язку з необхідністю здійснення розрахунків, моніторингу та аналізу фінансових змінних та показників для інвестиційного аналізу, прийняття інвестиційних рішень та корекції портфеля цінних паперів та стратегій інвестування, виникає потреба у автоматизації цих процесів з об'єднанням та графічним відображенням розрахованих даних. Це дозволяє спростити та зручно організувати інвестиційний менеджмент.

Для цього був створений новий клас програмних продуктів, який спрямований на автоматизацію розрахунків фінансових показників та змінних, збір та графічне відображення отриманих фінансових даних, щоб зробити їх зручними для інвесторів. Він також дозволяє здійснювати подальший інвестиційний менеджмент, зваження та прийняття інвестиційних рішень. Крім того, цей програмний продукт надає можливість першочергового вибору та коригування стратегій та підходів до інвестування.

Загалом, ці програмні продукти спрощують та зручно організовують інвестиційний менеджмент шляхом автоматизації розрахунків фінансових показників та змінних, збору та графічного відображення фінансових даних, що дозволяє інвесторам зважувати та приймати інвестиційні рішення та ефективно коригувати свої стратегії та підходи до інвестування.

З інтеграцією нових функцій у програмне забезпечення для автоматизації фінансових розрахунків та створення інфографічного представлення для аналізу, почали виникати питання про можливість автоматизації стратегій інвестування. Це означало, що програма не тільки могла здійснювати розрахунки та аналізувати дані, але й автоматично реагувати на "сигнали" - зміни відповідних фінансових показників, відповідно до заданих стратегій. Наприклад, коли значення або динаміка показника відповідали заданим у стратегії умовам, програма автоматично створювала транзакції на покупку або продаж певної кількості цінних паперів.

Таким чином, з'явилися програмні продукти, які замість простої функціональності та інструментів для розрахунків та аналітики, надавали можливість реалізації стратегій інвестування. Вони дозволяли використовувати певну стратегію або їх комбінацію, автоматично слідувати за нею та здійснювати покупку/продаж активів на основі програмно прийнятих рішень. Такі програми стали популярними і були використані трейдерами. Цей підхід до торгівлі на біржі з використанням таких програмних продуктів привів до появи нового напрямку - автоматизованого алгоритмічного трейдингу, який залишається актуальним і по сьогоднішній день.

На сьогоднішній день інвестори та трейдери виявляють значний інтерес до програмних систем, які об'єднують функціональні можливості обох вищезгаданих типів інформаційних систем управління цифровими активами. Ці системи надають можливості та інструменти для аналітики, а також механізми автоматизації інвестиційних стратегій, користувацьких преференцій та підходів до торгівлі цінними паперами.

Розробка таких комплексних систем та їх поява на ринку програмних продуктів є досить новим явищем. Незважаючи на їх обмежену кількість, вони мають великий попит серед інвестиційної та трейдингової аудиторії. Це пов'язано зі швидким розвитком, вдосконаленням, збільшенням складності та різноманіттям функціоналу, що втілюється у нових системах.

На сьогоднішній день існують такі види систем управління персональним портфелем цінних паперів:

1. Системи для розрахунку та аналітики фінансових показників.
2. Системи для автоматизації деяких процесів інвестування, зокрема механізмів стратегій та програмного прийняття інвестиційних рішень.
3. Системи, які поєднують в собі функціонал обох вищезгаданих типів інформаційних систем управління цифровими активами.

1.3 Огляд і аналіз існуючих систем

1.3.1 MorningStar

Один із передових інструментів для відстеження інвестицій, який називається Morningstar Portfolio Manager, надає інвесторам можливість імпортувати або вручну керувати своїми активами та отримувати цінну інформацію (див. рис. 1.3.1.1).

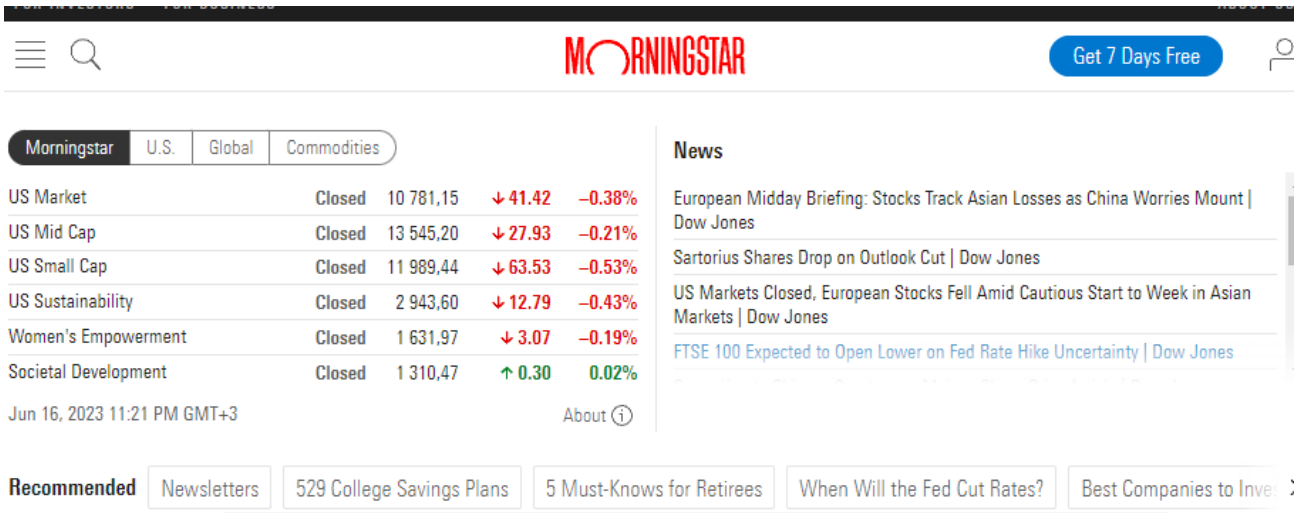

Рис. 1.3.1.1 - Портфель MorningStar в особистому кабінеті.

Менеджер портфеля Morningstar використовує інший підхід, порівняно з фінансовими терміналами аналогічних ресурсів, оскільки він не має функції автоматичного імпортування даних від брокера чи банку. У Morningstar необхідно вручну вводити кожну інвестицію в інструмент, навіть при оголошенні компанією про введення автоматизованого рішення.

Хоча введення даних інвестиційного рахунку в портфельний менеджер Morningstar може забирати більше часу та зусиль, аналіз, який надає цей інструмент, вартий зусиль. Він надає основну інформацію про кожну з ваших інвестицій, включаючи поточну ціну, щоденні зміни вартості та її відсоткову вагу у загальному портфелі. Крім того, ви можете додати кілька портфелів, включаючи акції, облігації та готівкові заощадження.

Менеджер портфеля Morningstar також надає надійні дані про загальну ефективність портфеля. Він відображає загальну прибутковість вашого портфеля за місяць і рік, порівнюючи її з вибраним еталоном. Однак основною сильною стороною Morningstar є його функціональний пакет Instant X-Ray.

Функція Instant X-Ray надає користувачу докладну інформацію про розподіл активів його портфеля (див. рис. 1.3.1.2). Вона відображає стиль інвестування портфеля як для акцій, так і для облігацій. Instant X-Ray аналізує розподіл інвестицій за секторами, типами акцій і навіть за регіонами. В результаті, користувач отримує середньозважений коефіцієнт витрат фонду його портфеля. Крім того, Instant X-Ray показує відсоткове співвідношення кожної окремої інвестиції в портфелі загалом. Це особливо корисно для тих, хто інвестує в пайові фонди.

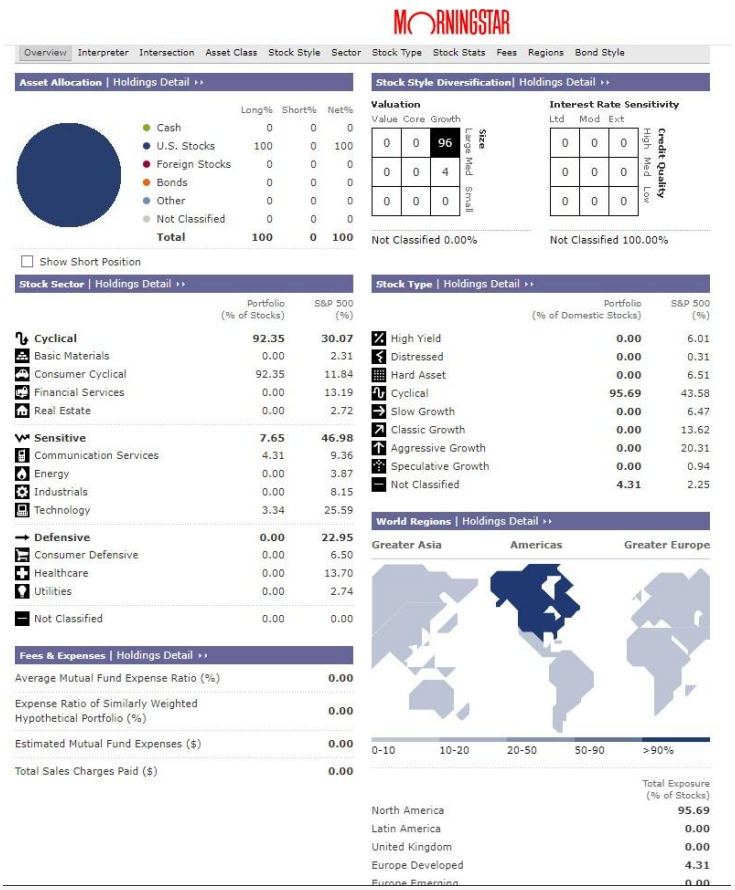

Рис. 1.3.1.2 - Результат портфелю після опрацювання Instant X-Ray

Morningstar є безкоштовна підписка, а є і платна.

Використання Portfolio Monitor відкрите для всіх користувачів, незалежно від їхнього членства, проте деякі інструменти вимагають платного членства або підписки. Наприклад, детальний аналіз Instant X-Ray, окремий інструмент X-Ray, інсайти, розширений скрінінг (пошук активів за певними параметрами) та пошук схожих активів доступні лише через платні послуги. Мінімальна вартість підписки становить почті $250 на рік використання.

1.3.2 Yahoo! Finance

Це є медіаресурс, що входить до мережі Yahoo!. Він надає широкий спектр фінансових новин, даних та коментарів, включаючи котирування акцій, прес-релізи, фінансові звіти та оригінальний контент. Ресурс також пропонує ряд онлайн-інструментів для управління особистими фінансами. Yahoo! Finance також має функцію перегляду новин та курсів крипто валют (див. рис. 1.3.2.1). Цей ресурс доступний через веб-версію, яка працює в будь-якому браузері незалежно від платформи, а також мобільні додатки для Android та iOS.

Однак Yahoo! Finance не обмежується лише наданням новин та котирувань акцій. Він також дозволяє користувачам створювати власний портфель активів для більш детального аналізу та активного відстеження їх інвестицій. Ця функція дозволяє збирати в одному місці активи, які мають найбільше значення для користувача, та отримувати актуальну інформацію про них. Yahoo! Finance пропонує такі можливості:

- Створення декількох портфелів та їх редагування за потребою.
- Додавання, редагування та видалення транзакцій (купівля/продаж) акцій компаній, які цікавлять користувача.
- Відстеження готівкових рахунків та непублічних акцій.
- Налаштування відображення графіків та інших параметрів.
- Створення власних звітів та аналітичних даних, вибравши необхідні

показники.

- Використання скрінерів для пошуку активів, що відповідають заданим параметрам.

- Експорт та імпорт даних портфеля у форматі CSV.

Це лише частина можливостей, які пропонує Yahoo! Finance. Ресурс також забезпечує детальну інформацію про ринки та окремі компанії, включаючи історичні дані, статистику та рекомендації аналітиків. Це робить Yahoo! Finance популярним серед фінансистів та інвесторів, і він входить до списків найвідвідуваніших веб-сайтів у категорії "Новини та медіа".

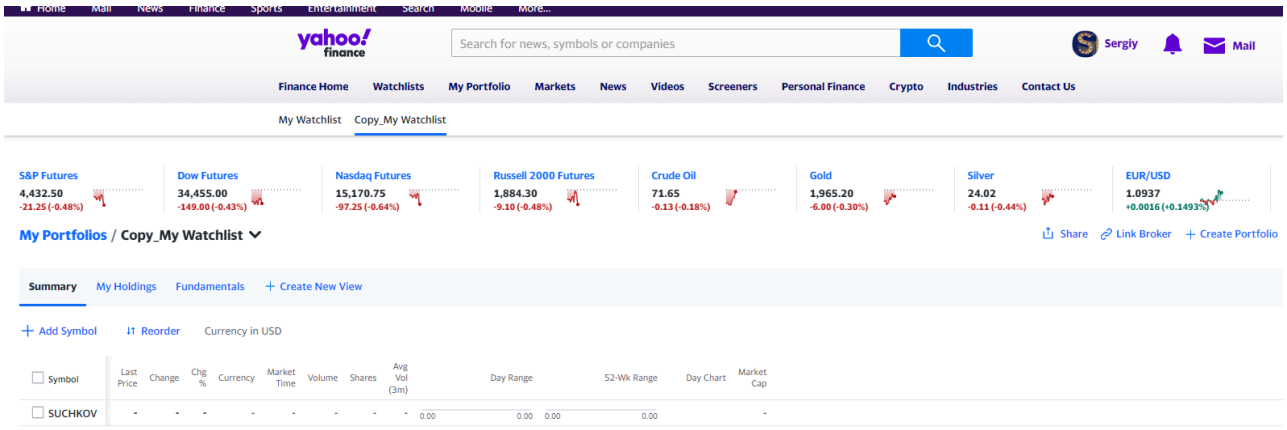

Рис. 1.3.2.1 - Представлення свого портфелю в YF.

1.3.3 Google Finance

Google Finance є платформою, яка надає фінансову інформацію про багато міжнародних компаній, що торгуються на різних фондових біржах. Цей сервіс забезпечує доступ до курсів та рейтингів цінних паперів, прес-релізів та фінансових звітів цих організацій. Він також включає інструменти, такі як Google News, для пошуку і надання відповідної інформації про компанії (див. рис 1.3.3.1).

Крім того, на платформі можна знайти історичні дані у вигляді графіків, а також використовувати функцію порівняння графіків акцій різних компаній,

індексів, портфелів та іншого. Google Finance також надає інформацію про тенденції на ринку, популярні акції в Google, календар дат публікацій фінансових звітів компаній та інші корисні функції..

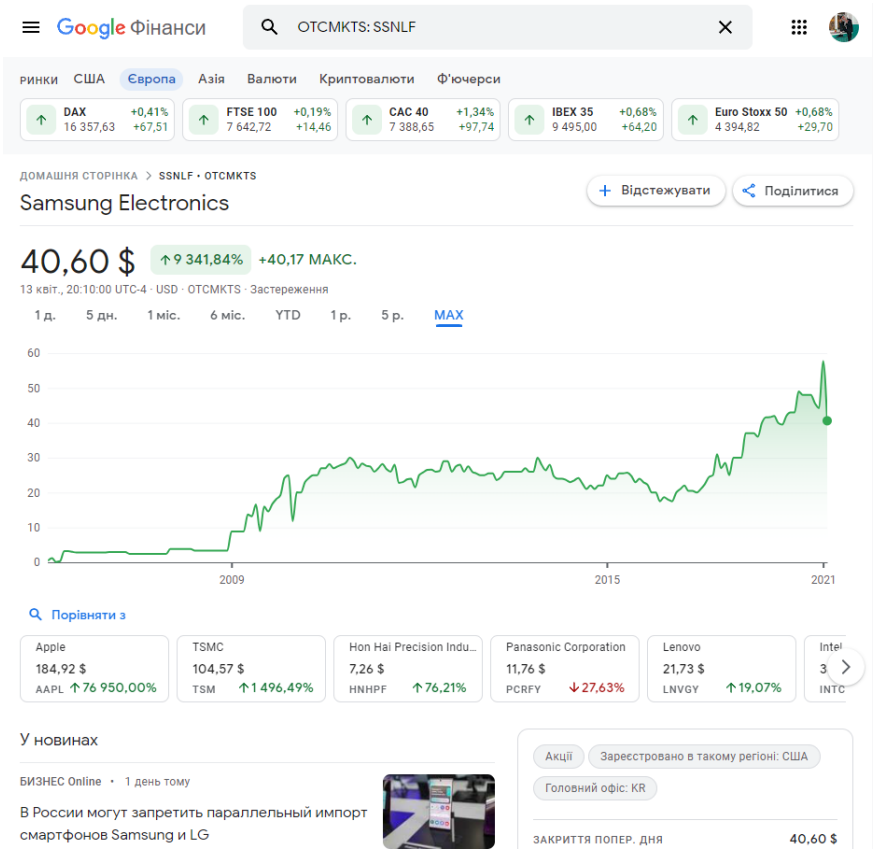

Рис. 1.3.3.1 - Інформацій на Google Finance про компанію Samsung Electronics, для прикладу.

Платформа надає широкий спектр сервісів для управління особистою фінансовою інформацією. Один з цих сервісів дозволяє користувачам створювати власні списки відстеження або портфелі (див. рис. 1.3.3.2), включаючи акції, які їх цікавлять або якими вони володіють. Це дозволяє користувачам бути в курсі останніх подій, переглядати публічну звітність компаній та основні фінансові показники.

Крім того, доступний перегляд основної інформації про портфель, такий як відсоток у складі портфелю компаній, що належать до домінуючої категорії за капіталізацією, їх дивідендну характеристику та співвідношення "ціна-прибуток". Крім того, можна оцінити характеристики портфеля з точки зору ваги компаній з

високим екологічним рейтингом.

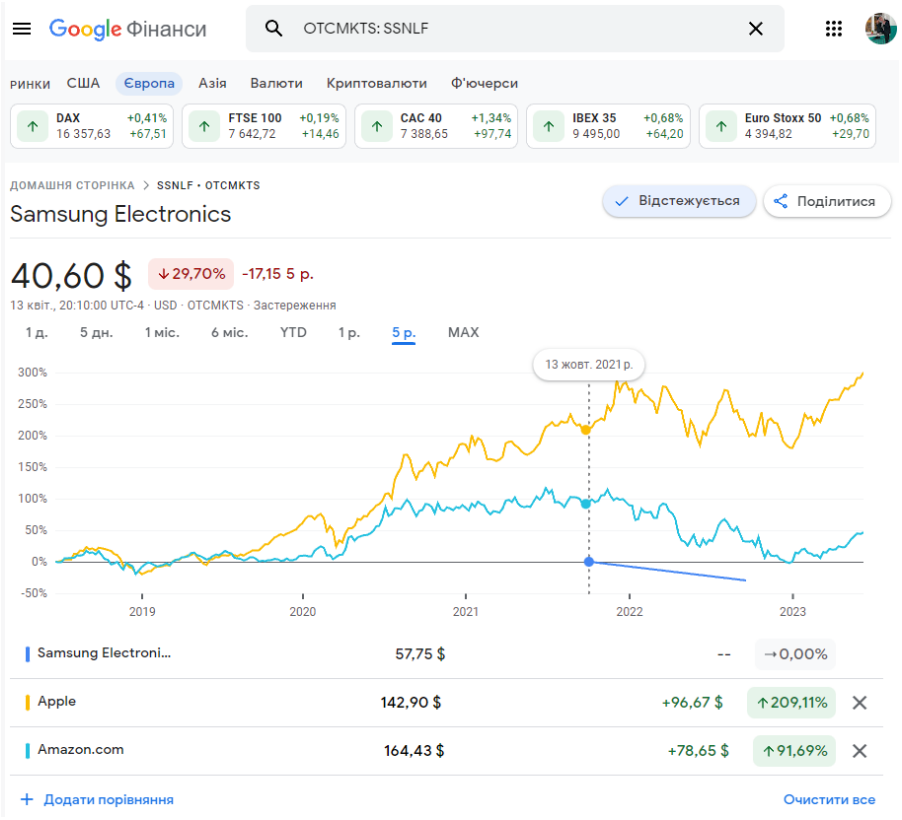

Рис. 1.3.3.2 - Порівняння ціни акцій Samsung Electronics, Apple, Amazon.com на Google Finance відносно останніх 5 років.

Отже, Google Finance надає користувачеві основну інформацію, яка дозволяє бути в курсі подій у фінансовому світі, а також забезпечує поверхневий аналіз його портфелю.

1.4 Підсумки огляду і аналізу існуючих веб додатків

Перш ніж представити результати огляду та порівняння наявних інструментів для ведення обліку акційного портфеля, важливо відзначити один суттєвий аспект. При виборі розглянутих ресурсів для огляду, крім їх поширеності, широкого функціоналу та можливостей, також враховувалась доступність.

На ринку існує значна кількість аналогічних інструментів, рекомендованих аналітиками, що допомагають вирішувати складні завдання, такі як пенсійні заощадження, інвестиції для освіти тощо. Вони враховують цілі користувача, його перспективу планування та бюджет, надаючи детальну, цільову аналітику і плани для досягнення поставлених цілей.

Очевидно, що такі можливості є більш затребуваними серед користувачів, оскільки програмні можливості таких інструментів спрямовані на виконання конкретних планів. Нажаль, багато з них накладають певні обмеження, наприклад, необхідність підтвердження адреси проживання в США або ЄС шляхом пред'явлення відповідних документів. Іншою суттєвою перешкодою може бути обов'язковість зв'язку з принаймні одним банківським або брокерським рахунком у організаціях Західної Європи або США. Очевидно, що лише обмежена кількість українських громадян може задовольнити ці вимоги. З урахуванням цього факту, вибір інструменту для віртуальної торгівлі акціями суттєво обмежується.

Розглянуті ресурси, зі свого боку, надають широкий функціонал, і кожен з них має свої переваги та недоліки. Кожен з цих ресурсів надає користувачеві значну кількість фінансової інформації про показники компаній, зміну цін акцій з часом, відбирає новини, що стосуються безпосередньо конкретних компаній, дозволяє групувати активи в портфелі та редагувати їх. Усе це забезпечується зручним для користувача способом з використанням візуально привабливого представлення. Проте, кожен з цих ресурсів має свої власні недоліки, які можуть бути індивідуальними для кожного сервісу.

Наприклад, основний потенціал ресурсу Morningstar прихований за платною версією, плата за яку може бути проблематичною для новачка. Навіть у безкоштовній версії можуть виникнути труднощі – велика кількість фінансової інформації, наданої ресурсом, розміщена на багатьох вкладках та панелях, що може заплутати нового користувача; крім того, документація для користувача не є повною, що створює багато питань. Але навіть із наявною документацією, у новачка можуть виникнути проблеми.

Важливо відзначити, що цей аспект більше відноситься до інструменту управління портфелем. Сторінки конкретних компаній на цьому ресурсі презентовані зрозумілою структурою та грамотним поданням інформації. При їх перегляді необхідно мати розуміння фінансової термінології та значення показників, що є важливим для інвесторів. Проте, для портфельних менеджерів потрібно також розбиратися в конкретних полях та інструментах, які пропонуються на даному ресурсі.

Окремо варто зазначити, що графічний інтерфейс цього менеджера портфеля відстає від його конкурентів та сучасних технологій у візуальному представленні інформації. Відсутня достатня інтерактивність та гнучкість під час роботи з таблицями та графіками, а стилістичне оформлення та деякі візуальні елементи залишають враження застарілості. Це не можна вважати сильним аспектом для ресурсу, за який користувачі безпосередньо платять.

Google Finance приваблює своїм візуальним представленням даних та покриттям компаній (включаючи біржі Північної Америки, Європи та Азії). Однак, проблема Google Finance полягає в обмеженій кількості фінансових даних. Під час перегляду сторінки компанії користувач може побачити лише два десятки показників, а при перегляді портфеля - ще менше. У такому складному процесі, як управління та облік акційного портфеля, такий обсяг інформації може бути недостатнім.

Також варто зазначити, що практично відсутня аналітика портфеля в Google Finance. Єдине, що відображається на сторінці портфеля, це прибуток/збиток портфеля за весь час і один день, а також деякі дані про приналежність компаній до різних категорій, таких як капіталізація, дивідендність та екологічний рейтинг.

Ще одним недоліком є обмеженість можливості зміни діапазону даних на графіках. Користувач може обирати лише певні проміжки (1 або 5 днів, 1 або 6 місяців, тощо), і не може налаштувати відображення даних за власним бажанням.

У порівнянні з Google Finance, Yahoo! Finance вирішує проблему надмірного завантаження графічного інтерфейсу даними. Вони надають користувачу можливість вибору того, що відображати, а саме створювати власні представлення (views), в яких він може відбрати необхідні показники зі списку понад 70 доступних та впорядкувати їх за власним розсудом.

Графіки зміни ціни акцій у Yahoo! Finance не мають обмежень, які є у Google Finance. Користувач може широко масштабувати, налаштовувати, додавати статистичні індикатори та порівнювати їх з курсами інших компаній. Однак, це стосується лише графіків на рівні компаній, тоді як графік на рівні портфеля представлений тільки на сторінці самого портфеля порівняно з індексом S&P 500. Крім того, розмір цього графіка надто малий та немасштабований для проведення аналізу.

Особливий недолік Yahoo! Finance полягає в тому, що він не надає можливості проводити аналітику портфеля в цілому, а лише відображає прибуток або збиток.

У контексті повної масштабності ресурсу та обширного обсягу наданої інформації, це представляє собою важливу проблему, оскільки вона знижує сутність самого портфеля і не забезпечує користувача необхідною інформацією.

Таблиця 1.1 -Порівняння систем обліку інвестицій.

Назва	Переваги	Недоліки
MorningStar	Загальна кількість показників та інформації про компанії є обширною. Підтримка різних типів активів, таких як акції, облігації, готівка та інші. Можливість створення та управління кількома портфелями одночасно. Чітка аналітика портфелю, що дозволяє оцінити його стан та результативність. Наявність інструменту Instant X-Ray, який здійснює оцінку та аналіз портфелю, стилю інвестування та розподілу активів за різними критеріями.	Обмежена доступність детальних аналітичних результатів, яка доступна лише через платну підписку. Відстарілий інтерфейс користувача з графічним дизайном. Недостатня візуалізація інформації, оскільки значна частина її представлена у вигляді таблиць. Розкидання значного обсягу інформації по різних вкладках та панелях. Недостатньо ефективне навчання користувачів засобами документації.

Yahho! Finance	Ефективний добір новин, фінансової інформації та показників. Можливість користувача створювати власне представлення, яке відображатиме лише ті дані, що мають значення для користувача. Застосування скрінерів для пошуку компаній за різними критеріями. Широкі можливості налаштування та відображення графіків.	Поверхнева аналітика портфелю.

Google Finance	Привабливе графічне оформлення. Агрегація новин та статей, що стосуються компанії або портфеля. Можливість накладання графіків зміни ціни акцій різних компаній. Широкий охоплення різних бірж. Можливість відстеження та обліку різних типів активів.	Недостатній обсяг фінансової інформації. Обмежений перелік періодів для перегляду графіків.

1.5 Постановка задачі для розроблення веб додатку

Ураховуючи вищезазначене, розробка веб-додатка набуває значущості і актуальності. Таким чином, головною метою дипломного проекту є створення прототипу інформаційної системи для управління та обліку персонального портфеля цінних паперів з метою здійснення всебічної оцінки ефективності інвестицій. Відповідно до цієї мети, в дипломній роботі необхідно вирішити наступні завдання:

Розробити веб додаток для управління та обліку персонального портфеля цінних паперів. Цей прототип буде інструментом для моделювання процесів інвестування, сприяючи засвоєнню користувачем фінансової грамотності та інвестиційних знань, а також спостереженню за ефектами різних підходів до інвестиційних процесів на практиці. Прототип не передбачає підтримку реальних торгів активами, але має враховувати можливість підключення до програмного модуля для реальних торгів на біржі в майбутньому.

Встановити вимоги до розроблюваного програмного продукту на основі аналізу аналогів, їх порівняння, огляду недоліків існуючих систем, а також аналізу специфіки предметної області.

Забезпечити розширення веб додатку, звертаючи особливу увагу на створення потрібної інфраструктури, що підтримуватиме високі показники розширюваності, переносимості, гнучкості та масштабованості як з точки зору архітектури, так і з точки зору технічної реалізації.

Спроектувати програмний продукт таким чином, щоб забезпечити його ефективну підтримку в майбутньому, легке розширення функціоналу та масштабування. Зокрема, передбачити можливість зміни платформи та операційної системи, на яких буде працювати програмний продукт, а також потенційний перехід до використання "хмарних" сервісів.

Розробити простий і зрозумілий інтерфейс користувача, який міститиме необхідну інформацію, не перевантажуватиме користувача та легко

сприйматиметься.

Отже, дипломна робота буде орієнтована на розробку веб додатку для управління та обліку персонального портфеля цінних паперів. Прототип буде розроблятись для ОС Windows як десктопний додаток, але будуть передбачені можливості зміни платформи та операційної системи в майбутньому, а також переносу інфраструктури в "хмарну" інфраструктуру.

Також, одним з ключових аспектів розробки цього веб-додатка буде інтеграція з API IEX. IEX API є фінансовим інтерфейсом програмування, який надає доступ до різноманітних фінансових даних, таких як котирування цінних паперів, історичні дані, фінансові звіти та багато іншого.

Інтеграція API IEX у веб-додаток дозволить отримувати актуальні дані про ціни цінних паперів, зміни цін, обсяги торгів та іншу фінансову інформацію, необхідну для користувачів при управлінні їхнім портфелем. Завдяки цьому, користувачі зможуть отримувати оновлені дані про свої інвестиції у режимі реального часу, а також отримувати доступ до історичних даних для аналізу та вивчення ринкових тенденцій.

Для інтеграції з API IEX необхідно буде встановити з'єднання з сервером API та використовувати відповідні HTTP-запити для отримання потрібної інформації. Дані, отримані з API IEX, можна буде використовувати для відображення веб-додатку, аналізу даних, створення графіків та забезпечення користувачам зручного інтерфейсу для перегляду та аналізу їхнього портфеля.

Інтеграція API IEX до веб-додатку значно розширить його функціональність та користувацький досвід, дозволяючи користувачам отримувати актуальну інформацію про свої інвестиції та ринкові тренди, що допоможе їм приймати більш обґрунтовані рішення з управління своїм портфелем цінних паперів.

1.5.1 Додаткові вимоги до функціоналу розроблюваного веб додатку

Можливість додавання акцій конкретної компанії до портфеля з попередньо визначеного списку організацій, що торгуються на фондовій біржі. Користувач може вказати назву компанії (символ або ідентифікатор), дату купівлі, обсяг та ціну акцій. Додавання здійснюється як окремі транзакції, тобто користувач вводить інформацію тільки про нові покупки, а не повторно вводить всі дані щодо вартості портфеля.

Відображення транзакцій продажу акцій в загальному списку транзакцій, розрахунок та відображення різниці між загальною сумою покупок та отриманою сумою від продажу акцій компанії.

Функціонал для автоматичного розміщення транзакцій купівлі/продажу акцій компанії на основі заданих користувачем преференцій або стратегій.

Графічне відображення прибутковості, яке показує зміну ціни акцій конкретної компанії порівняно з їхньою купівельною ціною. Це може бути відображено як для окремої компанії або окремої транзакції, так і для всього портфеля.

Обчислення та відображення середньої ціни придбаних акцій та різниці з актуальним значенням.

Представлення історичної інформації про зміну ціни акцій компанії, включаючи мультиплікатори та інші дані для технічного аналізу. Цю інформацію можна відображати як у текстовому, так і у графічному вигляді.

Можливість відображення зміни ціни акції компанії за заданий користувачем період, починаючи від певної дати до певної дати.

Реалізація цих функцій у прототипі програмного продукту дозволить залучити більше користувачів, адже будуть надані можливості як для трейдерів, так і для інвесторів. Це збільшить потенційну аудиторію користувачів і дозволить отримати більш повну оцінку прототипу, а також більше ідей для подальшого розвитку та вдосконалення програмного продукту.

Крім того, для інтеграції з зовнішніми джерелами фінансових даних, такими як інформація про ціни акцій компаній, рекомендується використання API IEX. Це дозволить отримувати актуальні та достовірні дані для аналізу та моніторингу цінних паперів у реальному часі. Застосування API IEX збільшить функціональність та цінність програмного продукту для користувачів, надаючи їм доступ до надійних даних з фондового ринку.

1.6 Монолітна архітектура

У монолітній архітектурі внутрішня структура системи не виражена явно. Вона представляє собою набір процедур, які оперують загальними глобальними даними та взаємодіють один з одним, а також зовнішніми ресурсами та користувачами. . Приклад такої архітектури (див. рис. 1.6.1).

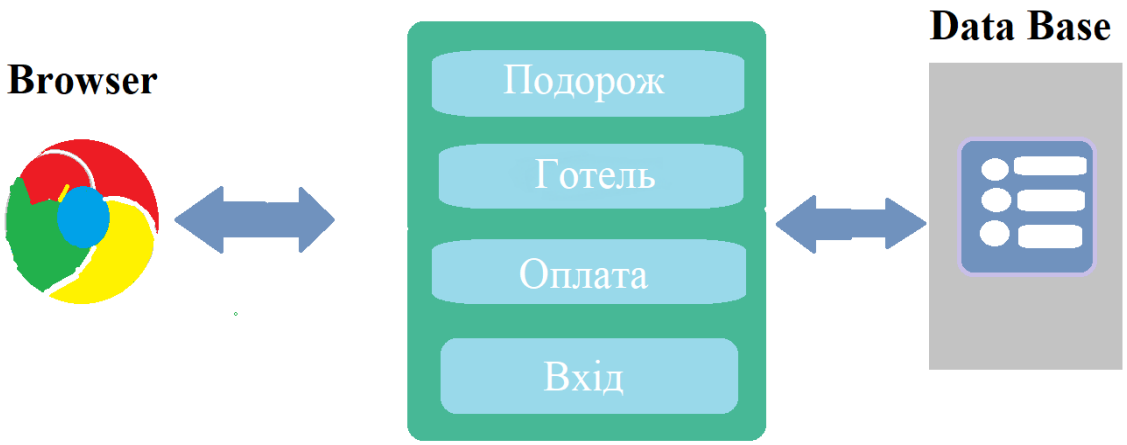

Рис. 1.6.1 - Схема структури монолітної системи.

Монолітна архітектурна структура програмних систем має ряд недоліків. Одним з них є нездатність моноліту відповідати потребам бізнесу щодо прискорення, оскільки зміна та адаптація великих та сильно залежних модулів в

системі стає складним завданням.

При використанні монолітної архітектури апаратно залежний та апаратно незалежний код тісно переплітаються, що ускладнює розвиток системи або її перенесення на іншу апаратну платформу. Крім того, присутність безсистемних розгалужених зв'язків між компонентами системи з монолітною архітектурою призводить до необхідності зміни багатьох компонентів системи, навіть якщо потрібна зміна лише одного з них. Це може спричинити ефект снігової лавини, де зміна однієї процедури вимагає зміни кількох інших, які, в свою чергу, вимагають змін ще кількох інших процедур, і так далі.

Однією з обмежень монолітної програмної системи є її повне замикання в контексті поведінки. Під час роботи система може взаємодіяти з іншими службами або сховищами даних, але основа її поведінки знаходиться у єдиному процесі, і вся система розгортається як один елемент. Для масштабування системи з такою архітектурою на горизонтальному рівні потрібно повністю дублювати всю систему на кількох серверах або віртуальних машинах.

Хоча простота та швидкість реалізації є перевагами такого підходу, оскільки не потрібно витрачати час на проектування компонентів, розподіл відповідальності та правил взаємодії, монолітна архітектура все ж має свої обмеження та проблеми, які можуть ускладнити подальший розвиток та масштабування системи.

1.7 Багаторівнева архітектура

У відповідь на обмеження, пов'язані з монолітною архітектурою щодо розширюваності, переносимості та майбутньої підтримки, з'явилася багаторівнева архітектура. Основна концепція багаторівневої архітектури полягає у тому, що рівні взаємодіють між собою лише через їхні інтерфейси, приховуючи внутрішню реалізацію кожного рівня від інших. Це дозволяє змінювати внутрішню реалізацію функціоналу одного рівня без необхідності змінювати інші компоненти системи та без ризику порушити їх правильну роботу, що забезпечує високу розширюваність системи. З такою архітектурою навіть можна повністю замінити весь рівень, просто забезпечивши стандартний інтерфейс його взаємодії з іншими рівнями програмної системи. .

Приклад багаторівневої архітектури, (див. рис 1.7.1).

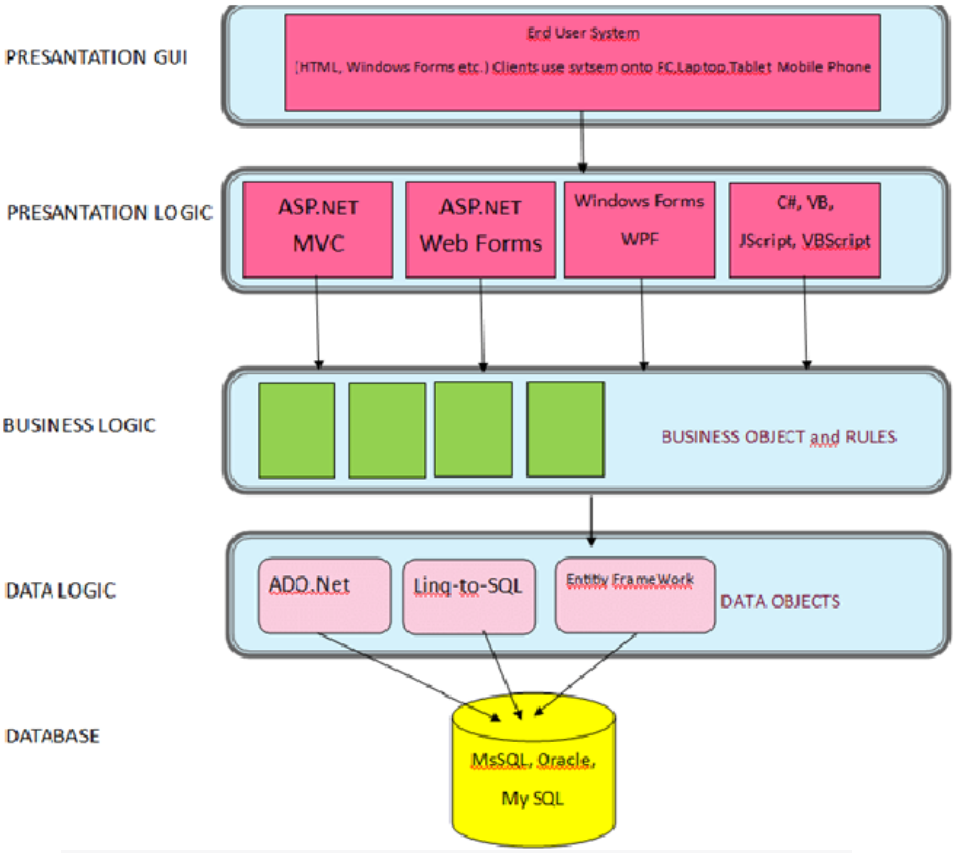

Рис. 1.7.1 - Багаторівнева архітектура, та її схема.

Багаторівнева архітектура володіє наступними перевагами:

Централізована бізнес-логіка дозволяє змінювати правила обробки даних без необхідності модифікувати клієнтський додаток та розповсюджувати його на клієнтські машини. Бізнес-логіка знаходиться на рівні сервера програмної системи, тому зміни відразу відображаються на клієнтах.

Кожен рівень виконує конкретний набір функцій, який не дублюється у різних рівнях, і не залежить від реалізації та структури даних інших рівнів. Це дозволяє змінювати вміст кожного рівня без ризику конфліктів та порушення працездатності інших компонентів системи.

Висока гнучкість щодо конфігурації та розгортання програмної системи.

Для горизонтального масштабування системи з такою архітектурою достатньо дублювати лише клієнтську частину, а не всю систему на кожному сервері або віртуальній машині.

Однак, недоліком такої архітектурної конструкції є складніше та затратніше проектування у порівнянні з монолітною архітектурою.

1.8 Мікросервісна архітектура

Мікросервісна архітектура є одним з видів сервіс-орієнтованої архітектури програмного забезпечення, яке використовується для розробки розподілених програмних систем. Основна ідея мікросервісної архітектури полягає в створенні невеликих, слабко зв'язаних та легко змінюваних модулів, які називаються мікросервісами. У цій архітектурі мікросервіси взаємодіють між собою через мережу, працюючи на досягнення спільної мети.

Основні компоненти мікросервісної архітектури включають сервіси, сервісну шину, зовнішню конфігурацію, шлюз API та контейнери. Кожен сервіс виконує окрему відповідальність і є незалежним від інших сервісів. Зміни, внесені до одного сервісу, не повинні впливати на роботу інших.

Мікросервісна архітектура має декілька переваг:

Взаємозамінність та незалежність мікросервісів, що забезпечує слабке зв'язування завдяки високому рівню ізоляції та модульності.

Гнучкість та масштабованість системи.

Можливість реалізації мікросервісів за допомогою різних програмних засобів, які найкраще підходять для кожного окремого модуля, забезпечуючи при цьому взаєморозуміння за допомогою інтерфейсу.

Вирішення проблеми потоків даних, яка іноді виникає в багаторівневій архітектурі.

Всі ці переваги роблять мікросервісну архітектуру привабливим варіантом для розробки програмного забезпечення, особливо в складних та масштабованих системах (див. рис. 1.8.1).

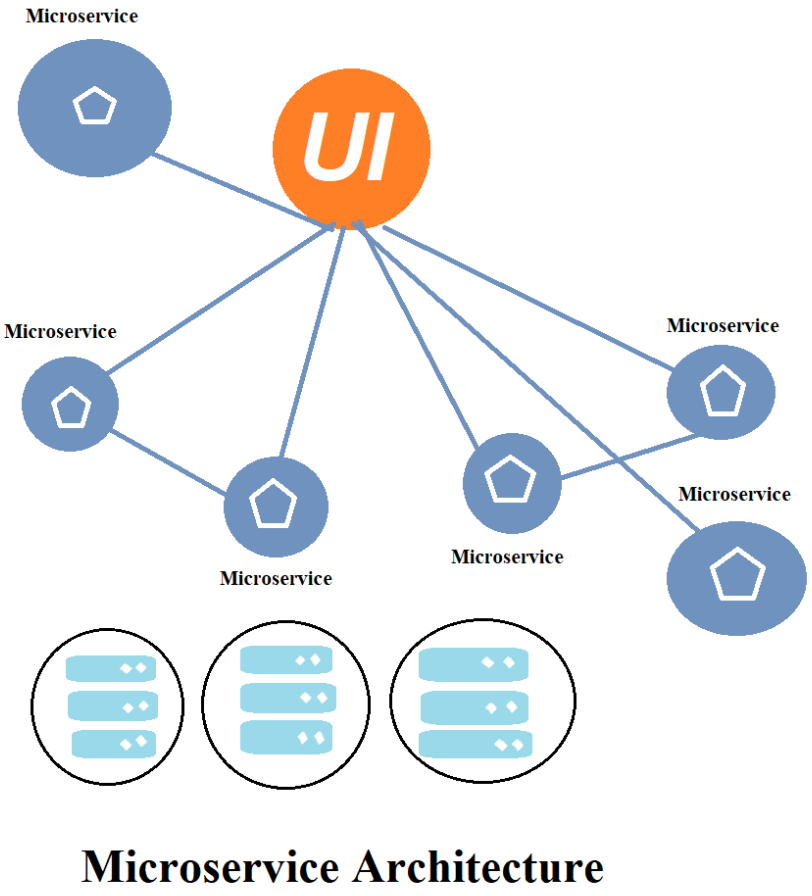

Рис. 1.8.1 - Схема мікросерверної архітектури.

1.9 Висновки до розділу

Дійсно, мікросервісна архітектура має свої недоліки. Основні недоліки мікросервісної архітектури включають:

Підвищений ризик збою під час обміну даними: Оскільки мікросервіси взаємодіють між собою через мережу, існує підвищений ризик збоїв під час передачі даних між сервісами. Нестабільна мережа або помилки в комунікації можуть призвести до проблем з доступністю та надійністю системи.

Складність керування багатьма сервісами: Зі зростанням кількості мікросервісів в системі стає складніше керувати та моніторити їх стан. Необхідно розробляти механізми керування, моніторингу та логування, щоб забезпечити ефективне управління всіма сервісами.

Складність тестування в розподіленому середовищі: Тестування мікросервісної архітектури вимагає комплексного тестування в розподіленому середовищі. Кожен сервіс може мати власні залежності та вимоги до інфраструктури, що робить тестування складнішим та вимагає більшої уваги до деталей.

Складність проектування та реалізації: Мікросервісна архітектура вимагає детального проектування та реалізації. Вона потребує вивчення взаємодії між сервісами, створення стабільних інтерфейсів та управління конфігурацією. Все це вимагає більше часу та зусиль у порівнянні з іншими архітектурними підходами.

Вирішення проблем розподіленої архітектури: Мікросервісна архітектура стикається з проблемами, які властиві розподіленій архітектурі, такими як затримки в мережі, балансування навантаження, управління транзакціями та збереження консистентності даних між сервісами.

Враховуючи ці недоліки, розробники повинні ретельно розглянути вибір мікросервісної архітектури для своїх проектів та забезпечити необхідні механізми для керування та підтримки складності цього підходу.

РОЗДІЛ 2. ОГЛЯД ІСНУЮЧИХ МЕТОДІВ

2.1 Вибір архітектурного рішення для розроблюваного програмного продукту

В результаті порівняння різних архітектурних підходів та аналізу вимог до програмного продукту та його функціоналу, для розроблюваного прототипу програмної системи було обрано трьохрівневу архітектуру з поділом системи на рівні та відповідальністю між ними і вважається найкращим для даного дипломного проекту, оскільки воно найбільш ефективно відповідає поставленим нефункціональним вимогам у порівнянні з іншими архітектурними рішеннями.

На рівні користувацького інтерфейсу (UI Layer) реалізується відображення даних користувача відповідно до його запитів та виконуваних дій в програмній системі. Тут також знаходиться логіка, яка відноситься виключно до користувацького інтерфейсу, наприклад, сортування та групування рядків та колонок у таблицях відображення даних (Data Grid View). Користувач безпосередньо взаємодіє з цим рівнем, який включає компоненти інтерфейсу користувача та механізми отримання введення від користувача.

Рівень бізнес-логіки (Business Logic Layer) складається з компонентів, що відповідають за обробку даних, отриманих від рівня представлення (рівня користувацького інтерфейсу). Він реалізує необхідну логіку веб-додатку, проводить обчислення, взаємодіє з рівнем доступу до даних та передає результати обробки на рівень представлення.

Рівень доступу до даних (Data Access Layer) містить моделі, що описують сутності, які використовує програмна система, із апити до чогось, наприклад, клас контексту даних Entity Framework.

2.2 Вибір мови програмування

На поставлене питання можна почути ряд відповідей, але однією із найпоширеніших з них буде C# . Підтримка узагальнень, лямбда-функцій, LINQ (Language Integrated Query – запити, інтегровані в мову) і асинхронних операцій з часом успішно перетворила C#, так що тепер вона класифікується як багатопарадигмова мова програмування. Код на C# може бути традиційно імперативним, або він може бути доповнений парадигмами декларативного або функціонального програмування .

На найнижчому рівні .NET забезпечує відмінно побудовану інфраструктуру для базових та складних типів даних C# та управління використанням апаратних ресурсів системи. В свою чергу обширна бібліотека класів .NET Framework надає підтримку для вирішення багатьох поширених завдань, які зустрічаються у різних типах програмування. До них відносяться: математичні обчислення, відображення, робота з колекціями, широкий вибір структур даних та проведення операцій над ними, файловий запис/зчитування, робота з потоками (керована та некерована багатопоточність) та асинхронність, зчитування і запис XML (Extensible Markup Language – розширювана мова розмітки) і JSON (JavaScript Object Notation – запис об'єктів JavaScript) тощо.

Платформа .NET з використанням мови програмування C# підтримує створення програмних продуктів для наступних платформ: Windows, MacOS, Linux, Android, iOS.

Один і той самий код, написаний на C#, може виконуватись на всіх перелічених вище платформах (операційних системах) завдяки специфічному механізму компіляції коду, який реалізований в середовищі виконання CLR (Common Language Runtime). CLR компілює вхідний C# код в проміжний код CIL (Common Intermediate Language), який за своєю структурою є близьким до машинного коду. Цей механізм дозволяє запускати один і той самий виконуваний файл програми середовища .NET незалежно від операційної системи пристроюю:

виклики API середовища .NET транслюються в системні виклики ОС [3]. Отже, мова програмування C# надає можливість розробляти один продукт для різних платформ в рамках єдиної екосистеми .NET.

2.3 Технологічні засоби для доступу до даних

Для реалізації доступу до даних, створення моделі бази даних та маніпуляцій з нею в рамках розроблюваного програмного продукту, одним з найпоширеніших та потужних інструментів є Entity Framework.

Entity Framework отримав широке використання порівняно з іншими ORM-інструментами завдяки своїм численним перевагам.

Цей фреймворк підтримує різноманітні системи керування базами даних, що дає можливість працювати з будь-якою СУБД через відповідні провайдери, що надаються EF.

Entity Framework також надає уніфікований API для роботи з даними. Якщо, наприклад, з'явиться необхідність змінити цільову СУБД, основні зміни в проекті стосуватимуться переважно конфігурації та налаштування підключення до відповідних провайдерів. Код, який безпосередньо працює з даними, отримує їх, додає до бази даних і т.д., залишиться без змін.

Технологія доступу до даних Entity Framework може бути використана на різних платформах стека .NET, включаючи Windows Forms, консольні програми, WPF, UWP та ASP.NET Core. Будучи кросплатформеним, EF дозволяє використовувати його в розробці програмних продуктів для будь-якої підтримуваної платформи .NET.

Однією з особливостей Entity Framework як технології ORM є використання запитів LINQ для отримання даних з бази даних. LINQ є потужним інструментом мови C#, що дозволяє створювати різні запити для вибірки об'єктів, пов'язаних з різними асоціативними зв'язками, включаючи складні фільтрацію, групування,

сортування. Entity Framework, при виконанні запиту, транслює LINQ-вирази в вирази, зрозумілі для конкретної СУБД.

З урахуванням вищезазначених особливостей Entity Framework можна стверджувати, що ця технологія найкраще підходить для вирішення задач, пов'язаних з доступом до даних та їх обробкою в рамках розроблюваного прототипу програмної системи, а також відповідає поставленим вимогам.

2.4 Технологічні засоби для розробки інтерфейсу користувача

Для реалізації графічних інтерфейсів на платформі .NET використовуються різні технології, зокрема WinForms, WPF та UWP. Проте, серед них найпростішою і зручною є технологія WinForms.

Підхід до розробки програм на платформі Windows Forms базується на графічному інтерфейсі GDI (Graphics Device Interface), який відповідає за представлення графічних об'єктів та їх відображення на пристроях [11]. WinForms надає широкі можливості для створення графічних інтерфейсів, дизайну та взаємодії з ними.

Основними перевагами WinForms є високий рівень придатності для розробки і широкі можливості роботи з нею в сучасних редакторах IDE, наявність документації по розробці і простота створення різних елементів інтерфейсу користувача, а також легкість реалізації логіки представлення даних.

Windows Presentation Foundation (WPF) є великим API-інтерфейсом для створення десктопних графічних програмних інтерфейсів. Однією з особливостей WPF є можливість декларативного визначення графічного інтерфейсу з використанням мови розмітки XAML, що базується на XML. Це надає альтернативу програмному створенню графіки та елементів управління.

Порівняно з WinForms, WPF має ширші можливості для створення складних веб додатків, але процес розробки на базі цієї технології є складнішим.

2.5 Формування технічної специфікації програмного продукту та вимоги щодо використання API IEX

Формування технічної специфікації програмного продукту є заключним етапом проектування. Цей процес заснований на результатів огляду технологій розробки різних компонентів програмної системи та їх порівняльного аналізу.

Технічна специфікація включає такі основні елементи:

1. Архітектурна конструкція: програмний продукт базується на багаторівневій архітектурі, що включає три рівні. Перший рівень - рівень доступу до даних (Data Access Layer), другий - рівень бізнес-логіки (Business Logic Layer), а третій - рівень представлення (UI Layer).

2. Мова програмування: для реалізації програмного продукту використовується мова програмування C#.

3. Технологія для доступу до даних та створення моделі бази даних: використовується технологія Entity Framework, яка дозволяє реалізувати доступ до даних та взаємодію з базою даних.

4. Технологія для реалізації інтерфейсу користувача: використовується технологія WinForms, яка забезпечує створення графічного інтерфейсу для взаємодії з користувачем.

5. Для забезпечення логічності і послідовності структури програмного продукту рекомендується використовувати об'єктно-орієнтовані конструкції, такі як інтерфейси, абстрактні класи, делегати, події та їх обробники, а також патерни проектування.

6. Система управління базами даних: використовується MySQL Server як СУБД для зберігання та управління даними, а MySQL Workbench CE – як , інструмент для взаємодії з MySQL Server, для розробки та адміністрування бази даних MySQL.

7. Хостинг: програмний продукт буде розміщений на серверах з операційною системою Windows.

8. Використання API IEX: програмний продукт буде інтегруватись з API IEX для отримання фінансової інформації та даних про ринки акцій. Це дозволить отримати доступ до актуальних котирувань, фінансових звітів та інших важливих даних, необхідних для функціональності програмного продукту.

9. Аутентифікація та авторизація: для взаємодії з API IEX буде використовуватись аутентифікація та авторизація для забезпечення безпеки та обмеження доступу до фінансових даних. Запити до API будуть здійснюватись за допомогою ключа API, який надається IEX.

10. Обробка даних з API: програмний продукт повинен забезпечувати зручну обробку та інтеграцію даних, отриманих з API IEX. Це може включати обробку різних типів даних, перетворення форматів, зберігання в базі даних або використання для подальшого аналізу.

11. Механізми обробки помилок: програмний продукт повинен включати механізми обробки помилок та відповідних виключень, пов'язаних з взаємодією з API IEX. Це дозволить коректно обробляти ситуації, пов'язані з відсутністю з'єднання з API, невірними запитами або непередбачуваними відповідями.

12. Моніторинг та керування обмеженнями: програмний продукт повинен включати механізми моніторингу та керування обмеженнями, накладеними API IEX. Це дозволить ефективно використовувати обмежені ресурси API та уникнути перевищення обмежень швидкості або обсягу запитів.

13. Тестування інтеграції з API: програмний продукт повинен пройти відповідне тестування для переконання у правильній роботі взаємодії з API IEX. Це дозволить виявити та виправити помилки або неполадки, пов'язані з інтеграцією з API, забезпечивши стабільну та надійну роботу програмного продукту.

Ці вимоги щодо використання API IEX доповнюють технічну специфікацію та враховують необхідність отримання фінансової інформації для успішної реалізації функцій програмного продукту.

2.6 Висновки до розділу

У даному розділі дипломної роботи були детально розглянуті етапи проектування розроблюваного прототипу програмної системи, враховуючи задачі для розробки та вимоги до програмного продукту. Першочергово, проведений огляд і порівняння різних поширених архітектурних підходів, що вважаються ефективними в розробці різноманітних інформаційних систем. З урахуванням проведеного огляду та порівняльного аналізу, а також враховуючи поставлені задачі та вимоги, було обрано архітектурне рішення для розроблюваного продукту.

Далі були розглянуті різноманітні задачі та перешкоди, що можуть виникати під час розробки програмних систем з аналогічними вимогами. Також був обґрунтований взаємозв'язок між цими проблемами, пов'язаними з вибором мови програмування та використовуваних технологій для розробки окремих компонентів системи. Зважаючи на проведений огляд та порівняльний аналіз різних технологій розробки, були обрані ті, що найкращим чином задовольняють вимоги та задачі, поставлені перед розроблюваним продуктом.

На основі отриманих результатів була сформульована технічна специфікація розроблюваного програмного продукту. Вона визначає вимоги, пов'язані безпосередньо з реалізацією продукту, та зв'язує їх з функціональними та нефункціональними вимогами, а також поставленими задачами, що були визначені та описані в першому розділі.

Крім того, у процесі проектування було виявлено необхідність використання API IEX для отримання фінанссової інформації. API IEX було обрано з метою забезпечення доступу до актуальних даних про фінансові ринки, що є важливим елементом для успішної реалізації функцій програмного продукту. З метою забезпечення правильної роботи взаємодії з API IEX, необхідно провести тестування інтеграції та забезпечити конфіденційність отриманих фінансових даних.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБ ДОДАТКУ

3.1 Поділ на функціональні частини, підсистеми

У процесі розробки веб додатку було проведено детальний аналіз задачі та функціональних вимог до програмного продукту. Для забезпечення ефективності та логічності розробки, функціонал системи було розподілено на кілька підсистем.

Першою підсистемою є «Підсистема управління транзакціями». Вона відповідає за обробку операцій купівлі та продажу акцій компаній. Для цього використовується API IEX для отримання актуальної інформації про доступні для купівлі акції.

Друга підсистема, "Підсистема надання інформації про акції компаній", відповідає за збір та відображення інформації про акції різних компаній, які доступні для інвестицій. Ця підсистема також використовує API IEX для отримання актуальних даних про компанії та їх ціни акцій.

«Підсистема розрахунку та відображення характеристик та статистики портфеля користувача» відповідає за розрахунок та відображення фінансових показників та статистики інвестиційного портфеля користувача. Вона включає розрахунки за отриманими даними з API IEX та використовує алгоритми для розрахунку фінансових показників.

«Підсистема моніторингу прибутковості компонентів інвестиційного портфеля» відповідає за моніторинг та аналіз прибутковості окремих компонентів інвестиційного портфеля. Вона використовує дані з API IEX для оновлення та аналізу цін акцій компаній.

"Підсистема автоматизованих визначених користувачем стратегій" забезпечує автоматизовану реалізацію стратегій, які визначає користувач.

Вона використовує дані з API IEX для виконання необхідних операцій та стратегій, що базуються на актуальних фінансових даних.

У процесі реалізації програмного продукту було створено структуру

проекту та файли програмного коду. Для забезпечення логічної структури та відповідності трьохрівневій архітектурній моделі, були створені три проекти: IMSPrototype.DAL, IMSPrototype.BLL та IMSPrototype.UIL. Кожен проект відповідає відповідному рівню архітектури та взаємодіє з іншими проектами за допомогою встановлених посилань. Загальна схема взаємодії підсистем на структурному рівні (див. рис. 3.1.1).

Проект IMSPrototype.DAL є реалізацією рівня доступу до даних (Data Access Layer) і містить необхідні класи для роботи з базою даних. Він також використовує файли конфігурації EntityFramework та міграцій для налагодження зв'язку з базою даних.

Бібліотека класів IMSPrototype.BLL є реалізацією рівня бізнес-логіки. Вона включає в себе класи, що реалізують функції бізнес-логіки, а також взаємодіють з API IEX для отримання фінансових даних та їх обробки. Класи реалізують алгоритми розрахунку показників, десеріалізацію даних, фільтрацію та сортування даних.

Проект IMSPrototype.UIL містить класи, що відповідають за відображення елементів інтерфейсу користувача, обробку дій користувача та взаємодію з рівнем бізнес-логіки. Ці класи взаємодіють з класами рівня BLL та отримують результати обчислень та перетворень для подальшого відображення користувачу.

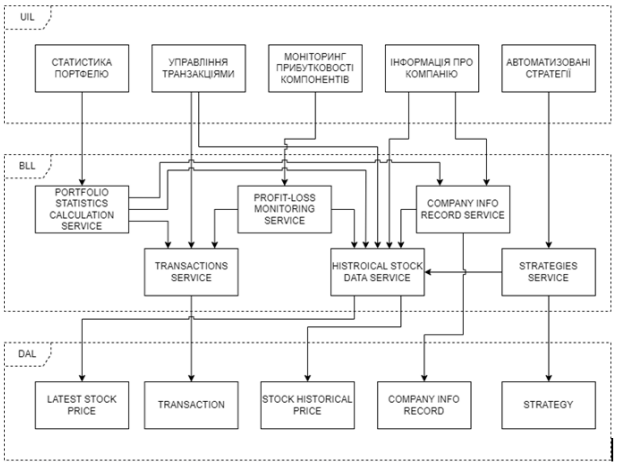

Рис. 3.1.1 - Загальна схема взаємодії підсистем на структурному рівні.

Така структура програмного продукту дозволяє ефективно реалізувати функціонал системи та забезпечити зручний інтерфейс для користувача, (див. рис. 3.1.2).

Застосування API IEX дозволяє отримувати актуальні фінансові дані для виконання розрахунків та стратегій користувача, що покращує функціональність та цінність системи.

Базуючись на вищевикладеному, програмна реалізація прототипу системи успішно включає в себе розбиття функціоналу на логічні підсистеми, створення відповідної структури проекту та файлив програмного коду, а також використання API IEX для отримання актуальних фінансових даних. Такий підхід дозволяє забезпечити функціональність, ефективність та зручний інтерфейс системи, що є важливими аспектами розробки інформаційних систем.

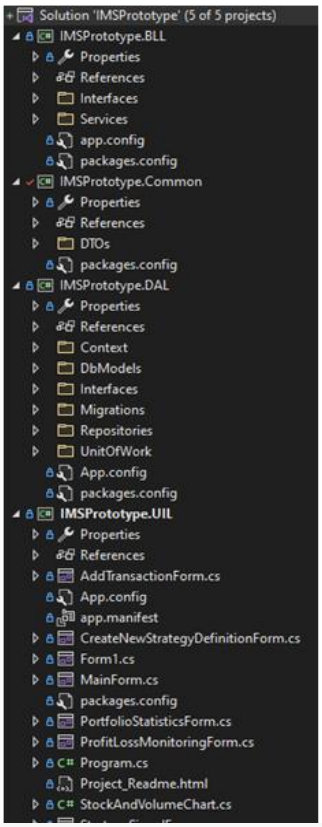

Рис. 3.1.2 - Загальна схема взаємодії підсистем на структурному рівні.

3.2 Створення та налаштування бази даних

Перш за все, мені потрібно було створити та запустити сервер, як інструмент я обрав MySQL Workbench CE, як виглядає головний екран із запущеним і активним локальним сервером, (див. рис. - 3.2.1).

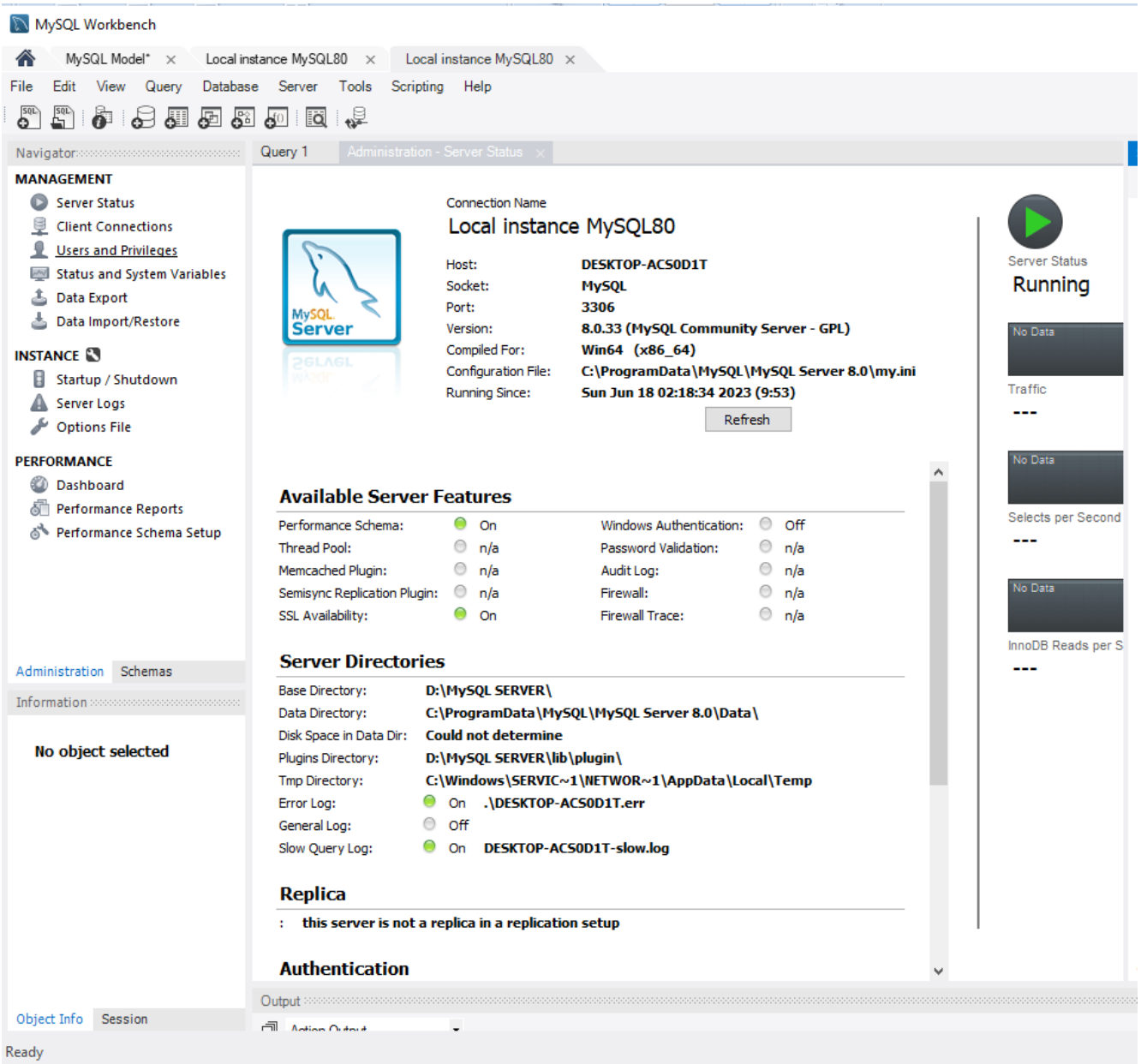

Рис. 3.2.1 - Головний екран із запущеним і активним локальним сервером у MySQL Workbench.

Відповідно до розробки, було визначено, що бібліотека класів IMSPrototype.DAL повинна містити компоненти, які дозволяють отримати доступ до даних. У цьому контексті включаються класи моделей або сутностей програмної системи, клас DbContext, який володіє колекціями сутностей, відповідних таблицям бази даних, та надає можливість виконувати запити до бази даних.

Крім того, є клас Configuration, який використовується в ORM EntityFramework для налаштування та заповнення таблиць бази даних початковими даними, а також для внесення змін до схеми бази даних при модифікації класів моделей сутностей, використовуючи підхід Code First з механізмом міграцій.

Процес розробки прототипу передбачав створення набору сутностей, на основі яких формувалась схема бази даних, (див. рис. 3.2.2). Одним з важливих аспектів розробки було використання API IEX для отримання актуальних фінансових даних, які використовувалися для обробки та аналізу у системі. Це дозволяло забезпечити користувачам доступ до оновлених даних та покращити функціональність системи.

Результати проектування показали, що реалізація IMSPrototype.DAL містить необхідні компоненти для доступу до даних, а також використовує API IEX для отримання фінансових даних. Це робить систему ефективною та забезпечує користувачам актуальну інформацію для прийняття рішень. Структура бази даних була сформована на основі набору сутностей, що забезпечує відповідність даних програмним моделям і полегшує управління базою даних.

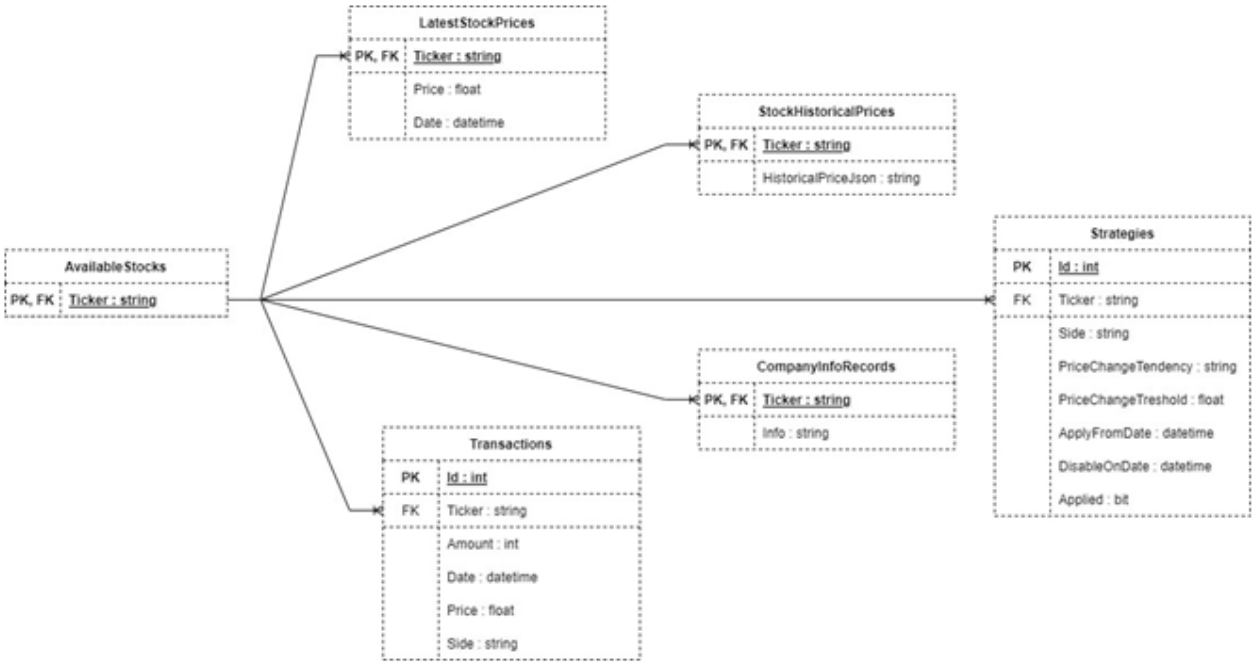

Рис. 3.2.2 - Схема бази даних розроблюваного веб додатку.

Інформація, необхідна для роботи прототипу, зберігається у данних сутностей, які включають дані про транзакції, стратегії, компанії та історичні зміни цін на акції. Для останніх двох сутностей використовується цікавий підхід, де значуща інформація зберігається у вигляді JSON-об'єкта, отриманого через API провайдера фінансової інформації IEX. Таке рішення спрощує створення та зберігання даних, а також зменшує обсяг і час запитів користувача при роботі з історичними даними акцій компаній.

Процес створення моделей та внесення змін призводить до накопичення міграцій, які створюють послідовну модель бази даних. Це "ланцюжок" зі знімків конфігурації та схеми бази даних, який відображає всі зміни, що були внесені з часом.

Також, було розроблено "обгортку" над класами, що працюють з базою даних, яка включає реалізації репозиторіїв та Unit Of Work. Репозиторій дозволяє абстрагуватися від конкретних джерел даних і слугує зв'язком між класами, які взаємодіють з даними та бізнес-логікою рівня BLL. Це зменшує залежність компонентів програмного коду і спрощує можливість заміни компонентів у майбутньому з подальшим розвитком програмного продукту.

Unit Of Work додає логіку роботи з джерелами даних, реалізованими репозиторіями, і дозволяє зручно керувати різними репозиторіями. Це особливо корисно, коли в системі присутні багато сутностей і моделей, а також забезпечує використання одного контексту даних для всіх репозиторіїв.

Крім того, використовується принцип інверсії залежностей, де деталі залежать від абстракцій. Функціонал UnitOfWork та Repository представлені у вигляді інтерфейсів IUnitOfWork та IRepository, що додають ще один рівень абстракції над реалізацією бізнес-логіки. Взаємодія з даними відбувається через екземпляри цих інтерфейсів, діаграма, (див. рис. 3.2.3).

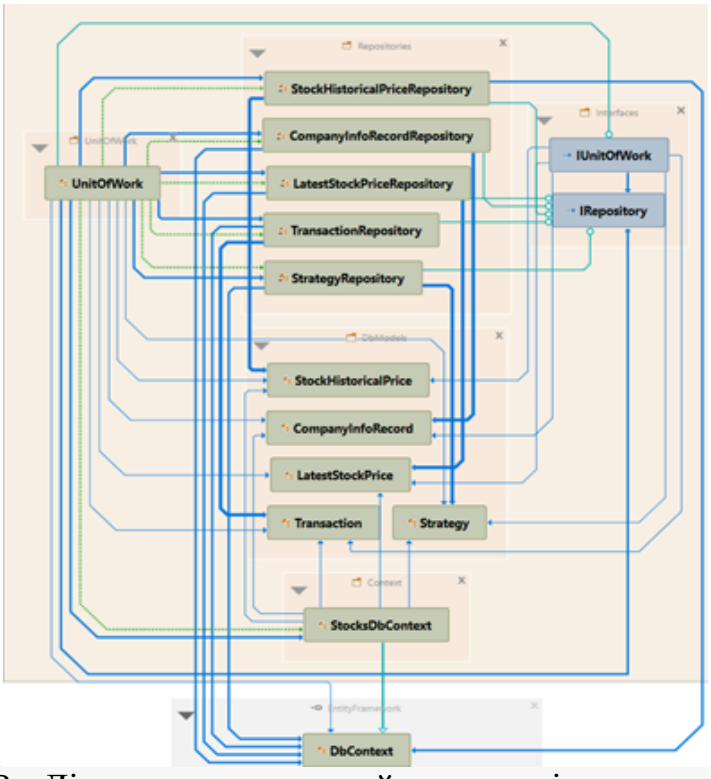

Рис. 3.2.3 - Діаграма залежностей , що реалізують моделі та механізм взаємодії з базою даних веб додатка.

3.3 Розробка функціональності прототипу та інтерфейсу користувачів

В процесі розробки програмної системи для дипломної роботи був використаний підхід Code First, що дозволяє автоматично генерувати схему бази даних на основі програмного коду класів моделей. Цей підхід є зручним для створення та зміни схеми бази даних, оскільки додавання або видалення полів в програмному коді моделей автоматично відображається як зміни в таблицях бази даних. Для збереження та застосування цих змін використовуються міграції, які є набором атомарних процедур, що відслідковують зміни в моделях та оновлюють схему бази даних.

Такий підхід Code First надає широкі можливості та гнучкість при розробці та модифікації схеми бази даних з програмного коду. Він особливо підходить для нових та невеликих програмних систем, які знаходяться на етапі початкової розробки. Використання цього підходу в розробці прототипу є належним і виправданим, оскільки він спрощує процес створення та зміни схеми бази даних зручним і ефективним способом.

Додатково, в проекті використовується API IEX для отримання даних про транзакції, стратегії, компанії та історичні зміни цін на акції. Для зручності, значуща інформація отримується у форматі JSON-об'єкта через API провайдера фінансової інформації IEX. Це спрощує зберігання та використання даних, а також зменшує обсяг і час запитів до провайдера при роботі з історичними даними акцій компаній.

3.4 Реалізація підсистеми управління транзакціями

Розроблена програмна система включає підсистему управління транзакціями (Transactions Manager), яка надає користувачам два інтерфейсні вікна для зручного взаємодії. Перше вікно знаходиться на головній формі прототипу (MainForm.cs) і містить таблицю з користувацькими транзакціями, які можна групувати та сортувати за різними колонками (див. рис. 3.4.1). Друге вікно є окремою формою (AddTransactionForm.cs) і призначене для додавання нових транзакцій. Це вікно активується при натисканні кнопки "Add Transaction" (див. рис. 3.4.2).

Крім того, у розробці веб-додатка було використано API IEX для отримання актуальної фінансової інформації. Завдяки цьому API, програма отримує доступ до даних про транзакції, стратегії, компанії та історичні зміни цін на акції. Ця функціональність розширює можливості додатку, дозволяючи користувачам отримувати актуальні дані і аналізувати їх у зручному форматі. Використання API IEX забезпечує швидкий та зручний доступ до фінансової інформації, що підвищує ефективність та корисність веб-додатку для користувачі.

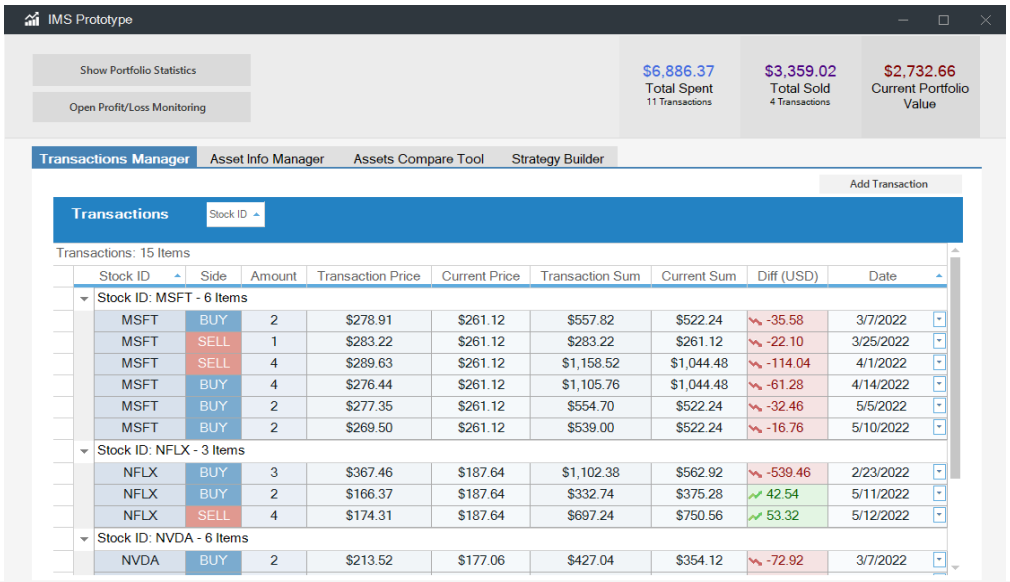

Рис. 3.4.1 - Таблиці з транзакціями, яка забезпечує можливість групування за назвою компанії та сортування дати транзакцій у межах кожної групи.

Файли MainForm.cs та AddTransactionForm.cs, що входять до складу бібліотеки класів IMSPrototype.UIL, містять логіку для відображення елементів на формі. Вони включають ініціалізацію елементів, встановлення їх властивостей щодо кольорів, розмірів, положення, вирівнювання та поведінки залежно від дій користувача.

Крім того, у розробці веб-додатка було використано API IEX, яке дозволяє отримувати актуальну фінансову інформацію. Це API надає доступ до даних про транзакції, стратегії, компанії та історичні зміни цін на акції. Використання API IEX дозволяє розширити функціональність веб-додатка, надаючи користувачам можливість отримувати актуальні дані та аналізувати їх у зручному форматі. Завдяки цьому API, веб-додаток стає більш ефективним та корисним для користувачів.

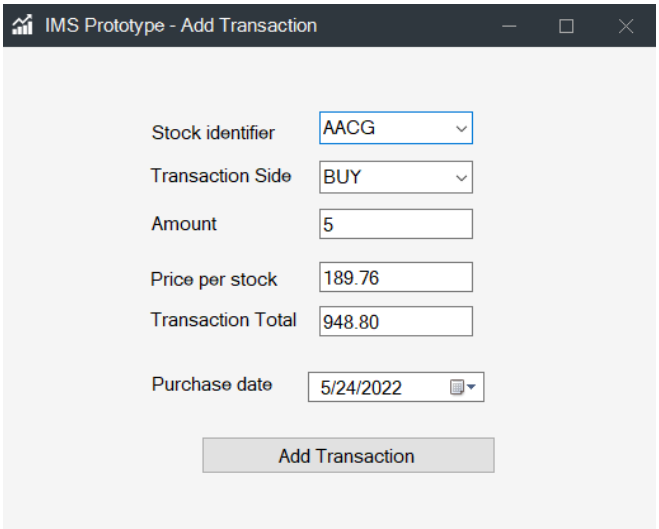

Рис. 3.4.2 - Форма додавання нової транзакції.

У веб-додатку, як і в усіх інших формах, використовується механізм подій та їх обробників для функціонування форм. Кожна зміна стану елемента форми відповідає певній конструкції в програмному коді. Для цього використовується екземпляр .NET класу Event, який є властивістю об'єкта форми, і може мати прикріплений обробник події. Обробник події представляє собою метод у мові C#, який містить логіку і виконується при виникненні події, до якої цей обробник прикріплений.

Класи форм, що належать до підсистеми Transactions Manager, містять приватні поля, які зберігають посилання на об'єкти відповідного класу рівня бізнес-логіки, TransactionsService. Шляхом виклику методів цього класу, форма отримує дані та результати обчислень для відображення на користувацькому інтерфейсі. Наприклад, поля Transaction Sum та Diff (USD) є обчислюваними полями, значення яких отримуються через виклик методів TransactionsService. Крім того, цей клас містить метод для додавання нової транзакції до бази даних, який виконується після заповнення користувачем полів форми додавання транзакції та натискання кнопки "Add Transaction".

У розробці веб-додатка також використовується API IEX, яке забезпечує отримання актуальної фінансової інформації. Це API надає доступ до даних про транзакції, стратегії, компанії та історичні зміни цін на акції. Використання API IEX дозволяє розширити функціональність веб-додатка, дозволяючи користувачам отримувати актуальні дані та аналізувати їх у зручному форматі.

3.5 Реалізація підсистеми надання інформації про акції компаній

Підсистема, що забезпечує надання інформації про акції компаній, складається з двох компонентів користувацького інтерфейсу (UI) - сторінок, розташованих на головній формі під вкладками Asset Info Manager (див. рис. 3.5.1) та Assets Compare Tool (див. рис. 3.5.2). Ці сторінки дозволяють отримати доступ до важливих даних про акції та забезпечують зручний інструмент для порівняння активів.

У розробці веб-додатка використовується API IEX, що дозволяє отримувати актуальну інформацію про фінансові ринки та акції компаній. Це API надає доступ до широкого спектру даних, включаючи статистику акцій, фінансові показники, новини та історичні дані. Використання API IEX у розробці веб-додатка дозволяє забезпечити користувачів актуальною інформацією про акції компаній, спрощує аналіз ринку та допомагає приймати обґрунтовані рішення щодо інвестування.

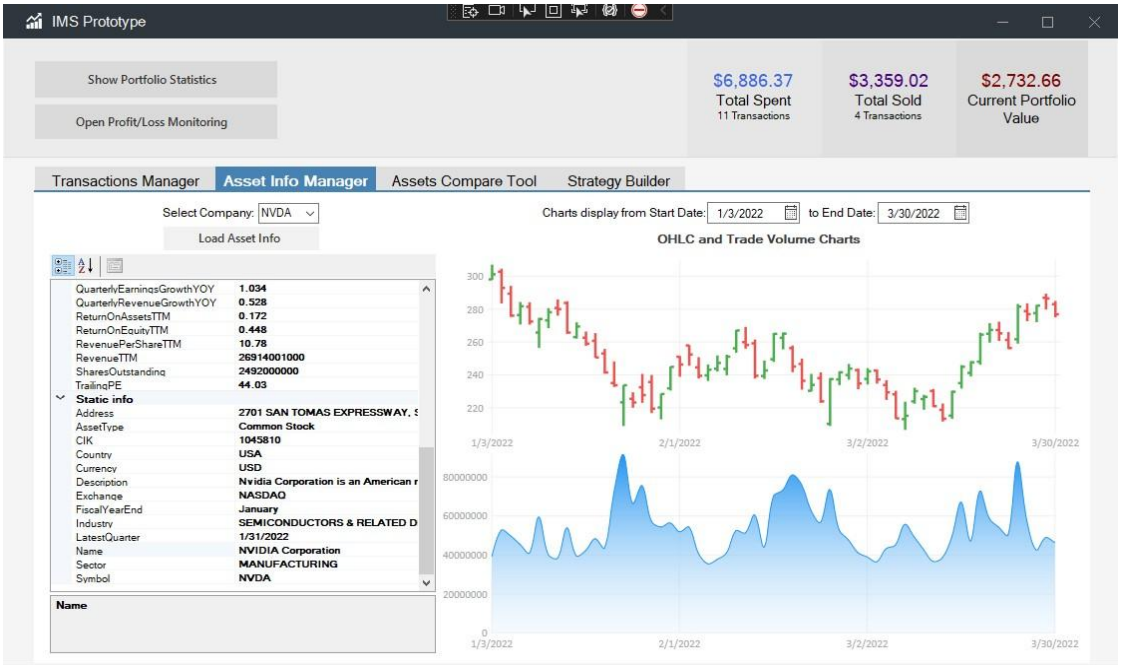

Рис. 3.5.1 - Інтерфейс користувача Asset Info Manager.

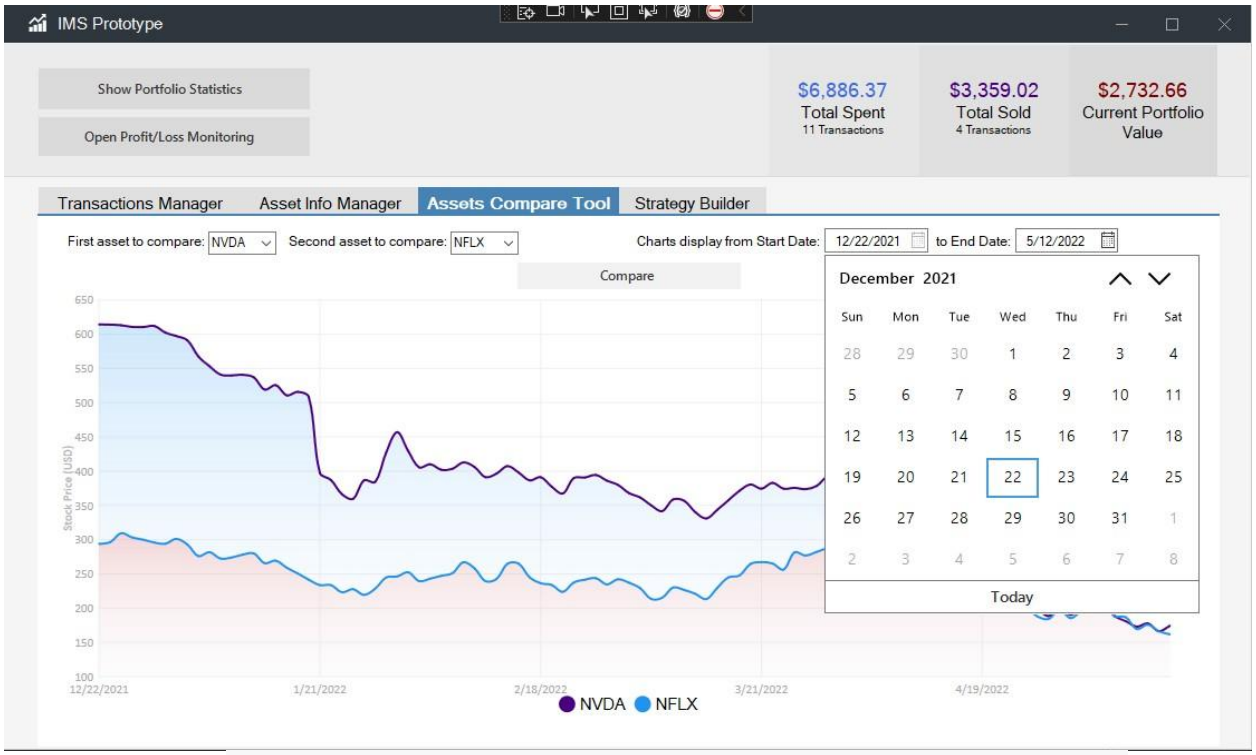

Рис. 3.5.2 - Інтерфейс користувача Assets Compare Tool.

Функціонал менеджера інформації про активи надає користувачу можливість отримувати та переглядати різноманітну інформацію про обрані компанії з певного переліку. Цей перелік включає такі характеристики та фінансові показники, як сектор економіки, до якого вони належать, обчислені значення простого ковзного середнього ціни акцій за 50 та 200 днів, розмір дивідендів, капіталізацію, коефіцієнт "ціна-прибуток" та інші коефіцієнти та показники, необхідні для технічного аналізу компаній-емітентів.

При розробці веб-додатку використовується API IEX для отримання фінансової інформації про компанії. Програмна реалізація логіки цієї підсистеми розміщена в класах CompanyInfoRecordsService та HistoricalStockDataService. Використовуючи асинхронні запити до API, ці класи в окремих логічних потоках отримують дані за назвою компанії та вказаним часовим інтервалом. Отримані результати десеріалізуються та формуються в колекції екземплярів C#-класів, які потім повертаються для відображення на користувацькому інтерфейсі.

Крім того, менеджер інформації про активи надає графічне представлення зміни вартості акцій обраної компанії протягом заданого періоду. Графік включає в себе значення зміни ціни день у день, а також значення відкриття, максимуму, мінімуму та закриття акцій за кожен торговий день. Крім цього, графік відображає зміну обсягу торгів протягом того самого періоду. Це зручне та наглядне представлення зміни цих двох показників дозволяє користувачеві проводити фінансовий аналіз компаній з більшою ефективністю.

Інструмент порівняння активів дозволяє користувачу візуально порівнювати графіки зміни цін різних компаній. Це особливо важливо для виявлення кореляцій між тенденціями цін. Вивчення таких кореляцій має велике значення для фінансового та інвестиційного аналізу загалом, адже воно допомагає в прийнятті рішень щодо вибору компаній для інвестування та управління інвестиційним портфелем з часом.

3.6 Реалізація підсистеми відображення статистики портфеля

У рамках розробки веб-додатку була реалізована підсистема, яка відповідає за відображення статистики портфеля користувача. Ця підсистема містить сторінку, де представлена інфографіка, що відображає обчислені значення різних показників, які характеризують інвестиційний портфель цінних паперів. Також на цій сторінці показані графіки, які демонструють зміни деяких з цих показників

протягом часу. За допомогою цих графіків користувач може отримати візуальне представлення про зміни в своєму портфелі (див. форму "Portfolio Statistics" на рисунку 3.6.1).

Проте, на цій сторінці відображаються три обчислені значення: загальна кількість інвестованого капіталу в акціях, загальна сума всіх продажів цінних паперів та поточне значення собівартості акцій, що знаходяться в портфелі (тобто не були продані). Ці значення відображаються на стрічці, що розташована вгорі головної форми і має підписи "Total Spent", "Total Sold" і "Current Portfolio Value".

Окремо варто зазначити, що в рамках "Portfolio Statistics" використовується графічний елемент GeoHeatMap, який відображає розподілення компаній, акції яких присутні у портфелі користувача, за країнами, де ці компанії здійснюють свою діяльність (під назвою "Portfolio Holdings by Countries"). Крім того, два графіки типу PieChart ілюструють розподіл загального інвестованого капіталу між всіма компаніями, що присутні у портфелі (під назвою "Portfolio Assets Allocation"), а також розподіл капіталу між секторами економіки (під назвою "Portfolio Stocks' Sectors").

У розробці цієї підсистеми було використано API IEX, яке забезпечує зручний доступ до фінансової інформації про компанії та надає потрібні інструменти для аналізу та порівняння активів. Використання цього API значно спрощує роботу з фінансовими даними та дозволяє побудувати функціональний та інформативний веб-додаток для керування інвестиційним портфелем.

Інтеграція API IEX в веб-додаток здійснюється за допомогою використання токена. Токен - це унікальний ідентифікатор, який надається розробнику при реєстрації на платформі IEX Cloud. Цей токен використовується для аутентифікації та авторизації запитів до API IEX.

Після отримання токена, його можна використовувати для створення запитів до API IEX. Для цього веб-додаток використовує HTTP-запити, наприклад, GET-запити, для отримання фінансової інформації про акції, компанії, індекси тощо. Запити можна виконувати за допомогою різних мов програмування, таких як JavaScript, Python, Ruby тощо.

У запитах до API IEX веб-додаток включає токен аутентифікації, який передається в заголовку або параметрі запиту. Наприклад, токен може мати наступний вигляд, (див. рис. 3.6.2)

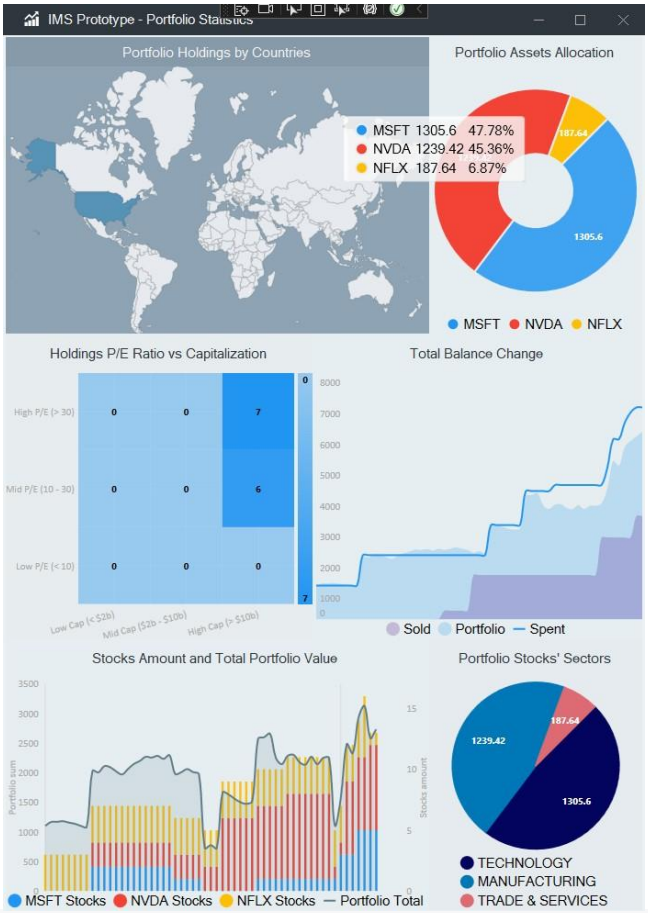

Рис. 3.6.1 - Інтерфейс веб додатку відображення статистики портфеля.

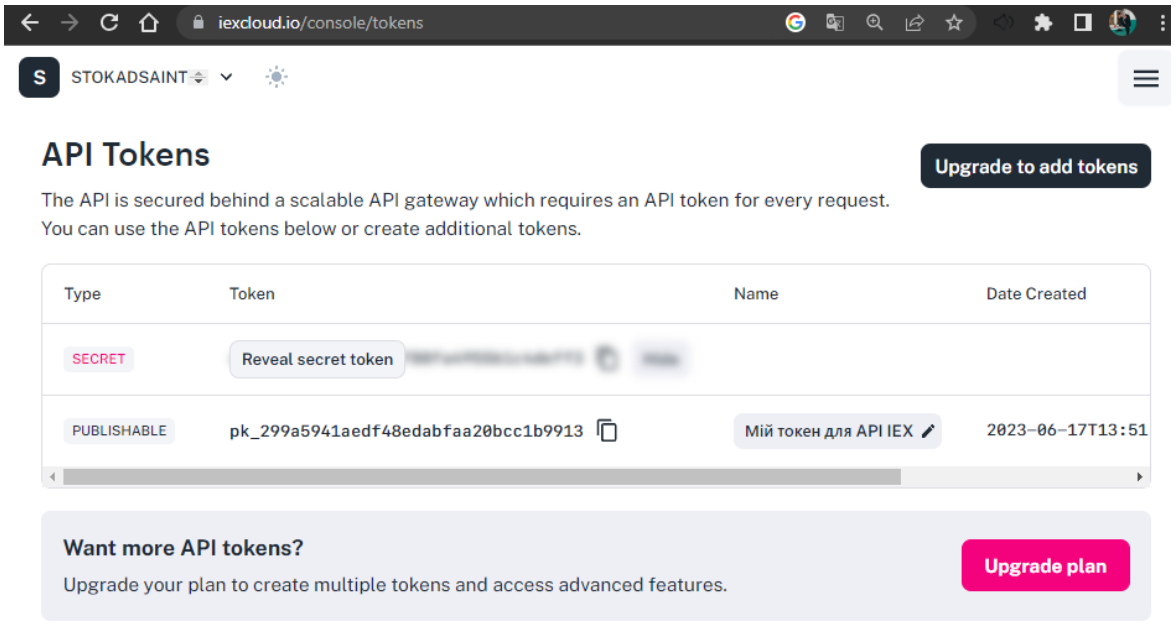

Рис. 3.6.2 - Токен API IEX та інтерфес сайту.

Токен API використовується для ідентифікації користувача або додатку, а також для встановлення рівня доступу та прав доступу до ресурсів, що надаються через API. Використання токенів дозволяє забезпечити безпеку та контроль доступу до API, а також дозволяє стежити за використанням ресурсів та обмежувати обсяги викликів.

Зазвичай токени API обмежуються в часі і можуть мати обмеження на обсяг запитів або доступ до конкретних функцій API. Це дозволяє власникам API контролювати використання своїх послуг та забезпечувати безпеку та ефективність взаємодії з додатками користувачів.

3.7 Реалізація підсистеми моніторингу прибутковості

Розроблення підсистеми моніторингу прибутковості (Profit/Loss Monitoring) дозволяє розраховувати та відображати зміни показника прибутковості від моменту першої покупки акцій вибраної користувачем компанії з його портфеля до поточного моменту часу. Цей розрахунок здійснюється з використанням API IEX.

Алгоритм підсистеми включає обчислення середнього значення інвестованого капіталу в акції обраної компанії на кожний день з врахуванням ціни акцій на момент їх покупки. Для цього здійснюється вибірка транзакцій по компанії та обчислення фактичної кількості акцій в портфелі на кожній ітерації, а також середньої ціни кожної акції. Отримані значення використовуються для побудови графічної залежності, яка відображається на графіку разом зі зміною ціни акції на фондовій біржі. Далі обчислюється показник прибутковості як різниця між поточною ціною акції на біржі та середньозваженим значенням інвестованого капіталу. Цей показник також відображається на графіку.

Веб-додаток використовує API IEX для отримання актуальних даних про ціни акцій та здійснення необхідних розрахунків. Інтеграція API IEX дозволяє забезпечити точність та актуальність фінансової інформації, що використовується в підсистемі моніторингу прибутковості, (див. рис. 3.7.1).

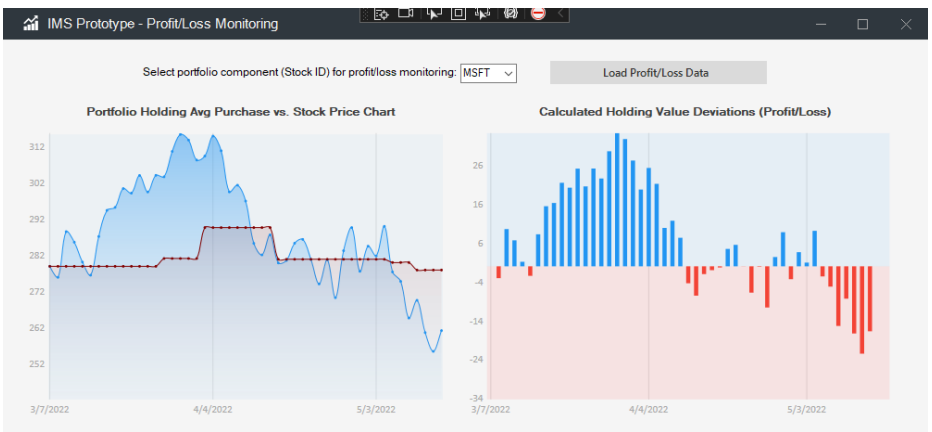

Рис. 3.7.1 - Profit/Loss Monitoring.

3. 8 Висновки до розділу

У даному розділі було проведено аналіз завдання для розробки веб-додатку з урахуванням функціональних вимог до продукту. Було розглянуто поділ функціоналу прототипу інформаційної системи на логічні складові та визначено перелік необхідних елементів. Також була створена структура файлів розроблюваного рішення, що забезпечує досягнення поставлених цілей та відповідну реалізацію обраної архітектури програмної системи. Це сприяє більш гнучкому та організованому процесу розробки. Крім того, було описано перелік класів, на яких базується логіка додатку, їх взаємозв'язок та структуру, а також реалізацію механізмів роботи з базою даних прототипу та його схему. Окрім цього, було надано детальний опис функціоналу, який було розроблено для прототипу. Результати роботи готового прототипу інформаційної системи були представлені на рисунках.

У процесі розробки веб-додатку було використано API IEX для отримання фінансових та інвестиційних показників. Цей API дозволив отримувати актуальну інформацію про ринки, акції та інші фінансові дані, що було важливим для реалізації функціональності додатку. Завдяки використанню API IEX, користувачі зможуть отримати доступ до актуальних даних та здійснювати аналіз фінансових показників для кращого управління своїм персональним портфелем цінних паперів.

ВИСНОВКИ

У ході дослідження для дипломного проекту було проведено аналіз засобів аналізу фінансових та інвестиційних показників і їх використання в існуючих системах управління персональним портфелем цінних паперів. Був зроблений порівняльний аналіз існуючих систем та виявлені їх недоліки.

Також були розглянуті шаблони та підходи до проектування архітектури програмних систем, технології для налаштування та взаємодії програмного продукту з базою даних, а також механізми модифікацій схеми бази даних при зміні вимог до функціональності та сутностей системи. Було проведено дослідження мов програмування та технологій створення інтерфейсу користувача для розробки інформаційних систем.

На основі проведеного аналізу були вибрані найбільш підходящі засоби, підходи та технології для реалізації дипломного проекту з урахуванням функціональних та нефункціональних вимог. Було визначено, що для імплементації функціональної частини розроблюваного прототипу буде використана технологія платформи .NET, для роботи з базою даних – Entity Framework, а для створення інтерфейсу користувача – WinForms.

Результатом дипломного проекту є розроблений прототип інформаційної системи управління та обліку персонального портфелю цінних паперів. Прототип дозволяє моделювати процеси інвестування та спостерігати ефект від застосування різних підходів до інвестиційної діяльності. Він має гнучку інфраструктуру, яка сприяє подальшому розвитку, модифікаціям та розширенню.

У процесі розробки веб-додатку було використано API IEX для отримання фінансових та інвестиційних показників. Це API забезпечило отримання актуальної інформації про ринки та акції, що було важливим для реалізації функціональності додатку. За допомогою API IEX користувачі мають можливість отримувати доступ до актуальних даних та аналізувати фінансові показники для кращого управління своїм персональним портфелем цінних паперів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. "Investment Analysis and Portfolio Management" - Frank K. Reilly, Keith C. Brown
2. "Modern Portfolio Theory and Investment Analysis" - Edwin J. Elton, Martin J. Gruber, Stephen J. Brown, William N. Goetzmann
3. "Financial Management and Analysis: Principles and Techniques" - Gurcharan Singh
4. "Financial Modeling" - Simon Benninga
5. "Financial Statement Analysis and Security Valuation" - Stephen H. Penman
6. Markowitz, H. (1952). "Portfolio Selection." The Journal of Finance, 7(1), 77-91.
7. Fama, E. F., & French, K. R. (1993). "Common risk factors in the returns on stocks and bonds." Journal of Financial Economics, 33(1), 3-56.
8. Black, F., & Scholes, M. (1973). "The Pricing of Options and Corporate Liabilities." The Journal of Political Economy, 81(3), 637-654.
9. Sharpe, W. F. (1964). "Capital asset prices: A theory of market equilibrium under conditions of risk." The Journal of Finance, 19(3), 425-442.
10. IEX Cloud API Documentation: https://iexcloud.io/docs/api/
11. Microsoft .NET Documentation: https://docs.microsoft.com/en-us/dotnet/
12. Entity Framework Documentation: https://docs.microsoft.com/en-us/ef/
13. Дослідження та звіти фінансових організацій таких як:

- World Economic Outlook - International Monetary Fund (IMF)
- Global Financial Stability Report - International Monetary Fund (IMF)
- Market Insights and Research Reports - Investment Banks (e.g., Goldman Sachs, J.P. Morgan, Morgan Stanley)

ДОДАТКИ

Додаток А

Програмна реалізація класу StrategiesService

```
namespace IMSPrototype.BLL.Services
{
public class StrategiesService : IStrategiesService
{
public IUnitOfWork UnitOfWork { get; set; } = new UnitOfWork();
public List<Strategy> UserDefinedStrategies { get; set; } = new List<Strategy>();

public StrategiesService()
{
LoadUserDefinedStrategies();
}

public void AddNewStrategy(string stockID, string side, string priceChangeTendency, decimal priceChangeThreshold, DateTime applyFromDate, DateTime? disableOnDate = null, bool applied = false)
{
var strategy = new Strategy { StockID = stockID, Side = side, PriceChangeTendency = priceChangeTendency, PriceChangeThreshold = priceChangeThreshold, ApplyFromDate = applyFromDate, DisableOnDate = disableOnDate, Applied = applied };

UnitOfWork.Strategies.Create(strategy);
}

public void ApplyStrategy(Strategy strategy)
{
strategy.Applied = true; UnitOfWork.Strategies.Update(strategy, strategy.Id.ToString());
}

public void DisableStrategy(Strategy strategy)
{
strategy.Applied = false; UnitOfWork.Strategies.Update(strategy,
```

```
strategy.Id.ToString());
        }

        public string[] GetPriceChangeTendencyNames()
        {
        return Enum.GetNames(typeof(PriceChangeTendency));
        }

        public void LoadUserDefinedStrategies()
        {
        UserDefinedStrategies = UnitOfWork.Strategies.Get();
        }

        public void ExecuteStrategy(Strategy strategy, decimal strategyAppliedDatePrice, decimal currentStockPrice)
        {
        if (strategy.PriceChangeTendency == PriceChangeTendency.Increased.ToString())
        {
        if (currentStockPrice >= strategyAppliedDatePrice * (1 + strategy.PriceChangeThreshold / 100))
        {
        StrategyThresholdReachedEventArgs args = new StrategyThresholdReachedEventArgs(); args.Strategy = strategy;
        args.DateTimeReached = DateTime.Now;OnThresholdReached(args);
        }
```

Продовження додатку А

```
}
else if (strategy.PriceChangeTendency == PriceChangeTendency.Dropped.ToString())
{
if (currentStockPrice <= strategyAppliedDatePrice * (1 - strategy.PriceChangeThreshold / 100))
{
StrategyThresholdReachedEventArgs args = new StrategyThresholdReachedEventArgs(); args.Strategy = strategy;
args.DateTimeReached = DateTime.Now; OnThresholdReached(args);
}
}
}

public FormattedStrategyDefinitionInfo GetFormattedStrategyDefinitionInfo(Strategy strategy)
{
var definitionParameters = $"Stock ID: {strategy.StockID} Side: {strategy.Side} Condition: When price
{strategy.PriceChangeTendency} by {strategy.PriceChangeThreshold}%";

    var strategyDisableOnDate = strategy.DisableOnDate != null ? ((DateTime)strategy.DisableOnDate).ToString("d")
 "-
-"; var applyDisableDates = $"Apply from date: {strategy.ApplyFromDate:d} Disable on date:
{strategyDisableOnDate}";

return new FormattedStrategyDefinitionInfo(definitionParameters, applyDisableDates);
}

public string GetStrategyRecordHeaderName(Strategy strategy)
{
var strategyIndex = UserDefinedStrategies.FindIndex(x => x.Id ==
```

```
strategy.Id);

      return $"Strategy {strategyIndex + 1}";
      }

      protected                              virtual                              void
OnThresholdReached(StrategyThresholdReachedEventArgs e)
      {
      EventHandler<StrategyThresholdReachedEventArgs>        handler        =
StrategyThresholdReached; if (handler != null)
      {
      handler(this, e);
      }
      }

      public      event      EventHandler<StrategyThresholdReachedEventArgs>
StrategyThresholdReached;
      }

      public class StrategyThresholdReachedEventArgs : EventArgs
      {
      public Strategy Strategy { get; set; }
      public DateTime DateTimeReached { get; set; }
      }
      }
```

Додаток Б

Програмна реалізація класу StrategiesService

```
namespace IMSPrototype.UIL
{
public partial class CreateNewStrategyDefinitionForm : SfForm
{
private ITransactionsService transactionsService = new TransactionsService(); private IStrategiesService strategiesService = new StrategiesService();

public CreateNewStrategyDefinitionForm()
{
InitializeComponent();

this.Style.TitleBar.Height = 35;
this.Style.TitleBar.BackColor = System.Drawing.Color.FromArgb(47, 55, 62); //2f373e this.BackColor = System.Drawing.Color.WhiteSmoke;
this.Style.TitleBar.ForeColor = System.Drawing.Color.WhiteSmoke; this.Style.TitleBar.CloseButtonForeColor = System.Drawing.Color.DarkGray; this.Style.TitleBar.MaximizeButtonForeColor = System.Drawing.Color.DarkGray; this.Style.TitleBar.MinimizeButtonForeColor = System.Drawing.Color.DarkGray; this.Style.TitleBar.Font = this.Font = new System.Drawing.Font("Microsoft Sans Serif", 11F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)(0))); this.Style.TitleBar.TextHorizontalAlignment = HorizontalAlignment.Left;
}

private void CreateNewStrategyDefinitionForm_Load(object sender, EventArgs e)
{
var stocksSource = transactionsService.GetStocks();
var stocksAutoCompleteStringCollection = new AutoCompleteStringCollection();
stocksAutoCompleteStringCollection.AddRange(stocksSource);
comboBox1.DataSource = stocksSource;
```

```
comboBox1.AutoCompleteMode = AutoCompleteMode.SuggestAppend;
comboBox1.AutoCompleteSource = AutoCompleteSource.CustomSource;
comboBox1.AutoCompleteCustomSource = stocksAutoCompleteStringCollection;

var transactionSidesSource =
transactionsService.GetTransactionSides();
var transactionSidesAutoCompleteStringCollection = new
AutoCompleteStringCollection();
transactionSidesAutoCompleteStringCollection.AddRange(transactionSidesSourc
e); comboBox2.DataSource = transactionSidesSource;
comboBox2.AutoCompleteMode = AutoCompleteMode.SuggestAppend;
comboBox2.AutoCompleteSource = AutoCompleteSource.CustomSource;
comboBox2.AutoCompleteCustomSource =
transactionSidesAutoCompleteStringCollection;
}

private void CreateNewStrategyDefinitionButton_Click(object sender,
EventArgs e)
{
var stockID = comboBox1.SelectedValue.ToString(); var side =
comboBox2.SelectedValue.ToString();
var priceChangeTendency = comboBox3.SelectedValue.ToString(); var
priceChangeThreshold = Convert.ToDecimal(textBox1.Text); var applyFromDate
= sfDateTimeEdit1.Value;
var disableOnDate = sfDateTimeEdit2.Value;

strategiesService.AddNewStrategy(stockID, side, priceChangeTendency,
priceChangeThreshold, (DateTime)applyFromDate, disableOnDate);

this.Close();
}
}
}
```

Додаток В

Програмна інтеграція API IEX

```
using IEXCloud;
using IEXCloud.Client;

namespace IMSPrototype.Services
{
    public class StrategiesService : IStrategiesService
    {
        private readonly IEXCloudClient iexCloudClient;

        public StrategiesService(string 
pk_299a5941aedf48edabfaa20bcc1b9913)
        {
            iexCloudClient = new IEXCloudClient(apiKey);
        }

        public void AddNewStrategy(string stockID, string side, 
string priceChangeTendency, decimal priceChangeThreshold, DateTime 
applyFromDate, DateTime? disableOnDate)
        {

            var stockQuote = 
iexCloudClient.StockPrices.GetQuote(stockID);
            var stockPrice = stockQuote.LatestPrice;

        }
    }
}
```