МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
Факультет кібербезпеки, комп'ютерної та програмної інженерії
Кафедра комп'ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ

Завідувач кафедри

Аліна САВЧЕНКО

"__"_____2022 р.

ДИПЛОМНИЙ ПРОЕКТ
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ

"БАКАЛАВРА"

ЗА СПЕЦІАЛЬНІСТЮ 122 «КОМП'ЮТЕРНІ НАУКИ»

Тема: "Прототип інформаційної системи управління та обліку персонального портфелю цінних паперів"

Виконавиця: Саттарова Маргарита Леонідівна

Керівник: зав. каф., д.т.н., доцент Савченко Аліна Станіславівна

Нормоконтролер: Олександр ШЕВЧЕНКО
(підпис)

Київ 2022

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії

Кафедра комп'ютерних інформаційних технологій

Освітній ступінь: "Бакалавр"

Галузь знань, спеціальність, освітньо-професійна програма:

12 "Інформаційні технології, 122 "Комп'ютерні науки", "Інформаційні управляючі системи та технології" .

ЗАТВЕРДЖУЮ

Завідувач кафедри

Аліна САВЧЕНКО

"____" _____ 2022 р.

ЗАВДАННЯ

на виконання дипломного проекту студентки

Саттарової Маргарити Леонідівни

(прізвище, ім'я, по батькові)

1. **Тема роботи:** «Прототип інформаційної системи управління та обліку персонального портфелю цінних паперів» затверджена наказом ректора №454/ст. від 29.04.2022.
2. **Термін виконання роботи:** 09.05.2022 – 13.06.2022.
3. **Вихідні дані до роботи:** дані про фінансові показники, документація програмних засобів, методи, засоби та підходи інвестиційного аналізу.
4. **Зміст пояснювальної записки (перелік питань, що підлягають розробці):** аналіз існуючих систем управління персональним портфелем цінних паперів, огляд архітектур програмних систем, порівняльний аналіз програмних засобів імплементації, проектування та розробка прототипу інформаційної системи.
5. **Перелік обов'язкового графічного матеріалу:** слайди презентації MS PowerPoint

6. Календарний план-графік

№ з/п	*Завдання*	*Термін виконання*	*Підпис керівника*
1.	Аналіз та дослідження предметної області	09.05.2022р. – 11.05.2022р.	
2	Опрацювання літературних джерел, що відносяться до теми дипломного проекту	12.05.2022р. – 16.05.2022р.	
3	Аналіз існуючих програмних продуктів для управління та обліку персонального портфелю цінних паперів та технічних засобів реалізації поставленої задачі	17.05.2022р. – 18.05.2022р.	
4	Розробка структури дипломного проекту відповідно до обраної архітектури програмних систем	19.05.2022р. – 20.05.2022р.	
5	Розробка функціональної частини проекту та інтерфейсу користувача	21.05.2022р. – 26.05.2022р.	
6	Оформлення пояснювальної записки.	27.05.2022р. – 01.06.2022р.	
7	Підготовка необхідної документації. Підготовка презентації та доповіді.	02.06.2022р. – 13.06.2022р.	

Дата видачі завдання: 09.05.2022 р.

Керівник дипломного проекту _____ Аліна САВЧЕНКО
(підпис керівника)

Завдання прийняла до виконання _____ Маргарита САТТАРОВА
(підпис випускника)

РЕФЕРАТ

Пояснювальна записка до дипломного проекту «Прототип інформаційної системи управління та обліку персонального портфелю цінних паперів» представлена на 66 сторінках, містить 22 рисунка, 15 наукових джерел.

Мета дипломного проекту: Розробити прототип інформаційної системи управління та обліку персонального портфелю цінних паперів для отримання вичерпної оцінки ефективності інвестицій.

Об'єкт дослідження: Процес управління та обліку персонального портфелю цінних паперів

Предмет дослідження: Автоматизація управління та обліку персонального портфелю цінних паперів на базі платформи .NET та Entity Framework

Метод дослідження: Аналітичний огляд існуючих систем управління персональним портфелем цінних паперів, дослідження засобів аналізу фінансових показників, порівняльний аналіз програмних засобів для реалізації інструментів для розрахунку та аналітики інвестиційних та фінансових показників

Результат проекту: Розроблений програмний продукт – прототип інформаційної системи управління та обліку персонального портфелю цінних паперів для моделювання процесів інвестування з метою закріплення та розширення інвесторами та трейдерами знань в фінансовій грамотності та спостереження ефекту від використання різних підходів до процесу інвестування на практиці. Прототип має сприятливу інфраструктуру для його подальшого розвитку, модифікацій (за потребою) та розширення, характеризуючись високими показниками розширюваності, переносимості, гнучкості та масштабованості з точки зору як архітектури, так і безпосередньої технічної реалізації.

ІНВЕСТИЦІЙНИЙ ПОРТФЕЛЬ, АНАЛІЗ ФІНАНСОВИХ ПОКАЗНИКІВ, СТРАТЕГІЇ ІНВЕСТУВАННЯ, АРХІТЕКТУРА, .NET, ENTITY FRAMEWORK.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

S&P 500 – фондовий індекс Standard & Poor's

API – прикладний програмний інтерфейс

ORM – технологія програмування, яка пов'язує бази даних із концепціями об'єктно-орієнтованих мов програмування

СУБД – комплекс програмних засобів, що дозволяють створювати бази даних та керувати даними

XAML – декларативна мова розмітки

CLR – середовище виконання байт-коду, в який компілюються програми, написані на .NET-сумісних мовах програмування

CIL – проміжна мова (байт-код) для платформи .NET

ВСТУП

Інвестиції є однією з найважливіших економічних категорій, що визначають шляхи розвитку економіки. Роль інвестицій висока, оскільки завдяки ним здійснюється накопичення громадського капіталу, розширюється база для нових виробничих можливостей. Для досягнення соціально значимих цілей необхідне активне інвестування. Інвестиції визначають процес розширеного відтворення. Будівництво нових підприємств, зведення житлових будинків, прокладання доріг, а, отже, і створення нових робочих місць суттєвим чином залежать від процесу інвестування.

Завдяки інвестиціям компанії знаходять гроші на реалізацію нових бізнес-проектів, створюють нові робочі місця, розробляють нові технології. Гроші, які інвестори заробили на біржі, вони витрачають на споживання товарів, робіт та послуг, тобто підвищують попит, а це в свою чергу стимулює виробництво. Стабільний фондовий ринок залучає іноземних інвесторів у країну, а іноземні інвестиції – важливий двигун економіки та фактор зростання ВВП.

В свою чергу, для інвестора інвестиції – це гарна можливість забезпечити пасивний дохід та фінансову безпеку у довгостроковій перспективі. Вони можуть допомогти накопичити на освіту дітям, створити фінансову подушку безпеки для непередбачених ситуацій, або швидше здійснити масштабні плани та мрії такі як, наприклад, купівля нерухомості. Фінансові кризи трапляються регулярно, але варто пам'ятати, що в довгостроковій перспективі фондовий ринок зростає і ті, хто купив по кілька акцій усіх компаній Нью-Йоркської фондової біржі на початку минулого століття і не продавали їх досі, в багато разів примножили свій капітал, і весь цей час ці гроші не просто лежали, а працювали, допомагали економіці розвиватись. Таким чином, всі, хто вкладають гроші у фондовий ринок, окрім можливого примноження свого власного капіталу, також роблять внесок у прогрес та допомагають робити життя суспільства загалом благополучнішим, різноманітнішим і комфортнішим.

Мета дипломного проекту – розробка прототипу інформаційної системи управління та обліку персонального портфелю цінних паперів для отримання вичерпної оцінки ефективності інвестицій.

Зростаюча популярність сфери інвестицій та трейдингу разом з поширеною проблемою недостатніх знань фінансової грамотності та практичних навичок виконання, моніторингу та управління цими процесами призвела до розробки достатньо великої кількості програмних систем для аналітики фінансових показників, управління та обліку персональним портфелем цифрових активів. Відповідно, було поставлено завдання в процесі дипломного проектування розробити прототип програмної системи, яка поєднує вдалі рішення вже наявних аналогів та в той же час є зручнішою у користуванні та такою, що надає більш вичерпну інформацію для успішного управління та обліку персонального інвестиційного портфелю.

Успішне виконання даного завдання потребує здійснення наступних кроків:

а) огляд і аналіз існуючих рішень в області управління інвестиціями, що дозволяють зберігати і корегувати дані, надавати користувачу необхідну статистичну інформацію;

б) дослідження підходів та патернів архітектур, а також програмних засобів створення інформаційних систем;

в) проектування і розробка власного програмного засобу, що реалізує функціонал аналітики та автоматизації процесів управління персональним портфелем цінних паперів.

Практичне значення отриманих результатів. Реалізований в процесі дипломного проектування прототип дозволяє інвесторам та трейдерам полегшити процес обліку персонального портфелю акцій, інтерфейс та інфографіка якого є простими, достатньо інформативними та інтуїтивно зрозумілими для користувача.

РОЗДІЛ 1
ЗАСОБИ АНАЛІЗУ ФІНАНСОВИХ ПОКАЗНИКІВ ТА ЇХ РЕАЛІЗАЦІЇ

1.1. Аналітичний огляд предметної області

Невід’ємними складовими процесу інвестування є формування портфеля цінних паперів, його регулярний перегляд та оцінка його ефективності.

Формування портфеля (portfolio construction) цінних паперів включає визначення конкретних активів для вкладення коштів, а також пропорцій розподілу капіталу, що інвестується між цими активами. При цьому інвестор стикається з проблемами селективності, вибору часу операцій і диверсифікації. Селективність (selectivity), звана також мікропрогнозуванням, відноситься до аналізу цінних паперів і пов'язана з прогнозуванням динаміки цін окремих видів цінних паперів. Вибір часу операцій (timing), або макропрогнозування, включає прогнозування зміни рівня цін на акції порівняно з цінами для фондових інструментів з фіксованим доходом, такими, як корпоративні облігації. Диверсифікація (diversification) полягає у формуванні інвестиційного портфеля таким чином, щоб за певних обмежень мінімізувати ризик [4].

Процес перегляду портфеля (portfolio revision) як необхідна складова інвестиційного процесу має місце тому, що з часом цілі інвестування можуть змінитися, в результаті чого портфель перестане задовольняти потребам та цілям, що ставить перед собою його власник. В такому випадку інвестору потрібно буде сформувати новий портфель шляхом продажу частини цінних паперів та покупки деяких нових. Іншою підставою для перегляду портфеля є зміна курсу цінних паперів з плином часу.

У зв'язку з цим деякі цінні папери, що спочатку були непривабливими для інвестора, можуть стати вигідним об'єктом інвестування, і навпаки. Тоді інвестор захоче придбати перші, одночасно продавши останні з свого портфеля. Рішення про перегляд портфеля залежить, поміж інших факторів, також від розміру транзакційних витрат і очікуваного зростання прибутковості переглянутого портфеля [4].

Оцінка ефективності портфеля (portfolio performance evaluation) включає періодичну оцінку значень та показників прибутковості інвестиційного портфеля, часто в комбінації з оцінкою показників ризику, з якими стикається інвестор [4]. При цьому необхідним є використання прийнятних показників прибутковості та ризику, а також відповідні стандарти (своєрідні «еталонні» значення) для порівняння.

Ще одним типом гравців на фондовому ринку є трейдери. Інвестори і трейдери оперують на одному і тому ж самому ринку, використовують одні і ті ж самі напрямки та методи аналізу, і в принципі ставлять перед собою однакову ціль – щонайбільше примноження капіталу, проте використовують для досягнення своїх цілей різні стратегії. Втім, для повноцінної нормальної роботи фондового ринку потрібні й ті, й інші.

Різницю між інвестиціями та трейдингом у біржовому світі визначає перш за все часовий фактор. Вважається, що угоди, закриті протягом року, належать до трейдингу, а розраховані на більш тривалий термін — до інвестицій. На відміну від інвесторів, трейдери у виборі активів орієнтуються не на фундаментальний аналіз та відповідно фундаментальні показники компаній, а на стан ринку, користуючись його короткостроковою волатильністю (коливаннями ціни активів) для отримання прибутку. Також характерним лише для трейдерів є те, що вони можуть отримувати вигоду як на зростаючих, так і на спадаючих ринках. Час трейдера обмежений, тому йому, на відміну від інвестора взагалі не важлива якість фінансових інструментів як така — він аналізує лише їхню поточну позицію на ринку, використовуючи для цього технічний аналіз.

Саме трейдери забезпечують ліквідність для інвесторів, стаючи протилежною стороною угоди. Незалежно від ролі та підходу, що використовується, трейдери – необхідні учасники фондового ринку. Очевидно, що для функціонування ринку необхідні і трейдери, і інвестори. Без трейдерів інвестори не мали б ліквідності, що дозволяє купувати та продавати акції. Без інвесторів трейдери не мали б підстав купувати чи продавати. Взаємодія цих двох груп учасників і утворює відомі сьогодні фінансові ринки.

Варто зазначити, що при тому, що інвестиції спрямовані на отримання інвестором прибутку, вони все ж таки не є гарантованим способом його отримати. Різні способи інвестування забезпечують різні гарантії отримання доходу, але завжди існує ризик того, що замість прибутку інвестор отримає збиток. Невід'ємною складовою успішних з точки зору дохідності інвестицій є прийняття грамотних та зважених інвестиційних рішень, які базуються та фактично самі по собі є результатом ретельного та виваженого фінансового аналізу та, як наслідок, грамотного інвестиційного менеджменту, який являє собою сукупність таких рішень.

Базовим для прийняття інвестиційних рішень та побудови інвестиційної стратегії, яка складає основу інвестиційного менеджменту є процес оцінки активів. Одним з методів, що застосовуються для цього є фундаментальний аналіз. Оцінка знаходиться у центрі уваги фундаментального аналізу та його основною метою фундаментального аналізу для приватних інвесторів є вибір цінних паперів, які зростуть у ціні або за якими емітенти платитимуть стабільно високі дивіденди.

Іншим методом проведення оцінки активів є технічний аналіз. Основним положенням цього методу є розрахунок фундаментальних фінансових змінних та проведення оцінки активів на основі залежностей ціни від цих змінних. Припущення, що лежить в основі цього методу, полягає в тому, що ціни рухаються, слідуючи певним зразкам, так званим «моделям» [5].

Технічні аналітики використовують на практиці широке різноманіття моделей – від найелементарніших до дуже складних. Припущення, що лежать в

основі цих моделей, дуже часто розрізняються, та все ж моделі мають спільні риси, які можна віднести до кількох широких категорій.

Взагалі, існують два основні підходи до оцінки в технічному аналізі. Перший з них - оцінка дисконтованих грошових потоків (discounted cash flow - DCF) - співвідносить вартість активу з поточною вартістю очікуваних у майбутньому грошових потоків, що припадають на цей актив.

Згідно з другим підходом, що визначається як порівняльна оцінка, вартість активу обчислюється в ході аналізу ціноутворення подібних активів, пов'язуючи його з будь-якою змінною (наприклад, з доходами, грошовими потоками, балансовою вартістю або об'ємом продажів) [5].

Результати оцінки можуть бути різними залежно від застосовуваного підходу. Вибір конкретної моделі та підходу до оцінки відповідно робиться на основі конкретних поставлених інвестором задач.

Процес обрахування загальної оцінки активу включає в себе процеси обрахунку ряду наступних фінансових показників, таких як: дохідність власного капіталу (return on equity, ROE), чиста приведена вартість (net present value, NPV), індекс рентабельності (profitability index, PI), дисконтований строк окупності інвестицій (discounted payback period, 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 управління цифровими активами

Розрахунок, моніторинг та подальший аналіз значень всіх необхідних фінансових змінних та показників для інвестиційного аналізу, прийняття інвестиційних рішень та корекції портфеля цінних паперів та стратегій інвестування потребує багато часу, тому питання автоматизації цих розрахунків на ряду з об’єднанням та формуванням графічного представлення обчислених даних

для зручності та спрощення інвестиційного менеджменту є дуже актуальним вже довгий час.

В ході вирішення цього питання був створений новий клас програмних продуктів, основна задача розробки та функціонал яких націлений на автоматизацію розрахунків фінансових показників та змінних, збір та відображення отриманої множини фінансових даних в зручному для інвестора та виконання ним подальшого інвестиційного менеджменту, зважения та прийняття інвестиційних рішень, а також першочерговий вибір та подальше коригування стратегій та підходів до інвестування.

З подальшим розвитком такого програмного забезпечення поступово почало підніматись питання можливості окрім автоматизації розрахунків фінансових показників та об'єднання отриманих даних в зручне інфографічне представлення для подальшого аналізу, які вже мали на той час імплементацію в існуючих програмних системах вищезгаданої спеціалізації, ще й автоматизації в цілому механізмів стратегій інвестування як таких, який включав би програмний моніторинг зміни значень необхідних фінансових показників та відповідну реакцію програми на так звані «сигнали» (коли поточне значення або динаміка зміни того чи іншого показника збігається з заданими в специфікації стратегії) у вигляді автоматичного створення транзакцій з покупки або продажу заздалегідь визначеної кількості цінних паперів.

Таким чином почали з'являтись програмні продукти спеціалізацією яких замість надання сукупності функціоналу, інструментів та представлень суто для розрахунків та аналітики, є реалізація стратегій інвестування, можливості застосування тієї чи іншої стратегії або їх сукупності та автоматизації слідування ним та покупки / продажу активів на базі програмно прийнятих рішень. Такі програмні продукти були популярними і використовувались та використовуються до теперішнього моменту здебільшого саме трейдерами, згодом такий підхід до торгування на біржі з використанням вищезгаданого типу програмних продуктів ліг в основу абсолютно нового напрямку трейдингу та отримав назву автоматизованого алгоритмічного трейдингу (automated algorithmic trading).

На сьогоднішній день серед інвесторів та трейдерів є великий запит на програмні системи, які поєднують в собі функціонал обох вищезгаданих видів інформаційних систем управління цифровими активами, надаючи одночасно як можливості та інструменти для аналітики, так і механізми автоматизації інвестиційних стратегії, користувацьких преференцій та підходів до процесу торгівлі цінними паперами.

Такі комплексні системи почали розроблятись та з'являтись на ринку програмних продуктів відносно недавно та присутні на ринку програмних продуктів в невеликій кількості, але великим попитом серед інвестиційної та трейдингової аудиторії обумовлений швидкий розвиток, вдосконалення, збільшення складності та різноманіття реалізованого функціоналу в розробці нових таких систем.

Отже, на сьогоднішній день представлені наступні види систем управління персональним портфелем цінних паперів:

– системи для розрахунку та аналітики фінансових показників;

– системи для автоматизації деяких процесів інвестування (здебільшого механізму стратегій) та програмного прийняття інвестиційних рішень;

– системи, які поєднують в собі функціонал обох вищезгаданих видів інформаційних систем управління цифровими активами.

1.3. Огляд і порівняльний аналіз існуючих систем

1.3.1. MorningStar

Morningstar Portfolio Manager – один із передових інструментів для відстеження інвестицій, який дозволяє інвесторам імпортувати або вручну управляти своїми активами та отримувати цінну інформацію [8].

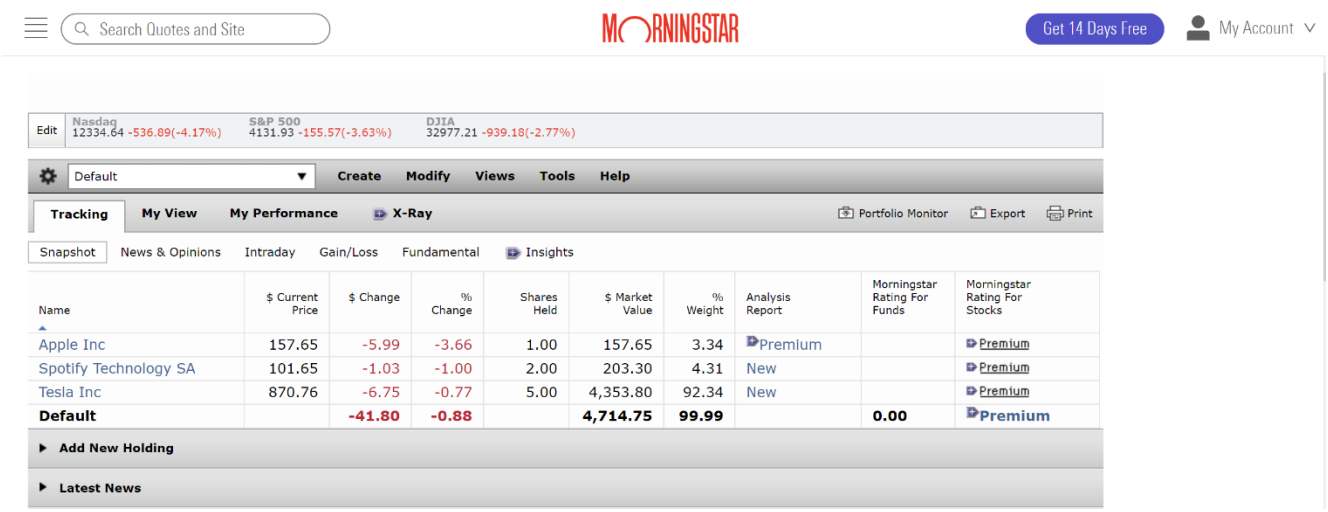

Рис. 1.1 Стартове вікно MorningStar при огляді портфелю

Менеджер портфеля Morningstar використовує дещо інший підхід у порівнянні з фінансовими терміналами аналогічних ресурсів – у деяких із них поширеною є функція автоматичного імпортування даних від брокера та/або банку. В Morningstar же потрібно вручну вводити кожну зі своїх інвестицій в інструмент «Диспетчер портфеля» (хоча компанія анонсує введення автоматизованого рішення).

Хоч введення даних інвестиційного рахунку в портфельний менеджер Morningstar потребує додаткових витрат часу та зусиль, аналіз, наданий інструментом, вартий зусиль. Інструмент надає основні дані про кожну з ваших інвестицій, включаючи її поточну ціну, щоденні зміни вартості та її відсоткову вагу у загальному портфелі. Також надається можливість додавання кількох портфелів, зберігання у них не тільки акцій, але і облігацій та облік готівкових збережень.

Менеджер портфеля Morningstar також надає надійні дані про загальну ефективність портфеля. Він показує загальну прибутковість вашого портфеля за місяць і рік і порівнює його з попередньо обраним еталоном. Однак основною сильною стороною Morningstar є його функціональний пакет Instant X-Ray.

Instant X-Ray надає користувачеві детальну інформацію про розподіл активів його портфеля. Він показує стиль інвестування портфеля як для акцій, так і для облігацій. Instant X-Ray розбиває інвестиції за секторами, типами акцій і навіть за регіонами. В результаті цього користувачу відображається середньозважений

коефіцієнт витрат фонду вашого портфеля.

До того ж, Instant X-Ray показує, який відсоток кожна окрема інвестиція у вашому портфелі становить у вашому портфелі в цілому. Це може бути особливо корисно для тих, хто інвестує в пайові фонди.

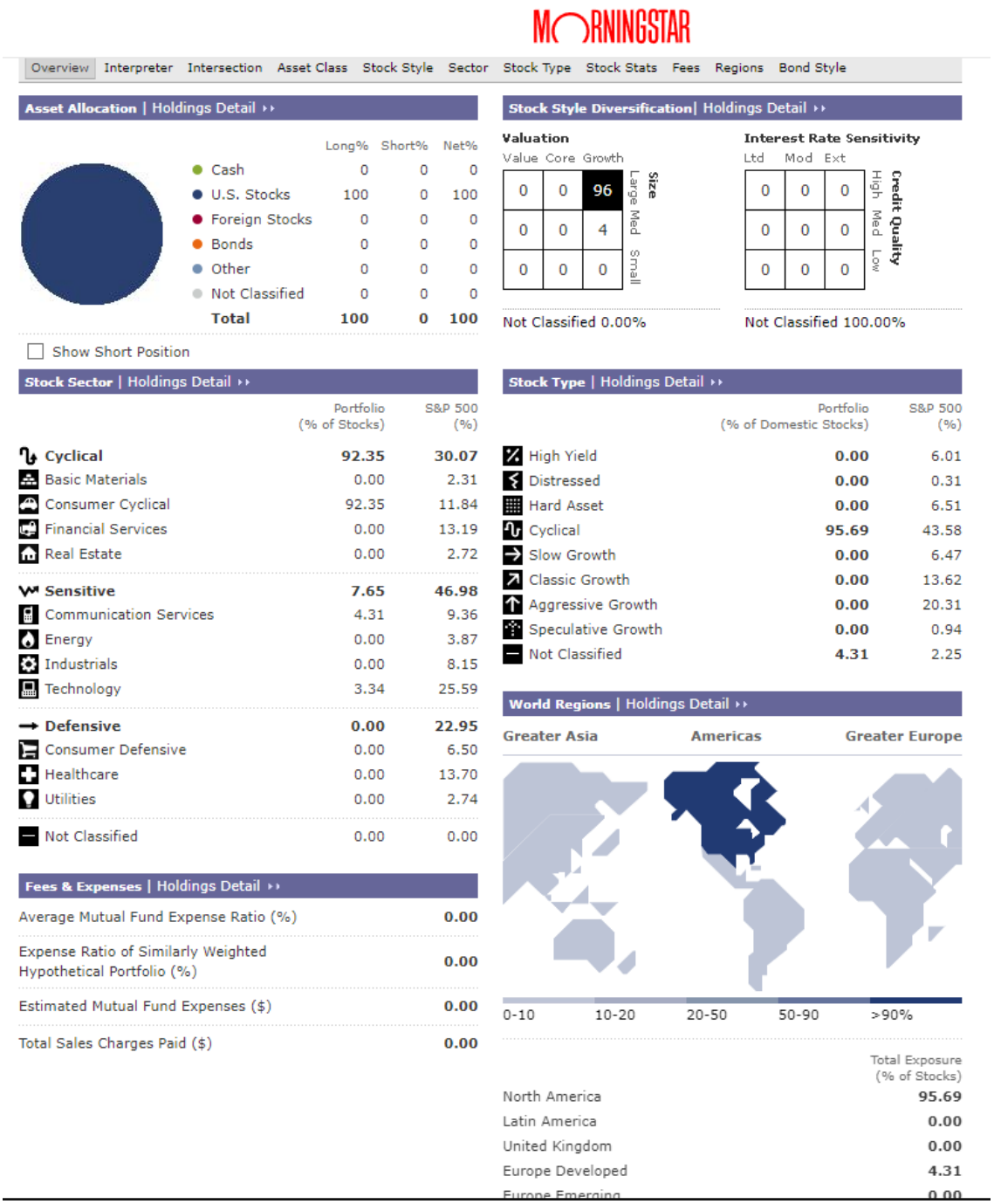

Рис. 1.2. Результат опрацювання портфелю послугою Instant X-Ray

Morningstar пропонує як безкоштовне, так і платне членство. Ви можете

використовувати його Portfolio Monitor з будь-яким членством, хоча деякі інструменти вимагають платного членства (підписки). Зокрема, практично усі результати детальної аналітики Instant X-Ray, X-Ray (це окремий інструмент), інсайти, розширені скрінінг (пошук відповідних активів за підібраними параметрами) та пошук схожих активів відноситься до платних послуг. Мінімальна вартість підписки - $249 за рік користування.

1.3.2. Yahoo! Finance

Yahoo! Finance — це медіаресурс, що є частиною мережі Yahoo!. Він надає фінансові новини, дані та коментарі, включаючи котирування акцій, прес-релізи, фінансові звіти та оригінальний контент. Він також пропонує деякі онлайн-інструменти для управління особистими фінансами. Окрім розміщення партнерського контенту з інших веб-сайтів, він публікує оригінальні публікації своєї команди штатних журналістів. Yahoo! Finance (надалі YF) також впровадили функцію перегляду новин та курсів криптовалют. YF представлений веб-версією, що доступна з браузера незалежно від платформи пристрою, а також мобільними застосунками для Android та iOS [10].

Однак YF надає користувачеві можливість не лише переглядати новини та курси акцій, але і створювати свій портфель активів для більш ґрунтовного аналізу та активного відстеження своїх вкладень. Така опція дозволяє зібрати в одному місці ті активи, які відіграють найбільшу роль для користувача та отримувати найбільш релевантну для нього інформацію. В рамках цього YF надає наступні можливості:

– створювати декілька портфелів та редагувати їх при потребі;

– додавати, редагувати та видаляти лоти (транзакції купівлі / продажу) тих компаній, що цікавлять;

– додатково відстежувати готівкові рахунки та приватні (непублічні) акції;

– налаштовувати відображення графіків;

– створювати свої власні представлення даних, обравши потрібні показники

із десятків відібраних;

– створювати та використовувати скрінери (інструменти для пошуку активів, що відповідають конкретним характеристикам, заданих користувачем);

– проводити експорт та імпорт даних портфеля у вигляді CSV-файлу.

Це лише незначна частина із тих всіх можливостей, що надаються YF. Даний ресурс також надає надзвичайно обширну інформацію щодо ситуації на ринку та окремих компаній, включаючи історичні, статистичні дані, рекомендації аналітиків тощо. Недарма YF багатьма фінансистами та інвесторами признаний одним із найкращих для відповідних цілей, і входить в рейтинги найбільш відвідуваних сайтів у світі в категорії «Новини та медіа-вебсайти».

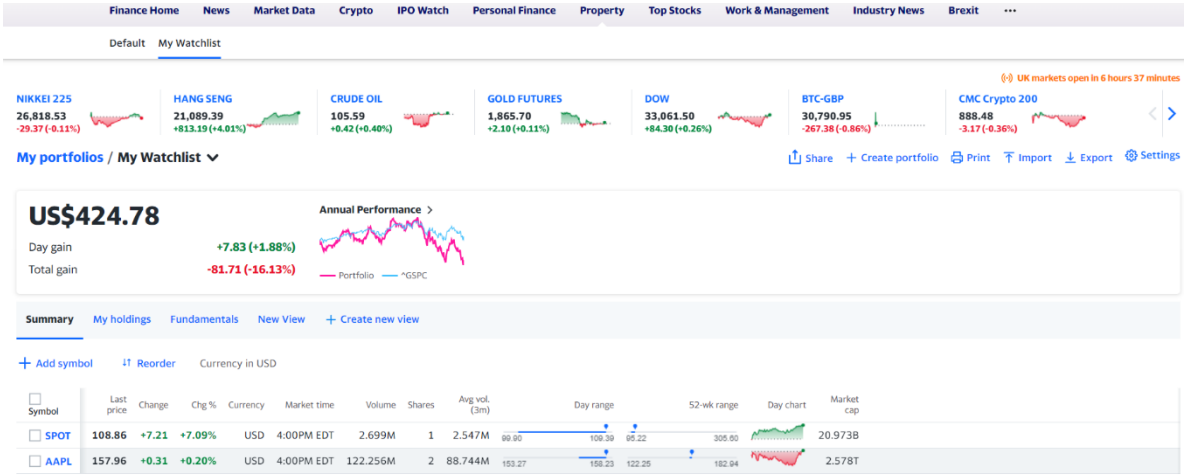

Рис. 1.3. Представлення створеного портфелю в YF

1.3.3. Google Finance

Google Finance – ресурс для надання фінансової інформації. Даний сервіс надає доступ до фінансової інформації значної кількості міжнародних компаній, що представлені на ряді фондових бірж. Доступна інформація щодо курсів та рейтингів цінних паперів, прес-релізи та фінансові звіти цих організацій. За допомогою уже розроблених інструментів, такий як Google News, відбувається пошук та видача відповідних інформаційних матеріалів про ту чи іншу компанію. Історичні дані доступні у вигляді графіків, також доступною є функція накладання на одну

координатну площину кількох графіків, що представляють акції різних компаній, індексів, портфелів тощо. Сервіс також відображає тенденції ринку, найбільш популярні акції в Google, календар дат публікації звітів про фінансові результати компаній тощо [9].

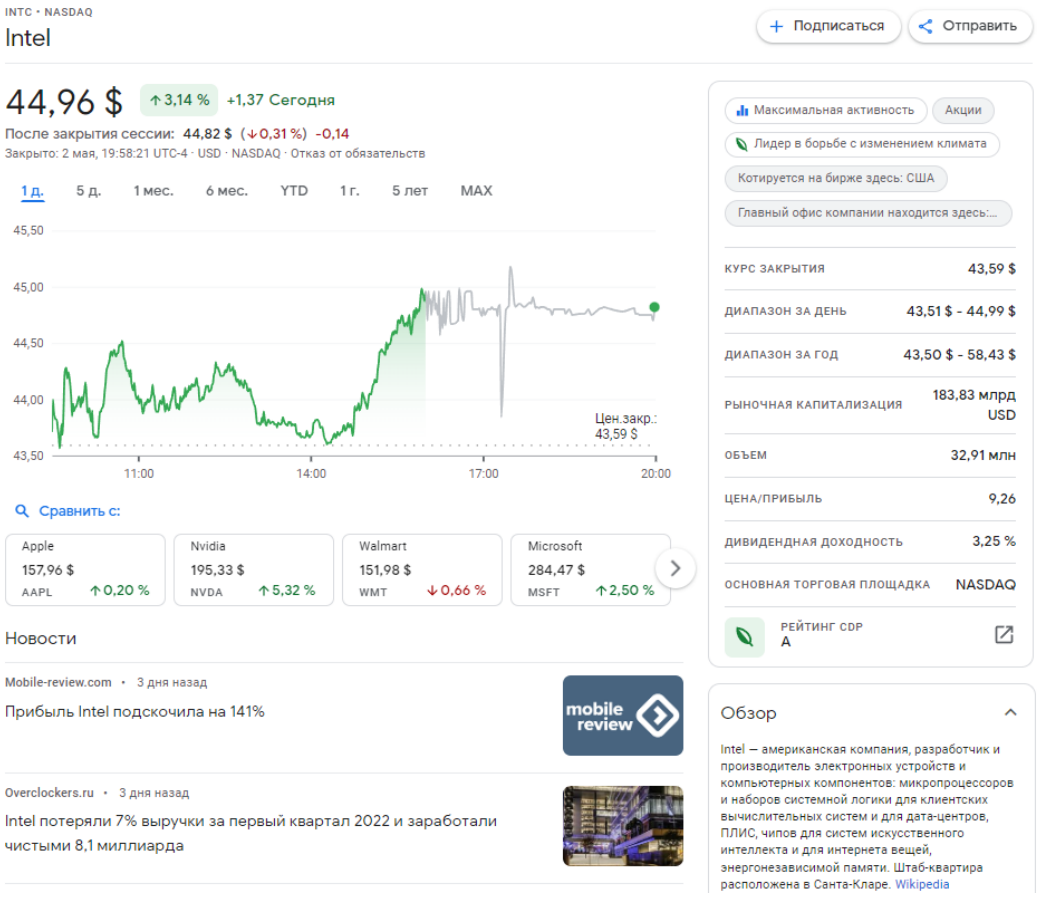

Рис. 1.4. Огляд інформації Google Finance про актив на прикладі акцій компанії Intel

Сайт пропонує низку сервісів для управління персональною фінансовою інформацією. Зокрема, у користувача є можливість створити свій список відстеження або портфель, заповнивши їх тими акціями які їх цікавлять або якими володіють відповідно. Це допомагає залишатись в курсі подій, переглядати публічну звітність компаній та основні фінансові показники. Також доступний перегляд основної інформації за портфелем, як-от указання долі в портфелі компаній, що належать до домінуючої категорії капіталізації, їх дивідендну характеристику та співвідношення «ціна-прибуток», а також характеристику

портфеля з точки зору ваги в них компаній з високим екологічним рейтингом.

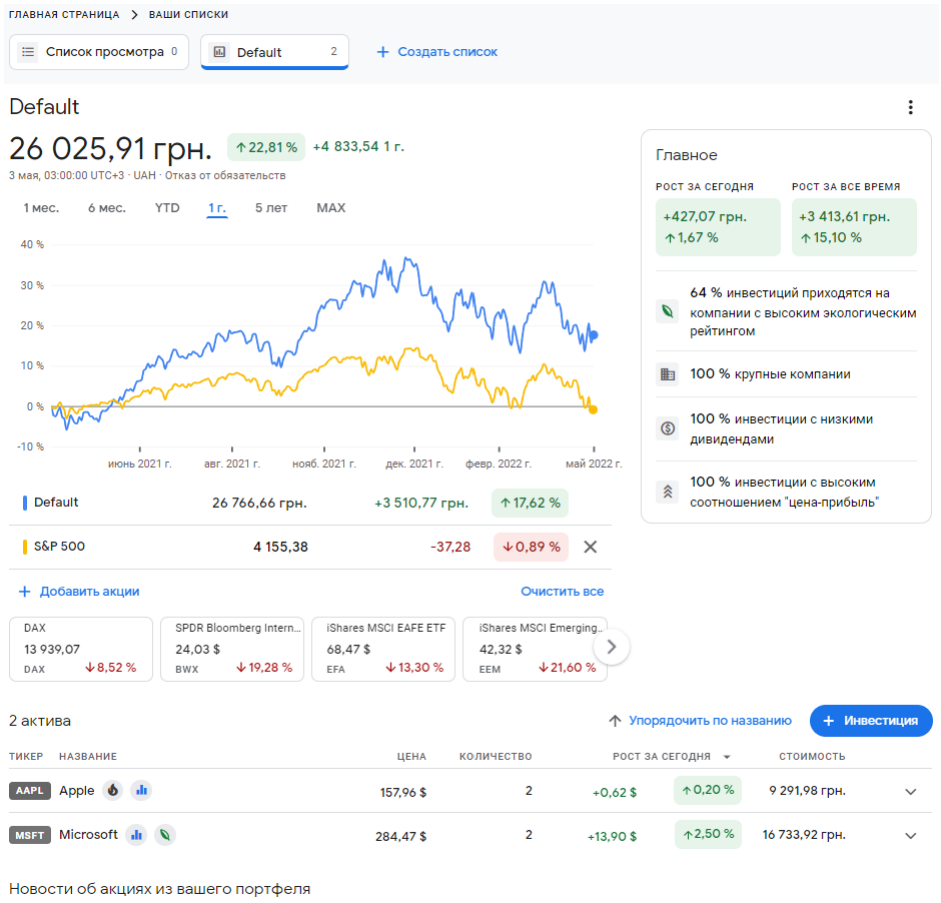

Рис. 1.5. Огляд портфелю у порівнянні з індексом S&P 500 в Google Finance

Таким чином, користувачеві Google Finance надається основна інформація, необхідна для того, щоб бути в курсі подій фінансового світу, а також приводиться хоч і поверхнева, та все ж аналітика його портфелю.

1.3.4. Підсумки огляду існуючих систем

Перш ніж привести результати огляду та порівняння існуючих інструментів для обліку портфеля акцій, варто відзначити один важливий аспект. На вибір для огляду вищезгаданих ресурсів, окрім їх поширеності, широкого функціоналу та можливостей, повпливав ще нюанс доступності.

На ринку присутня значна кількість аналогів, рекомендованих аналітиками, що допомагають вирішувати більш комплексні задачі, такі як заощадження та

інвестиції для забезпечення пенсії, освіти тощо. Вони з'ясовують цілі користувача, його горизонт планування та бюджет, надаючи взамін більш детальну, цільову аналітику і плани для досягнення цілей. Очевидно, що такі можливості є більш затребуваними для користувачів, оскільки програмні потужності інструменту локалізуються для виконання конкретного чіткого плану. На жаль, значна частина з них накладає певні обмеження, зокрема, необхідність валідації акаунту за допомогою документів, що підтверджують проживання в США або ЄС. Іншою суттєвою перешкодою може бути вимога прив'язати щонайменше один банківський або брокерський рахунок у організаціях Західної Європи або США. Очевидно, вкрай мала частина українців може похвалитись можливістю вирішення цих вимог. З огляду на це, вибір інструменту для так званого paper trading значно звужується.

Розглянуті ж ресурси надають доволі широкий функціонал, і кожен з них має свої переваги та недоліки. Кожен із них надає користувачу значний об'єм фінансової інформації про показники компанії, зміну ціни її акцій з часом, відбирає новини, що стосуються безпосередньо конкретної компанії, дозволяють групувати активи в портфелі та редагувати їх, і все це – зручним для користувача способом, з чудовим візуальним представленням. Недоліки ж зазвичай є індивідуальними для кожного сервісу.

Зокрема, Morningstar основний свій потенціал приховує за платною версією, оплата якої може бути проблематичною для початківця. Але навіть із безкоштовним функціоналом можуть виникнути труднощі – обширний об'єм фінансової інформації, наданої ресурсом, розміщений на значній кількості вкладок та панелей, що дезорієнтує новачка; до того ж, посібник користувача розвинутий слабо, що залишає багато запитань при спробі розібратись із інтерфейсом сайту та його наповненням.

Варто відзначити, що це швидше відноситься до інструменту управління портфелем – в свою чергу, сторінки конкретних компаній представлені зрозуміло, із грамотною структурою подання інформації. При їх перегляді достатньо розуміти фінансову термінологію та суть показників (що само по собі являється необхідним

при занятті інвестиціями). У випадку ж менеджера портфелю потрібно ще з’ясовувати, що означають ті чи інші поля або інструменти в рамках самого ресурсу.

Окремо варто згадати про графічний інтерфейс цього менеджера портфелю. На фоні своїх конкурентів та сучасних технологій представлення візуальної інформації Morningstar доволі відчутно відстає – не вистачає інтерактивності та гнучкості при роботі з таблицями та графіками, до того ж, стилістичне оформлення та деякі візуальні елементи нагадують сайти 15-річної давності. Сильною стороною ресурсу, якому частина користувачів безпосередньо платить свої гроші, це не назвеш.

Google Finance виглядає чудово зі сторони візуального представлення даних та охоплення компаній (охоплює біржі Північної Америки, Європи та Азії), однак ситуація із наданням саме фінансових даних протилежна проблемі Morningstar: у випадку останніх інформації моментами було надто багато, а у Google Finance навпаки – надто мало. При перегляді сторінки компанії користувачу окрім графіку із зміною курсу акцій відображається близько двох десятків показників, при перегляді портфелю – ще менше. При такому багатогранному та складному процесі як управління та облік портфелем акцій така кількість інформації може бути недостатньою.

Також практично відсутня аналітика портфелю як такого – за винятком прибутку/збитку портфелю за весь час і один день, на його сторінці відображаються лише дані щодо приналежності компаній портфелю до тих чи інших категорій, як-от капіталізації, дивідендності та екологічного рейтингу.

Ще один незначним, однак все ж неприємним недоліком є можливість зміни горизонту даних на графіках лише у визначених проміжках (1 або 5 днів; 1 або 6 місяців; з початку поточного року до сьогодні; протягом минулого року або п’яти; за всю історію). Таким чином, ніяким чином налаштувати відображення даних на відмінному від стандартного проміжку не вдасться.

Yahoo! Finance в свою чергу якнайкраще вирішили проблему перевантаженості графічного інтерфейсу даними – вони делегували можливість

вибору того, що відображати, самому користувачу. Таким чином, окрім кількох базових вкладок із мінімально необхідним наповненням, користувач сам може створювати свої власні представлення (views), в яких він власноруч відбере необхідні показники з понад 70 доступних та впорядкує їх на власний розсуд.

Графіки зміни ціни акцій не мають вищезгаданої проблеми Google Finance – надається можливість їх широкого масштабування, налаштування, додавання статистичних індикаторів, порівняння з курсами інших компаній тощо. Однак усе це стосується графіків на рівні компаній – графік на рівні портфеля представлений тільки на сторінці самого портфеля у порівнянні з індексом S&P 500, до того ж, він надто малий і не масштабований, щоб по ньому можна було проводити хоч якийсь аналіз. Відкрити його у повному розмірі, провести налаштування також не є можливим.

Саме на рівні обліку портфеля і проявляється ще один недолік YF – даний ресурс не надає можливості провести аналітику портфелю загалом, натомість відображається лише його прибуток або збиток. На фоні всієї потужності ресурсу та об'єму інформації, що він надає, це є доволі значимою прогалиною, тому що таким чином нівелюється сама суть портфелю як такого, і користувач не отримує тієї інформації, яка йому вкрай потрібна.

Результат аналізу розглянутих існуючих систем в термінах виявлених в них переваг та недоліків відображено в таблиці 1.1.

Таблиця 1.1

Порівняння систем обліку інвестицій

	Переваги	Недоліки
MorningStar	- Широкий перелік показників та даних про компанії; - Підтримка кількох видів активів (акції, облігації, готівка тощо); - Можливість створення кількох портфелів;	- Результати детальної аналітики доступні лише в рамках платної підписки; - Застарілий графічний інтерфейс; - Нестача візуального подання (значна частина інформації

Закінчення таблиці 1.1

	- Чітка аналітика портфелю; - Інструмент Instant X-Ray, що проводить оцінку та аналіз портфелю, стилю інвестування, розподіл активів за рядом критеріїв;	припідноситься в табличному представленні); - Значний об'єм інформації розкиданий по ряду вкладок та панелей; - Недостатньо розвинене керівництво користувача.
Yahoo! Finance	- Вдалий підбір новин, фінансової інформації та показників; - Користувач може створити своє представлення, в якому відображатимуться ті дані, які релевантні для користувача; - Скрінери, що надають можливість шукати компанії за рядом критеріїв; - Широкі можливості налаштування та відображення графіків.	- Аналітика портфелю практично відсутня.
Google Finance	- Чудове графічне оформлення; - Агрегація новин та статей, що стосуються компанії / портфелю; - Можливість накладання графіків зміни ціни акцій різних компаній; - Широке охоплення бірж; - Можливість відстеження / обліку кількох типів активів.	- Аналітика портфелю поверхнева; - Недостатній об'єм фінансової інформації; - Перегляд графіків доступний лише за обмежений перелік періодів.

1.4. Постановка задачі для розробки

Зважаючи на вищенаведене розробка прототипу інформаційної системи є актуальною.

Відповідно, метою дипломного проекту є розробка прототипу інформаційної системи управління та обліку персонального портфелю цінних паперів для отримання вичерпної оцінки ефективності інвестицій.

Відповідно до поставленої мети в дипломному проекті необхідно вирішити нижче наведені задачі.

Загалом задача полягає в розробці прототипу інформаційної системи управління та обліку персонального портфелю цінних паперів. Це має бути інструмент для paper trading, основним призначенням якого є саме моделювання процесів інвестування для закріплення користувачем знань в фінансовій грамотності та інвестиціях, а також спостереження ефекту від використання тих чи інших підходів до цього процесу на практиці, підтримка проведення реальних торгів активами в рамках розроблюваного прототипу не передбачається. Тим не менш можливість підключення до розробленого прототипу програмного модуля для проведення реальних торгів на реальній біржі має бути передбачена з точки зору архітектури програмного продукту, так як є можливим розширенням до нього в перспективі.

На основі проведеного аналізу аналогів, їх порівняння та огляду виявлених недоліків існуючих програмних систем, а також огляду та аналізу специфіки предметної області було сформовано вимоги до розроблюваного програмного продукту.

Основною метою розробки прототипу є його подальше розширення до повнофункціональної інформаційної системи, а отже особливо важливою вимогою до нього є забезпечення необхідної та максимально сприятливої для цього інфраструктури, забезпечення високих показників розширюваності, переносимості, гнучкості та масштабованості з точки зору як архітектури, так і безпосередньої технічної реалізації. Загалом програмний продукт має бути

спроектований таким чином, щоб забезпечити його ефективну підтримку в майбутньому, можливість легкого подальшого розширення функціоналу та планомірного масштабування.

Розробка прототипу має бути орієнтована на ОС Windows та представляти з себе десктопний додаток. Проте необхідно також передбачити з точки зору формованої інфраструктури програмного продукту можливу зміну платформи та операційної системи, на якій буде працювати програмний продукт, адже дуже ймовірним є створення різних версій продукту для різних платформ (наприклад, веб- та мобільної версії) з часом. Також потрібно забезпечити належну основу для потенційного переносу інфраструктури розроблюваного прототипу в «хмарну» інфраструктуру, адже зі збільшення кількості користувачів, об’єму даних в системі та загального навантаження на неї питання переходу до використання хмарних сервісів буде дуже актуальним.

Аналіз аналогів та виявлених недоліків існуючих систем показав, що перш за все серед представлених програмних продуктів немає «золотої середини» в плані повноти інформації та функціоналу, що надається користувачеві, цього або замало, що не дозволяє провести повноцінну аналітику фінансових інвестиційних показників та портфеля, або ж навпаки забагато, що перевантажує користувача завеликою кількістю інформації, опцій та функцій, з якими буває важко зосередити увагу на дійсно необхідному та взагалі розібратись із тим що означає та за що відповідає той чи інший показник або функціонал. Також спостерігається і те, що функціоналу, показників та інформації буває просто забагато, це виглядає складно, але фактично користувач більшою частиною з цього не користується, так як це не є необхідним для вирішення поставлених перед ним задач.

Отже, однією з основних вимог до розроблюваного прототипу інформаційної системи також є створення простого та зрозумілого інтерфейсу користувача, який би з одного боку містив в собі достатньо інформації, яка дійсно необхідна, а з іншого – не був перевантаженим та складним для розуміння і сприйняття.

Крім того, враховуючи два різних основних напрямки проведення торгів цифровими активами, розглянуті в аналізі предметної області – інвестування та

трейдинг, в розроблюваному продукті потрібно реалізувати функціонал, який може бути використовуваним і трейдерами, і інвесторами. Прототип програмного продукту має включати і об’єднювати в собі інструменти як для аналітики, так і для деякої автоматизації інвестиційних процесів. Для прототипу достатнім є реалізація функціоналу лише автоматизації простих трейдингових стратегій та програмного додавання відповідних моделей транзакцій продажу та покупки активів на основі них. Цього немає в розглянутих існуючих системах-аналогах, тому використання та надання користувачам автоматизованих стратегій як прикладу такого функціоналу в складі прототипу дозволить оцінити в цілому затребуваність наявності такого класу функціоналу, а також його реальну використовуваність та популярність серед користувачів.

Це дозволить також залучити більшу кількість користувачів, адже надаючи функціонал і для трейдерів, і для інвесторів збільшується об’єм потенційної цільової аудиторії користувачів програмного продукту, а це в свою чергу забезпечить отримання більш повної та вичерпної оцінки прототипу та більшої кількості ідей для подальшого вдосконалення, розширення та розвитку прототипу в повноцінну інформаційну систему.

Функціонал розроблюваного прототипу має включати:

1. Додавання в портфель акцій конкретної компанії з попередньо визначеного списку організацій, представлених на фондовій біржі:

– указання назви (символа / ідентифікатора компанії), дати купівлі, об’єму та ціни (загальної або за акцію);

– додавання у вигляді разових транзакцій – інформація вказується щодо однієї конкретної покупки акцій компанії X за конкретною ціною за раз (при подальшому поповненні портфелю дані потрібно вводити виключно щодо нової покупки, а не перераховувати вручну всю вартість).

2. Відображення транзакцій продажу в загальному (випадаючому) списку транзакцій, розрахунок та відображення різниці між загальною сумою покупки та сумою отриманою з продажу акцій компанії X.

3. Функціонал для автоматизованого розміщення транзакцій купівлі / продажу акцій компанії X на основі заданих користувачем преференцій (стратегій).

4. Графічне відображення прибутковості (графіку зміни ціни акцій конкретної компанії в порівнянні з ціною, за яку користувачем вона була куплена) як для окремо взятої компанії (або транзакції), так і для портфеля в цілому.

Додаткові вимоги щодо відображення:

1. Обчислювати та відображати середню ціну придбаних акцій та різницю з актуальним значенням.

2. Представити функціонал, що дозволятиме отримувати історичну інформацію щодо зміни ціни акцій компанії та інформації про неї (в текстовому та графічному поданні), мультиплікатори, інші дані для технічного аналізу тощо.

3. Відображення на графіку зміни ціни акції компанії X за заданий користувачем період (від деякої дати до деякої дати).

1.5. Висновки до розділу

В розділі було проведено огляд предметної області та визначено які проблеми та задачі є актуальними в розробці програмних систем для управління та обліку персонального портфелю цінних паперів. Крім того було розглянуто декілька вже існуючих інформаційних систем такого призначення, проведено їх порівняння та аналіз виявлених в них недоліків.

В результаті огляду предметної області та аналізу аналогічних вже розроблених систем та їх недоліків було означено перелік задач, які є актуальними для вирішення, виділено чинники та характеристики, які є важливими для розроблюваних програмних систем цього напрямку та спеціалізації.

На основі отриманих даних було проведено постановку задач для розроблюваного програмного продукту та визначено для нього функціональні та нефункціональні вимоги.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

На основі поставленої задачі перш за все було проведено аналіз області рішень поставлених задач. Коли визначено принципові способи вирішення всіх поставлених завдань (можливо, в якихось обмеженнях), основною проблемою стає спосіб організації програмної системи, який дозволив би реалізувати всі ці рішення і при цьому задовольнити вимоги, що стосуються нефункціональних аспектів програми, що розробляється. Визначення способу організації програмного забезпечення у вигляді системи взаємодіючих компонентів зводиться до задачі проектуванням архітектури програмного забезпечення [15].

2.1. Вибір архітектури розроблюваної системи

Найбільш важливим етапом проектування програмного продукту є вибір його архітектури, адже саме від цього напряму залежить можливість забезпечення його ефективної підтримки в майбутньому, можливість полегшеного та незатратного з точки зору часу та зусиль подальшого розширення функціоналу програмного продукту, а також створення та підтримки нових його версій для інших платформ (наприклад мобільної або веб-версії), а також його планомірного масштабування.

Архітектура програмного забезпечення визначається складністю розв'язуваних завдань, ступенем універсальності програмного забезпечення, що розробляється, і кількістю користувачів, що одночасно працюють з однією його копією [6].

Архітектурними конструкціями програмних систем, використання яких має

широке застосування, є монолітна (Monolithic), багаторівнева (N-Layered, або також її ще називають N-Tiered) та мікросервісна (Microservice).

2.1.1. Монолітна архітектура

При монолітній архітектурі система немає будь-якої явно вираженої внутрішньої структури. Це просто набір процедур, що використовують загальні глобальні дані, що викликаються один одним, зовнішнім ресурсом або користувачем [16]. Приклад реалізації такої архітектури представлено на рис. 2.1.

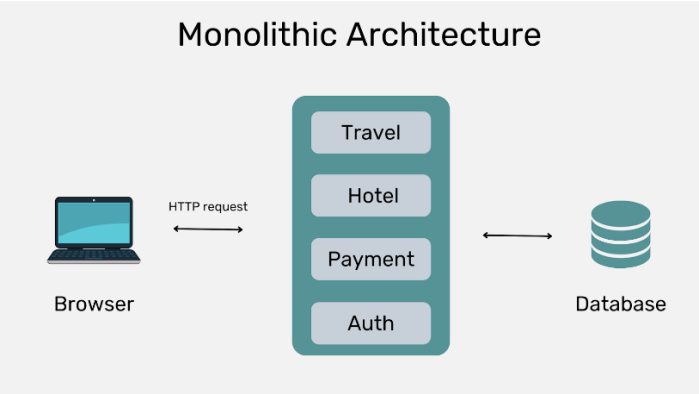

Рис. 2.1. Схема структури програмної системи з монолітною архітектурою

Використання монолітної архітектурної структури для програмних систем має багато недоліків. Монолітна архітектура не справляється з потребою бізнесу прискорюватись, оскільки великі та дуже залежні між собою модулі складно швидко змінювати та адаптувати. При використанні монолітної архітектури апаратно залежний і апаратно незалежний код у складі системи тісно переплітаються, що вкрай ускладнює розвиток системи або її перенесення на іншу апаратну платформу. Наявність безсистемних розгалужених зв'язків між компонентами системи з монолітною архітектурою призводить до необхідності корекції багатьох компонентів системи за умови реальної потреби зміни лише одного з них. При цьому може спостерігатися ефект снігової лавини – зміна однієї процедури вимагає зміни кількох інших, кожна з яких вимагає зміни ще кількох інших процедур тощо.

Крім того, оскільки монолітна програмна система є повністю замкнутою в контексті поведінки, під час роботи вона може взаємодіяти з іншими службами або сховищами даних, однак основа її поведінки реалізується у єдиному власному процесі, а вся система при цьому розгортається як один елемент. Для горизонтального масштабування програмної системи з такою архітектурою необхідно всю цю систему в повному обсязі дублювати на декількох серверах або віртуальних машинах.

Перевагою використання такого архітектурного підходу є проста та швидка реалізація, оскільки не потрібно витрачати час на проектування компонентів, розділення відповідальності між ними та правил (інтерфейсів) їх взаємодії.

2.1.2. Багаторівнева архітектура

Багаторівнева архітектура виникла у відповідь обмеженням монолітної архітектури в контексті розширюваності, переносимості і майбутньої підтримки. Основна ідея багаторівневої архітектури полягає в наступному:

Багаторівнева архітектура передбачає взаємодію між рівнями виключно через їх інтерфейси, при цьому внутрішня реалізація рівнів прихована від інших рівнів. Це дозволяє у разі потреби змінювати внутрішні реалізації функціоналу рівня без потреби змінювати інші компоненти системи та без ризику порушити їх правильність роботи, що забезпечує високий показник розширюваності системи. Дуже легко з такою архітектурою навіть повністю замінити весь рівень, потрібно лише забезпечити стандартний інтерфейс його взаємодії з іншими рівнями програмної системи [16]. Приклад реалізації такої архітектури представлено на рис. 2.2.

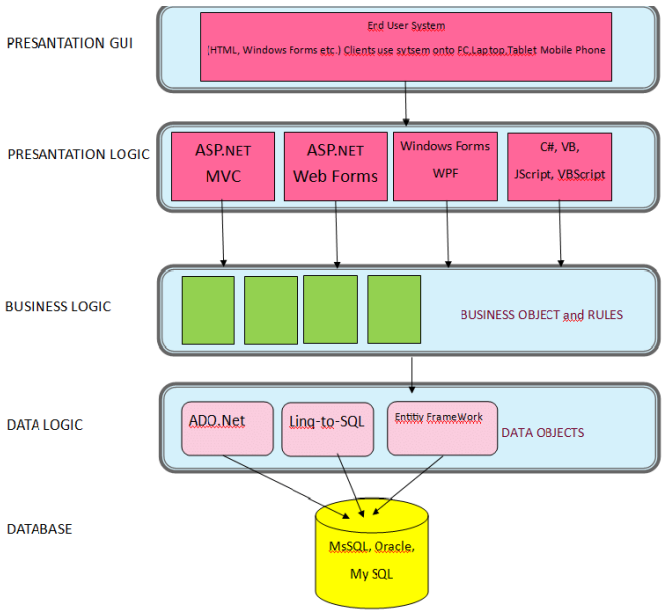

Рис. 2.2. Схема структури системи з багаторівневою архітектурою

Переваги такого архітектурного рішення:

– централізована бізнес-логіка (правила обробки даних). При необхідності зміни тих чи інших правил, не потрібно переробляти клієнтський додаток, потім витрачати зусилля його поширення на клієнтські машини. Тепер ця бізнес-логіка зберігається на рівні сервера програмної системи, і при її зміні клієнти одразу отримують можливість працювати за новими правилами;

– кожен рівень цієї архітектури виконує суворо обмежений набір функцій (які не повторюються від рівня до рівня) і не знає деталей реалізації та структур даних інших рівнів. Тому вміст кожного з рівнів можна змінювати без ризику глобальних конфліктів між рівнями та порушення коректної працездатності інших компонентів системи;

– висока гнучкість щодо конфігурації та розгортання програмної системи;

– для горизонтального масштабування програмної системи з такою архітектурою замість дублювання всієї системи в повному обсязі на декількох серверах або віртуальних машинах достатньо продублювати лише клієнтську її частину.

Недоліком такої архітектурної конструкції є необхідність в більш складному та затратному по часу проектуванні, в порівнянні з монолітною, та складніша її реалізація.

2.1.3. Мікросервісна архітектура

Мікросервісна архітектура являє собою вид сервіс-орієнтованої архітектури програмного забезпечення та є застосовною у побудові розподілених програмних систем. Основна ідея архітектури мікросервісів полягає в проектуванні та реалізації наскільки це можливо невеликих, слабко зв’язаних та легко змінюваних модулів – мікросервісів. Модулі в мікросервісній архітектурі взаємодіють через мережу, при цьому виконуючи єдину мету [6].

Архітектура працює за принципом компонентизації сервісів. Вона поділяє програмне забезпечення різні ізольовані компоненти, кожен із яких несе єдину відповідальність. Зміни в одному сервісі не повинні впливати на інші.

Мікросервісна архітектура складається із наступних компонентів: сервіси (Services), сервісна шина (Service Bus), зовнішня конфігурація (External configuration), шлюз API (API Gateway), контейнери (Containers).

Приклад реалізації такої архітектури представлено на рис. 2.3.

Переваги такої архітектурної конструкції:

– взаємозамінність мікросервісів та їх незалежність один від одного, що забезпечує слабку зв'язаність завдяки високому ступеню ізоляції та високу модульність;

– забезпечення високої гнучкості та масштабованості;

– можливість написання мікросервісів за допомогою будь-яких програмних засобів, найбільш підходящих для кожного конкретного модуля, при цьому вони добре розуміють один одного завдяки інтерфейсу;

– вирішення проблеми з потоками даних, яка іноді буває у багаторівневій архітектурі.

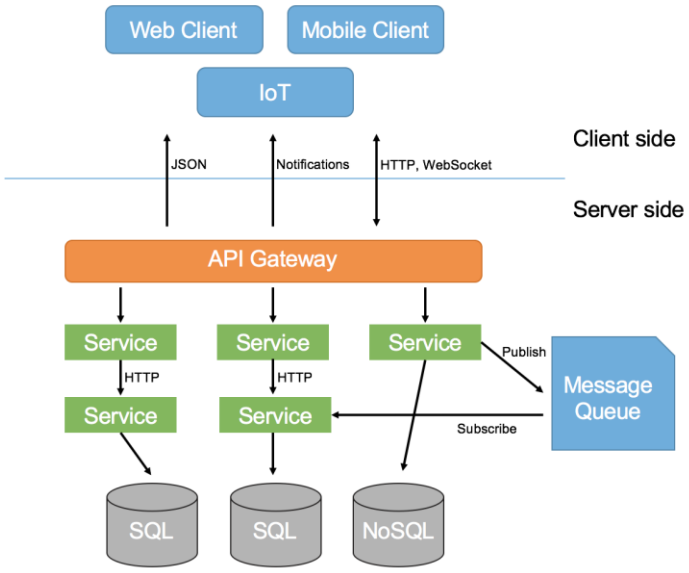

Рис. 2.3. Схема структури системи з мікросервісною архітектурою

Недоліками мікросервісної архітектури є підвищений ризик збою під час обміну даними між сервісами, складність керування великою кількістю сервісів, потреба в комплексному тестуванні в розподіленому середовищі та висока складність проектування та реалізації в загальному та зокрема реалізації інтерфейсу взаємодії мікросервісів [6]. На реалізацію такої архітектури потрібно значно більше часу в порівнянні з монолітною та багаторівневою.

Крім того, така архітектура вимагає вирішення таких проблем, як затримки в мережі, балансування навантаження та інші труднощі, властиві розподіленій архітектурі.

2.1.4. Вибір архітектурного рішення для розроблюваного програмного продукту

На основі наведеного порівняння архітектурних підходів та сформованих вимог до програмного продукту та його функціоналу, в якості архітектурного рішення для розроблюваного прототипу програмної системи було обрано трьохрівневу архітектуру зі схемою поділу системи на рівні (layers) та

розподіленням відповідальності та залежностей між ними, відображену на відповідній діаграмі (рис. 2.4).

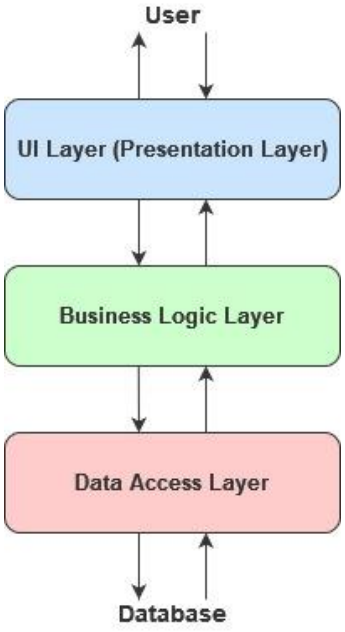

Рис. 2.4. Схема структури застосунку з трьохрівневою архітектурою

Такий вибір архітектурного рішення обумовлений тим, що така архітектура є найбільш вдалою у випадку розроблюваного дипломного проекту та якнайкраще забезпечить його відповідність поставленим нефункціональним вимогам в порівнянні з іншими архітектурними рішеннями.

В рівні користувацького інтерфейсу (UI Layer) знаходиться реалізація правил відображення даних користувача відповідно до його запитів та виконуваних ним дій в програмній системі, а також логіка яка є виключно логікою користувацького інтерфейсу. Наприклад, функціонал сортування та групування рядків та колонок в таблицях відображення даних (Data Grid View). Це рівень, з яким безпосередньо взаємодіє користувач. Цей рівень включає окрім компонентів інтерфейсу користувача, також і механізми отримання введення від користувача.

Рівень бізнес-логіки (Business Logic Layer) містить набір компонентів, які відповідають за обробку отриманих від рівня представлення (рівня користувацького інтерфейсу) даних, реалізує всю необхідну логіку програмної

системи, всі обчислення, взаємодіє з рівнем доступу до даних та передає рівню представлення результати обробки.

Рівень доступу до даних (Data Access Layer) зберігає моделі, що описують використовувані програмною системою сутності, також тут розміщуються специфічні класи для роботи з різними технологіями доступу до даних, наприклад клас контексту даних Entity Framework. Тут також зберігаються репозиторії, через які рівень бізнес-логіки взаємодіє з базою даних.

2.2. Огляд технологій для розробки компонентів системи

Основною метою, виходячи з означених вимог, розробки прототипу інформаційної системи є її подальше розширення та розвиток у повноцінну комплексну та повнофункціональну систему, з перспективою подальшого створення та розвитку різних версій однієї і тієї самої системи для різних платформ (веб та мобільних версій, наприклад). Для цього при розробці прототипу необхідним та найбільш важливим є закладення фундаменту для цього майбутнього розвитку шляхом забезпечення якомога більших показників гнучкості та розширюваності програмного продукту.

Також важливим є закладення належної основи для потенційного переносу програмної системи на інші апаратні та програмні платформи в майбутньому. Зі збільшенням кількості користувачів, навантаження на систему, об’єму даних, що в ній зберігаються, достатньо очікуваним є прийняття рішення для перенесення та розгортання програмної системи наприклад в хмарному середовищі, адже за останні роки стрімко зросла зацікавленість бізнесу у переході до використання розподілених та хмарних технологій. На сьогоднішній день вони вже стали одним з найефективніших інструментів для бізнесу. Однак отримання виграшу та дійсно суттєвих переваг від переходу до використання хмарних технологій у вигляді гнучкості та більш раціонального використання ресурсів може забезпечити лише розгортання системи побудованої з правильно підібраною архітектурою та використовуваними технологіями розробки.

При проектуванні такого рішення для прототипу програмної системи, яке б дало можливість реалізації всього описаного вище потенціалу виникають три наступні перешкоди.

Проблема 1: Різні платформи включають аналогічні способи представлення GUI і взаємодії користувача з системою, але є багато відмінностей в деталях. Кожна платформа має різні способи навігації навколо програм і сторінок, різних умов для представлення даних, тощо.

Проблема 2: Різні платформи засновані на різних операційних системах з різними API. У багатьох випадках, різні платформи реалізують подібні типи об’єктів інтерфейсу користувача, але кожен з них має різну назву. Природно, що такі відмінності не обмежуються іменами і проявляються і в самих програмних інтерфейсах.

Проблема 3: Розробники мають певну гнучкість у виборі мови програмування для кожної з цих трьох платформ, але, загалом, кожна платформа дуже тісно пов'язана з певною мовою програмування.

Ця «мовна» проблема є особливо неприємною, проте вона має своє рішення, і суть його в наступному: якщо є можливість використовувати одну мову програмування для різних платформ, то це дозволить поширити програмний код (принаймні, значну його частину) між платформами. Цей «спільний» код, ймовірно, не буде пов'язаний з інтерфейсом користувача, тому що кожна платформа має різні *API*, проте завдяки вибору на користь трьохрівневої архітектури частина такого коду від загальної кількості є максимальною.

Також наявність великої частини спільного між різними платформами програмного коду (який крім того, згідно обраному архітектурному рішенню, містить у собі реалізацію всієї бізнес-логіки програмного продукту) полегшує та пришвидшує розширення функціоналу, адже логіку нового функціоналу в такому випадку необхідно спроектувати та написати лише один раз для всіх підтримуваних платформ. Таким чином, вся описана вище проблема зводиться до питання вибору мови програмування, яка б надавала таку можливість.

2.2.1. Вибір мови програмування

На поставлене питання можна почути ряд відповідей, але однією із найпоширеніших з них буде C# [1]. 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 [7].

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 пристрою: 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.

2.2.2. Технології для доступу до даних

Для реалізації доступу до даних, створення моделі бази даних розроблюваного програмного продукту та роботи з нею однією з найбільш поширених та потужних технологій є Entity Framework.

В порівнянні з іншими ORM-інструментами Entity Framework набув особливо широкого використання через ряд своїх переваг над ними.

Entity Framework підтримує багато різних систем баз даних. Таким чином, надаючи можливість через EF працювати з будь-якою СУБД, якщо для неї є потрібний провайдер.

Також EF надає універсальний API для роботи з даними. І якщо, наприклад, виникне потреба змінити цільову СУБД, то основні зміни в проекті стосуватимуться насамперед конфігурації та налаштування підключення до відповідних провайдерів. А код, який безпосередньо працює з даними, отримує дані, додає їх до БД тощо, залишиться тим самим.

Як технологія доступу до даних Entity Framework можна використовувати на різних платформах стека .NET. Це і стандартні платформи типу Windows Forms, консольні програми, WPF, UWP та ASP.NET Core. При цьому кросплатформна природа EF дозволяє використовувати його в розробці програмних продуктів для всіх підтримуваних .NET платформ [14].

Відмінною рисою Entity Framework як технології ORM є використання запитів LINQ для вибірки даних з БД. LINQ є дуже потужним інструментом мови C#, за допомогою якого можна створювати різні запити на вибір об'єктів пов'язаних різними асоціативними зв'язками, в тому числі зі складними правилами їх фільтрації, групування та сортування. Entity Framework при виконанні запиту транслює вирази LINQ на вирази, зрозумілі для конкретної СУБД.

Враховуючи вищенаведені властивості Entity Framework, можна сказати, що

для вирішення поставлених для розроблюваного прототипу програмної системи задач та його відповідності поставленим вимогам в аспекті реалізації доступу до даних та роботи з ними, ця технологія підходить якнайкраще.

2.2.3. Технології для розробки інтерфейсу користувача

Для створення графічних інтерфейсів за допомогою платформи .NET застосовуються різні технології – WinForms, WPF, UWP. Однак найбільш простою і зручною технологією досі залишається WinForms.

Підхід до розробки прикладних програм на Windows Forms ґрунтується на графічному інтерфейсі GDI (Graphics Device Interface, Graphical Device Interface) - це інтерфейс для представлення графічних об'єктів та передачі їх на пристрої відображення [11]. WinForms надає можливості для створення графічних інтерфейсів, імплементації їх дизайну та інтерактивності.

Перевагами даної технології є передусім високий рівень пристосованості та широкі можливості для роботи з нею в відповідних редакторах сучасних IDE, достатньо повна документація по розробці та простота в створенні різноманітних елементів інтерфейсу користувача, а також в імплементації логіки представлення даних.

Windows Presentation Foundation (WPF) являє собою великий API-інтерфейс для створення десктопних графічних програмних інтерфейсів, головною особливість якого є можливість декларативного визначення графічного інтерфейсу за допомогою спеціальної мови розмітки XAML, що базується на xml і представляє альтернативу програмному створенню графіки та елементів управління [12].

В порівнянні з WinForms, WPF має більш широкі можливості для побудови дійсно складних та інтерактивних додатків, проте складність розробки на базі цієї технології є значно вищою.

Враховуючи наведене порівняння, для реалізації інтерфейсу користувача розроблюваного прототипу інформаційної системи краще підійде технологія WinForms, оскільки її можливості є достатніми для вирішення поставлених задач

розроблюваного продукту в аспекті інтерфейсу користувача в той час як безпосередньо процес розробки на базі цієї технології є значно простішим та швидшим.

2.3. Формування технічної специфікації програмного продукту

Заключним етапом проектування є формування технічної специфікації програмного продукту базуючись на результатах проведеного огляду технологій розробки різних компонентів програмної системи та їх порівняльного аналізу.

Технічна специфікація:

– архітектурна конструкція – багаторівнева архітектура зі схемою поділу на три рівні: рівень доступу до даних (Data Access Layer), рівень бізнес-логіки (Business Logic Layer) та рівень представлення (UI Layer);

– мова програмування – C#;

– технологія для реалізації доступу до даних та створення моделі бази даних (ORM) – Entity Framework;

– технологія для реалізації інтерфейсу користувача – WinForms;

– для підвищення логічності і послідовності структури програмного продукту рекомендується використовувати такі конструкції об’єктно-орієнтованих мов, як інтерфейси і абстрактні класи, делегати, події і їх обробники, патерни проектування тощо;

– СУБД – Microsoft SQL Server;

– хостинг – на серверах ОС Windows;

– система контролю версій – Git.

2.4. Висновки до розділу

В даному розділі було проведено та описано етапи проектування розроблюваного прототипу програмної системи, засновуючись на поставлені задачі для розробки та вимоги до програмного продукту. Перш за все було

проведено огляд і порівняння поширених архітектурних підходів, які вважаються гарними практиками в розробці інформаційних систем будь-якого призначення. На основі проведеного огляду і порівняльного аналізу, а також задач та вимог, які ставляться для розроблюваного прототипу інформаційної системи було обрано архітектурне рішення для розроблюваного продукту.

Після цього було розглянуто різні задачі та перешкоди, які виникають при розробці програмних систем зі схожими вимогами до розроблюваної та обгрунтовано залежність між цими перешкодами та проблемами вибору мови програмування та використовуваних технологій для розробки тих чи інших компонентів систем. На основі проведеного огляду та порівняльного аналізу різних технологій для розробки було обрано такі, які надають можливості створити програмний продукт, який буде найкращим чином відповідати поставленим задачам та вимогам.

Базуючись на отриманих результатах було сформовано технічну специфікацію розроблюваного програмного продукту, яка визначає вимоги безпосередньо до реалізації продукту та зв’язує їх з функціональними та нефункціональними вимогами та поставленими задачами, які були визначені та сформовані в розділі 1.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ ПРОТОТИПУ СИСТЕМИ

3.1. Поділ на функціональні підсистеми

Базуючись на поставленій задачі для розробки та функціональних вимог до розроблюваного програмного продукту було проведено поділ функціоналу прототипу інформаційної системи на логічні складові (підсистеми), які включають наступні:

1. Підсистема управління транзакціями.
2. Підсистема надання інформації про акції компаній представлені у списку доступних для купівлі.
3. Підсистема розрахунку та відображення характеристик та статистики портфеля користувача.
4. Підсистема моніторингу прибутковості компонентів інвестиційного портфеля.
5. Підсистема автоматизованих визначених користувачем стратегій.

Згідно обраної архітектурної конструкції розроблюваного програмного продукту було створено структуру проекту та файли його програмного коду. У відповідності до трьох рівнів обраної архітектурної моделі було створено три проекти – бібліотеки класів .NET Framework, названі IMSPrototype.DAL, IMSPrototype.BLL та IMSPrototype.UIL та встановлено посилання (зв'язки) між ними таким чином, як це описано на схемі архітектури (рис. 2.4).

Проект IMSPrototype.DAL є програмною реалізацією першого рівня обраної трьохрівневої архітектурної моделі – рівня доступу до даних (Data Access Layer – DAL) та містить в собі необхідні класи для роботи з базою даних, а також файли

конфігурації EntityFramework та міграцій.

Бібліотека класів IMSPrototype.BLL є реалізацією другого рівня трьохрівневої архітектурної моделі та містить в собі інтерфейси та реалізації безпосередньо бізнес-логіки розроблюваної системи, розділені на окремі класи-сервіси у відповідності з виділеними логічними підсистемами розроблюваного прототипу. Вона включає в себе імплементації алгоритмів розрахунку фінансових показників, виконання запитів до API при потребі в оновленні даних про компанії, ціни акцій тощо, реалізацію логіки дисеріалізації отриманих результатів запитів для забезпечення можливості подальшої роботи з ними з C#-коду, логіку для виконання необхідних перетворень даних, таких як фільтрація, сортування, групування, приведення (mapping) класів моделей сутностей отриманих з DAL в DTO (Data Transfer Objects), класи яких знаходяться окремій бібліотеці класів IMSPrototype.Common, для подальшої їх передачі в коректному вигляді на рівень логіки відображення – UIL).

Класи цього проекту взаємодіють як з рівнем DAL, реалізованим в IMSPrototype.DAL, отримуючи з нього дані необхідні для виконання бізнес-логіки через інтерфейс UnitOfWork, так і з рівнем інтерфейсу користувача UIL, виконуючи розрахунки та проводячи інші перетворення над даними за запитами з нього, після чого віддаючи результати цих обчислень/перетворень для подальшого їх відображення користувачеві рівнем UIL.

Проект IMSPrototype.UIL містить в собі класи Windows форм, в яких реалізовано логіку створення та відображення елементів інтерфейсу користувача, інфографіки та даних, а також механізми обробки дій користувача в системі та перенаправлення його запитів на отримання даних або проведення операцій/розрахунків на рівень бізнес-логіки. Класи цього рівня взаємодіють з класами рівня BLL, викликаючи їх методи та отримуючи результати з них.

Загальна схема взаємодії виділених підсистем на структурному рівні з урахуванням архітектурної конструкції наведена на рис. 3.1.

Описана вище структура програмного рішення розроблюваного продукту представлена на рис. 3.2.

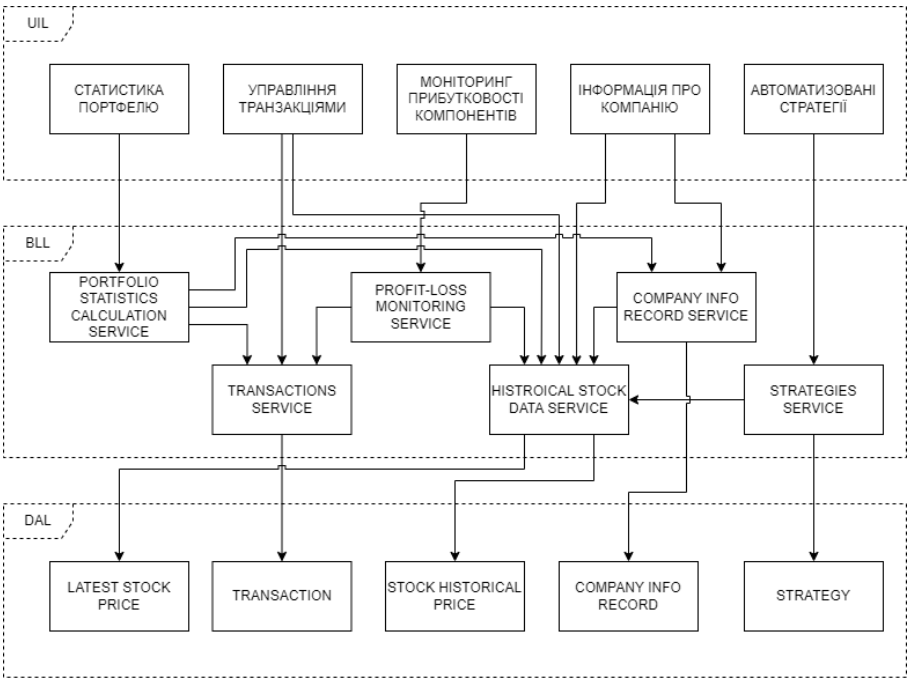

Рис. 3.1. Загальна схема взаємодії підсистем на структурному рівні з урахуванням архітектурної конструкції

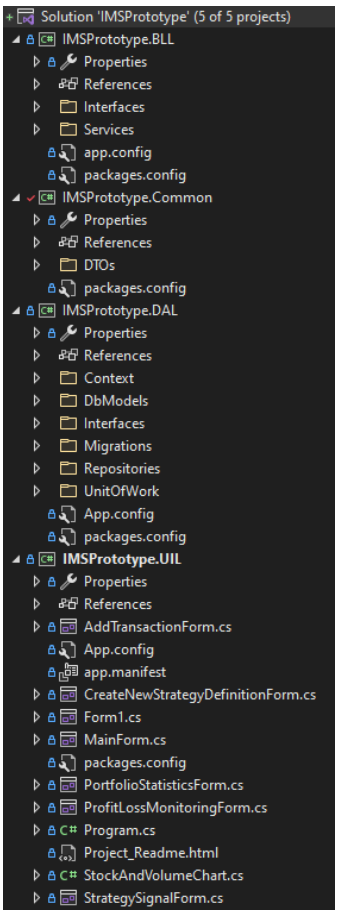

Рис. 3.2. Структура рішення розроблюваної програмної системи

3.2. Створення та налаштування механізмів роботи з базою даних

Згідно з результатами проектування, в бібліотеці класів IMSPrototype.DAL повинні знаходитись інструменти для доступу до даних. Це, зокрема, класи моделей (сутностей) програмної системи, нащадок класу DbContext, що зберігатає в собі набори сутностей з відповідних їм таблиць бази даних системи та надає можливість виконувати запити до неї, клас Configuration, який використовується ORM EntityFramework для налаштування, заповнення визначеним набором даних таблиць БД при її первинному створенні та подальших модифікацій схеми бази даних при внесенні змін до класів моделей сутностей в рамках підходу Code First за допомогою механізму міграцій.

Першочергово при розробці прототипу було створено набір сутностей, на основі яких будувалась безпосередньо сама схема БД (рис. 3.3)

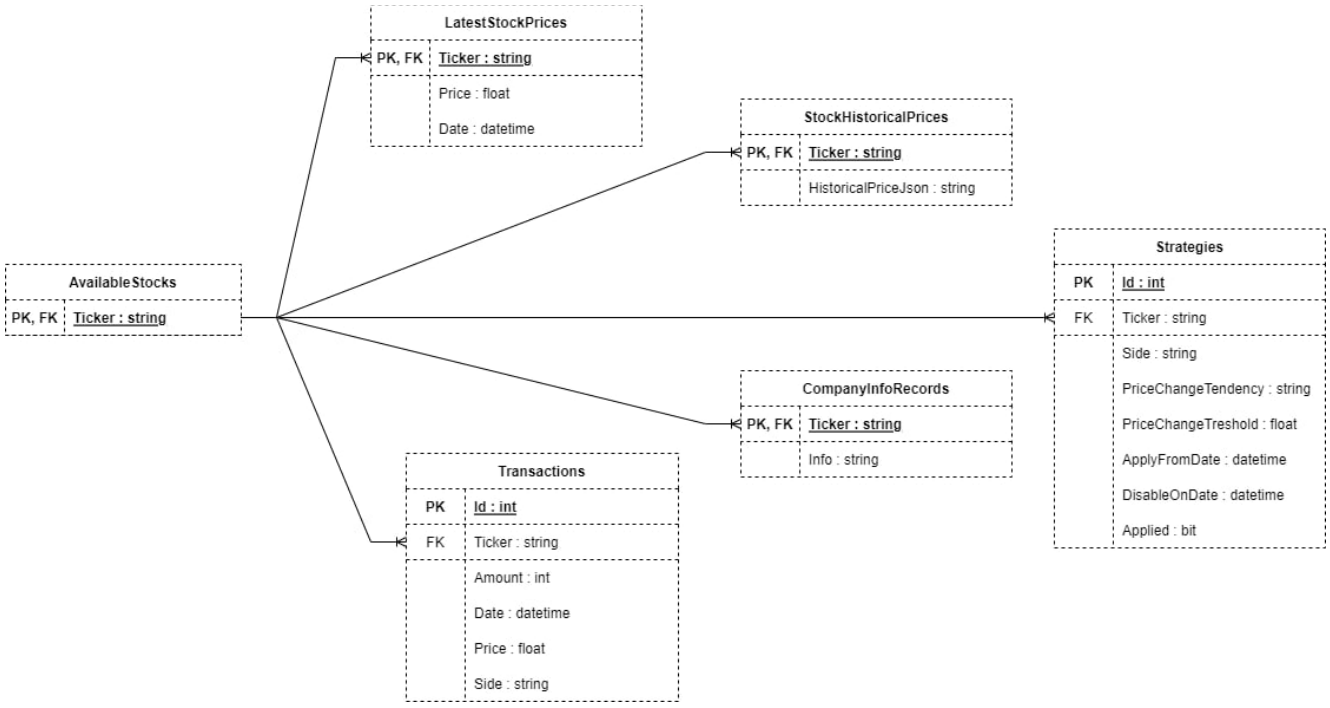

Рис. 3.3. Схема бази даних розроблюваного прототипу

Дані сутності зберігають всю необхідну для роботи прототипу інформацію, зокрема дані про транзакції, стратегії, компанії та історичну інформацію щодо зміни ціни їх акцій. Останні дві із перелічених сутностей виділяються тим, що в

значущому полі інформація зберігається у вигляді рядка, що являє собою об’єкт JSON, отриманий з API провайдера фінансової інформації. Так було вирішено для того, щоб упростити створювані сутності та полегшити процес зберігання та вилучення інформації. Наприклад, масив історичних даних про ціну акцій певної компанії може містити тисячі значень, що при роботі з інформацією про декілька компаній та довгим горизонтом оцінки може призвести до підвищеного часу очікування на запит користувача. Даний же підхід дає застосунку більше гнучкості та компактності даних.

Поступово по мірі створення моделей та внесення до них змін, накопичувався об’єм міграцій, що в своїй сукупності створює так званий «ланцюжок» із знімків конфігурації та схеми розроблюваної бази даних. Це дозволяє отримати лінійну (послідовну) модель всієї бази даних як набір декларацій про те, як і що саме змінювалось в БД з часом.

Над класами, що працюють безпосередньо з базою даних (StocksDBContext, класи моделей сутностей) було створено «обгортку», що включає реалізації так званих репозиторіїв (патерн Repository) та Unit Of Work. ЇЇ створення та взагалі використання вищеназваних патернів обумовлене перш за все тим, що Репозиторій дозволяє абстрагуватися від конкретних підключень до джерела даних, з якими працює програмна система, і є проміжним зв’язком між класами, безпосередньо взаємодіючи з даними та рештою програмного коду (в даному випадку – бізнес-логікою рівня BLL). Створення цього додаткового рівня абстракції зменшує ступінь зв’язності компонентів програмного коду, що в свою чергу робить процес їх потенційної заміни в майбутньому з подальшим розвитком програмного продукту значно легшим та швидшим процесом.

Unit Of Work, будучи доповненням до механізму інкапсуляції логіки роботи з джерелами даних, що реалізується репозиторіями, в свою чергу дозволяє значно спроситити роботу з різними репозиторіями, особливо якщо в системі багато сутностей та, відповідно, класів моделей (що є справедливим відносно розроблюваного прототипу і також є очікуваним збільшення кількості сутностей при його подальшому розширенні) і крім того дає впевненість, що всі репозиторії

будуть використовувати один і той самий контекст даних, що також є важливим аспектом для розроблюваного продукту.

В доповнення до перелічених вище класів, керуючись загальноприйнятими практиками та принципами розробки програмних систем, було вирішено дотриматись принципу інверсії залежностей. Згідно із ним, деталі повинні залежати від абстракцій; саме тому основний функціонал з класів UnitOfWork та Repository було представлено в інтерфейсах IUnitOfWork та IRepository, що являють собою додатковий рівень абстракції над імплементацією бізнес-логіки застосунку. Саме з екземплярами цих інтерфейсів і проводиться взаємодія при роботі з даними.

Діаграма залежностей між створеними класами та інтерфейсами наведена на рис. 3.4.

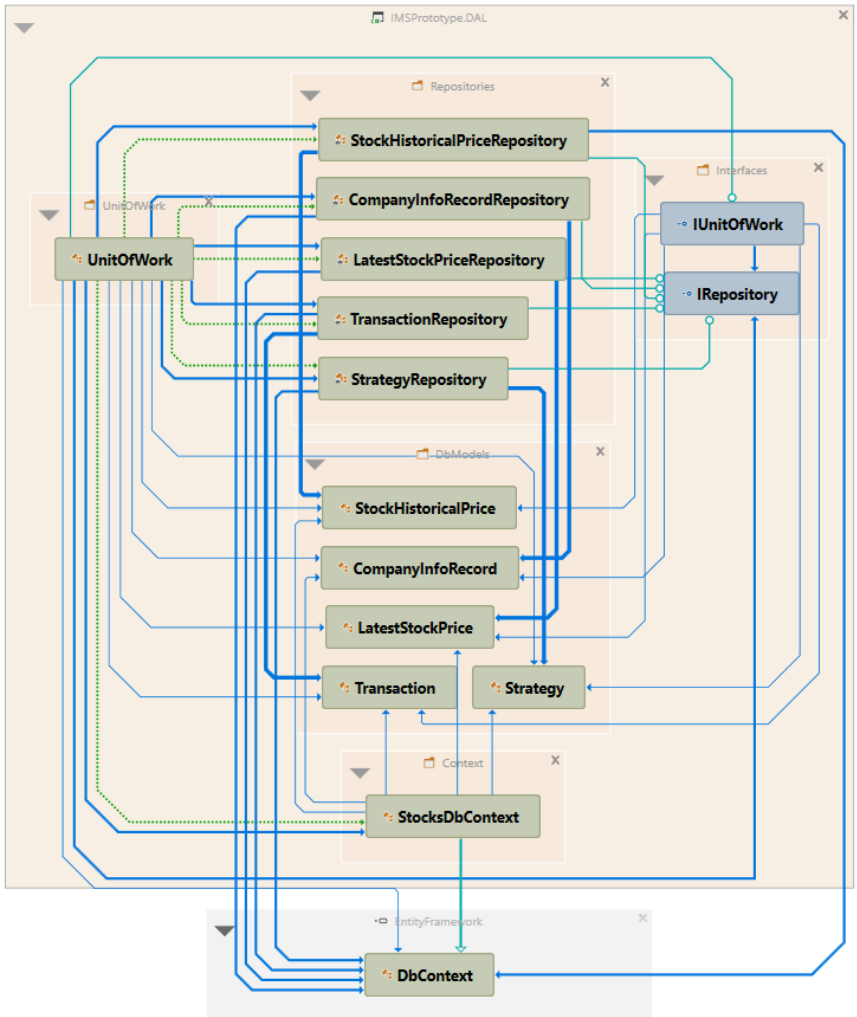

Рис. 3.4. Діаграма залежностей між класами, що реалізують моделі та механізм взаємодії з базою даних розроблюваної програмної системи

Таким чином, для створення та подальшої роботи з сутностями та схемою бази даних розроблюваної програмної системи було використано підхід Code First, основною ідеєю якого є генерація схеми бази даних на основі попередньо написаного програмного коду класів моделей сутностей програмної системи. В подальшому для внесення змін в схему БД достатнім є змінити модель (програмний клас) шляхом додавання або видалення полів цього класу, які власне і стануть в результаті полями таблиць бази даних. Всі зміни, зроблені в класах моделей відслідковуються технологією EF та можуть бути збережені та застосовані як набір атомарних процедур, що називаються міграціями.

В силу надання широких можливостей та гнучкості для полегшеного створення та подальших модифікацій схеми бази даних з програмного коду такий підхід найкращим чином підходить для нових та відносно невеликих програмних систем, які знаходяться на етапі первинної розробки. Саме тому використання цього підходу для розроблюваного прототипу є максимально виправданим.

3.3. Розробка функціональності прототипу та інтерфейсу користувачів

3.3.1. Реалізація підсистеми управління транзакціями

Програмна реалізація підсистеми управління транзакціями (Transactions Manager) включає два вікна користувацького інтерфейсу, одне з яких знаходиться на головній формі розробленого прототипу (файл програмного коду MainForm.cs) на відповідній вкладці та містить таблицю зі створеними користувачем транзакціями з можливістю їх групування та сортування за обраними колонками (рис. 3.5), а інше – вікно додавання нової транзакції, знаходиться на окремій формі (AddTransactionForm.cs), яка відкривається при натисканні кнопки «Add Transaction» (рис. 3.6).

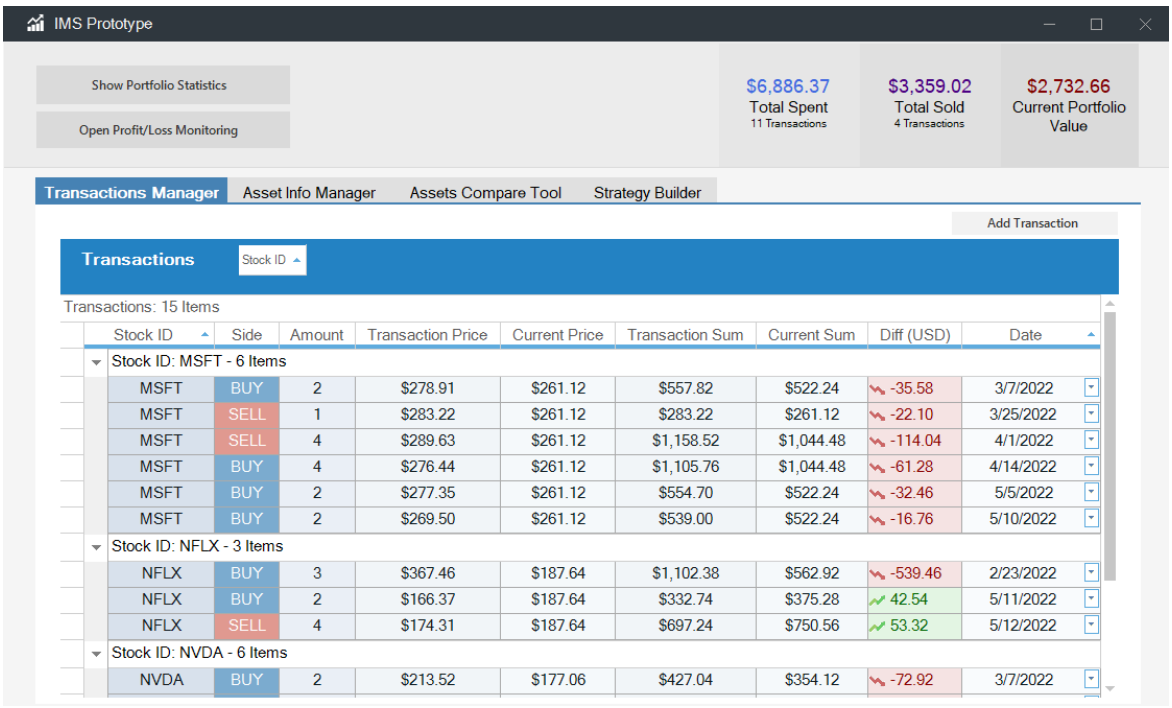

Рис. 3.5. Таблиця з транзакціями на головному вікні, на якій виконано групування по назві компанії та сортування дати транзакцій за зростанням в межах груп

Файли MainForm.cs та AddTransactionForm.cs бібліотеки класів IMSPrototype.UIL містять в собі логіку відображення елементів форми, таку як ініціалізація елементів, задання їм властивостей, які управляють кольорами, розмірами, положенням, вирівнюванням та поведінкою конкретного елемента на формі в залежності від тих чи інших дій користувача.

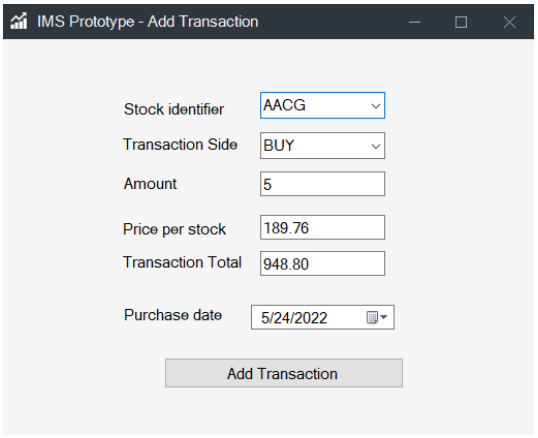

Рис. 3.6. Форма додавання нової транзакції

Вищезгадані форми працюють, як і всі інші форми розробленого додатку, по механізму подій та їх обробників. Будь яка зміна стану елементу форми має відповідну собі конструкцію в програмному коді – екземпляр .NET класу Event, який є властивістю об’єкта форми, та може мати обробник, який прикріплений до цієї події. Обробник події являє собою C# метод, який містить в собі деяку логіку та виконується тоді, коли виникає подія, до якої цей обробник прикріплений.

Класи двох форм підсистеми Transactions Manager містять приватні поля, що зберігають в собі посилання на об’єкти відповідного класу рівня бізнес-логіки – TransactionsService, викликом методів якого форма отримує дані та результати обчислень значень обчислюваних полів (наприклад поля Transaction Sum та Diff (USD)), які необхідно відобразити на UI. Також цей клас містить метод додавання нової транзакції до бази даних, який відпрацьовує після того як користувач заповнює поля форми додавання транзакції та натискає кнопку «Add Transaction».

3.3.2. Реалізація підсистеми надання інформації про акції компаній

Підсистема надання інформації про акції компаній складається з двох UI-компонентів – сторінок, які знаходяться на головній формі під вкладками Asset Info Manager (рис. 3.7) та Assets Compare Tool (рис. 3.8).

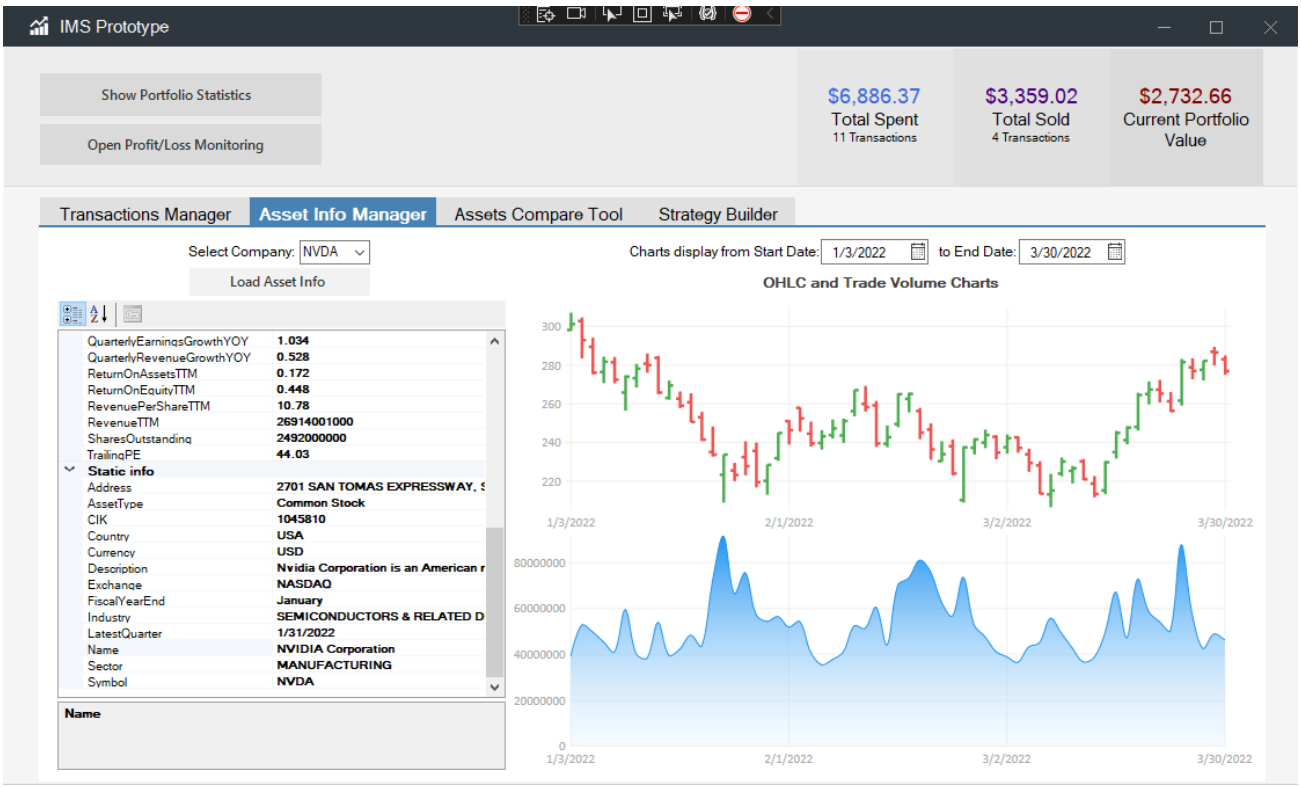

Рис. 3.7. Інтерфейс користувача функціоналу Asset Info Manager

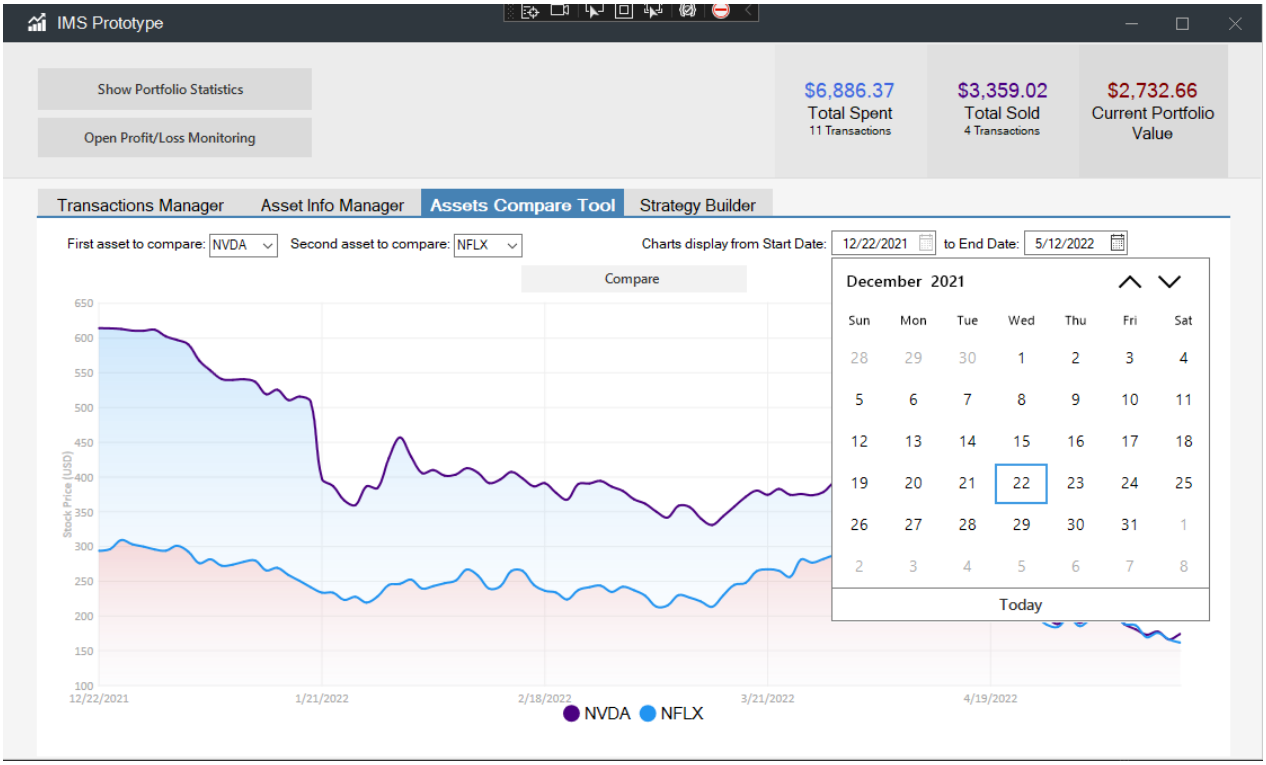

Рис. 3.8. Інтерфейс користувача функціоналу Assets Compare Tool

Функціонал Asset Info Manager надає користувачеві можливість отримання та перегляду інформації про будь-яку обрану ним компанію з визначеного переліку,

яка включає в себе такі характеристики та фінансові показники компанії, як сектор економіки, до якого вона відноситься, обчислені значення простого ковзного середнього (Moving Average) ціни акцій компанії за періоди в 50 та 200 діб, розмір дивідендів з її акцій, значення капіталізації, коефіцієнт «ціна-прибуток» (PE Ratio) та інші коефіцієнти та показники, які є необхідними для проведення користувачем технічного аналізу тієї чи іншої компанії-емітента.

Крім цього Asset Info Manager надає користувачеві графічне представлення зміни вартості акцій обраної компанії за визначений ним проміжок часу. Дане відображення включає в себе як значення зміни ціни день у день, так і впродовж проміжку часу з моменту відкриття торгів на біржі в конкретний день та до їх закриття (Open High Low Closed Chart). Крім OHLC графічно відображена також зміна за той самий проміжок часу, що і ціна, і значення об’єму торгів (Trade Volume), адже ці перелічені показники мають достатньо тісний зв’язок між собою в термінах проведення фінансового аналізу, розглядаються здебільшого в зв’язці один з одним, тому важливим є надання користувачеві зручного, наглядного та суміщеного представлення зміни з часом цих двох показників.

Assets Compare Tool являє собою інструмент для порівняння відображених графічно Цей інструмент є важливим перш за все тому, що за допомогою нього користувач матиме можливість спостерігати та знаходити кореляції між тенденціями зміни цін різних компаній. Дослідження кореляцій грає дуже важливу роль в фінансовому інвестиційному аналізі загалом і зокрема в процесі прийняття інвестиційних рішень про вибір компаній для інвестування на тому чи іншому етапі, а також подальшого перегляду та балансування інвестиційного портфеля з часом.

Програмна реалізація логіки функціонування цієї підсистеми знаходиться в класах CompanyInfoRecordsService та HistoricalStockDataService, які у відповідних методах асинхронно (в окремих логічних потоках виконання, не блокуючи при цьому головний потік, в якому працює логіка інтерфейсу користувача) виконують запити до API за назвою компанії та за значенням початку та кінця часового інтервалу, які задав користувач, отримують результати запитів, десеріалізують їх,

формують відповідні колекції вже екземплярів C#-класів та повертають результат для відображення на UI.

3.3.3. Реалізація підсистеми відображення статистики портфеля

Підсистема відображення статистики портфеля включає сторінку відображення інфографіки розрахованих значень показників, які з різних сторін характеризують інвестиційний портфель цінних паперів користувача та ілюстрацію зміни деяких з цих показників з часом (форма Portfolio Statistics) (рис. 3.9), а також три розраховані значення – загальну кількість інвестованого в акції капіталу, загальну суму з усіх продажів цінних паперів і поточне значення собівартості наявних в портфелі (не проданих) акцій, які закріплені на стрічці вгорі головної форми (Total Spent, Total Sold і Current Portfolio Value).

Portfolio Statistics містить GeoHeatMap, який показує розподілення кількості компаній у відсотках, акції яких наявні в портфелі користувача по країнах в яких ці компанії ведуть свою діяльність (Portfolio Holdings by Countries). Два графіка типу PieChart ілюструють розподілення загального інвестованого на поточний момент капіталу між всіма присутніми в портфелі компаніями (Portfolio Assets Allocation) та між секторами економіки (Portfolio Stocks’ Sectors), таким чином графічно ілюструючи баланс капіталу між компонентами портфеля та дозволяючи зробити висновок про ступінь диверсифікації інвестицій користувача.

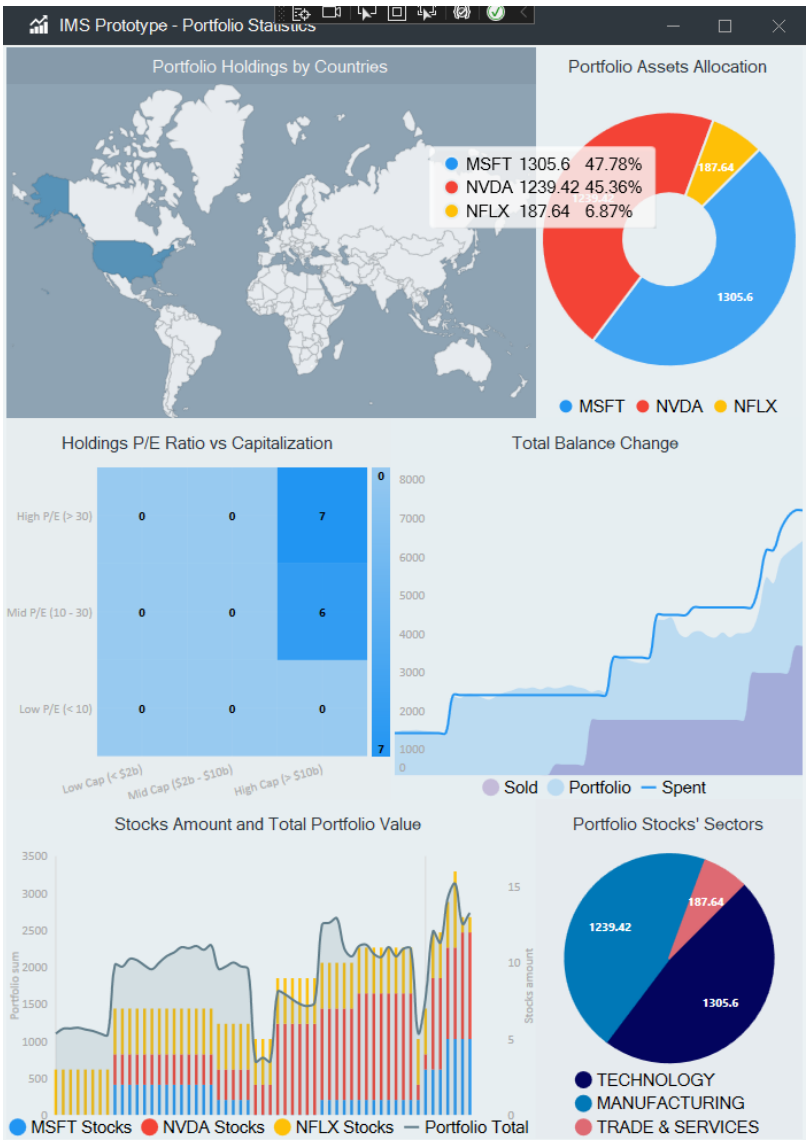

Рис. 3.9. Інтерфейс підсистеми відображення статистики портфеля

3.3.4. Реалізація підсистеми моніторингу прибутковості

Реалізація підсистеми моніторингу прибутковості (форма Profit/Loss Monitoring) направлена на розрахунок та відображення з моменту найпершої покупки акцій вибраної користувачем компанії з наявних в його портфелі і до теперішнього моменту часу зміни показника прибутковості, яке розраховується за наступним алгоритмом.

Перш за все для кожного дня з цього проміжку обчислюється середнє значення інвестованого капіталу в акції обраної компанії, які є наявними в портфелі станом на цей конкретний день. При цьому в розрахунках використовуються саме

ті значення ціни кожної акції, якими вони були на момент покупки кожної з них. Для цього в методі реалізуючому цей алгоритм робиться вибірка всіх наявних транзакцій по цій компанії, далі відбувається ітеративний попарний перебір всіх транзакції продажу, обчислення фактичної кількості наявних в портфелі акцій компанії з моменту першої транзакції (з поточної пари на поточній ітерації) до наступної та середньої ціни кожної акції. Таким чином на виході алгоритму отримується набір значень середнього (зваженого по кількісній частці різновартісних, куплених в різні дні, акцій) значення інвестованого капіталу.

З цього набору формується та відображається графічна залежність (червона лінія на графіку зліва (рис. 3.10)), яка накладається на графік зміни ціни цієї ж акції на фондовій біржі. Далі обчислюється власне значення прибутковості як різниця між поточною ціною акції на біржі та обчисленим на попередньому кроці середньозваженим значенням інвестованого капіталу та відображається на відповідному графіку (графік справа на рис. 3.10).

Рис. 3.10. Вікно Profit/Loss Monitoring

3.3.5. Реалізація підсистеми автоматизованих стратегій

Функціонал цієї підсистеми надає можливість створення певних визначених користувачем преференцій (стратегій), а також можливість застосувати та вимкнути будь-яку зі створених ним стратегій в будь-який час (рис. 3.11 та 3.12).

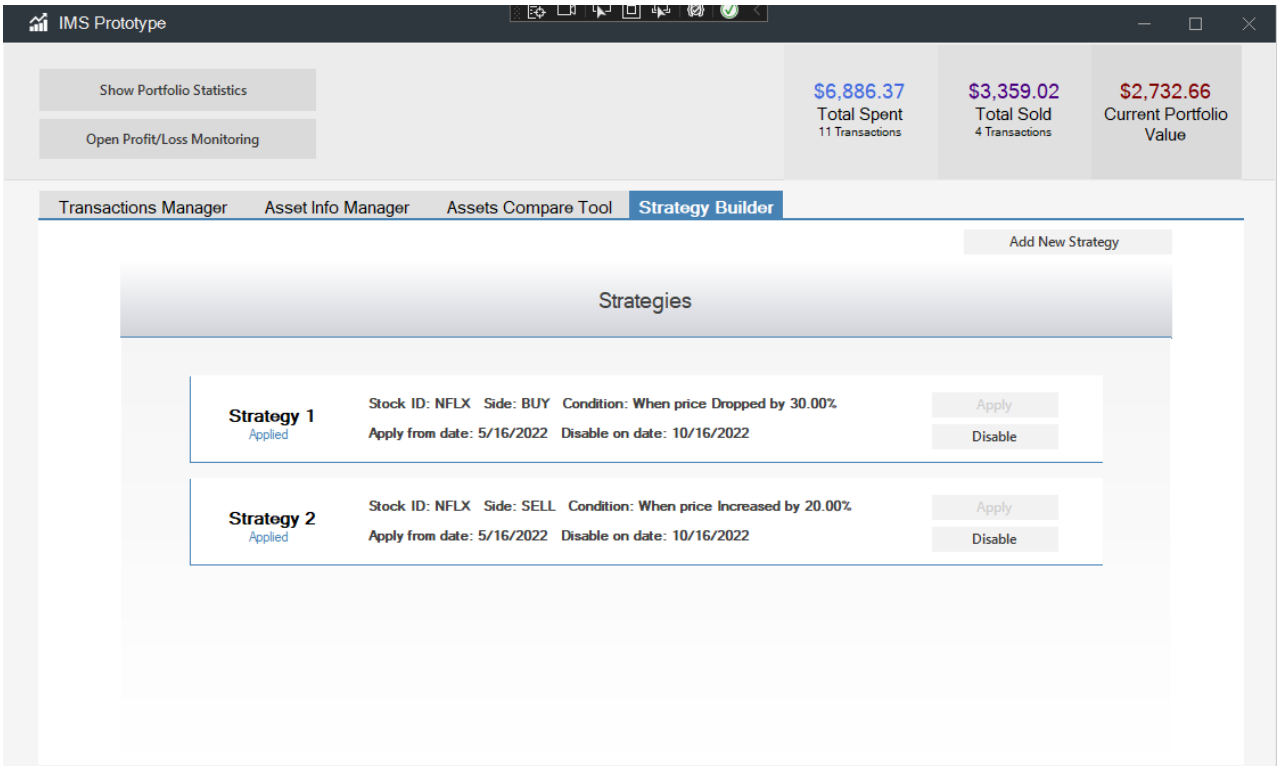

Рис. 3.11. Сторінка відображення створених користувачем стратегій

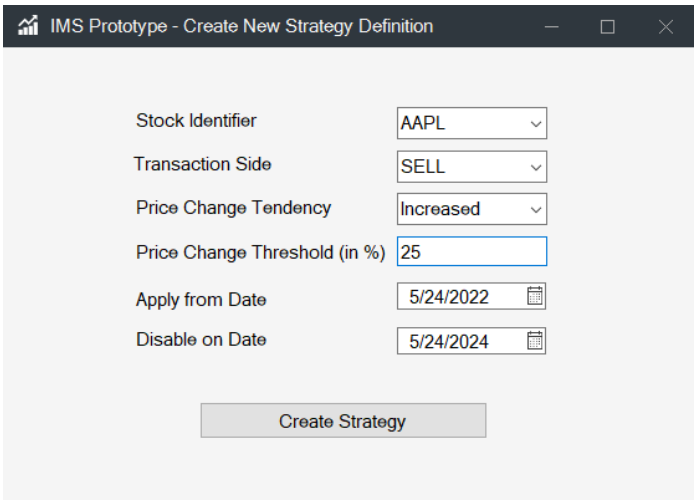

Рис. 3.12. Форма створення нової стратегії

Кожна створена та застосована користувачем стратегія має визначений керований параметр (найбільш використовуваним є значення ціни активу, тому в розроблюваному прототипі реалізована підтримка функціоналом стратегій саме цього параметру) і, якщо вона застосована користувачем, виконує слідкування за значенням цього керованого параметру. Крім цього стратегія має в собі певну визначену користувачем умову щодо тенденції зміни параметру (наприклад, ціна акції зростає або спадає) та граничного значення параметру, по досягненню якого подається відповідний сигнал (відкривається форма Strategy Signal Notification) користувачеві та надається можливість автоматичного створення транзакції на покупку або продаж визначеної ним кількості акцій на базі стратегії.

Програмно даний функціонал реалізовано за допомогою механізму подій та обробників. В кожній стратегії реалізована відповідна подія та прикріплений до неї метод-обробник. При кожному оновлені значень ціни акцій компаній відбувається прохід по колекції всіх створених та ввімкнених користувачем стратегій та перевіряється умова та граничне значення керованого параметру стратегії (ціни акції компанії відносно якої стратегію було створено) і у випадку якщо поточне значення ціни співпадає із заданим в стратегії, спрацьовує згадана раніше подія, а метод-обробник цієї події виводить користувачеві форму сповіщення з інформацією про те, що умова стратегії спрацювала, та пропонується створити транзакцію на базі стратегії (рис. 3.13).

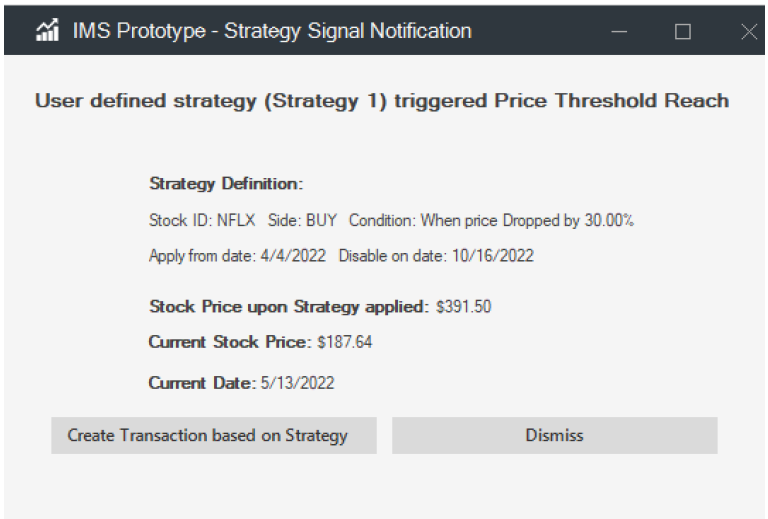

Рис. 3.13. Форма з повідомленням про виконання умови стратегії

3.4. Висновки до розділу

В даному розділі було відповідним чином проаналізовано завдання для розробки, на базі якого, враховуючи також функціональні вимоги до розроблюваного продукту, було проведено поділ функціоналу прототипу інформаційної системи на логічні складові (підсистеми) та встановлено перелік необхідних елементів, що повинні бути в ньому представлені. Було розроблено структуру файлів розроблюваного рішення, яка забезпечить виконання поставлених цілей та реалізацію обраної архітектури програмної системи, що також дозволило зробити процес розробки більш гнучким та організованим. Описано перелік класів, на яких будується логіка роботи додатку, представлено їх взаємозв’язок та структуру, а також реалізацію механізмів роботи з базою даних прототипу та її схему. Крім цього було наведено опис та деталі імплементації переліку функціоналу прототипу, який було розроблено. Результати роботи готового прототипу інформаційної системи представлено на рисунках.

ВИСНОВКИ

В рамках дипломного проекту було досліджено засоби аналізу фінансових та інвестиційних показників та їх реалізації в існуючих системах управління персональним портфелем цінних паперів, проведено порівняльний аналіз існуючих систем та виявлених в них недоліків.

Було розглянуто шаблони та підходи до проектування архітектури програмних систем, технології для реалізації механізмів налаштування та взаємодії розроблюваного програмного продукту з базою даних, а також програмних механізмів модифікацій її схеми в процесі розробки та зміни вимог до функціональності та сутностей програмної системи. Було досліджено мови програмування та технології створення інтерфейсу користувача, які використовуються для розробки зазначених інформаційних систем.

На основі аналізу було обрано найбільш підходящі засоби, підходи та технології для реалізації поставленої задачі для розробки дипломного проекту, а також забезпечення його відповідності функціональним та нефункціональним вимогам, визначеним для нього виходячи з потреб цільової аудиторії потенційних користувачів та недоліків існуючих рішень. Цими засобами стали технології платформи .NET для імплементації функціональної частини розроблюваного прототипу, Entity Framework – для реалізації механізмів роботи та модифікацій сутностей та схеми бази даних, та WinForms – для створення елементів та логіки інтерфейсу користувача.

Результатом виконання дипломного проекту є програмний продукт – прототип інформаційної системи управління та обліку персонального портфелю цінних паперів для моделювання процесів інвестування з метою спостереження ефекту від використання різних підходів до процесу інвестування на практиці. Прототип має сприятливу інфраструктуру для його подальшого розвитку, модифікацій (за потребою) та розширення, характеризуючись високими показниками розширюваності, переносимості, гнучкості та масштабованості з точки зору як архітектури, так і безпосередньої технічної реалізації.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Троелсен, Эндрю, Джепикс, Филипп. Язык программирования C# 7 и платформы .NET и .NET Core / пер. с англ. Ю. Артеменко — СПб. : ООО «Диалектика», 2018. — 1328 с.

2. Мартін, Роберт. Чистий код: створення і рефакторинг за допомогою Agile / пер. з англ. І. Бондаренко-Терещенко. — Харків: Вид-во «Ранок»: Фабула, 2021. — 448 с.

3. Richter, Jeffrey CLR via C#. Fourth edition / Jeffrey Richter — Redmond, Washington: Microsoft Press, 2012. — 896 p.

4. Sharpe, William F., Alexander, Gordon J., Bailey, Jeffery V. Investments, Fifth edition / William F. Sharpe — Englewood Cliffs, New Jersey: Prentice Hall International, Inc., 1995. — 1058 p.

5. Дамодаран А. Инвестиционная оценка. Инструменты и методы оценки любых активов / А. Дамодаран — «Альпина Диджитал», 2008. — 1338 с.

6. Baptista, G. and Abbruzzese, F. Software Architecture with C# 9 and .NET 5: Architecting software solutions using microservices, DevOps, and design patterns, 2nd Edition. 2nd edn. / G. Baptista, F. Abbruzzese — Birmingham: Packt Publishing, 2020. — 586 p.

7. Price, M. J. C# 9 and .NET 5 - modern cross-platform development - fifth edition: Build intelligent apps, websites, and services with Blazor, ASP.NET Core, and Entity Framework Core using Visual Studio Code. 5th edn. / M. J. Price — Birmingham: Packt Publishing, 2020. — 822 p.

8. Morningstar | Empowering Investor Success [Електронний ресурс]. Режим доступу: https://www.morningstar.com/ (дата звернення 26.05.2022) — Назва з екрана.

9. Google Finance – Stock Market Prices, Real-time Quotes & Business News [Електронний ресурс]. Режим доступу: https://www.google.com/finance/?hl=en (дата звернення 26.05.2022) — Назва з екрана.

10. Yahoo Finance – Stock Market Live, Quotes, Business & Finance News [Електронний ресурс]. Режим доступу: https://finance.yahoo.com/ (дата звернення 26.05.2022) — Назва з екрана.

11. WPF vs. WinForms [Електронний ресурс]. Режим доступу: https://wpf-tutorial.com/ (дата звернення 27.05.2022) — Назва з екрана.

12. Windows Forms vs WPF [Електронний ресурс]. Режим доступу: http://sonyks2007.blogspot.com/2013/12/windows-forms-vs-wpf.html (дата звернення 28.05.2022) — Назва з екрана.

13. Entity Framework documentation | Microsoft Docs [Електронний ресурс]. Режим доступу: https://docs.microsoft.com/en-us/ef/ (дата звернення 28.05.2022) — Назва з екрана.

14. Martin, Robert C. Clean Architecture: A Craftsman’s Guide to Software Structure and Design / Robert C. Martin — Pearson, 2017. — 432 p.

15. Richards, Mark Software Architecture Patterns / Mark Richards — O'Reilly Media, Inc., 2015. — 54 p.

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);
            }
```

```
            }
            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; }
    }
}
```

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(transactionSidesSource);
         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();
      }
   }
}
```