Вінницький національний технічний університет
Факультет комп'ютерних систем і автоматики
Кафедра автоматизації та інтелектуальних інформаційних технологій

# Пояснювальна записка

до магістерської кваліфікаційної роботи

на тему: «Розробка клієнт-серверної системи для прогнозування поведінки фінансових інструментів(Серверна частина)»

Виконав: студент групи 1ІСТ-21м
Спеціальність: 126 – Інформаційні системи та технології.
Курніцький Д.П.
Керівник: к.т.н., доц. каф. АІІТ
Коцюбинський В. Ю.
Рецензент к.т.н., доц. каф. КСУ
Биков М.М.

Вінниця ВНТУ – 2022 рік

# АНОТАЦІЯ

В даній роботі розглянуто та проаналізовано метод розробки серверної частини клієнт-серверної системи для прогнозування фінансових інструментів мовою програмування C# на основі платформи .NET6 та за допомогою фреймворка EntityFramework

На основі отриманих результатів спроектовано архітектуру серверної частини клієнт-серверної системи для прогнозування фінансових інструментів

Розглянута розробка основних модулів системи, таких як бізнес-логіка роботи з інвестиційними портфелями, система логування дій користувачів. Розроблені UML діаграми цих систем.

# ANNOTATION

This work considers and analyzes the method of developing the server part of the client-server system for forecasting financial instruments in the C# programming language based on the .NET6 platform and using the EntityFramework framework

Based on the obtained results, the architecture of the server part of the client-server system for forecasting financial instruments was designed

The development of the main modules of the system, such as business logic for working with investment portfolios, a system for logging user actions, is considered. UML Diagrams and charts of these systems have been developed.

# ЗМІСТ

# ВСТУП

Актуальність. Останнім часом, в період стрімкого розвитку фінансових технологій, надзвичайно актуальним є питання оцінки фінансових ризиків, та засобів, інструментів для її здійснення. Одним із найефективніших інструментів, які допомагають здійснити оцінку фінансової установи є спеціалізовані системи для передбачення поведінки фінансових активів, на основі якої, можна здійснити оцінку.

Поширення подібних систем для передбачення поведінки фінансових активів з роками буде лише зростати через різкий стрибок у розвитку новітніх фінансових технологій. Темпи користування такими системами в наш час стрімко зростають, адже фінансові установи бажають спростити процес оцінки фінансових ризиків.

Мета макроекономічного аналізу не обмежується тільки абстрактним відображенням реальних явищ і процесів в економіці, оскільки його призначення полягає і у визначенні закономірностей, їх формування і розвитку на основі макроекономічного моделювання. Саме на базі макроекономічного моделювання можна отримати досить повне уявлення про сутність тих чи інших макроекономічних явищ і процесів, скласти прогноз їх розвитку, оцінити можливості впливу на дані явища і події, обґрунтувати рекомендації для макроекономічної політики з їх нейтралізації або прискореному розвитку.

В теперішній час новим та перспективним напрямком проведення сценарного аналізу на фінансових ринках є розробка узагальненого підходу для проведення моделювання з використанням апарату регресійного аналізу. Можна виділити наступні переваги використання узагальненого підходу до моделювання: при великих об’ємах інформації, система з чіткою логікою виконує задачі аналізу набагато швидше, та дозволяє змоделювати велику кількість сценаріїв і надає можливість, користувачеві аналізувати найкращі та найгірші варіанти розвитку. Враховуючи актуальність використання узагальненого підходу до процесу моделювання в контексті даної досліджуваної роботи є практична необхідність

однозначного та чіткого опису всіх кроків виконання сценарного стрес-тестування та вхідних даних, що підвищить ефективність розробленої методології.

Мета дослідження. Метою даної роботи є підвищення ефективності прогнозування поведінки фінансових інструментів, а саме портфелів цінних паперів.

Об’єкт дослідження - процес прогнозування стану портфелю та цінних паперів з використанням сценарного аналізу та моделювання впливу макроекономічних факторів.

Предмет дослідження – методи збирання даних для передбачення, методи прогнозування стану портфелю цінних паперів в умовах впливу макроекономічних факторів, та обробки прогнозованих даних.

Методи дослідження - метод статистичної обробки даних, методи прикладної математики, методи збору та фільтрації цифрових даних, метод макроекономічного моделювання.

Задачі дослідження. Для досягнення поставленої мети необхідно розв’язати такі задачі:

Аналіз застосування технологій автоматичного збору та фільтрації історичних даних для подальшого використання в прогнозуванні.

Огляд та аналіз існуючих систем для проведення сценарного аналізу на фінансових ринках за допомогою макроекономічного моделювання.

Розроблення методів збирання інформації про користувачів, з використанням методів фінансової валідації;

Розроблення надійної та гнучкої архітектури системи;

Практичне значення одержаних результатів. Дослідження виконані у даній роботі, дозволяють розробити автоматизовану систему для збору вхідних даних та проведення сценарного аналізу зі знаходженням найбільш ефективної математичної моделі для кожної із вхідних змінних. З практичної точки зору, розроблена система дозволить підвищити якість сценарного аналізу з використанням

макроекономічного моделювання та автоматизувати всі процеси збору даних для уникнення помилок спричинених людським фактором.

Апробація результатів роботи. Отримані результати впроваджені та прийняті для використання в виробничих процесах на НВП ТОВ «Спільна Справа».

# 1 ТЕХНІКО-ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ ДОЦІЛЬНОСТІ РОЗРОБКИ СИСТЕМИ АВТОМАТИЗАЦІЇ СЦЕНАРНОГО АНАЛІЗУ

## 1.1 Суть технічної проблеми, що виникла

Суть технічної проблеми полягає у відсутності розподілених систем для автоматизації проведення прогнозування портфелю цінних паперів, шляхом ідентифікації впливу різних макроекономічних факторів, які могли б швидко виконати аналіз економічного середовища, та провести сценарне стрес-тестування з використанням апарату регресійного аналізу.

Враховуючи те, що фінансові ринки активно розвиваються, мають властивість постійно змінюватись і несуть великі ризики для інвесторів цінних паперів, з’являється необхідність прогнозування можливих нестандартних подій та застосування відповідних стратегій управління для уникнення негативних варіантів розвитку чи можливих фінансових втрат.

На сьогоднішній день існує великий спектр програмних продуктів, які вирішують задачі з проведення сценарного стрес-тестування портфелів цінних паперів, але вони мають обмежені функціональні можливості, та необхідність ручного збору вхідних даних для проведення моделювання. Основним недоліком існуючих систем є відсутність можливості вибору методу аналізу, та порівняння результатів отриманих різними методами для оцінки найбільш оптимального методу реалізації чисельної регресії для даного конкретного рішення.

В загальному метою технічної проблеми є створення програмного продукту, кращого за існуючі аналоги за своїми техніко-економічними показниками.

## 1.2 Огляд сучасних аналогів

На сьогоднішній день існує велика кількість програмних продуктів, які виконують сценарне стрес-тестування. Стислу порівняльну характеристику аналогів наведено в таблиці 1.1.

Таблиця 1.1 – Стислі характеристики існуючих систем

| Параметр | Аналоги розробки | | | Розроблюваний продукт |
|---|---|---|---|---|
| | IBM Predictive-Market-Stress-Testing | Check Risk | Black Rock | |
| 1 | 2 | 3 | 4 | 5 |
| Ліцензія | Безкоштовна | Комерційна | Комерційна | Комерційна |
| Кросплатформенність | + | - | + | + |
| Вибір методу проведення аналізу | - | - | - | + |
| Інтерфейс незалежний від інших програмних продуктів | + | + | + | + |
| Порівняльна характеристика отриманих результатів | - | - | - | + |

| | | | | |
|---|---|---|---|---|
| Автоматичний збір даних для проведення аналізу | - | + | + | + |

1.2.1 Короткий опис аналогів

IBM Predictive-Market-Stress-Testing – система для автоматизації стрес-тестування фінансових портфелів. Даний продукт використовує служби IBM cloud для запуску веб-додатка, який проводить стрес-тестування, в результаті роботи додатка, користувач отримує csv файл з розрахованими факторами ризику та розмірами можливих втрат [1].

CheckRisk – система для проведення стрес-тестування з використанням сценаріїв, на основі існуючих історичних даних різних варіантів розвитку подій. Використовує байєсівські мережі для моделювання макроекономічних факторів та формування можливих факторів ризику.

BlackRock - система для автоматизації процесу стрес-тестування, яка дозволяє виявляти більше 30 можливих факторів ризику та надає користувачеві доступ до історичних даних основних макроекономічних показників.

1.2.2 Обґрунтування актуальності аналогу

З поданої вище таблиці 1.1 можна зробити певні висновки по кожному програмному продукту, зазначеному в ньому.

IBM Predictive-Market-Stress-Testing – має хороші характеристики, основними перевагами даного продукту є те, що рішення для проведення аналізу влаштовані як веб-додаток, який розміщується на зовнішніх розподілених серверах та дозволяє

проводити масштабні розрахунки без необхідності використання фізичних ресурсів комп'ютера користувача. Основними недоліками даної системи є: залежність від сервісу IBM Cloud, необхідність ручного збору та завантаження вхідних даних для проведення сценарного аналізу.

CheckRisk – комплексне рішення, яке надає користувачеві можливість проведення сценарного стрес-тестування без необхідності збору та обробки вхідних даних. Основними недоліками даної системи є відсутність можливості управління процесом проведення стрес-тестування, дана система є повністю закритою, та дає користувачеві доступ тільки до результатів моделювання.

BlackRock – дана система пропонує користувачеві стандартний набір факторів ризику, та не надає можливостей гнучкого налаштування системи для кожного конкретного випадку.

У всіх розглянутих аналогах програмних продуктів відсутні методи для визначення похибок отриманих результатів та можливості керування методами аналізу.

У даній роботі використовується алгоритм оцінки отриманих результатів прогнозування різними методами реалізації чисельної регресії, що дозволяє застосувати до кожної змінної оптимальний метод аналізу відповідно до показників похибок отриманих під час прогнозування вже існуючих історичних показників. Окрім того використання клієнт-серверного підходу та розподілених хмарних сервісів для проведення розрахунків, дозволяє користувачеві мати доступ до системи з будь-якого пристрою, та виключає необхідність великої кількості обчислювальних ресурсів для проведення сценарного прогнозування незалежно від кількості вхідних даних.

Отож, нова розробка має кращі характеристики в порівнянні з її аналогами, а саме автоматичний збір та фільтрацію вхідних даних, вибір горизонту стрес-тестування, різноманітні засоби візуалізації результатів, оцінку похибок моделювання на основі історичних даних..

### 1.2.3 Основна задача, що вирішується в роботі

Основною задачею даної роботи є підвищення ефективності збору інформації для передбачення, прогнозування портфелю цінних паперів, шляхом ідентифікації впливу різних макроекономічних факторів, розробка серверної частини продукту, який має переваги над існуючими системами, та дозволяє покращити якість результатів, що отримуються.

### 1.2.4 Використання наукового підходу для аналізу історичних даних

Одною з найважливіших задач прогнозування є перевірка дійсності отриманих результатів та розрахунок ймовірності того, що система буде давати коректні данні при аналізі на реальних вибірках часових рядів. Хоча тестування на даних поза межами вибірки може до деякої міри показати, чи витримає система іспит більш новими даними, за допомогою наукових методів можна одержати додаткову інформацію. Велику користь з наукових методів приносить статистика. Вона дозволяє визначити, чи результат випадковий чи він заснований на реальних перевагах системи. Статистичні розрахунки можуть бути використані для виявлення підгонки під історичні дані, тобто можуть визначити, чи є ефективність моделі, що спостерігається реальною чи ж вона — результат підгонки [1].

Слід зазначити, що в статистиці, як правило, робляться деякі теоретичні припущення щодо зразків даних і вибірок, до яких можна адекватно застосовувати статистичні методи. При роботі з торговими системами ці правила частково приходиться порушувати, причому деякі порушення правил не мають ніякий практичної цінності, тоді як більш важливі правила часто вдається обійти без

компромісу. При використанні додаткового аналізу часом удається обійти чи компенсувати навіть дуже важкі невідповідності даних вимогам статистичного аналізу.

Загалом, статистика може допомогти користувачеві перевірити з якою ймовірністю система буде працювати правильно в майбутньому, на основі порівняння зі статистичними даними .

Серед методів статистичного аналізу, найбільш корисним можна назвати перевірку по критерію Стьюдента, кореляційний аналіз і деякі види не параметричного статистичного аналізу.

Перевірка за критерієм Стьюдента необхідна в тих випадках, коли треба визначити імовірність того, що середнє чи сума деякого ряду незалежних значень (отриманих з вибірки) більше чи менше деякого числа чи знаходиться в деяких межах від нього. Ці критерії також корисні для вибору періодичності даних. Крім того, перевірка за критерієм Стьюдента допомагає установити границі продуктивності системи в майбутньому.

Кореляційний аналіз допомагає визначити ступінь взаємозв'язку двох різних змінних. При використанні для прийняття рішень він також допомагає визначити, чи є зв'язки «статистично значимими» чи просто випадковими. За допомогою таких методів можна визначити довірчі інтервали границь «реальної» кореляції, тобто кореляції по вибірці даних за деякий період часу [1].

### 1.2.5 Види даних

В області прогнозування поведінки фінансових ринків не можна зробити висновок про працездатність чи придатність того чи іншого методу моделювання без якісних даних для тестів і симуляцій. Для розробки системи, яка може адекватно

оцінювати ситуацію на ринках; як мінімум необхідні історичні цінові дані відносно необхідних секторів ринку.

Дані можуть використовуватися у своїх природних тимчасових рамках чи перераховуватися в інший масштаб. У залежності від масштабу, що використовується й особливостей торгової системи можуть знадобитися денні, тижневі, двотижневі, місячні, квартальні і навіть річні дані. Звичайно джерело даних має природні тимчасові обмеження. [1]. У зв’язку з цим, було прийнято рішення, використовувати данні різних часових рядів.

## 1.3 Висновки до розділу

В даному розділі було проаналізовано суть технічної проблеми, що виникла і доведено доцільність розробки нового узагальненого підходу до автоматизації проведення сценарного стрес-тестування, а також алгоритмічного та програмного забезпечення продукту. Проведено детальний аналіз існуючих програмних продуктів. представлені основні недоліки сучасних систем та шляхи їх вирішення, які дозволяють підвищити ефективність прогнозування портфелів цінних паперів, шляхом ідентифікації впливу різних макроекономічних факторів.

# 2 ПРОЕКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ ТА ВИБІР ТЕХНОЛОГІЙ РЕАЛІЗАЦІЇ

## 2.1 Аналіз інформаційної системи

Інформаційна система - організований набір елементів, що збирає, обробляє, передає, зберігає та надає дані. Інформаційна система складається із людей, обладнання, процесів, процедур, даних та операцій. Кожна інформаційна система включає в себе наступні компоненти:

- структура системи;
- функції кожного елемента системи;
- вхід і вихід кожного елементу і системи в цілому;
- мета і обмеження системи та її окремих елементів.

Серверна частина клієнт-серверної системи для передбачення поведінки фінансових інструментів повністю відповідає за забезпечення усіх заходів безпеки інформації, генерацію статистики, збір даних інвестиційних портфелів, перевірка та валідація даних автоматично в реальному часі, створення пакетів прогнозованих даних у форматі JSON та  доставку цих даних на клієнтську частину

Аналіз інформаційної системи наведений на рисунку 2.1.


Рисунок 2.1 – Інформаційна схема клієнт-серверної системи для передбачення поведінки фінансових інструментів

## 2.2 Розробка архітектури системи

Клієнт-серверна система для передбачення поведінки фінансових інструментів буде складатися із таких головних модулів:

- Модуль по роботі із інвестиційними портфелями
- Модуль генерації звітів
- Модуль по роботі зі змінними
- Модуль по роботі із відношеннями змінних та сценаріям даних

- Модуль керування компанією
- Модуль логування дій користувачів

Модуль по роботі із інвестиційними портфелями включає в себе функціонал, який дозволяє створювати, редагувати, налаштовувати та розраховувати портфелі. Сюди також входить створення компонентів портфелю, наприклад Revenue чи Expenses, та створення даних для цих компонентів. Прив’язка сценаріїв до портфелів, валідація даних портфелів та експорт даних калькуляції у вигляді xlsx чи csv файлу.


Рисунок 2.2 – Функціональна схема модулю по роботі з інвестиційними портфелями

Модуль генерації звітів включає в себе функціонал по генерації великої кількості різноманітних звітів, на основі розрахованих портфелів. Наприклад це можуть бути таки звіти, як HeatMap report, який показує відношення даних економічних змінних до даних по змінних порфелю у вигляді теплової карти. Також, одним із прикладів є Variable Historical Report, який виводить звіт по історичних даних економічних змінних.

Модуль по роботі зі змінними являє собою своєрідний клієнтський модуль, що звертається до стороннього сервісу, який спеціалізується на роботі зі змінними, по потрібну інформацію. Це може бути як і список змінних з короткою інформацією про них, так і повна інформація по одній змінній із маленькою вибіркою історичних даних. Також за допомогою цього модулю можна оновлбвати змінні та інформацію про них.

Модуль по роботі із відношеннями змінних та сценаріями даних, являє собою функціонал, що дозволяє створювати відношення змінних, куди входять головні змінні, та змінні від яких вони залежать. Далі головні змінні будуть розраховуватись на основі залежних змінних та інформації з портфелю.

Модуль курування компанією – модуль, який дозволяє налаштовувати компанію, керувати користувачами системи, які прив’язані до даної компанії, також провести налаштування компанії, наприклад вказати валюти, у яких будуть проводитись розрахунки, вибрати кольори та відрізки даних для HeatMap Report, налаштувати формат дати, та інші налаштування.

Модуль логування дій користувачів, являє собою middleware, який приймає запит, обробляє його, записує дані запиту в спеціальний singletone об’єкт, потім приймає відповідь на запит, обробляє, також записує в singletone об’єкт та відправляє відповідь далі. Потім спеціальний фоновий сервіс записує дані із singletone об’єкта в базу даних.

Також у системі будуть присутні такі модулі як Common, який буде відповідати за розміщення усіх об’єктів, моделей, текстів, які є спільні для усіх

модулів, DAL (Data Access Layer) – модуль, у якому за допомогою EntityFramework описані всі таблиці бази даних, та налаштоване підключення до неї а також налаштовані самі таблиці та зв’язки між ними, модуль Tests, де знаходяться та запускаються юніт-тести, та модуль Auth, де відбувається контроль над авторизацією користувача та валіідація його даних.

Підсумовуючи, потрібно зазначити, що кожен модуль має свої підсистеми та містить певний об’єм функцій

Для прикладу розглянемо функціональну схему модулю по роботі з інвестиційними портфелями. Функціональна схема модулю по роботі з інвестиційними портфелями на Рисунку 2.2.

Як зазначено на рисунку, даний модуль буде виконувати доволі великий обсяг роботи для забезпечення стабільності та коректності роботи системи.

UML діаграму архітектури системи наведено у додатку Є.

## 2.3 Проектування бази даних системи та її розробка за допомогою EntityFramework

Data Access Layer – шар доступу до даних У ньому містяться усі сутності системи, контекст бази даних, міграції бази даних та усі функції, що напряму взаємодіють з БД.

Оскільки у проекті використовується Entity Framework Core, то створення бази даних даної системи та налагодження взаємодії бази із системою можливе за допомогою двох підходів:

1. Code First;
2. Datebase First.

Code First це стиль моделювання бази даних, коли сутності сворюються у вигляді C#-класів, та за допомогою налаштувань у контексті та певних консольних команд, австоматично створюється база даних.

Контекст бази даних DbContext - це основний клас, який координує функціональні можливості EF для певної моделі даних. Цей клас є похідним від класу Microsoft.EntityFrameworkCore.DbContext. Похідний клас DbContext вказує сутності, які включаються в модель даних. Деякі розширення функціональності EF можна налаштувати окремо. В контексті визначаються усі таблиці, що будуть в базі, та налаштовуються зв’язки між певними таблицями. Також можливо налаштувати самі таблиці, що можна зробити і в класі сутності.

Підхід Datebase First – абсолютний антипод підходу Code First. У даному випадку, спочатку створюється база даних за допомогою SQL, потім, за допомогою консольних команд, макет бази транслюється у C#-код.

Основними сутностями системи є інвестиційні портфелі(Portfolio), сценарії(Scenario), дані компонентів портфелів(PortfolioComponentData), власне, самі компоненти портфелів(PortfolioComponentConfiguration), користувач системи (User) та компанія(Company). Діаграма зв’язків між таблицями в базі даних наведено в додатку Е

У базі даних системи присутні такі зв’язки між таблиями:

- Company – Portfolio – один до багатьох, оскільки до однієї компанії може буде прив’язано декілька портфелів;
- Portfolio – PortfolioComponentConfiguration – один до багатьох, оскільки портфель може містити декілька компонентів;
- Portfolio – PortfolioComponentData – один до багатьох, бо портфель можу містити багато даних компонентів. Кількість обмежена тільки ламітуванням під час валідації
- Company – Scenario – один до багатьох

- Company – User – один до багатьох
- User – Role – багато до багатьох, оскільки як один користувач може мати багато ролей, як і одну і ту саму роль може мати декілька користувачів.

## 2.4 Організація роботи сервісу логування дій користувачів за принципом middleware

Логування дій користувачів це одна із найголовніших частин системи, як з боку статистики так і з боку безпеки та відслідковування помилок. Якщо у якого користувача виникне якась помилка, та він надішле її опис, у адміністратора системи також будуть дані про запит до сервера та відповідь від нього, тож адміністратор чи розробник, завдяки цьому, зможе оперативно виправити помилку, та допомогти користувачу.

Логування дій користувача реалізовано за допомогою спеціальної підсистеми middleware, що є найкращим варіантом для цього функціоналу. Middleware працює наступним чином: спочатку до middleware надходить запит, роблять всі необхідні маніпуляції з інформацією, та викликається наступний middleware, поки вони не закінчаться. Потім відповідь проходить усі middleware, починаючи з останнього і до першого, перед тим як відправитись до клієнта. Спочатку послідовність проходження запиту через middleware виглядає як черга, а потім послідовність проходження відповіді виглядає як стек.

Власне, в даному випадку, спочатку до middleware логування дій користувачів попадає запит, його дані десеріалізуються, та записуються у модель, яку в свою чергу система записує в спеціальний singletone, який буде зберігати дані в системі, поки спеціальний фоновий сервіс не запише дані в базу. Це зроблено для того, щоб не перевантажувати навіть найлегший запит, оскільки записи в базу є доволі затратними по часу і робити їх кожен раз при кожному запиті було б недоцільно.

Потім, після збереження запиту, система викликає наступний в черзі middleware, а цей чекає допоки до нього не надійде відповідь. Коли надходить відповідь, відбуваються схожі маніпуляції, відповідь десеріалізується і записується в singletone.

Коротка схема роботи middleware та блок-схема методу логування зображені в додатках Д та Г відповідно.

## 2.5 Вибір програмної платформи та мови програмування

.NET Core — модульна платформа для розробки кросплатформного програмного забезпечення, з відкритим вихідним кодом. Базується на .NET Framework. Відрізняється від неї модульністю, кросплатформністю, можливістю застосування хмарних технологій.

Основою платформи є загальномовне середовище виконання Common Language Runtime (CLR), завдяки чому .NET підтримує кілька мов: поряд з C # це також VB.NET, C ++, F #, а також різні діалекти інших мов, прив'язані до .NET, наприклад, Delphi. NET. При компіляції код на будь-якому з цих мов компілюється в збірку спільною мовою CIL (Common Intermediate Language) - свого роду асемблер платформи .NET. Тому за певних умов можливо зробити окремі модулі однієї програми на окремих мовах.

C# – це об’єктно-орієнтовна мова програмування, розроблена компанією Microsoft для роботи з платформою Microsoft .NET Framework та пізніше розробленою .NET Core.

C# є дуже близьким родичом мови програмування Java. Мова Java була створена компанією Sun Microsystems, коли глобальний розвиток інтернету поставив задачу розосереджених обчислень. Взявши за основу популярну мову C++, Java виключила з неї потенційно небезпечні речі (типу вказівників без

контролю виходу за межі). Для розосереджених обчислень була створена концепція віртуальної машини та машинно-незалежного байт-коду, свого роду посередника між вихідним текстом програм і апаратними інструкціями комп'ютера чи іншого інтелектуального пристрою.[2]

LINQ (англ. Language Integrated Query - запити, інтегровані в мову) - компонент Microsoft .NET Framework, який додає нативні можливості виконання запитів даних до мов, що входять у .NET. Хоча порти існують для PHP (PHPLinq), JavaScript(linq.js), TypeScript (linq.ts), і ActionScript (ActionLinq), - жоден з них не є абсолютно еквівалентним LINQ в C# (де LINQ - не просто додаткова бібліотека, а частина мови).

LINQ розширює можливості мови, додаючи до неї вирази запитів, що є схожими на твердження SQL та можуть бути використані для зручного отримання та обробки даних масивів, XML документів, реляційних баз даних та сторонніх джерел. LINQ також визначає набір імен методів (що називаються стандартними операторами запитів, або стандартними операторами послідовностей), а також правила перекладу, що має використовувати компілятор для перекладу текучих виразів у звичайні, використовуючи їх назву, лямбда-вирази та анонімні типи.[3]

## 2.6 Вибір бази даних, та засобів взаємодії з нею

PostgreSQL (вимовляється «Пост-грес-К'ю-ель» або «постгрес») — об'єктно-реляційна система керування базами даних (СКБД). Є альтернативою як комерційним СКБД (Oracle Database, Microsoft SQL Server, IBM DB2 та інші), так і СКБД з відкритим кодом (MySQL, Firebird, SQLite).

Порівняно з іншими проєктами з відкритим кодом, такими як Apache, FreeBSD або MySQL, PostgreSQL не контролюється якоюсь однією компанією, її розробка

можлива завдяки співпраці багатьох людей та компаній, які хочуть використовувати цю СКБД та впроваджувати у неї найновіші досягнення.

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

Entity Framework Core (EF Core) являє собою об'єктно-орієнтовану, легковажну технологію від компанії Microsoft для доступу до даних. EF Core є ORM-інструментом (object-relational mapping - відображення даних на реальні об'єкти). Тобто EF Core дозволяє працювати базами даних, але є більш високий рівень абстракції: EF Core дозволяє абстрагуватися від самої бази даних і її таблиць і працювати з даними незалежно від типу сховища. Якщо на фізичному рівні ми оперуємо таблицями, індексами, первинними і зовнішніми ключами, але на концептуальному рівні, який нам пропонує Entity Framework, ми вже працюємо з об'єктами.

Entity Framework Core підтримує безліч різних систем баз даних. Таким чином, ми можемо через EF Core працювати з будь-якої СУБД, якщо для неї є потрібний провайдер.

За замовчуванням на даний момент Microsoft надає ряд вбудованих провайдерів: для роботи з MS SQL Server, для SQLite, для PostgreSQL. Також є провайдери від сторонніх постачальників, наприклад, для MySQL.

Також варто відзначити, що EF Core надає універсальний API для роботи з даними. І якщо, наприклад, ми вирішимо змінити цільову СУБД, то основні зміни в проекті будуть стосуватися насамперед конфігурації і настройки підключення до відповідних провайдерам. А код, який безпосередньо працює з даними, отримує дані, додає їх в БД і т.д., залишиться колишнім.

Центральною концепцією Entity Framework є поняття сутності або entity. Сутність визначає набір даних, які пов'язані з певним об'єктом. Тому дана технологія передбачає роботу не з таблицями, а з об'єктами і їх колекціями.[5]

## 2.7 Вибір архітектурного стилю зв’язку серверної системи з клієнтською

Representational State Transfer - це архітектура, тобто принципи побудови розподілених систем гіпермедіа включаючи універсальні способи обробки та передачі станів ресурсів по HTTP

Переваги REST:

- Відсутність додаткових внутрішніх прошарків, що означає передачу даних в тому ж вигляді, що і самі дані. Тобто дані не обертаються в XML, як це робить SOAP і XML-RPC, не використовується AMF, як це робить Flash і т.д. Просто віддаються самі дані;
- Кожна одиниця інформації (ресурс) однозначно визначається URL - це значить, що URL по суті є первинним ключем для одиниці даних. Причому абсолютно не має значення, в якому форматі знаходяться дані за адресою - це може бути і HTML, і jpeg, і документ Microsoft Word;
- Як відбувається управління інформацією ресурсу - це цілком і повністю ґрунтується на протоколі передачі даних. Найбільш поширений протокол звичайно ж HTTP. Для HTTP дію над даними задається за допомогою методів: GET (отримати), PUT (додати, замінити), POST (додати, змінити, видалити), DELETE (видалити). Таким чином, дії CRUD (Create-Read-Update-Delete) можуть виконуватися як з усіма 4-ма методами, так і тільки за допомогою GET і POST;

Сама архітектура REST не прив'язана до конкретних технологій і протоколів, але в реаліях сучасного Веб, побудова RESTful API майже завжди має на увазі

використання HTTP і будь-яких поширених форматів представлення ресурсів, наприклад JSON, або, менш популярного сьогодні, XML.

За визначенням, безпечні операції ідемпотентні, так як вони призводять до одного і того ж результату на сервері. Безпечні методи реалізовані як операції тільки для читання. Однак безпека не означає, що сервер повинен повертати той же самий результат кожного разу.[7]

## 2.8 Впровадження залежностей із використанням вбудованого DI-контейнера .NET Core

Залежність або сервіс – це об'єкт, від якого залежить будь-який інший об'єкт.

Dependency injection (DI) або впровадження залежностей представляє механізм, який дозволяє зробити взаємодіючі в додатку об'єкти слабкозв'язаними. Такі об'єкти пов'язані між собою через абстракції, наприклад, через інтерфейси, що робить всю систему більш гнучкою, більш адаптованою і розширюється.

Нерідко для налаштування залежностей в подібних системах використовуються спеціальні контейнери - IoC-контейнери (Inversion of Control). Такі контейнери служать свого роду фабриками, які встановлюють залежності між абстракціями і конкретними об'єктами і, як правило, управляють створенням цих об'єктів.

Основним вибором для реалізації впровадження залежностей на платформі .NET Core –є використання вбудованого IoC-контейнеру.

ASP.NET Core вже має вбудований контейнер впровадження залежностей, який представлений інтерфейсом IServiceProvider. А самі залежності ще називаються сервісами, власне тому контейнер можна назвати провайдером сервісів. Цей контейнер відповідає за зіставлення залежностей з конкретними типами і за впровадження залежностей в різні об'єкти.[9]

Для роботи із даним контейнером слід використовувати такі ключові методи, як: AddScoped, AddSingleton та AddTransient. Усі вони створюють об’єкти з відповідним часом існування

Впровадження залежностей усуває проблеми сильного зв’язку між об’єктами наступним чином:

## 2.9 Висновки до розділу

В даному розділі було розглянуто та обгрунтовано вибір мови програмування для розв’язання поставленої задачі. Проаналізовано можливість реалізації зв’язку між клієнтом та сервером за допомогою використання архітектурного стилю REST API.

Розглянуто реалізацію системи із використанням технологій .NET Core, ASP.NET Core, EntityFramework Core, PostgreSQL.

Було описано реалізацію принципу впровадження залежностей у додатках, що використовують .NET Core.

# 3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 3.1 Розробка базових компонентів функціоналу програми

Для розробки серверної частини буде використовуватись мова програмування C#, ASP .NET Core, EntityFramework на платформі .NET 6.

Детальніше про технології, які будуть використовуватись під час розробки програми описано у розділі 2.

Основним архітектурним патерном для розробки серверної частини програми було вибрано патерн MVC. [19]

Основні вимоги, щодо взаємодії компонентів патерну MVC, та невід'ємні частини реалізації, наведені на Рисунку 3.1.

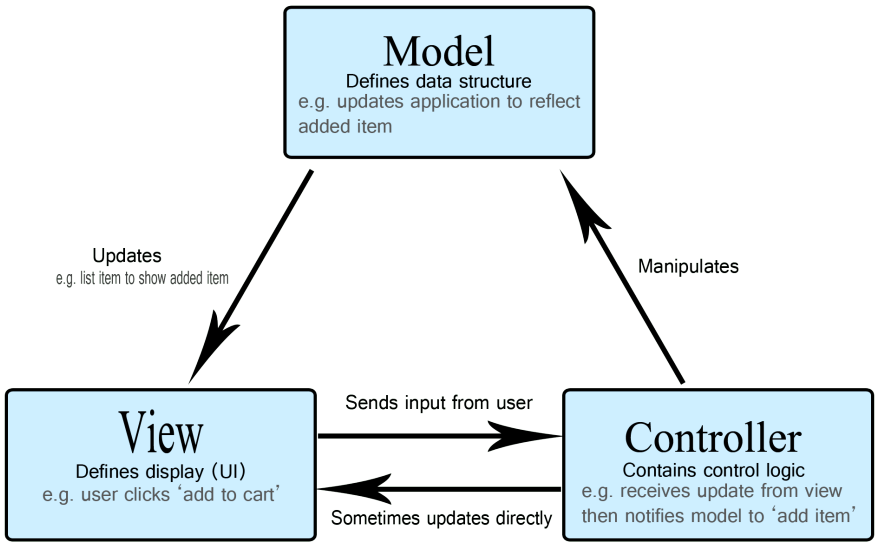



Рисунок 3.1 – Структура патерну MVC, та зв'язки взаємодії між структурними одиницями

Архітектура програми буде складатися з набору класів, що взаємодіють один з одним.

Усі класи серверної частини поділяються на такі типи:

- менеджер – класи, які інкапсулюють бізнес-логіку певної логічної сутності програми;
- модель – класи, що виступають описом логічних сутностей програми. Використовуються для представлення та створення таблиць в базі даних за допомогою EntityFramework;
- модель-представлення – модель, що являє собою об’єкт, що буде надсилати клієнт, або об’єкт, який буде надходити до клієнта з серверу
- клієнт – класи, містять у собі набір функцій для отримання результатів з зовнішніх ресурсів по заданому URL;
- контролер – класи, представляють собою набір методів, що виступають кінцевими точками, для доступу до бізнес логіки зі сторони клієнта; [44]

```
[HttpPost( template: "")]
[UserActionLog("Portfolio", description: "Create empty portfolio", containsSensitiveData: false, RequestSensitivityLevel.ApplicationSection)]
2 usages  Serhii Borodkin +2
public async Task<PortfolioViewModel> CreatePortfolio(string type)
{
    try
    {
        CheckPortfolioTypeRoles(type);
        var vm :PortfolioViewModel = await ServiceProvider.LogicProvider.PortfolioManager.CreatePortfolio(type.GetEnum<PortfolioType>());
        await LogicHub(vm.CompanyId, vm.PortfolioTypeEnum.Value).PortfolioCreated(vm, stamp: string.Empty);
        return vm;
    }
    catch (Exception ex)
    {
        ServiceProvider.Logger.LogErrorCreate(ex, CurrentUserId, viewModel: null);
        throw;
    }
}
```

Рисунок 3.2 – Метод створення портфелю даних

Розглянемо декілька прикладів функцій, які будуть реалізовувати основний функціонал програми.

Метод створення портфелю даних наведений на Рисунку 3.2.

З клієнтської частини приходить тип портфелю, який потрібно створити, потім, за допомогою методу CheckPortfolioTypeRoles визначається, чи доступне створення портфелю цього типу користувачу, після цього відбувається виклик методу CreatePortfolio у відповідному менеджері, який інкапсулює логіку, пов’язану із роботою з портфелями, по веб-сокету відправляється повідомлення про створення портфелю, та його модель, щоб клієнтська частина одразу відобразила його.

Ще один важливий сервіс – це сервіс логування дій користувачів, збереження їх в базі даних, та подальшому відображенні в спеціальному меню адміністратора.

Дані функції наведені на Рисунку 3.3.

```
public static void UseRequestResponseLogging(this IApplicationBuilder app, ILogger logger, IAppAuthorizeService appAuthorizeService)
{
    app.Use(async (context, next) =>
    {

        if (GlobalVars.RequestResponseLogType != Common.Enums.UserActionLogType.None)
        {
            try
            {
                if (GlobalVars.RequestResponseLogType == Common.Enums.UserActionLogType.All || _checkFilters(context))
                {
                    var userActionManager = context.RequestServices // IServiceProvider
                        .GetService(typeof(IUserActionLogManager)) as IUserActionLogManager;
                    var userActionLogListSingleton = context.RequestServices // IServiceProvider
                        .GetService(typeof(UserActionLogListSingletonModel)) as UserActionLogListSingletonModel;


                    var actionModel = new UserActionDetailedViewModel();
                    var userId :long? = _getUserId(await context.GetTokenAsync("access_token"));

                    actionModel = await FormatRequestAsync(context, actionModel, logger, userId);


                    var originalBody :Stream = context.Response.Body;

                    using (var newResponseBody :MemoryStream = new RecyclableMemoryStreamManager().GetStream())
                    {
                        context.Response.Body = newResponseBody;

                        await next.Invoke();

                        newResponseBody.Seek( offset: 0,  loc: SeekOrigin.Begin);
                        await newResponseBody.CopyToAsync(originalBody);

                        newResponseBody.Seek( offset: 0,  loc: SeekOrigin.Begin);
                        actionModel = await FormatResponseAsync(context, newResponseBody, actionModel, appAuthorizeService);


                        //logger.LogInformation(actionModel.RequestBody + Environment.NewLine + actionModel.ResponseBody);
                    }

                    userActionLogListSingleton.UserActionList.Add(actionModel);
                    //await userActionManager./*UserActionLogManager.*/SaveUserLog(actionModel);
                }
                else
                    await next.Invoke();
```

Рисунок 3.3 – Функції отримання загального життєвого циклу серверів

Даний функціонал розроблений у вигляді middleware. Це спеціальна підсистема, яка спрацьовує під час кожного запиту, що ідеально підходить для реалізації системи логування дій користувачів.

Метод спочатку приймає запит з клієнта, записує в змінні посилання на об’єкт менеджеру для роботи із діями користувачів, що інкапсулює відповідну логіку, із DI-контейнера, та посилання на спеціальний статичний об’єкт, що буде зберігати в пам’яті дії користувачів доти, доки сервіс по запису логів не запише цю інформацію в базу даних, та очистить пам’ять.

За допомогою методу FormatRequestAsync, система запише дані запиту до об’єкту actionModel, та викликом next.Invoke, передасть запит на інший middleware.

Потім за допомогою FormatResponseAsync, система запише дані відповіді сервера до actionModel, та запише цей об’єкт до userActionLogSingletone.

Структурна UML діаграма патерну MVC наведена у додатку Б на Рисунку Б.5. UML діаграма класів, розробленого модулю моніторингу інтернет сервісів, наведена у додатку Б на Рисунку Б.6.

## 3.2 Організація роботи із портфелями

Портфелі – основний компонент системи. Навколо них відбувається більшість процесів, додатку, оскільки вони відображають набір даних, та змінних, на основі яких відбувається передбачення поведінки певних активів.

Метод створення портфелю було розглянуто у підрозділі 3.1

Для того, щоб виконати розрахунки, в портфелі потрібно створити та заповнити відношення змінних, тобто вказати головні змінні, та змінні, від яких вони залежать. На основі залежних змін та інформації в портфелі буде відбуватись прогнозування головних змінних. Також потрібно заповнити хоча б один із компонентів інвестиційного портфелю, цього буде достатньо для розрахунку. Далі потрібно прив’язати відношення змінних до портфелю, та можна буде відправляти його на розрахунок.

Розглянемо один із компонентів портфелю, код якого зображено на рисунку 3.4.

```
public class PortfolioComponentSegment : PortfolioComponentBase
{
    [DisplayName("Minimum")]
    [Edit]
    [JsonProperty("minimum")]
    12 usages
    public double Minimum { get; set; }

    [DisplayName("Maximum")]
    [Edit]
    [JsonProperty("maximum")]
    9 usages
    public double Maximum { get; set; }
    Serhii Borodkin
    public PortfolioComponentSegment()
    {
    }
    Serhii Borodkin
    public PortfolioComponentSegment(
        string[] header,
        string[] csv,
        PortfolioComponentDataValueViewModel data,
        Dictionary<long, string> variables)
        : base(header, csv, data, variables)
    { }

    0+1 usages   Serhii Borodkin
    public override IPortfolioComponentCalculatorViewModel GetComponentCalculatorViewModel(
    PortfolioComponentDataValueViewModel dataValues,
    Dictionary<long, string> variables)
    {
        return new PortfolioComponentSegmentCalculatorViewModel()
        {
            Minimum = Minimum,
            Maximum = Maximum
        };
    }
```

Рисунок 3.4 Сегмент – один із компонентів портфелю

Сегмент є одним із компонентів портфелю з типом Efficient Frontier. Він є обов'язковим для заповнення, та його можна прив'язувати до даних в інших компонентах. Він потрібен для деталізації інших компонентів.

Розглянемо механізм створення компонентів. Код методу буде представлений на рисунку 3.5

```
public async Task<bool> CreatePortfolioComponentData([FromRoute]long portfolioId, [FromRoute]byte componentId, [FromBody]PortfolioComponentDataViewModel portfolioComponentData)
{
    try
    {
        await ValidatePortfolioBeforeAction(portfolioId);
        var component = PortfolioComponent.GetComponent(componentId);
        if (component.ModelType == typeof(PortfolioComponentFixedIncome) ||
            component.ModelType == typeof(PortfolioComponentEfFixedIncome))
        {
            var yieldCurves :Dictionary<string,Dictionary<...>> = await ServiceProvider.LogicProvider.VariablesManager // IVariablesManager
                .GetMappingField(CurrentCompanyId, func: m :MappingViewModel => m.YieldCurves); // Task<Dictionary<...>>
            if (yieldCurves == null ||
                yieldCurves.Count == 0 ||
                yieldCurves.First().Value == null ||
                yieldCurves.First().Value.Count == 0)
                throw new EmptyYieldCurvesException();
        }

        var dataValues = await GetPortfolioComponentDatasList();
        var result :PortfolioComponentDataViewModel = await ServiceProvider.LogicProvider.PortfolioManager // IPortfolioManager
            .CreatePortfolioComponentData(portfolioId, componentId, portfolioComponentData, SimpleVariables, dataValues); // Task<PortfolioComponentDataViewModel>
        await LogicHub(CurrentCompanyId).PortfolioComponentDataCreated(result, portfolioId, componentId);
        await UpdatePortfolioConfigurationAndStatus(portfolioId, componentId);

        await SetChildPortfoliosToReady(portfolioId, scenarioId: null);
        return true;
    }
    catch (Exception ex)
    {
        ServiceProvider.Logger.LogError(ex, message: $"Error while creating portfolio {portfolioId} component data", portfolioComponentData);
        throw;
    }
}
```

Рисунок 3.5. Метод створення компонента портфеля.

В даному методі спочатку відбувається перевірка, чи дозволено користувачу проводити дану дію з цим портфелем, потім отримуються налаштування компанії та записуються у змінну dataValues, далі в змінну result записується результат, який повернув метод CreatePortfolioComponentData, який потім по веб-сокету відправляється на клієнтську частину. Під час створення, компонент мусить пройти валідацію, щоб відповідати певним, визначеним наперед вимогам. Валідація сегмента зображена на рисунку 3.6

```
public override bool Validate(PortfolioComponentDataValueViewModel data,
    Dictionary<long, SimpleVariableNameViewModel> variables)
{
    var errors = new List<string>();


    if (Minimum < 0 || Minimum > 1)
        errors.Add( item: string.Format(ExceptionMessages.ComponentValidationError, nameof(Minimum),
            "Value should be from 0 to 1"));

    if (Maximum < 0 || Maximum > 1)
        errors.Add( item: string.Format(ExceptionMessages.ComponentValidationError, nameof(Maximum),
            "Value should be from 0 to 1"));

    if (Minimum > Maximum)
        errors.Add( item: string.Format(ExceptionMessages.ComponentValidationError, nameof(Minimum),
            "Value should be less than Maximum"));

    if (Minimum == 1.0)
        errors.Add( item: string.Format(ExceptionMessages.ComponentValidationError, nameof(Minimum),
            "Value can't be 1. It should be less than 1"));

    return Validate(errors);
}
```

Рисунок 3.6. Валідація сегмента

## 3.3 Тестування програмного забезпечення

Тестування системи, яка проектується в межах даної роботи, буде базуватись на написанні юніт-тестів за допомогою фреймворка XUnit.

Використання XUnit обумовлюється такими перевагами:

- створення «моків», тобто штучних класів і інтерфейсів;
- перевірка виклика методу і значень переданих методу в якості параметрів;

Також цей фреймворк має набір методів визначення поведінки функції в залежності від тих чи інших параметрів. [28][29]

За допомогою XUnit можна робити заміну оригінального методу на метод-заглушку, в який можна передавати наперед визначені для тестування параметри, а також вказати, який об’єкт буде повертати цей метод. Для цього у бібліотеці цього фреймворку є функція Setup() та Returns().

З їх допомогою можливо не тільки робити підміну значень та функцій, а й обраховувати час виконання методу, підраховувати скільки разів даний метод був викликаний і при яких умовах.

Приклад тесту з використанням фреймворку XUnit наведений на Рисунку 3.7.

```
public async Task GeneralTest()
{
    //Arrange
    long portfolioId = 1;
    byte componentId = 1;
    var component = PortfolioComponent.GetComponent(componentId);
    var data = JsonConvert.DeserializeObject(value: "{}", component.ModelType) as Models.Interfaces.IPortfolioComponentDataValidatableObject;
    switch (componentId)
    {
        case 1:
            var obj = data as PortfolioComponentLoan;
            obj.RatingId = 1;
            //obj.OriginationDate = DateTime.Now.AddDays(-2);
            //obj.MaturityDate = DateTime.Now.AddDays(2);
            obj.Tenor = 1;
            obj.TimeToMaturity = 1;
            obj.SectorId = 1;
            obj.CurrencyId = 1;
            obj.ValueOfCollateral = 1;
            obj.Lgd = 1;
            obj.TypeId = 1;
            obj.InitialDrawn = 1;
            obj.MaxFundedDrawnDefault = 1;
            obj.TimeToMaxDrawn = 1;
            obj.DrawnAmount = 1;
            obj.TotalCommitment = 2;
            obj.CouponOrSpread = 1;
            obj.ReturnTypeId = 1;
            obj.RecoveryIndexId = 1;
            break;
    }
    var portfolioComponentData = new PortfolioComponentDataViewModel() { Id = 1, ComponentData = data, Amount = 2, Name = "TestName" };

    var portfolio = new List<Portfolio>() { new Portfolio() { Id = portfolioId } };
    var datas = new List<PortfolioComponentData>()
    {
        new PortfolioComponentData() { PortfolioId = 1, Id = 1, Name = "Loan 1", Amount = 100, ComponentData = portfolioComponentData.ComponentDataString },
        new PortfolioComponentData() { PortfolioId = 2, Id = 2, Name = "Loan 2", Amount = 200, ComponentData = portfolioComponentData.ComponentDataString },
    };
    var portfolioComponentDataValue = new PortfolioComponentDataValueViewModel()
    {
        Currencies = new List<PortfolioComponentCurrencyDataDropDownViewModel>() { new PortfolioComponentCurrencyDataDropDownViewModel() { Id = 1, Name = "USD", Code = "USD" } },
        Ratings = new List<Common.Models.IdNameViewModel>() { new Common.Models.IdNameViewModel() { Id = 1, Name = "A" } },
        Sectors = new List<Common.Models.IdNameViewModel>() { new Common.Models.IdNameViewModel() { Id = 1, Name = "Sector" } },
        RequiredVariables = new List<long>() { 1, 2, 3 }
    };
    var variables = new Dictionary<long, SimpleVariableNameViewModel>()
    {
        {1, new SimpleVariableNameViewModel(){ Name = "variable1", Lognormal = Lognormal.Lognormal.ToString()} }
    };
    Init(portfolio, isPermissionGranted: true, datas, isEmpty: false);
    //Act
    var result :PortfolioComponentDataViewModel = await PortfolioManagerMock.Object.CreatePortfolioComponentData(portfolioId, componentId, portfolioComponentData, variables, portfolioComponentDataValue);
    //Assert
    var isCorrect = result.Id == 0 && result.Amount == portfolioComponentData.Amount && result.ComponentDataString == portfolioComponentData.ComponentDataString;
    Assert.True(isCorrect);
}
```

Рисунок 3.7 – Приклад юніт-тесту з використанням XUnit

Кожен тест помічається атрибутом [Fact], який вказує XUnit на те, що даний метод є юніт-тестом.

Кожен тест можна запускати окремо, не запускаючи увесь проект.

Для прикладу, запустим усі тести класу CreatePortfolioComponentDataText, який був розроблений для тестування частини менеджера по роботі з портфелем, а саме функціоналу створення компонентів портфелю, результат виконання тестів наведено на Рисунку 3.8.

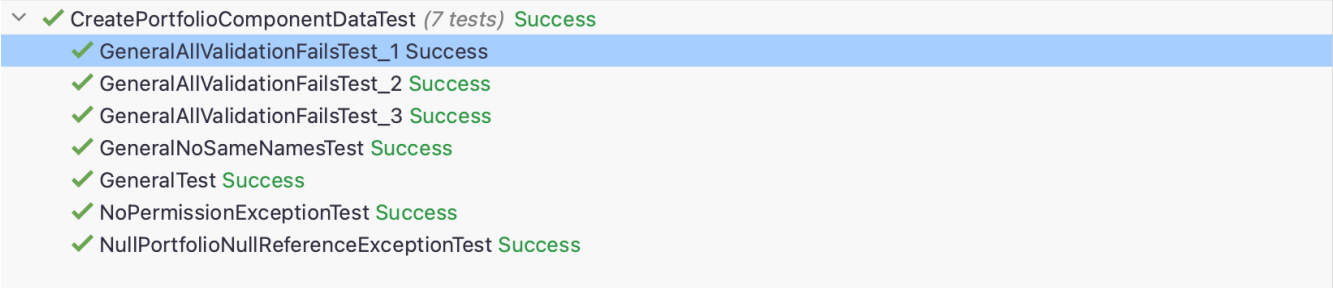



Рисунок 3.8 – Результат виконання тестів класу CreatePortfolioComponentDataText

Після покриття тестами певної частини коду, ми можемо дивитись на відсоток покритих методів.

## 3.4 Висновки до розділу

У даному розділі розглянуто основні етапи розробки базових компонентів системи прийняття рішення, щодо підбору віртуальної машини.

Розроблено UML Flow діаграму одного із основних компонентів програми – системи логування дій користувачів, діаграму роботи механізму middleware, діаграму роботи сервісу авторизації.

Розроблено базові компоненти серверної частини програми за допомогою мови програмування C#, та на базі платформи .NET 6.

Лістинг основного компоненту програми, а саме контролеру по роботі з портфелями наведено у додатку Г. [47]

# 4 ЕКОНОМІЧНА ЧАСТИНА

Науково-технічна розробка має право на існування та впровадження, якщо вона відповідає вимогам часу, як в напрямку науково-технічного прогресу та і в плані економіки. Тому для науково-дослідної роботи необхідно оцінювати економічну ефективність результатів виконаної роботи.

## 4.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Метою проведення комерційного і технологічного аудиту дослідження за темою «Розробка клієнт-серверної системи для прогнозування поведінки фінансових інструментів (Серверна частина)» є оцінювання науково-технічного рівня та рівня комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності.

Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням 5-ти бальної системи оцінювання за 12-ма критеріями.[52]

Таблиця 4.1 – Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки експертами

| Критерії | Експерт (ПІБ, посада) | | |
|---|---|---|---|
| | 1 | 2 | 3 |
| | Бали: | | |
| 1. Технічна здійсненність концепції | 5 | 5 | 5 |
| 2. Ринкові переваги (наявність аналогів) | 1 | 2 | 2 |
| 3. Ринкові переваги (ціна продукту) | 1 | 1 | 1 |
| 4. Ринкові переваги (технічні властивості) | 1 | 2 | 2 |
| 5. Ринкові переваги (експлуатаційні витрати) | 2 | 2 | 2 |

| | | | |
|---|---|---|---|
| 6. Ринкові перспективи (розмір ринку) | 3 | 3 | 3 |
| 7. Ринкові перспективи (конкуренція) | 3 | 2 | 3 |
| 8. Практична здійсненність (наявність фахівців) | 5 | 5 | 5 |
| 9. Практична здійсненність (наявність фінансів) | 3 | 4 | 3 |
| 10. Практична здійсненність (необхідність нових матеріалів) | 5 | 5 | 5 |
| 11. Практична здійсненність (термін реалізації) | 4 | 4 | 4 |
| 12. Практична здійсненність (розробка документів) | 4 | 4 | 4 |
| Сума балів | 37 | 39 | 39 |
| Середньоарифметична сума балів $СБ_с$ | 38,3 | | |

За результатами розрахунків, наведених в таблиці 4.1, зробимо висновок щодо науково-технічного рівня і рівня комерційного потенціалу розробки. При цьому використаємо рекомендації, наведені в [52]

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Розробка клієнт-серверної системи для прогнозування поведінки фінансових інструментів (Серверна частина)» становить 38,3 бала, що відповідно до [52] свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки вище середнього).

## 4.2 Визначення рівня конкурентоспроможності розробки

В процесі визначення економічної ефективності науково-технічної розробки також доцільно провести прогноз рівня її конкурентоспроможності за сукупністю параметрів, що підлягають оцінюванню.

Загальні технічні та економічні характеристики розробки представлено в таблиці 4.2.

Таблиця 4.2 – Основні техніко-економічні показники аналога та розробки, що проектується

| Показники (параметри) | Одиниця вимірю-вання | Аналог | Проектований пристрій | Відношення параметрів нової розробки до аналога | Питома вага показника |
|---|---|---|---|---|---|
| Об'єм памяті дистрибутивів для запуску серверу (на сервері) | МБ | 640 | 400 | 1,6 | 0,2 |
| Об'єм оперативної пам'яті | МБ | 310 | 150 | 2,1 | 0,25 |
| Кількість таблиць бази даних | шт | 75 | 33 | 2,27 | 0,2 |
| Об'єм бази даних | ГБ | 310 | 5 | 62 | 0,1 |
| Кількість мікросервісів | шт | 15 | 4 | 3,75 | 0,25 |

| | | | | | |
|---|---|---|---|---|---|
| Експлуатаційні витрати | грн. | 500 | 320 | 0,64 | 0,5 |
| Запланована вартість підписки (доступу до ресурсу) | грн. | 700 | 780 | 1,11 | 0,5 |

Одиничний параметричний індекс розраховуємо за формулою[52]:

$$q_i = \frac{P_i}{P_{базі}} \tag{4.1}$$

де $q_i$ – одиничний параметричний індекс, розрахований за $i$-м параметром;

$P_i$ – значення $i$-го параметра виробу;

$P_{базі}$ – аналогічний параметр аналога, з яким проводиться порівняння.

Нормативні параметри оцінюємо показником, який отримує одне з двох значень: 1 – пристрій відповідає нормам і стандартам; 0 – не відповідає.

За нормативними параметрами розроблюваний пристрій відповідає вимогам ДСТУ, тому $I_{нп} = 1$.

Значення групового параметричного індексу за технічними параметрами визначаємо з урахуванням вагомості (частки) кожного параметра [52]:

$$I_{ТП} = \sum_{i=1}^{n} q_i \cdot \alpha_i \,, \tag{4.2}$$

де $I_{тп}$ – груповий параметричний індекс за технічними показниками (порівняно з виробом-аналогом);

$q_i$ – одиничний параметричний показник $i$-го параметра;

$\alpha_i$ – вагомість $i$-го параметричного показника, $\sum_{i=1}^{n} \alpha_i = 1$;

*n* – кількість технічних параметрів, за якими оцінюється конкурентоспроможність.

Проведемо аналіз параметрів згідно даних таблиці 4.2.

$I_{mn}$ = 1,6·0,2+2,1·0,25+2,27·0,2+62·0,1+3,75·0,25 = 8,44.

Груповий параметричний індекс за економічними параметрами розраховуємо за формулою[52]:

$$= I_{ЕП} = \sum_{i=1}^{m} q_i \cdot \beta_i \,, \tag{4.3}$$

де $I_{ЕП}$ – груповий параметричний індекс за економічними показниками;

$q_i$ – економічний параметр *i*-го виду;

$\beta_i$ – частка *i*-го економічного параметра, $\sum_{i=1}^{m} \beta_i = 1$;

*m* – кількість економічних параметрів, за якими здійснюється оцінювання.

Проведемо аналіз параметрів згідно даних таблиці .

$I_{ЕП}$=0,64 · 0,5 + 1,11 · 0,5 = 0,88.

На основі групових параметричних індексів за нормативними, технічними та економічними показниками розрахуємо інтегральний показник конкурентоспроможності за формулою[52]**:**

$$K_{ІНТ} = I_{НП} \cdot \frac{I_{ТП}}{I_{ЕП}}, \tag{4.4}$$

$К_{ІНТ}$ = 1 · 8,44 / 0,88 = 9,64.

Інтегральний показник конкурентоспроможності $К_{ІНТ}$ > 1, отже розробка переважає відомі аналоги за своїми техніко-економічними показниками.

## 4.3 Розрахунок витрат на проведення науково-дослідної роботи

Витрати, пов'язані з проведенням науково-дослідної роботи на тему «Розробка клієнт-серверної системи для прогнозування поведінки фінансових інструментів (Серверна частина)», під час планування, обліку і калькулювання собівартості науково-дослідної роботи групуємо за відповідними статтями.

### 4.3.1 Витрати на оплату праці

Основна заробітна плата дослідників

Витрати на основну заробітну плату дослідників ($З_о$) розраховуємо у відповідності до посадових окладів працівників, за формулою[52]:

$$З_о = \sum_{i=1}^{k} \frac{M_{ni} \cdot t_i}{T_p}, \qquad (4.5)$$

де $k$ – кількість посад дослідників залучених до процесу досліджень;

$M_{ni}$ – місячний посадовий оклад конкретного дослідника, грн;

$t_i$ – число днів роботи конкретного дослідника, дн.;

$T_p$ – середнє число робочих днів в місяці, $T_p$=21 дні.

$З_о = 16500{,}00 \cdot 32 / 21 = 25142{,}86$ *грн.*

Проведені розрахунки зведемо до таблиці.

Таблиця 4.3 – Витрати на заробітну плату дослідників

| Найменування посади | Місячний посадовий оклад, грн | Оплата за робочий день, грн | Число днів роботи | Витрати на заробітну плату, грн |
|---|---|---|---|---|
| Керівник проекту | 16500,00 | 785,71 | 32 | 25142,86 |
| Інженер-розробник | 15300,00 | 728,57 | 26 | 18942,86 |

| | | | | |
|---|---|---|---|---|
| інформаційних систем | | | | |
| Інженер-розробник програмного забезпечення | 15350,00 | 730,95 | 26 | 19004,76 |
| Консультант (менеджер-аналітик фінансових систем) | 20000,00 | 952,38 | 3 | 2857,14 |
| Технік | 7500,00 | 357,14 | 30 | 10714,29 |
| Всього | | | | 76661,90 |

Основна заробітна плата робітників

Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт НДР на тему «Розробка клієнт-серверної системи для прогнозування поведінки фінансових інструментів (Серверна частина)» розраховуємо за формулою:

$$З_р = \sum_{i=1}^{n} C_i \cdot t_i, \qquad (4.6)$$

де $C_i$ – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

$t_i$ – час роботи робітника при виконанні визначеної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду $C_i$ можна визначити за формулою:

$$C_i = \frac{M_M \cdot K_i \cdot K_c}{T_p \cdot t_{зм}}, \qquad (4.7)$$

де $М_М$ – розмір мінімальної місячної заробітної плати, приймемо $Мм$=6700,00 грн;

$К_і$ – коефіцієнт міжкваліфікаційного співвідношення (табл. Б.2, додаток Б) [52];

$К_с$ – мінімальний коефіцієнт співвідношень місячних тарифних ставок;

$Т_р$ – середнє число робочих днів в місяці, приблизно $Т_р$ = 21 дн;

$t_{зм}$ – тривалість зміни, год.

$С_1 = 6700,00 \cdot 1,10 \cdot 1,65 / (21 \cdot 8) = 72,38$ грн.

$З_{р1} = 72,38 \cdot 8,20 = 593,55$ грн.

Таблиця 4.4 – Величина витрат на основну заробітну плату робітників

| Найменування робіт | Тривалість роботи, год | Розряд роботи | Тарифний коефіцієнт | Погодинна тарифна ставка, грн | Величина оплати на робітника грн |
|---|---|---|---|---|---|
| Установка обладнання для моделювання та проектування інформаційної системи | 8,20 | 2 | 1,10 | 72,38 | 593,55 |
| Підготовка робочого місця розробника програмного забезпечення | 4,70 | 3 | 1,3 | 88,83 | 417,52 |

| | | | | | |
|---|---|---|---|---|---|
| Підготовка серверного обладнання | 3,20 | 4 | 1,50 | 98,71 | 315,86 |
| Інсталяція програмного забезпечення для моделювання та розробки інформаційної системи | 6,22 | 4 | 1,50 | 98,71 | 613,95 |
| Компіляція програмних блоків | 8,00 | 5 | 1,70 | 111,87 | 894,93 |
| Налагодження програмних блоків | 4,60 | 6 | 2,00 | 131,61 | 605,39 |
| Тестування системи | 10,00 | 2 | 1,10 | 72,38 | 723,84 |
| Всього | | | | | 4165,04 |

Додаткова заробітна плата дослідників та робітників

Додаткову заробітну плату розраховуємо як 10 … 12% від суми основної заробітної плати дослідників та робітників за формулою:

$$З_{дод} = (З_о + З_р) \cdot \frac{Н_{дод}}{100\%}, \quad (4.8)$$

де $Н_{дод}$ – норма нарахування додаткової заробітної плати. Приймемо 10%.

$З_{дод} = (76661,90 + 4165,04) \cdot 10 / 100\% = 8082,69$ грн.

4.3.2 Відрахування на соціальні заходи

Нарахування на заробітну плату дослідників та робітників розраховуємо як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%} \qquad (4.9)$$

де $Н_{зп}$ – норма нарахування на заробітну плату. Приймаємо 22%.

Зн = (76661,90 + 4165,04 + 8082,69) · 22 / 100% = 19560,12 грн.

4.3.3 Сировина та матеріали

Витрати на матеріали ($М$), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою:

$$М = \sum_{j=1}^{n} Н_j \cdot Ц_j \cdot К_j - \sum_{j=1}^{n} В_j \cdot Ц_{вj}, \qquad (4.10)$$

де $Н_j$ – норма витрат матеріалу $j$-го найменування, кг;

$n$ – кількість видів матеріалів;

$Ц_j$ – вартість матеріалу $j$-го найменування, грн/кг;

$К_j$ – коефіцієнт транспортних витрат, ($К_j$ = *1,1 ... 1,15*);

$В_j$ – маса відходів $j$-го найменування, кг;

$Ц_{вj}$ – вартість відходів $j$-го найменування, грн/кг.

$М_1$ = 3,0 · 295,00 · 1,11 - 0 · 0 = 982,35 грн.

Проведені розрахунки зведемо до таблиці.

Таблиця 4.5 – Витрати на матеріали

| Найменування матеріалу, марка, тип, сорт | Ціна за 1 кг, грн | Норма витрат, кг | Величина відходів, кг | Ціна відходів, грн/кг | Вартість витраченого матеріалу, грн |
|---|---|---|---|---|---|

| | | | | | |
|---|---|---|---|---|---|
| Офісний папір Cristal A4 500 | 295,00 | 3,0 | - | - | 982,35 |
| Папір для записів Cristal LightPapers 65 A5 | 162,00 | 4,0 | - | - | 719,28 |
| Органайзер офісний Cristal | 195,00 | 4,0 | - | - | 865,80 |
| Набір офісний Cristal Base | 204,00 | 3,0 | - | - | 679,32 |
| Картридж для принтера Canon LBP5000 | 1420,00 | 1,0 | - | - | 1576,20 |
| Диск оптичний CD-R | 23,00 | 2,0 | - | - | 51,06 |
| Flesh-пам'ять Kingston 32 GB | 410,00 | 1,0 | - | - | 455,10 |
| Всього | | | | | 5329,11 |

4.3.4 Розрахунок витрат на комплектуючі

Витрати на комплектуючі *($К_в$)*, які використовують при проведенні НДР на тему «Розробка клієнт-серверної системи для прогнозування поведінки фінансових інструментів (Серверна частина)», розраховуємо, згідно з їхньою номенклатурою, за формулою:

$$К_в = \sum_{j=1}^{n} Н_j \cdot Ц_j \cdot К_j \quad (4.11)$$

де $Н_j$ – кількість комплектуючих $j$-го виду, шт.;

$Ц_j$ – покупна ціна комплектуючих $j$-го виду, грн;

$К_j$ – коефіцієнт транспортних витрат, ($К_j$ = *1,1 … 1,15*).

$К_в = 1 \cdot 3850{,}00 \cdot 1{,}12 = 4312{,}00$ грн.

Проведені розрахунки зведемо до таблиці.

Таблиця 4.6 – Витрати на комплектуючі

| Найменування комплектуючих | Кількість, шт. | Ціна за штуку, грн | Сума, грн |
|---|---|---|---|
| RouterBOARD TP-Link230 | 1 | 3850,00 | 4312,00 |
| База даних (тестувальна) БД64-А10 | 1 | 2680,00 | 3001,60 |
| Всього | | | 7313,60 |

4.3.5 Спецустаткування для наукових (експериментальних) робіт

Балансову вартість спецустаткування розраховуємо за формулою:

$$В_{спец} = \sum_{i=1}^{k} Ц_i \cdot С_{пр.i} \cdot К_i, \quad (4.12)$$

де $Ц_i$ – ціна придбання одиниці спецустаткування даного виду, марки, грн;

$С_{пр.i}$ –кількість одиниць устаткування відповідного найменування, які придбані для проведення досліджень, шт.;

$K_i$ – коефіцієнт, що враховує доставку, монтаж, налагодження устаткування тощо, ($K_i$ = *1,10…1,12);*

*k* – кількість найменувань устаткування.

$В_{спец}$ = 30360,00 · 1 · 1,12 = 34003,20 грн.

Отримані результати зведемо до таблиці:

Таблиця 4.7 – Витрати на придбання спецустаткування по кожному виду

| Найменування устаткування | Кількість, шт | Ціна за одиницю, грн | Вартість, грн |
|---|---|---|---|
| Серверне обладнання на основі EOM DELUX F43-A71BC | 1 | 30360,00 | 34003,20 |
| Всього | | | 34003,20 |

4.3.6 Програмне забезпечення для наукових (експериментальних) робіт

Балансову вартість програмного забезпечення розраховуємо за формулою:

$$B_{прг} = \sum_{i=1}^{k} Ц_{iпрг} \cdot C_{прг.i} \cdot K_i, \qquad (4.13)$$

де $Ц_{iпрг}$ – ціна придбання одиниці програмного засобу даного виду, грн;

$C_{пр.i}$– кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

$K_i$ – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, ($K_i$ = *1,10…1,12);*

*k* – кількість найменувань програмних засобів.

$В_{прг}$ = 650,00 · 1 · 1,1 = 715,00 грн.

Отримані результати зведемо до таблиці:

Таблиця 4.8 – Витрати на придбання програмних засобів по кожному виду

| Найменування програмного засобу | Кількість, шт | Ціна за одиницю, грн | Вартість, грн |
|---|---|---|---|
| Доступ (підписка) для вивчення аналогу клієнт-серверної системи для прогнозування поведінки фінансових інструментів | 1 | 650,00 | 715,00 |
| Емулятор серверної інтернет-платформи для моделювання поведінки інформаційного ресурсу | 1 | 6520,00 | 7172,00 |
| Всього | | | 7887,00 |

4.3.7 Амортизація обладнання, програмних засобів та приміщень

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо, розраховуємо з використанням прямолінійного методу амортизації за формулою:

$$A_{обл} = \frac{Ц_б}{T_в} \cdot \frac{t_{вик}}{12}, \quad (4.14)$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$T_в$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

$A_{обл} = (27400{,}00 \cdot 2) / (3 \cdot 12) = 1522{,}22$ грн.

Проведені розрахунки зведемо до таблиці.

Таблиця 4.9 – Амортизаційні відрахування по кожному виду обладнання

| Найменування обладнання | Балансова вартість, грн | Строк корисного використання, років | Термін використання обладнання, місяців | Амортизаційні відрахування, грн |
|---|---|---|---|---|
| Персональний комп'ютер | 27400,00 | 3 | 2 | 1522,22 |
| Обчислювально-аналітична система програмної розробки | 25300,00 | 3 | 2 | 1405,56 |
| Ноутбук HP Laptop 15s-eq2037ua Natural Silver (422G7EA) | 34800,00 | 3 | 2 | 1933,33 |
| Робоче місце розробника програмного забезпечення | 6900,00 | 5 | 2 | 230,00 |
| Пристрій виводу інформації HP-2100 | 6600,00 | 4 | 2 | 275,00 |
| Оргтехніка | 9350,00 | 5 | 2 | 311,67 |

| | | | | |
|---|---|---|---|---|
| Приміщення лабораторії | 790000,00 | 20 | 2 | 6583,33 |
| ОС Windows 11 | 8400,00 | 2 | 2 | 700,00 |
| Прикладний пакет Microsoft Office 2019 | 7600,00 | 2 | 2 | 633,33 |
| Прикладний пакет розробки інформаційних систем (мови Java, C#, C++, Perl, PHP) | 8350,00 | 2 | 2 | 695,83 |
| Всього | | | | 14290,28 |

4.3.8 Паливо та енергія для науково-виробничих цілей

Витрати на силову електроенергію ($B_e$) розраховуємо за формулою:

$$B_e = \sum_{i=1}^{n} \frac{W_{yi} \cdot t_i \cdot Ц_e \cdot K_{вni}}{\eta_i}, \quad (4.15)$$

де $W_{yi}$ – встановлена потужність обладнання на визначеному етапі розробки, кВт;

$t_i$ – тривалість роботи обладнання на етапі дослідження, год;

$Ц_e$ – вартість 1 кВт-години електроенергії, грн; приймемо $Ц_e$ = 6,15 грн;

$K_{вni}$ – коефіцієнт, що враховує використання потужності, $K_{вni} < 1$;

$\eta_i$ – коефіцієнт корисної дії обладнання, $\eta_i < 1$.

$B_e = 0,02 \cdot 200,0 \cdot 6,15 \cdot 0,95 / 0,97 = 24,60$ грн.

Проведені розрахунки зведемо до таблиці.

Таблиця 4.10 – Витрати на електроенергію

| Найменування обладнання | Встановлена потужність, кВт | Тривалість роботи, год | Сума, грн |
|---|---|---|---|
| RouterBOARD TP-Link230 | 0,02 | 200,0 | 24,60 |
| Серверне обладнання на основі EOM DELUX F43-A71BC | 0,56 | 200,0 | 688,80 |
| Персональний комп'ютер | 0,25 | 240,0 | 369,00 |
| Обчислювально-аналітична система програмної розробки | 0,32 | 200,0 | 393,60 |
| Ноутбук HP Laptop 15s-eq2037ua Natural Silver (422G7EA) | 0,04 | 200,0 | 49,20 |
| Робоче місце розробника програмного забезпечення | 0,10 | 240,0 | 147,60 |
| Пристрій виводу інформації HP-2100 | 0,06 | 3,0 | 1,11 |
| Оргтехніка | 0,60 | 2,0 | 7,38 |
| Всього | | | 1681,29 |

4.3.9 Службові відрядження

Витрати за статтею «Службові відрядження» розраховуємо як 20…25% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{св} = (З_о + З_р) \cdot \frac{Н_{св}}{100\%}, \quad (4.16)$$

де $Н_{св}$ – норма нарахування за статтею «Службові відрядження», приймемо $Н_{св}$ = 20%.

$В_{св}$ = (76661,90 + 4165,04) · 20 / 100% = 16165,39 грн.

4.3.10 Витрати на роботи, які виконують сторонні підприємства, установи і організації

Витрати розраховуємо як 30…45% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{сп} = (З_о + З_р) \cdot \frac{Н_{сп}}{100\%}, \quad (4.17)$$

де $Н_{сп}$ – норма нарахування за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації», приймемо $Н_{сп}$= 45%.

$В_{сп}$ = (76661,90 + 4165,04) · 45 / 100% = 36372,12 грн.

4.3.11 Інші витрати

Витрати за статтею «Інші витрати» розраховуємо як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$I_{в} = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (4.18)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати», приймемо $Н_{ів}$ = 50%.

$І_в$ = (76661,90 + 4165,04) · 50 / 100% = 40413,47 грн.

4.3.12 Накладні (загальновиробничі) витрати

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуємо як 100…150% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (4.19)$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати», приймемо $Н_{нзв}$ = 130%.

$В_{нзв}$ = (76661,90 + 4165,04) · 130 / 100% = 105075,02 грн.

Витрати на проведення науково-дослідної роботи на тему «Розробка клієнт-серверної системи для прогнозування поведінки фінансових інструментів (Серверна частина)» розраховуємо як суму всіх попередніх статей витрат за формулою:

$$В_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + В_{спец} + В_{прг} + А_{обл} + В_е + В_{св} + В_{сп} + І_в + В_{нзв}. \quad (4.20)$$

$В_{заг}$= 76661,90 +4165,04 +8082,69 +19560,11992 +5329,11 +7313,60 +34003,20 + 7887,00 + 14290,28 +1681,29 +16165,39 +36372,12 +40413,47 +105075,02 = 377000,24 грн.

Загальні витрати *ЗВ* на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховується за формулою:

$$ЗВ = \frac{В_{заг}}{\eta}, \quad (4.21)$$

де $\eta$ - коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи, приймемо $\eta$=0,9.

ЗВ = 377000,24 / 0,9 = 418889,15 грн.

## 4.4 Розрахунок економічної ефективності науково-технічної розробки при її можливій комерціалізації потенційним інвестором

Результати дослідження проведені за темою «Розробка клієнт-серверної системи для прогнозування поведінки фінансових інструментів (Серверна частина)» передбачають комерціалізацію протягом 4-х років реалізації на ринку.

В цьому випадку основу майбутнього економічного ефекту будуть формувати:

$\Delta N$ – збільшення кількості споживачів яким надається відповідна інформаційна послуга у періоди часу, що аналізуються;

| Показник | 1-й рік | 2-й рік | 3-й рік | 4-й рік |
|---|---|---|---|---|
| Збільшення кількості споживачів, осіб | 4500 | 9000 | 12000 | 7000 |

$N$ – кількість споживачів яким надавалась відповідна інформаційна послуга у році до впровадження результатів нової науково-технічної розробки, приймемо 12000 осіб;

$Ц_б$ – вартість послуги у році до впровадження інформаційної системи, приймемо 700,00 грн;

$\pm\Delta Ц_о$ – зміна вартості послуги від впровадження результатів, приймемо 85,55 грн.

Можливе збільшення чистого прибутку у потенційного інвестора $\Delta П_i$ для кожного із 4-х років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховуємо за формулою[52]:

$$\Delta П_i = (\pm\Delta Ц_о \cdot N + Ц_о \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \qquad (4.22)$$

де $\lambda$ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2022 році ставка податку на додану вартість складає 20%, а коефіцієнт $\lambda = 0,8333$;

$\rho$ – коефіцієнт, який враховує рентабельність інноваційного продукту). Приймемо $\rho$ =42%;

$\vartheta$ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2022 році $\vartheta = 18\%$;

Збільшення чистого прибутку 1-го року:

$\Delta П_1 = (85{,}55 \cdot 12000{,}00 + 785{,}55 \cdot 4500) \cdot 0{,}83 \cdot 0{,}42 \cdot (1\text{-}0{,}18/100\%) = 1303935{,}34$ грн.

Збільшення чистого прибутку 2-го року:

$\Delta П_2 = (85{,}55 \cdot 12000{,}00 + 785{,}55 \cdot 13500) \cdot 0{,}83 \cdot 0{,}42 \cdot (1\text{-}0{,}18/100\%) = 3324894{,}68$ грн.

Збільшення чистого прибутку 3-го року:

$\Delta П_3 = (85{,}55 \cdot 12000{,}00 + 785{,}55 \cdot 25500) \cdot 0{,}83 \cdot 0{,}42 \cdot (1\text{-}0{,}18/100\%) = 6019507{,}15$ грн.

Збільшення чистого прибутку 4-го року:

$\Delta П_4 = (85{,}55 \cdot 12000{,}00 + 785{,}55 \cdot 32500) \cdot 0{,}83 \cdot 0{,}42 \cdot (1\text{-}0{,}18/100\%) = 7591364{,}42$ грн.

Приведена вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \qquad (4.23)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

$T$ – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки;

$\tau$ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, $\tau$=0,25;

$t$ – період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$ПП = 1303935,34/(1+0,25)^1+3324894,68/(1+0,25)^2+6019507,15/(1+0,25)^3+$
$+7591364,42/(1+0,25)^4=1043148,27+2127932,60+3081987,66+3109422,87=$
$=9362491,39$ грн.

Величина початкових інвестицій $PV$, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки:

$$PV = k_{інв} \cdot ЗВ, \tag{4.24}$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, приймаємо $k_{інв}$=2;

$ЗВ$ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, приймаємо 418889,15 грн.

$PV = k_{інв} \cdot ЗВ$= 2 · 418889,15 = 837778,31 грн.

Абсолютний економічний ефект $E_{абс}$ для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV \tag{4.25}$$

де $ПП$ – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, 9362491,39 грн;

$PV$ – теперішня вартість початкових інвестицій, 837778,31 грн.

$E_{абс} = ПП - PV$ = 9362491,39 - 837778,31 = 8524713,09 грн.

Внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$E_в = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1, \quad (4.26)$$

де $E_{абс}$ – абсолютний економічний ефект вкладених інвестицій, 8524713,09 грн;

$PV$ – теперішня вартість початкових інвестицій, 837778,31 грн;

$T_ж$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, 4 роки.

$$E_в = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1 = (1+8524713,09/837778,31)^{1/4} = 0,83.$$

Мінімальна внутрішня економічна дохідність вкладених інвестицій $\tau_{мін}$:

$$\tau_{мін} = d + f, \quad (4.27)$$

де $d$ – середньозважена ставка за депозитними операціями в комерційних банках; в 2022 році в Україні $d$ =0,11;

$f$ – показник, що характеризує ризикованість вкладення інвестицій, приймемо 0,22.

$\tau_{мін}$= 0,11+0,22 = 0,33 < 0,83 свідчить про те, що внутрішня економічна дохідність інвестицій $E_в$, вища мінімальної внутрішньої дохідності. Тобто інвестувати в науково-дослідну роботу за темою «Розробка клієнт-серверної системи для прогнозування поведінки фінансових інструментів (Серверна частина)» доцільно.

Період окупності інвестицій $T_{ок}$ які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$T_{ок} = \frac{1}{E_в}, \quad (4.28)$$

де $E_в$ – внутрішня економічна дохідність вкладених інвестицій.

$T_{ок}$ = 1 / 0,83 = 1,21 р.

$T_{ок}$ <3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

## 4.5 Висновки до розділу

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Розробка клієнт-серверної системи для прогнозування поведінки фінансових інструментів (Серверна частина)» становить 38,3 бала, що, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки вище середнього).

При оцінюванні рівня конкурентоспроможності, згідно узагальненого коефіцієнту конкурентоспроможності розробки, науково-технічна розробка переважає існуючі аналоги приблизно в 9,64 рази.

Також термін окупності становить 1,21 р., що менше 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Отже можна зробити висновок про доцільність проведення науково-дослідної роботи за темою «Розробка клієнт-серверної системи для прогнозування поведінки фінансових інструментів (Серверна частина)».

# ВИСНОВКИ

В даній роботі була розглянута актуальність систем для передбачення поведінки банківських інструментів та перспективи їх використання.

Обґрунтовано вибір мови програмування для розв’язання поставленої задачі. Проаналізовано можливість реалізації зв’язку між клієнтом та сервером за допомогою використання архітектурного стилю REST API.

Розглянуто реалізацію системи із використанням технологій .NET Core, ASP.NET Core, EntityFramework Core, PostgreSQL.

Було описано реалізацію принципу впровадження залежностей у додатках, що використовують .NET Core.

Були наведені основні етапи та технології розробки серверної частини. Було розглянуто підхід реалізації архітектури системи серверної частини кілєнт-серверного додатку на базі .NET Core. Було проведено аналіз архітектури системи, та кожної із її частин. Здійснено аналіз основних технологій, що були використані для розробки клієнтської частини системи.

В результаті даної роботи було розроблено та протестовано клієнт-серверну систему для підтримки банківських транзакцій за допомогою існуючих технологій для створення клієнт-серверних систем.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Курніцький Д. П., Розробка клієнт-серверної системи для підтримки банківських транзакцій / Д. П. Курніцький // Матеріали доповідей L Науково-технічної конференції підрозділів Вінницького національного технічного університету, 16 березня. – Вінниця : ВНТУ, 2021.

2. Поняття клієнт-серверних систем [Електронний ресурс]. – Режим доступу: http://bourabai.kz/dbt/client1.html/ - Назва з екрану

3. Ranade J. Client/Server Architecture / Ranade J. – Boston: Mcgraw-Hill, 1992. - 452 p. – ISBN - 978-0070050761.

4. Клиент-серверная архитектура приложения [Електронний ресурс]. – Режим доступу: http://fkn.ktu10.com/?q=node/9330 - Назва з екрану.

5. Kyle M. Learning Material Design / Kyle M. – New York: Pack Publishing, 2015. – 186 p. – ISBN - 978-1785289811.

6. Serving Web Content with Spring MVC [Електронний ресурс]. – Режим доступу: https://spring.io/guides/gs/serving-web-content/ - Назва з екрану.

7. TutorialsTeatcher. MVC Architecture [Електронний ресурс]. – Режим доступу: https://www.tutorialsteacher.com/mvc/mvc-architecture - Назва з екрану.

8. Сравнение систем мониторинга Zabbix vs Nagios [Електронний ресурс]. – Режим доступу: https://www.netping.ru/Blog/sravnenie-sistem-monitoringa-zabbix-vs-nagios - Назва з екрану.

9. Barth W. Nagios, 2nd Edition: System and Network Monitoring / Barth W. – London: No Starch Press, 2015. – 719 p. – ISBN - 978-1593271794.

10. Olups R. Zabbix Network Monitoring / Olups R. – Ltd: Packt Publishing, 2016. – 754 p. – ISBN - 978-1782161295.

11. Анализ юзабилити сайта, изучение внешнего вида и эффективности работы сайта (usability) [Електронний ресурс]. – Режим доступу: http://www.m-mix.com.ua/marketing/analiz-usability.html – Назва з екрану.

12. Веб зсередини [Електронний ресурс]. – Режим доступу: http://www.zhu.edu.ua/mk_school/mod/page/view.php – Назва з екрану.

13. Веб-програмування. HTML5 [Електронний ресурс]. – Режим доступу: http://webstudio2u.net/ua/programming/489-html5-css3.html. – Назва з екрану.

14. Новые теги HTML5.0 [Електронний ресурс]. – Режим доступу: http://www.wisdomweb.ru/HTML5d/ – Назва з екрану.

15. Goldstein Alexis HTML5 & CSS3 ForTheRealWorld / Alexis Goldstein, Louis Lazaris, Estelle Weyl. – NY.: Science, 2015. – 350 p. – ISBN 978-0-9874674-9-2.

16. Статті / CSS [Електронний ресурс]. – Режим доступу: http://www.webostudio.com/ua/stats/CSS/ – Назва з екрану.

17. Настройки CSS, стилизация HTML-элементов с использованием расширенных классов, и передовая система разметки [Електронний ресурс]. – Режим доступу: http://bootstrap-3.ru/css.php – Назва з екрану.

18. Роббинс Дж. HTML5. Карманный справочник / Роббинс Дж. – М.: ВИЛЬЯМС, 2015. – 192с. – ISBN: 978-5-8459-1937-3.

19. Основні компоненти мови UML [Електронний ресурс]. – Режим доступу: http://um.co.ua/9/9-2/9-29941.html – Назва з екрану.

20. Spring Security [Електронний ресурс]. – Режим доступу: http://projects.spring.io/spring-security/ - Назва з екрану.

21. Refactoring Book / М. Фаулер, К. Бек, Д. Брант, У. Апдайк. – М.: Вильямс. – 2017. – 63 - 78 с. – ISBN 5-93286-045-6

22. Pro Angular. Second Edition / Adam Freeman – A.:Apress, 2017. – 788 c. – ISBN 978-1-4842-2307-9;

23. AngularJS Tutorial: A Comprehensive 10000 World Guide [Електронний ресурс]. – Режим доступу: https://www.airpair.com/angularjs – Назва з екрану.

24. AngularJS – MVC архітектура [Електронний ресурс]. – Режим доступу: http://thewebland.net/development/javascript/angularjs/angularjs-mvc/ – Назва з екрану.

25. Внедрение компонентого подхода в вебе: обзор веб – компонентов [Електронний ресурс]. – Режим доступу: https://habrahabr.ru/company/microsoft/blog/264791/ – Назва з екрану.

26. Компонентный подход к созданию приложения с помощью AngularJS 1.5 & 2.0 [Електронний ресурс]. – Режим доступу: http://bogdanovblog.ru/komponentnyj-podhod-k-sozdaniyu-prilozheniya-s-pomoshhyu-angularjs-1-5- 2-0/ – Назва з екрану.

27. Каскадные таблицы стилей [Електронний ресурс]. – Режим доступу: http://www.4stud.info/web-programming/css.html – Назва з екрану.

28. Software Testing Fundamentals [Електронний ресурс]. – Режим доступу: http://softwaretestingfundamentals.com/defect/ – Назва з екрану.

29. Unit tests with Mockito – Tutorial [Електронний ресурс]. – Режим доступу: http://www.vogella.com/tutorials/Mockito/article.html – Назва з екрану.

30. Документація Java по фреймворку Mockito 2017 [Електронний ресурс]. – Режим доступу: http://static.javadoc.io/org.mockito/mockitocore/1.10.19/org/mockito – Назва з екрану.

31. Spring Framework Overview [Електронний ресурс]. – Режим доступу: http://www.tutorialspoint.com/spring/spring_overview.htm - Назва з екрану.

32. Балансировка нагрузки: основные алгоритмы и методы [Електронний ресурс]. – Режим доступу: https://habr.com/ru/company/selectel/blog/250201/ – Назва з екрану.

33. Round-Robin Game Scheduling [Електронний ресурс]. – Режим доступу: http://gilmerbaseball.com/library/roundrobin.asp – Назва з екрану.

34. Amazon EC2 T3 Instances [Електронний ресурс]. – https://aws.amazon.com/ec2/instance-types/t3/ – Назва з екрану.

35. Brown N. Amazon Elastic Compute Cloud (EC2): Guide for Beginners. Easy to Understand / Nicholas Brown, – South Carolina: CreateSpace Independent Publishing Platform, 2017. – 74 с. – ISBN – 978-1542885621.

36. Broyles P. Amazon Web Services. The Ultimate Guide for Beginners, Intermediates and Expert / Phillip Broyles, - KDP Print US: Amazon Digital Services LLC, 2020. – 122 c. – ISBN – 979-8601352512.

37. Blokdyk G. Load Balancer: A Complete Guide / Gerardus Blokdyk, – South Carolina: CreateSpace Independent Publishing Platform, 2018. – 110 c. – ISBN – 978-1718616219.

38. Kopparapu C. Load Balancing Servers, Firewalls and Caches / Chandra Kopparapu, - Hoboken: John Wiley & Sons, 2002. – 224 c. – ISBN – 978-0471421283.

39. Virtual Instance. Future Trends and Research Directions [Електронний ресурс]. – https://www.sciencedirect.com/topics/computer-science/virtual-instance – Назва з екрану.

40. Пасічник В.В. Глобальні інформаційні системи та технології: моделі ефективного аналізу, опрацювання та захисту даних. Монографія / В.В. Пасічник, П. І. Жежнич, Р. Б. Кравець, А. М. Пелещишин, Д. О. Тарасов – Львів: Видавництво Львівської політехніки, 2006. – 348 с. ISBN: 966-553-578-1.

41. What is client-server architecture / Що таке клієнт-серверна архітектура [Електронний ресурс]. – Режим доступу: http://apachebooster.com/kb/what-is-client-server-architecture-and-what-are-its-types/ - Назва з екрану.

42. RESTful API Design [Електронний ресурс] – Режим доступу: https://hackernoon.com/restful-api-design-step-by-step-guide-2f2c9f9fcdbf – Назва з екрану.

43. Robert M. Clean Architecture: Guide to Software Structure and Design / Robert M. – Washington: Pearson 2017. – 432 p. ISBN 978-0134494166.

44. Clean Code Blog [Електронний ресурс] – Режим доступу: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html - Назва з екрану.

45. Use Case Diagrams [Електронний ресурс] – Режим доступу: https://www.smartdraw.com/use-case-diagram/ - Назва з екрану.

46. Regina O. PostgreSQL: Up and Running / Regina O. - Sebastopol: O'Reilly Media 2012. – 168 p. – ISBN 978-1449326333.

47. Robert M. Clean Code: A Handbook of Agile Software Craftsmanship / Robert M. – Washington: Pearson 2008. – 464 p. – ISBN 978-0132350884.

48. Ajit K. Sencha MVC Architecture / Ajit K. – Birmingham: Packt Publishing 2012. -126 p. – ISBN 978-1849518888.

49. Mark M. REST API Design Rulebook / Mark M. – Sebastopol: O'Reilly Media, 2011. -116 p. – ISBN 978-1449310509.

50. Методичні рекомендації з комерціалізації розробок, створених в результаті науково-технічної діяльності – К.: Наказ Державного комітету України з питань науки, інновацій та інформатики (Лист № 1/06-4-97 від 13.09.2010 р.).

51. Козловський В. О. Методичні вказівки до виконання студентами-магістрантами економічної частини магістерських кваліфікаційних робіт. – Вінниця: ВНТУ, 2012

52. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. – Вінниця : ВНТУ, 2021. – 42 с.

53. Кавецький В. В. Економічне обґрунтування інноваційних рішень: практикум / В. В. Кавецький, В. О. Козловський, І. В. Причепа – Вінниця : ВНТУ, 2016. – 113 с.

# ДОДАТКИ

Додаток А

(обов’язковий)

Технічне завдання

ЗАТВЕРДЖЕНО

Зав. кафедри АІІТ

_____ Бісікало О.В.

«__»_____ 2022 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на магістерську кваліфікаційну роботу

«Система прийняття рішень щодо підбору віртуальної машини на основі моніторингу інтернет-сервісів»

08-02.МКР.000.00.017 ТЗ

Керівник роботи:

к.т.н., доц каф АІІТ., Коцюбинський В.Ю.

“___” _____ 2022 р.

Виконавець: ст. гр. 1ІСТ-21м

Курніцький Д.П.

“___” _____ 2022 р.

Вінниця 2022

1. Назва та галузь застосування

«Клієнт-серверна система передбачення поведінки фінансових інструментів» - призначена для застосування в галузі автоматизованого моніторингу та масштабування серверів з метою покращення якості моніторингу інтернет сервісів, автоматизації процесів підбору конфігурацій та створення віртуальних машин, а також зменшення впливу людського фактору.

2. Підстави для розробки

Розробку системи здійснювати на підставі наказу по університету № 214 від 25.09.2020 та завдання до магістерської кваліфікаційної роботи, складеного та затвердженого кафедрою «Автоматизації та інтелектуальних інформаційних технологій».

3. Мета та призначення розробки

Метою даної роботи є підвищення ефективності прогнозування поведінки фінансових інструментів, а саме портфелів цінних паперів.

4. Джерела розробки

1. Пасічник В.В. Глобальні інформаційні системи та технології: моделі ефективного аналізу, опрацювання та захисту даних. Монографія / В.В. Пасічник, П. І. Жежнич, Р. Б. Кравець, А. М. Пелещишин, Д. О. Тарасов – Львів: Видавництво Львівської політехніки, 2006. – 348 с. ISBN: 966-553-578-1.

2. Поняття клієнт-серверних систем [Електронний ресурс]. – Режим доступу: http://bourabai.kz/dbt/client1.html/ - Назва з екрану

3. Ranade J. Client/Server Architecture / Ranade J. – Boston: Mcgraw-Hill, 1992. - 452 p. – ISBN - 978-0070050761.

4. Brown N. Amazon Elastic Compute Cloud (EC2): Guide for Beginners. Easy to Understand / Nicholas Brown, – South Carolina: CreateSpace Independent Publishing Platform, 2017. – 74 с. – ISBN – 978-1542885621.

5. Показники призначення

Клієнт-серверна система для передбачення поведінки фінансових інструментів має забезпечувати коректний збір даних інвкстиційних портфелів від користувачів, валідацію цих даних та коректний обрахунок передбачення.

Вхідні дані: дані інвестиційного портфелю (компоненти Revenue, Expenses, Cash, Segments), відношення змінних.

Вихідні дані: майбутні дані певних головних змінних у відношенні.

6. Економічні показники

- прогнозовані витрати на розробку – не більше 420 тис. грн;
- термін окупності витрат для інвестора – не більше 3 років.

7. Стадії розробки

1. Розділ 1 «Аналіз та порівняння існуючих систем та методів прийняття рішень» має бути виконаний до 29.10.2022.

2. Розділ 2 «Проектування архітектури системи та вибір технологій реалізації» має бути виконаний до 03.11.2022.

3. Розділ 3 «Розробка програмного забезпечення» має бути виконаний до 15.11.2022.

4. Економічний розділ має бути виконаний до 30.11.2022.

7. Порядок контролю та приймання

1. Рубіжний контроль провести до 15.11.2022

2. Попередній захист магістерської кваліфікаційної роботи провести до 01.12.2022.

3. Захист магістерської кваліфікаційної роботи провести в період з 07.12.2022 до 06.01.2022.

Додаток Б

(обов'язковий)

Графічна частина

Зав. кафедри АІІТ _____ д-р техн. наук, професор каф. АІІТ

Бісікало О.В.

(підпис) (науковий ступінь, вчене звання, ініціали та прізвище)

Науковий керівник _____ канд. техн. наук, доцент каф. АІІТ

Коцюбинський В. Ю.

(підпис) (науковий ступінь, вчене звання, ініціали та прізвище)

Тех. контроль _____ канд. техн. наук, доцент каф. АІІТ

Коцюбинський В. Ю.

(підпис) (науковий ступінь, вчене звання, ініціали та прізвище)

Нормоконтроль _____ канд. техн. наук, доцент каф. АІІТ

Маслій Р.В.

(підпис) (науковий ступінь, вчене звання, ініціали та прізвище)

Рецензент _____ канд. техн. наук, доцент каф. КСУ

Биков М. М.

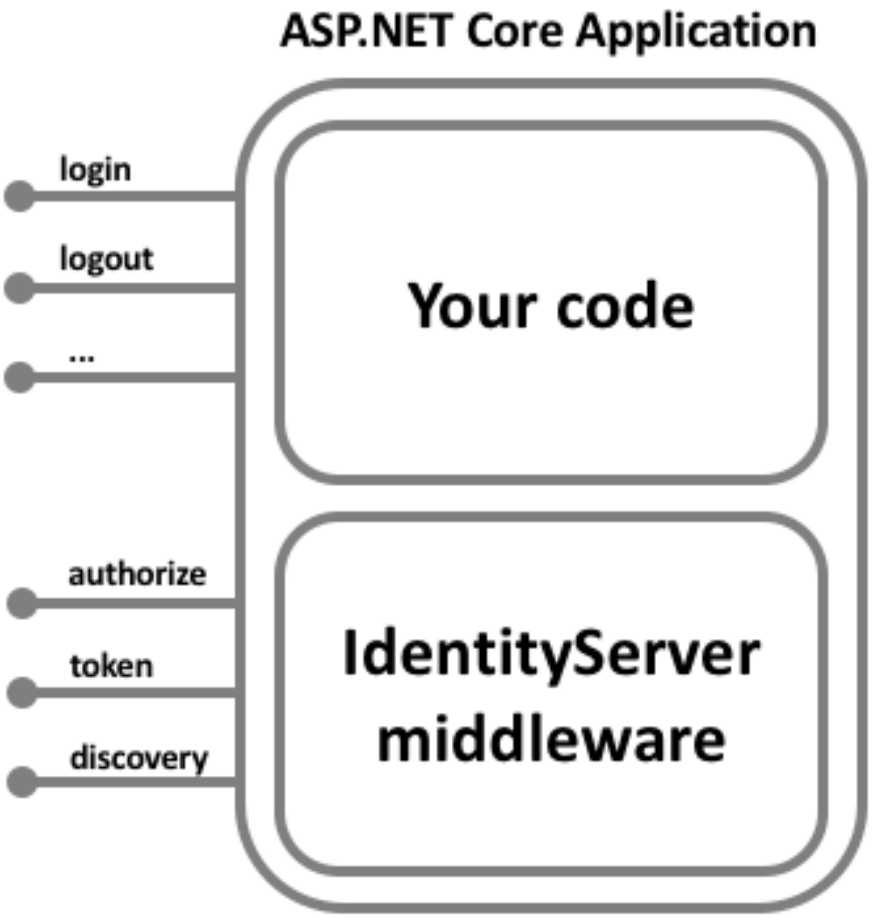
(підпис) (науковий ступінь, вчене звання, ініціали та прізвище)

Студент гр. 1ІСТ-21м _____ Курніцький Д.П.

(підпис) (ініціали та прізвище)

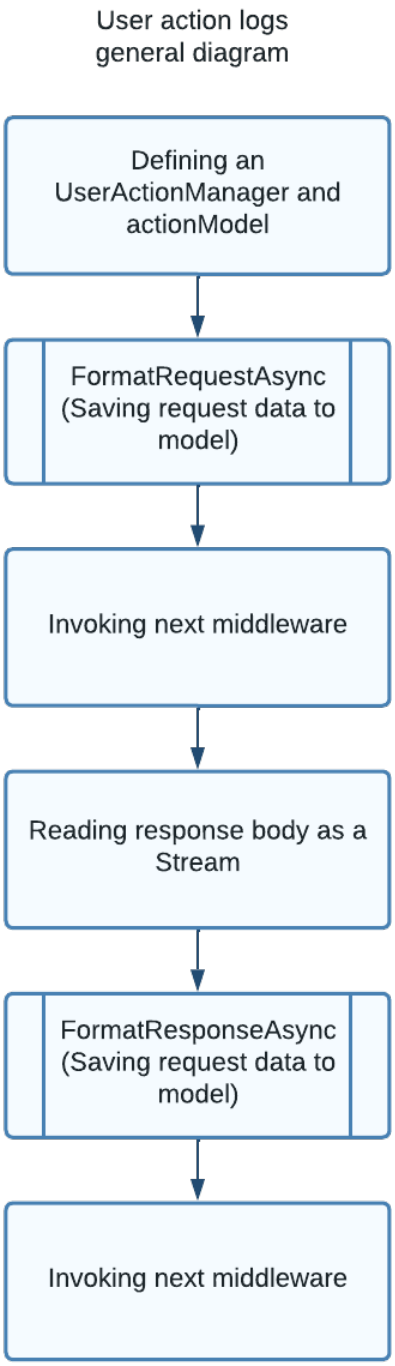
Додаток В

Схема реалізації сервісу авторизації



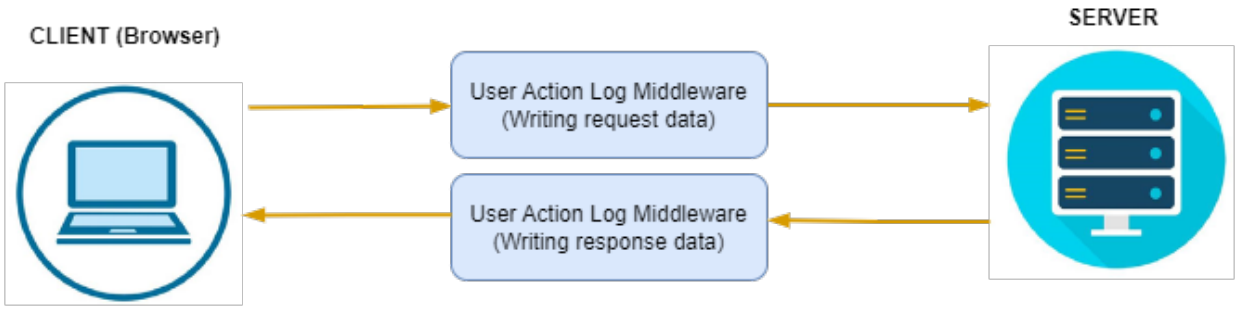
# Додаток Г

## UML – flow діаграма системи логування дій користувачів



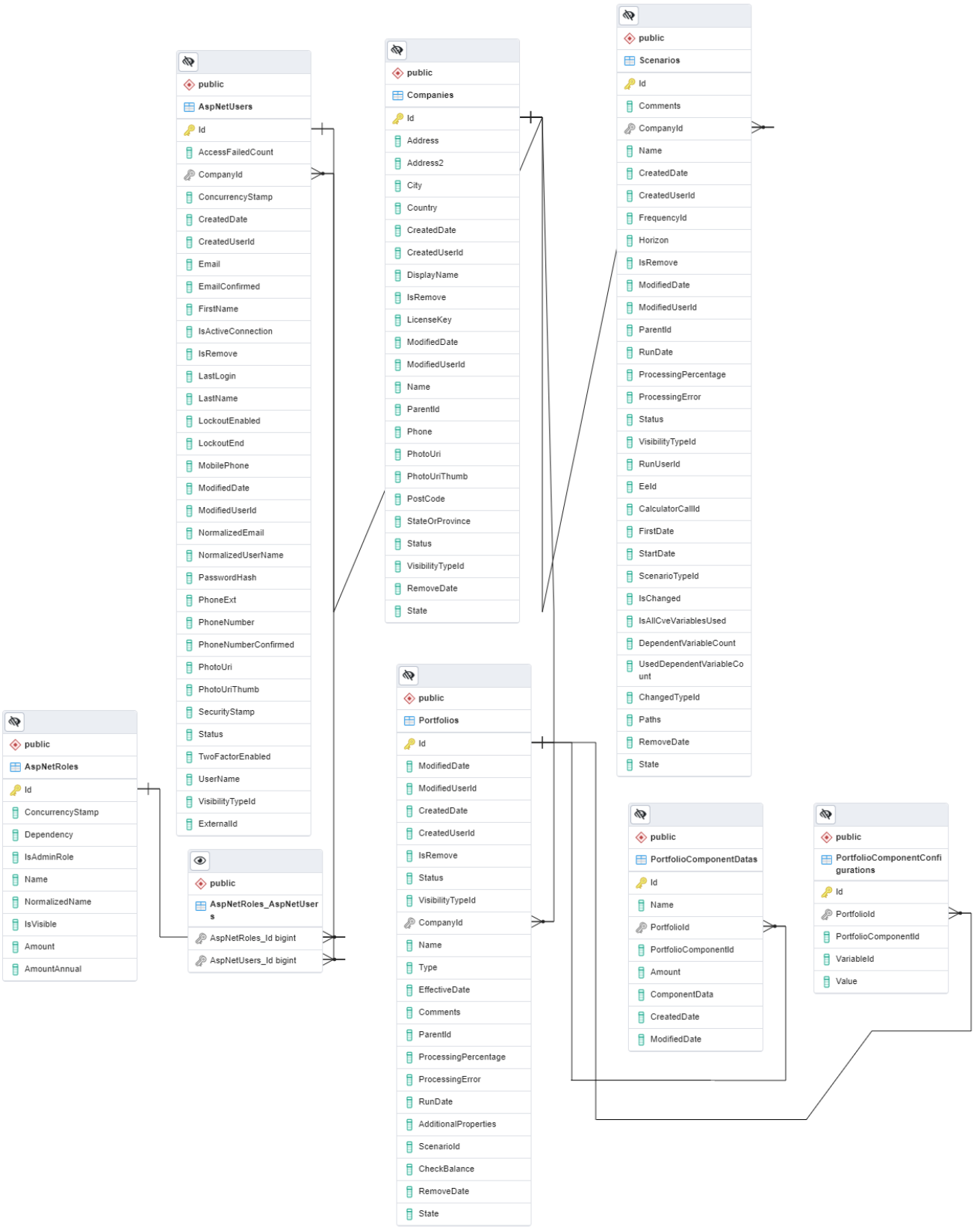
# Додаток Д

## Діаграма принципу роботи системи логування дій користувачів



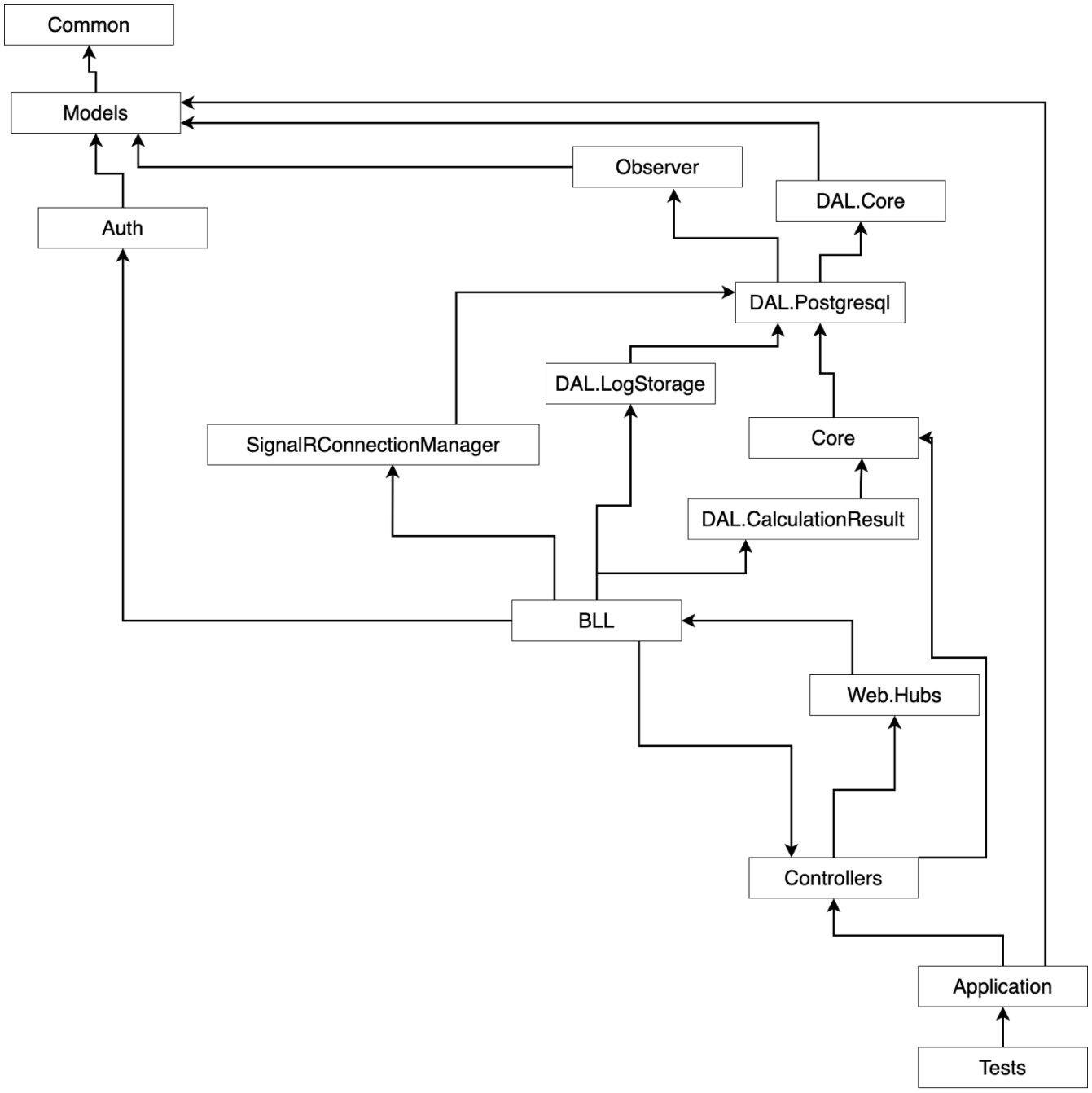
# Додаток Е

## Схема бази даних проекту



# Додаток Є

## UML-діаграма структури проекту



Додаток Ж

(обов'язковий)

Акт впровадження

Науково-виробниче підприємство
**"СПІЛЬНА СПРАВА"** Товариство з обмеженою відповідальністю
Україна, 21000, м. Вінниця, вул. 1 Травня, 36/3, тел. +380 (67) 974 73 77
код ЄДРПОУ 31041670, код платників податків 310416702282, свідоцтво № 0183329
IBAN : UA 41 322313 0000026004000003226 в філії АТ «Укрексімбанк» в м. Вінниці

**Затверджую**

**Директор**

**НВП «СПІЛЬНА СПРАВА», ТОВ**

_____

**Чикалова Анастасія Олександрівна**

«_____» _____ 20__ р.

**АКТ**

**впровадження результатів магістерської роботи**
**Курніцького Дмитра Петровича « Розробка клієнт-серверної системи для прогнозування поведінки фінансових інструментів(Серверна частина)»**

Комісія у складі головного спеціаліста С. Бородкіна, та керівника групи розробки інформаційних систем ,к.т.н. В. Сторчака склали цей акт про те, що у Науково-виробничому підприємстві "Спільна Справа", ТОВ впроваджуються результати, які отримані магістрантом Курніцьким Д. П. при виконанні магістерської кваліфікаційної роботи мають практичну цінність. Клієнт-серверні системи для передбачення поведінки фінансових інструментів знаходяться в стадії активного розвитку, та є дуже затребуваними у галузі фінансових технологій. З кожним роком збільшується, популярність подібних систем для передбачення поведінки фінансових інструментів. Таким чином, актуальність роботи Курніцького Д. П. не викликає сумнівів, а низка методик та підходів та результати їх застосування можуть бути застосованими у практичній діяльності.

Науково-виробничому підприємству "Спільна Справа", ТОВ Курніцьким Д. П. передано програмне забезпечення, яке дозволяє підвищити продуктивність розширень для Jira.

Члени комісії:

Головний спеціаліст. С. Бородкін

Керівник групи к.т.н. В. Сторчак

# Додаток З

## Лістинг програми розробленого модулю моніторингу інтернет сервісів

```
{
    /// <summary>
    /// Portfolio controller
    /// </summary>
    [Authorize(Roles = "SuperAdmin,Portfolio,EfficientFrontier", Policy =
"CheckRequirement")]
    [Route("api/[controller]")]
    [Produces("application/json")]
    public class PortfolioController : Core.ControllerBase
    {
        private readonly ICalculatorService _calculatorService;
        private readonly IExportService _exportService;
        private readonly IDownloadCacheService _downloadCacheService;

        /// <summary>
        /// Init portfolio controller
        /// </summary>
        /// <param name="connectionHelper">The Connection helper</param>
        /// <param name="loggerFactory">The logger</param>
        /// <param name="logicProvider">The logic provider</param>
        /// <param name="notificationHub">The notification hub</param>
        /// <param name="logicHub">The logic hub</param>
        /// <param name="notificationService">The notification service</param>
        /// <param name="calculatorService">The calculator service</param>
        /// <param name="exportService">The export service</param>
        /// /// <param name="downloadCacheService">The download service</param>
        public PortfolioController(
            IConnectionHelper connectionHelper,
            ILoggerFactory loggerFactory,
            IBusinessLogicManagerProvider logicProvider,
            IHubContext<NotificationControlHub, INotificationControlHub>
notificationHub,
            IHubContext<BusinessLogicHub, IBusinessLogicHub> logicHub,
            INotificationService notificationService,
            ICalculatorService calculatorService,
            IExportService exportService,
            IDownloadCacheService downloadCacheService) : base(connectionHelper,
logicProvider, loggerFactory, notificationHub, logicHub, notificationService,
TargetLocation.Portfolio)
        {
            _calculatorService = calculatorService;
            _exportService = exportService;
            _downloadCacheService = downloadCacheService;
        }

        /// <summary>
        /// Get Portfolios
        /// </summary>
        /// <returns>The Portfolio view model list</returns>
        [HttpGet("")]
        [UserActionLog("Portfolio", "Get Portfolios", false,
RequestSensitivityLevel.ApplicationSection)]
```

```
        public async Task<List<PortfolioViewModel>> GetPortfolios()
        {
            var allowedTypes = new List<int>();
            if (User.IsInRole(UserRole.Portfolio.ToString()))
                allowedTypes.Add((int)PortfolioType.Alternative);
            if (User.IsInRole(UserRole.InvestmentPortfolio.ToString()))
                allowedTypes.Add((int)PortfolioType.Investment);
            if (User.IsInRole(UserRole.EfficientFrontier.ToString()))
                allowedTypes.Add((int)PortfolioType.EfficientFrontier);
            if(User.IsInRole(UserRole.BalanceSheetProjection.ToString()))
               allowedTypes.Add((int)PortfolioType.BalanceSheetProjection);

            return await
ServiceProvider.LogicProvider.PortfolioManager.GetPortfolios(allowedTypes);
        }

        /// <summary>
        /// Create empty portfolio
        /// </summary>
        /// <param name="type">The portfolio type (Alternative, EfficientFrontier,
Investment). Default is Regular</param>
        /// <returns>The Portfolio view model</returns>
        [HttpPost("")]
        [UserActionLog("Portfolio", "Create empty portfolio", false,
RequestSensitivityLevel.ApplicationSection)]
        public async Task<PortfolioViewModel> CreatePortfolio(string type)
        {
            try
            {
                CheckPortfolioTypeRoles(type);
                var vm = await
ServiceProvider.LogicProvider.PortfolioManager.CreatePortfolio(type.GetEnum<Portfol
ioType>());
                await LogicHub(vm.CompanyId,
vm.PortfolioTypeEnum.Value).PortfolioCreated(vm, string.Empty);
                return vm;
            }
            catch (Exception ex)
            {
                ServiceProvider.Logger.LogErrorCreate(ex, CurrentUserId, null);
                throw;
            }
        }

        /// <summary>
        /// Check portfolio balance
        /// </summary>
        /// <param name="id">The portfolio Id</param>
        /// <returns>The portfolio object</returns>
        [HttpGet("{id:long}/balance")]
        [UserActionLog("Portfolio", "Check portfolio balance", false,
RequestSensitivityLevel.ApplicationSection)]
        public async Task<bool> CheckBalance(long id)
        {
            return await
ServiceProvider.LogicProvider.PortfolioManager.CheckBalance(id);
        }

        /// <summary>
```

```
        /// Get Portfolio
        /// </summary>
        /// <param name="id">The portfolio Id</param>
        /// <returns>The portfolio object</returns>
        [HttpGet("{id:long}")]
        [UserActionLog("Portfolio", "Get Portfolio", false,
RequestSensitivityLevel.ApplicationSection)]
        public async Task<PortfolioViewModel> GetPortfolio(long id)
        {
            await ValidatePortfolioBeforeAction(id);
            return await
ServiceProvider.LogicProvider.PortfolioManager.GetPortfolio(id);
        }

        /// <summary>
        /// Get portfolio configuration
        /// </summary>
        /// <returns>The Portfolio configuration view model</returns>
        [HttpGet("configuration")]
        [UserActionLog("Portfolio", "Get portfolio configuration", false,
RequestSensitivityLevel.InputOrResult)]
        public async Task<PortfolioConfigurationViewModel>
GetPortfolioConfiguration(long portfolioId)
        {
            await ValidatePortfolioBeforeAction(portfolioId);
            return await
ServiceProvider.LogicProvider.PortfolioManager.GetPortfolioConfiguration(portfolioI
d);
        }

        /// <summary>
        /// Set portfolio configuration
        /// </summary>
        /// <returns>The Portfolio component type view model</returns>
        [HttpPost("configuration")]
        [UserActionLog("Portfolio", "Set portfolio configuration", false,
RequestSensitivityLevel.InputOrResult)]
        public async Task<PortfolioComponentTypeViewModel>
SetPortfolioConfiguration(long portfolioId, byte componentId)
        {
            await ValidatePortfolioBeforeAction(portfolioId);
            SettingViewModel balanceSheetRiskWeights = null;
            if (PortfolioComponent.GetComponent(componentId).Name.Equals("Risk
Weights"))
            {
                var ratings = await
ServiceProvider.LogicProvider.VariablesManager.GetMappingField(CurrentCompanyId, m
=> m.Ratings);
                var settings = await
ServiceProvider.LogicProvider.SettingManager.GetUserCompanySettings(
                    true, UserCompanySettingCategory.BalanceSheet,
CurrentCompanyId);

settings.CompanySettings[AppSettingType.BalanceSheetConfiguration.ToString()] =

UserCompanySetting.ValidateBalanceSheetconfigurationValue((int)AppSettingType.Balan
ceSheetConfiguration,

settings.CompanySettings[AppSettingType.BalanceSheetConfiguration.ToString()],
```

```
ratings, true);
                balanceSheetRiskWeights = settings;
            }
            return await
ServiceProvider.LogicProvider.PortfolioManager.SetPortfolioComponent(portfolioId,
componentId, balanceSheetRiskWeights);
        }

        /// <summary>
        /// Link scenario to portfolio
        /// </summary>
        /// <returns></returns>
        [HttpPut("link")]
        [UserActionLog("Portfolio", "Link scenario to portfolio", false,
RequestSensitivityLevel.ApplicationSection)]
        public async Task<bool> LinkScenarioToPortfolio(long portfolioId, long
scenarioId)
        {
            try
            {
                await ValidatePortfolioBeforeAction(portfolioId);
                var result = await
ServiceProvider.LogicProvider.PortfolioManager.LinkScenarioToPortfolio(portfolioId,
scenarioId, await GetRequiredVariables());
                await LogicHub(result.CompanyId,
result.PortfolioTypeEnum.Value).ScenarioLinked(result, string.Empty);
                await _recalculateLinkedScenariosStatus(result);
                await LogicHub(result.CompanyId,
result.PortfolioTypeEnum.Value).PortfolioUpdated(result, string.Empty);
            }
            catch (Exception ex)
            {
                ServiceProvider.Logger.LogError(ex, $"Error while assigning
scenarioId {scenarioId} to portfolioId {portfolioId}");
                throw;
            }
            return true;
        }

        private async Task _recalculateLinkedScenariosStatus(PortfolioViewModel
portfolio)
        {
            var dataValues =
ServiceProvider.LogicProvider.VariablesManager.GetPortfolioComponentDatasList();
            for (var i = 0; i < portfolio.Subs.Count; i++)
            for (var j = 0; j < portfolio.Subs[i].Subs.Count; j++)
                portfolio.Subs[i] = await
ServiceProvider.LogicProvider.PortfolioManager

.CalculateLinkedScenarioToPortfolioStatus(portfolio.Subs[i].Subs[j].Id,
dataValues.RequiredVariables, CurrentCompanyId);
        }

        /// <summary>
        /// Get portfolio link view model
        /// </summary>
        /// <returns></returns>
        [HttpGet("link")]
        [UserActionLog("Portfolio", "Get portfolio link view model", false,
```

```
RequestSensitivityLevel.ApplicationSection)]
        public async Task<PortfolioScenarioLinkViewModel>
GetLinkScenarioToPortfolioModel(long portfolioId)
        {
            try
            {
                await ValidatePortfolioBeforeAction(portfolioId);
                return await ServiceProvider.LogicProvider.PortfolioManager
                    .GetLinkPortfolioScenarioViewModel(portfolioId,
User.IsInRole("EconomicEnvironment"),
                    GetAvailableRelationshipVariables(), await
GetRequiredVariables());
            }
            catch (Exception ex)
            {
                ServiceProvider.Logger.LogError(ex, $"Error while get link view
model portfolioId {portfolioId}");
                throw;
            }
        }

        /// <summary>
        /// Update Portfolio
        /// </summary>
        /// <param name="portfolioId">The portfolio id</param>
        /// <param name="model">The update view model</param>
        /// <returns></returns>
        [HttpPut("{portfolioId:long}")]
        [UserActionLog("Portfolio", "Update Portfolio", false,
RequestSensitivityLevel.ApplicationSection)]
        public async Task<bool> Update(long portfolioId, [FromBody] UpdateViewModel
model)
        {
            try
            {
                await ValidatePortfolioBeforeAction(portfolioId);
                var result = await
ServiceProvider.LogicProvider.PortfolioManager.UpdateAsyncObserved(portfolioId,
model.Field, model.Value);
                if (result == null)
                    throw new PortfolioHasNotBeenUpdatedException();
                await LogicHub(result.CompanyId,
result.PortfolioTypeEnum.Value).PortfolioUpdated(result, string.Empty);
            }
            catch (Exception ex)
            {
                ServiceProvider.Logger.LogErrorUpdate(ex, CurrentUserId,
portfolioId, model);
                throw;
            }
            return true;
        }

        /// <summary>
        /// Clone Portfolio
        /// </summary>
        /// <param name="portfolioId">The portfolio id</param>
        /// <param name="type">Type of the portfolio to clone, e.g.: Alternative,
Efficient Frontier</param>
```

```
        /// <returns></returns>
        [HttpPut("copy/{portfolioId:long}")]
        [UserActionLog("Portfolio", "Clone Portfolio", false,
RequestSensitivityLevel.ApplicationSection)]
        public async Task<bool> ClonePortfolio(long portfolioId, string type =
null)
        {
            try
            {
                await ValidatePortfolioBeforeAction(portfolioId);
                ValidateNewPortfolioTypeBeforeCloning(type);
                var result =
                    await
ServiceProvider.LogicProvider.PortfolioManager.ClonePortfolioAsyncObserved(portfoli
oId, type);
                await _recalculateLinkedScenariosStatus(result);
                await LogicHub(result.CompanyId,
result.PortfolioTypeEnum.Value).PortfolioCloned(result, string.Empty);
                await UpdatePortfolioStatus(result.Id);
            }
            catch (Exception ex)
            {
                ServiceProvider.Logger.LogErrorCreate(ex, CurrentUserId,
portfolioId);
                throw;
            }

            return true;
        }

        /// <summary>
        /// Unassign scenarios from portfolio
        /// </summary>
        /// <returns></returns>
        [HttpDelete("unlink")]
        [UserActionLog("Portfolio", "Unassign scenarios from portfolio", false,
RequestSensitivityLevel.ApplicationSection)]
        public async Task<bool> UnLinkScenarioFromPortfolio([FromQuery]long
portfolioId, [FromQuery]long scenarioId)
        {
            try
            {
                await ValidatePortfolioBeforeAction(portfolioId);
                var result = await
ServiceProvider.LogicProvider.PortfolioManager.UnLinkScenarioFromPortfolio(portfoli
oId, scenarioId);
                await LogicHub(CurrentCompanyId).ScenarioUnlinked(portfolioId,
result.RelationshipId, result.ScenarioId, string.Empty);
            }
            catch (Exception ex)
            {
                ServiceProvider.Logger.LogError(ex, $"Error while assigning
scenarioId {scenarioId} from portfolio");
                throw;
            }
            return true;
        }

        /// <summary>
```

```
        /// Delete portfolio
        /// </summary>
        /// <returns></returns>
        [HttpDelete("delete/{portfolioId:long}")]
        [UserActionLog("Portfolio", "Delete portfolio", false,
RequestSensitivityLevel.ApplicationSection)]
        public async Task<bool> DeletePortfolio(long portfolioId)
        {
            try
            {
                await ValidatePortfolioBeforeAction(portfolioId);
                var portfolio = await
ServiceProvider.LogicProvider.PortfolioManager.GetPortfolio(portfolioId);
                var result = await
ServiceProvider.LogicProvider.PortfolioManager.DeletePortfolio(portfolioId);
                await LogicHub(CurrentCompanyId,
portfolio.PortfolioTypeEnum.Value).PortfolioDeleted(portfolioId, string.Empty);
            }
            catch (Exception ex)
            {
                ServiceProvider.Logger.LogError(ex, $"Error while deleting
portfolio {portfolioId}", portfolioId);
                throw;
            }
            return true;
        }

        /// <summary>
        /// Get portfolio component list
        /// </summary>
        /// <returns>The Portfolio</returns>
        [HttpGet("{portfolioId:long}/configuration/{componentId:int}")]
        [UserActionLog("Portfolio", "Get portfolio component list", false,
RequestSensitivityLevel.InputOrResult)]
        public async Task<PortfolioComponentsViewModel>
GetPortfolioComponentDataList([FromRoute]long portfolioId, [FromRoute]byte
componentId)
        {
            try
            {
                await ValidatePortfolioBeforeAction(portfolioId);
                SettingViewModel balanceSheetRiskWeights = null;
                if (PortfolioComponent.GetComponent(componentId).Name.Equals("Risk
Weights") || PortfolioComponent.GetComponent(componentId).Name.Equals("Other
Balance Sheet Info"))
                {
                    var ratings = await
ServiceProvider.LogicProvider.VariablesManager.GetMappingField(CurrentCompanyId, m
=> m.Ratings);
                    var settings = await
ServiceProvider.LogicProvider.SettingManager.GetUserCompanySettings(
                        true, UserCompanySettingCategory.BalanceSheet,
CurrentCompanyId);

settings.CompanySettings[AppSettingType.BalanceSheetConfiguration.ToString()] =

UserCompanySetting.ValidateBalanceSheetconfigurationValue((int)AppSettingType.Balan
ceSheetConfiguration,
```

```
settings.CompanySettings[AppSettingType.BalanceSheetConfiguration.ToString()],
ratings, true);
                    balanceSheetRiskWeights = settings;
                }
                return await
ServiceProvider.LogicProvider.PortfolioManager.GetPortfolioComponentDataList(portfo
lioId, componentId, balanceSheetRiskWeights);
            }
            catch (Exception ex)
            {
                ServiceProvider.Logger.LogError(ex, $"Error while deleting
portfolio {portfolioId}", portfolioId);
                throw;
            }
        }

        /// <summary>
        /// Get segment Id - Name Dictionary
        /// </summary>
        /// <param name="portfolioId">Id of the portfolio</param>
        /// <returns></returns>
        [HttpGet("configuration/{portfolioId:long}/getSegmentComponentData")]
        [UserActionLog("Portfolio", "Get segment Id - Name Dictionary", false,
RequestSensitivityLevel.InputOrResult)]
        public async Task<List<IdNameViewModel>>
GetEfSegmentComponentData([FromRoute] long portfolioId)
        {
            try
            {
                await ValidatePortfolioBeforeAction(portfolioId);
                return await
ServiceProvider.LogicProvider.PortfolioManager.GetEfSegmentComponentData(portfolioI
d);
            }
            catch (Exception e)
            {
                ServiceProvider.Logger.LogError(e, $"Error while fetching portfolio
segment component data {portfolioId}", portfolioId);
                throw;
            }
        }

        /// <summary>
        /// Create new portfolio component
        /// </summary>
        /// <returns>The Portfolio component data view model</returns>

[HttpPost("configuration/{portfolioId:long}/{componentId:int}/createComponentData")
]
        [UserActionLog("Portfolio", "Create new portfolio component", false,
RequestSensitivityLevel.InputOrResult)]
        public async Task<bool> CreatePortfolioComponentData([FromRoute]long
portfolioId, [FromRoute]byte componentId, [FromBody]PortfolioComponentDataViewModel
portfolioComponentData)
        {
            try
            {
                await ValidatePortfolioBeforeAction(portfolioId);
                var component = PortfolioComponent.GetComponent(componentId);
```

```
                if (component.ModelType == typeof(PortfolioComponentFixedIncome) ||
                    component.ModelType == typeof(PortfolioComponentEfFixedIncome))
                {
                    var yieldCurves = await
ServiceProvider.LogicProvider.VariablesManager
                        .GetMappingField(CurrentCompanyId, m => m.YieldCurves);
                    if (yieldCurves == null ||
                        yieldCurves.Count == 0 ||
                        yieldCurves.First().Value == null ||
                        yieldCurves.First().Value.Count == 0)
                        throw new EmptyYieldCurvesException();
                }

                var dataValues = await GetPortfolioComponentDatasList();
                var result = await ServiceProvider.LogicProvider.PortfolioManager
                    .CreatePortfolioComponentData(portfolioId, componentId,
portfolioComponentData, SimpleVariables, dataValues);
                await
LogicHub(CurrentCompanyId).PortfolioComponentDataCreated(result, portfolioId,
componentId);
                await UpdatePortfolioConfigurationAndStatus(portfolioId,
componentId);

                await SetChildPortfoliosToReady(portfolioId, null);
                return true;
            }
            catch (Exception ex)
            {
                ServiceProvider.Logger.LogError(ex, $"Error while creating
portfolio {portfolioId} component data", portfolioComponentData);
                throw;
            }
        }

        /// <summary>
        /// Update portfolio component with whole model
        /// </summary>
        /// <returns>The Portfolio component data view model</returns>
        [HttpPut("{portfolioId:long}/{componentId:int}/updateComponentData")]
        [UserActionLog("Portfolio", "Update portfolio component with whole model",
false, RequestSensitivityLevel.InputOrResult)]
        public async Task<bool> UpdatePortfolioComponentData([FromRoute]long
portfolioId, [FromRoute]byte componentId, [FromBody]PortfolioComponentDataViewModel
portfolioComponentData)
        {
            try
            {
                await ValidatePortfolioBeforeAction(portfolioId);
                var dataValues = await GetPortfolioComponentDatasList();
                var result = await
ServiceProvider.LogicProvider.PortfolioManager.UpdateMultiplePortfolioComponentData
(portfolioId, componentId, new
List<PortfolioComponentDataViewModel>(){portfolioComponentData}, SimpleVariables,
dataValues);
                await
LogicHub(CurrentCompanyId).PortfolioComponentDataUpdated(result.First(),
portfolioId, componentId);
                await UpdatePortfolioConfigurationAndStatus(portfolioId,
componentId);
```

```
                await SetChildPortfoliosToReady(portfolioId, null);
                return true;
            }
            catch (Exception ex)
            {
                ServiceProvider.Logger.LogError(ex, $"Error while updating
portfolio {portfolioId} component data {componentId}", portfolioComponentData);
                throw;
            }
        }

        /// <summary>
        /// Update portfolio components with whole model
        /// </summary>
        /// <returns>The Portfolio component data view model</returns>

[HttpPut("{portfolioId:long}/{componentId:int}/updateMultipleComponentData")]
        [UserActionLog("Portfolio", "Update portfolio components with whole model",
false, RequestSensitivityLevel.InputOrResult)]
        public async Task<bool> UpdateMultiplePortfolioComponentData([FromRoute]
long portfolioId, [FromRoute] byte componentId, [FromBody]
List<PortfolioComponentDataViewModel> portfolioComponentDataList)
        {
            try
            {
                await ValidatePortfolioBeforeAction(portfolioId);
                var dataValues = await GetPortfolioComponentDatasList();
                var result = await
ServiceProvider.LogicProvider.PortfolioManager.UpdateMultiplePortfolioComponentData
(portfolioId, componentId, portfolioComponentDataList, SimpleVariables,
dataValues);
                foreach (var portfolioComponentDataViewModel in result)
                {
                    await
LogicHub(CurrentCompanyId).PortfolioComponentDataUpdated(portfolioComponentDataView
Model, portfolioId, componentId);
                }
                await UpdatePortfolioConfigurationAndStatus(portfolioId,
componentId);
                await SetChildPortfoliosToReady(portfolioId, null);
                return true;
            }
            catch (Exception ex)
            {
                ServiceProvider.Logger.LogError(ex, $"Error while updating
portfolio {portfolioId} component data {componentId}", portfolioComponentDataList);
                throw;
            }
        }


        /// <summary>
        /// Update portfolio component by field
        /// </summary>
        /// <returns>The Portfolio component data view model</returns>

[HttpPut("{portfolioId:long}/updateComponentDataField/{componentDataId:int}")]
        [UserActionLog("Portfolio", "Update portfolio component by field", false,
```

```
RequestSensitivityLevel.InputOrResult)]
        public async Task<bool> UpdatePortfolioComponentDataField([FromRoute]long
portfolioId, [FromRoute]long componentDataId, [FromBody]UpdateViewModel model)
        {
            try
            {
                await ValidatePortfolioBeforeAction(portfolioId);
                var dataValues = await GetPortfolioComponentDatasList();
                var result = await
ServiceProvider.LogicProvider.PortfolioManager.UpdatePortfolioComponentDataAsyncObs
erved(portfolioId, componentDataId, SimpleVariables, dataValues, model.Field,
model.Value);
                var portfolioComponentData = await
ServiceProvider.LogicProvider.PortfolioManager.GetPortfolioComponentData(portfolioI
d, componentDataId);
                await
LogicHub(CurrentCompanyId).PortfolioComponentDataUpdated(result, portfolioId,
portfolioComponentData.PortfolioComponentId);
                await UpdatePortfolioConfigurationAndStatus(portfolioId,
portfolioComponentData.PortfolioComponentId);
                await SetChildPortfoliosToReady(portfolioId, null);
                return true;
            }
            catch (Exception ex)
            {
                ServiceProvider.Logger.LogError(ex, $"Error while updating
portfolio {portfolioId} component data {componentDataId}", model);
                throw;
            }
        }

        /// <summary>
        /// Clone portfolio component
        /// </summary>
        /// <returns>The Portfolio component data view model</returns>

[HttpPut("configuration/{portfolioId:long}/cloneComponentData/{componentDataId:long
}")]
        [UserActionLog("Portfolio", "Clone portfolio component", false,
RequestSensitivityLevel.ApplicationSection)]
        public async Task<bool> ClonePortfolioComponentData([FromRoute]long
portfolioId, [FromRoute]long componentDataId)
        {
            try
            {
                await ValidatePortfolioBeforeAction(portfolioId);
                var result = await
ServiceProvider.LogicProvider.PortfolioManager.ClonePortfolioComponentData(portfoli
oId, componentDataId);
                var portfolioComponentData = await
ServiceProvider.LogicProvider.PortfolioManager.GetPortfolioComponentData(portfolioI
d, componentDataId);
                await
LogicHub(CurrentCompanyId).PortfolioComponentDataCloned(result, portfolioId,
portfolioComponentData.PortfolioComponentId);
                await UpdatePortfolioConfigurationAndStatus(portfolioId,
portfolioComponentData.PortfolioComponentId);
                await SetChildPortfoliosToReady(portfolioId, null);
                return true;
```

```
            }
            catch (Exception ex)
            {
                ServiceProvider.Logger.LogError(ex, $"Error while deleting
portfolio {portfolioId} component data {componentDataId}", componentDataId);
                throw;
            }
        }

        /// <summary>
        /// Delete portfolio component
        /// </summary>
        /// <returns>The Portfolio component data view model</returns>

[HttpDelete("configuration/{portfolioId:long}/deleteComponentData/{componentDataId:
long}")]
        [UserActionLog("Portfolio", "Delete portfolio component", false,
RequestSensitivityLevel.ApplicationSection)]
        public async Task<bool> DeletePortfolioComponentData([FromRoute]long
portfolioId, [FromRoute]long componentDataId)
        {
            try
            {
                await ValidatePortfolioBeforeAction(portfolioId);
                var portfolioComponentData = await
ServiceProvider.LogicProvider.PortfolioManager.GetPortfolioComponentData(portfolioI
d, componentDataId);
                var result = await
ServiceProvider.LogicProvider.PortfolioManager.DeleteMultiplePortfolioComponentData
(portfolioId, new List<long> {componentDataId});
                await
LogicHub(CurrentCompanyId).PortfolioComponentDataDeleted(componentDataId,
portfolioComponentData.PortfolioId, portfolioComponentData.PortfolioComponentId);
                await UpdatePortfolioConfigurationAndStatus(portfolioId,
portfolioComponentData.PortfolioComponentId);
                await SetChildPortfoliosToReady(portfolioId, null);
                return true;
            }
            catch (Exception ex)
            {
                ServiceProvider.Logger.LogError(ex, $"Error while deleting
portfolio {portfolioId} component data {componentDataId}", componentDataId);
                throw;
            }
        }

        /// <summary>
        /// Delete multiple components
        /// </summary>
        /// <param name="portfolioId">The portfolio Id</param>
        /// <param name="componentDataIds">Component IDS</param>
        [HttpPut("configuration/{portfolioId:long}/deleteMultipleComponentData")]
        [UserActionLog("Portfolio", "Delete multiple components", false,
RequestSensitivityLevel.ApplicationSection)]
        public async Task<bool>
DeleteMultiplePortfolioComponentData([FromRoute]long portfolioId, [FromBody]
List<long> componentDataIds)
        {
            try
```

```
            {
                await ValidatePortfolioBeforeAction(portfolioId);
                var portfolioComponentDatas =
                    await
ServiceProvider.LogicProvider.PortfolioManager.GetPortfolioComponentDataByIds(portf
olioId,
                        componentDataIds);
                var result = await
ServiceProvider.LogicProvider.PortfolioManager.DeleteMultiplePortfolioComponentData
(portfolioId, componentDataIds);
                foreach (var portfolioComponentData in portfolioComponentDatas)
                {
                    await
LogicHub(CurrentCompanyId).PortfolioComponentDataDeleted(portfolioComponentData.Id,
portfolioComponentData.PortfolioId, portfolioComponentData.PortfolioComponentId);
                }
                await UpdatePortfolioConfigurationAndStatus(portfolioId,
portfolioComponentDatas.First().PortfolioComponentId);

                await SetChildPortfoliosToReady(portfolioId, null);
                return true;
            }
            catch (Exception ex)
            {
                ServiceProvider.Logger.LogError(ex, $"Error while deleting
portfolio {portfolioId} component data {string.Join(", ", componentDataIds)}",
componentDataIds);
                throw;
            }
        }
        /// <summary>
        /// Get portfolio components list
        /// </summary>
        /// <returns>The Portfolio component data view model</returns>
        [HttpGet("configuration/component/lists")]
        [UserActionLog("Portfolio", "Get portfolio components list", false,
RequestSensitivityLevel.InputOrResult)]
        public async Task<PortfolioComponentDataValueViewModel>
GetPortfolioComponentDatas()
        {
            return await GetPortfolioComponentDatasList();
        }

        /// <summary>
        /// Import portfolio component data
        /// </summary>
        /// <returns>The Portfolio component data view model</returns>
        [RequestFormLimits(MultipartBodyLengthLimit = 50000000)]
        [RequestSizeLimit(50000000)]
        [DisableRequestSizeLimit]

[HttpPost("{portfolioId:long}/configuration/{componentId:int}/{importType:int}/impo
rtComponentData")]
        [UserActionLog("Portfolio", "Import portfolio component data", false,
RequestSensitivityLevel.InputOrResult)]
        public async Task<bool> ImportPortfolioComponentData([FromRoute]long
portfolioId, [FromRoute]byte componentId, [FromRoute]byte importType,
[FromQuery]string connectionId, IFormFile formFile)
        {
```

```
            await ValidatePortfolioBeforeAction(portfolioId);
            Func<string, Task> uploadingCallBack = async (i) => { await
Task.CompletedTask;};
            if (!string.IsNullOrEmpty(connectionId))
                uploadingCallBack = async (status) => await
ServiceProvider.LogicHub.Clients.Client(connectionId).UpdateImportComponentProgress
(status, "");
            try
            {
                await uploadingCallBack("Validating");
                var portfolio = await
ServiceProvider.LogicProvider.PortfolioManager.GetPortfolio(portfolioId);

                var component = PortfolioComponent.GetComponent(componentId);

                if (component.IsSingle)
                    throw new ImportDoesntSupportSingleException();

                formFile.CheckFile(Formats.Xlsx, Formats.Xls, Formats.Csv);
                var splitedName = formFile.FileName.Split('.');
                var ext = splitedName.Last();
                CheckAdminImportCsv(formFile.FileName);

                var stream = await
ServiceProvider.LogicProvider.ScenarioManager.GetMemoryStreamFromFile(formFile);

                var fullName = new string[splitedName.Length - 1];
                Array.Copy(splitedName, fullName, splitedName.Length - 1);
                var user = ServiceProvider.LogicProvider.UserManager.GetUserData();

                await
ServiceProvider.LogicProvider.PortfolioManager.UploadPortfolioComponentDataFileToS3
(user.UserName, formFile.FileName, stream);

                var dataValues = await GetPortfolioComponentDatasList();
                var newComponents =
                    await
ServiceProvider.LogicProvider.PortfolioManager.ValidateImportPortfolioComponentData
(
                        portfolioId, componentId, ext != Formats.Csv, stream,
SimpleVariables, dataValues);

                var existingComponentData =
                    await
ServiceProvider.LogicProvider.PortfolioManager.GetPortfolioComponentDataList(portfo
lioId, componentId);
                var result = new List<PortfolioComponentDataViewModel>();
                await
ServiceProvider.LogicProvider.PortfolioManager.ValidateImportedSegment(portfolio,
component, (ImportType)importType, newComponents,
                    existingComponentData);
                await
ServiceProvider.LogicProvider.PortfolioManager.ValidateCashFlows(portfolio,
component,
                    (ImportType) importType, newComponents, existingComponentData);

                await uploadingCallBack("Creating");

                switch (component.Name)
```

```
                {
                    case "Projected Liability Cashflows":
                    {
                        switch (importType)
                        {
                            case 0:
                            {
                                foreach (var newComponent in newComponents)
                                {
                                    await
ServiceProvider.LogicProvider.PortfolioManager
                                        .CreatePortfolioComponentData(portfolioId,
componentId, newComponent,
                                            SimpleVariables, dataValues);

                                }
                                break;
                            }
                            case 1:
                            {
                                await
ServiceProvider.LogicProvider.PortfolioManager

.DeleteMultiplePortfolioComponentData(portfolioId,

existingComponentData.ComponentDatas.Select(d => d.Id).ToList());
                                foreach (var newComponent in newComponents)
                                {
                                    await
ServiceProvider.LogicProvider.PortfolioManager
                                        .CreatePortfolioComponentData(portfolioId,
componentId, newComponent,
                                            SimpleVariables, dataValues);
                                }

                                break;
                            }
                        }
                        break;
                    }
                    default:
                    {
                        foreach (var newComponent in newComponents)
                        {
                            switch (importType)
                            {
                                case 0: // Append
                                    var append = await
ServiceProvider.LogicProvider.PortfolioManager
                                        .CreatePortfolioComponentData(portfolioId,
componentId, newComponent,
                                            SimpleVariables, dataValues);
                                    //result.Add(append);
                                    break;
                                case 1: // Replace
                                case 2: // Merge
                                    var existingComponent =
existingComponentData.ComponentDatas
                                        .FirstOrDefault(x => x.Name ==
```

```
newComponent.Name);
                                    if (existingComponent == null)
                                    {
                                        var replaceAppend = await
ServiceProvider.LogicProvider.PortfolioManager

.CreatePortfolioComponentData(portfolioId, componentId, newComponent,
                                                SimpleVariables, dataValues);
                                        //result.Add(replaceAppend);
                                    }
                                    else
                                    {
                                        newComponent.Id = existingComponent.Id;
                                        var replaceExist = await
ServiceProvider.LogicProvider.PortfolioManager

.UpdateMultiplePortfolioComponentData(portfolioId, componentId,
                                                new
List<PortfolioComponentDataViewModel>() {newComponent},
                                                SimpleVariables, dataValues);
                                        //result.Add(replaceExist);
                                        if (importType == 1)
                                            existingComponentData.ComponentDatas
                                                .Remove(existingComponent);
                                    }

                                    break;
                                }
                            }

                            if (importType == 2 || importType == 0)
                                existingComponentData.ComponentDatas = new
List<PortfolioComponentDataViewModel>();
                            foreach (var componentData in
existingComponentData.ComponentDatas)
                            {
                                await ServiceProvider.LogicProvider.PortfolioManager
                                    .DeleteMultiplePortfolioComponentData(portfolioId,
new List<long> {componentData.Id});
                            }

                            break;
                        }
                    }

                    existingComponentData = await
ServiceProvider.LogicProvider.PortfolioManager
                        .GetPortfolioComponentDataList(portfolioId, componentId);
                    await LogicHub(CurrentCompanyId)

.PortfolioComponentDataImported(existingComponentData.ComponentDatas, portfolioId,
componentId);
                    await UpdatePortfolioConfigurationAndStatus(portfolioId,
componentId);
                    await SetChildPortfoliosToReady(portfolioId, null);
                    return true;
                }
                catch (Exception ex)
                {
```

```
                ServiceProvider.Logger.LogError(ex, $"Error while importing
portfolio {portfolioId} component data. ComponentId: {componentId}");
                throw;
            }
        }

        /// <summary>
        /// Export portfolio Component data
        /// </summary>
        /// <param name="portfolioId">The portfolio Id</param>
        /// <param name="componentId">The portfolio component Id</param>
        /// <param name="format">The format of export (xlsx or csv)</param>

[HttpGet("{portfolioId:long}/configuration/{componentId:int}/exportComponentData")]
        [UserActionLog("Portfolio", "Export portfolio Component data", false,
RequestSensitivityLevel.InputOrResult)]
        public async Task<string> ExportSynthesisDownload([FromRoute]long
portfolioId, [FromRoute]int componentId, [FromQuery]string format)
        {
            await ValidatePortfolioBeforeAction(portfolioId);
            var dataValues = await GetPortfolioComponentDatasList();
            var fileContent = await
ServiceProvider.LogicProvider.PortfolioManager.ExportComponentData(portfolioId,
(byte)componentId, format,
                SimpleVariables.ToDictionary(m => m.Key, n => n.Value.Name),
dataValues);
            return await
_downloadCacheService.SetFileToCacheAndGetKey(fileContent.Content,
fileContent.FileDownloadName);
        }

        /// <summary>
        /// Send portfolio to synthesis
        /// </summary>
        /// <param name="id">The linked scenario to portfolio Id</param>
        [HttpPut("{id:long}/synthesis")]
        [UserActionLog("Portfolio", "Send portfolio to synthesis", false,
RequestSensitivityLevel.ApplicationSection)]
        public async Task<bool> RunSynthesis(long id)
        {
            await ValidatePortfolioBeforeAction(id);
            var dataValues = await GetPortfolioComponentDatasList();

            var portfolioVm = await
ServiceProvider.LogicProvider.PortfolioManager.CalculateLinkedScenarioToPortfolioSt
atus(id,
                dataValues.RequiredVariables, CurrentCompanyId);
            if (portfolioVm.Status != Status.Pending.ToString() &&
portfolioVm.Status != Status.Ready.ToString())
            {
                await LogicHub(portfolioVm.CompanyId,
portfolioVm.PortfolioTypeEnum.Value).PortfolioUpdated(portfolioVm, string.Empty);
                throw new LinkedScenarioToPortfolioNotReadyException();
            }
            var settings = await ServiceProvider.LogicProvider.SettingManager
                   .GetUserCompanySettings(true,
UserCompanySettingCategory.General);
            var model = await
ServiceProvider.LogicProvider.PortfolioManager.GenerateModelForSynthesis(id,
```

```
SimpleVariables, dataValues,

settings.GetSetting(AppSettingType.IsPortfolioIntermediateEnabled).Equals("true",
StringComparison.InvariantCultureIgnoreCase));
            if (model == null)
                return false;

            var vm = await
ServiceProvider.LogicProvider.PortfolioManager.StartProgress(id);
            await LogicHub(vm.CompanyId,
vm.PortfolioTypeEnum.Value).PortfolioUpdated(vm, string.Empty);

            if (await _calculatorService.Send(model, HttpContext.GetToken(), true))
            {
                vm = await
ServiceProvider.LogicProvider.PortfolioManager.UpdateLinkedScenarioCalculatorCallId
(id,
                _calculatorService.CalculatorCallViewModel.Id);
                await LogicHub(vm.CompanyId,
vm.PortfolioTypeEnum.Value).PortfolioUpdated(vm, string.Empty);
                await AvailableCallsUpdated(vm.CompanyId);
                return true;
            }

            vm = await ServiceProvider.LogicProvider.PortfolioManager
                .SetError(id, _calculatorService.CalculatorCallViewModel.Error);
            await LogicHub(vm.CompanyId,
vm.PortfolioTypeEnum.Value).PortfolioUpdated(vm, string.Empty);
            return false;
        }

        /// <summary>
        /// Cancel calculation
        /// </summary>
        /// <param name="id">The synthesis Id</param>
        [HttpPut("{id:long}/synthesis/cancel")]
        [UserActionLog("Portfolio", "Cancel calculation", false,
RequestSensitivityLevel.ApplicationSection)]
        public async Task<bool> CancelSynthesisCalculation(long id)
        {
            await ValidatePortfolioBeforeAction(id);
            var portfolio = await
ServiceProvider.LogicProvider.PortfolioManager.GetPortfolio(id);
            await _calculatorService.CancelTask(id, CalculatorAction.Synthesis);
            var vm = await
ServiceProvider.LogicProvider.PortfolioManager.UpdateStatus(id, Status.Pending);
            await LogicHub(null, vm.PortfolioTypeEnum.Value).PortfolioUpdated(vm,
string.Empty);
            await AvailableCallsUpdated(vm.CompanyId);
            return true;
        }

        /// <summary>
        /// Get synthesis result by path
        /// </summary>
        /// <param name="id">The linked scenario to portfolio Id</param>
        /// <param name="path">The linked scenario result path</param>
        [HttpGet("{id:long}/synthesis/result/{path:int}")]
        [UserActionLog("Portfolio", "Get synthesis result by path", false,
```

```
RequestSensitivityLevel.InputOrResult)]
        public async Task<List<RestClientSynthesisResultViewModel>>
GetSynthesisResult(long id, int path)
        {
            await ValidatePortfolioBeforeAction(id);
            return await
ServiceProvider.LogicProvider.PortfolioManager.GetSynthesisResult(id, path);
        }

        /// <summary>
        /// Run export synthesis
        /// </summary>
        /// <param name="id">The linked scenario to portfolio Id</param>
        /// <param name="type">The export type (excel/csv)</param>
        /// <param name="size">The export type size (Full, Partial,
BalanceSheetSummary). Default is full</param>
        [HttpPut("{id:long}/synthesis/download/{type}")]
        [UserActionLog("Portfolio", "Run export synthesis", false,
RequestSensitivityLevel.ApplicationSection)]
        public async Task<bool> RunSynthesisExport([FromRoute]long id,
[FromRoute]string type, PortfolioExportType size = PortfolioExportType.Full)
        {
            await ValidatePortfolioBeforeAction(id);
            await
ServiceProvider.LogicProvider.PortfolioManager.RunExportSynthesis(id, type,
HttpContext.GetToken(), size);
            return true;
        }


        private async Task UpdatePortfolioConfigurationAndStatus(long portfolioId,
byte componentId)
        {
            var result = await
ServiceProvider.LogicProvider.PortfolioManager.UpdatePortfolioComponentConfiguratio
n(portfolioId, componentId);
            await
LogicHub(CurrentCompanyId).PortfolioComponentConfigurationUpdated(result,
portfolioId, componentId);

            await UpdatePortfolioStatus(portfolioId);
        }
        private async Task UpdatePortfolioStatus(long portfolioId)
        {
            var portfolioVm = await
ServiceProvider.LogicProvider.PortfolioManager.UpdatePortfolioStatus(portfolioId);
            await LogicHub(CurrentCompanyId).PortfolioUpdated(portfolioVm,
string.Empty);
        }


        /// <summary>
        /// Download synthesis outputs
        ///  </summary>
        /// <param name="synthId">The synthesis Id</param>
        [Authorize(Roles = "DownloadPortfolioFiles", Policy = "CheckRequirement")]
        [HttpGet("downloadOutputArray/{synthId:long}")]
        [UserActionLog("Portfolio", "Download synthesis outputs", false,
RequestSensitivityLevel.InputOrResult)]
```

```
        public async Task<string> DownloadFolderArray(long synthId)
        {
            await ValidatePortfolioBeforeAction(synthId);
            var file = await
ServiceProvider.LogicProvider.PortfolioManager.DownloadFolder(synthId);
            return await
_downloadCacheService.SetFileToCacheAndGetKey(file.Content, file.FileDownloadName);
        }

        /// <summary>
        /// Fetch failed synthesis
        ///  </summary>
        [Authorize(Roles = "SuperAdmin", Policy = "CheckRequirement")]
        [HttpGet("FailedSynthesis")]
        [UserActionLog("Portfolio", "Fetch failed synthesis", false,
RequestSensitivityLevel.ApplicationSection)]
        public async Task<List<FailedEntityViewModel>> FetchFailedSynthesis()
        {
            try
            {
                return await ServiceProvider.LogicProvider.PortfolioManager
                    .FetchFailedSynthesises(new DateTime());
            }
            catch (Exception ex)
            {
                ServiceProvider.Logger.LogErrorCreate(ex, CurrentUserId, null);
                throw;
            }
        }

        /// <summary>
        /// Is completed Portfolio subsidiary objects?
        /// If at least just one Synthesis is in completed status - true. Otherwise
- false
        /// </summary>
        /// <param name="portfolioId">The parent portfolio ID. If portfolioId is
null - search by scenario ID</param>
        /// <param name="scenarioId">The scenario or relationship ID. If scenario
id is null - search by portfolio ID</param>
        [UserActionLog("Portfolio", "Is completed Portfolio subsidiary objects?",
false, RequestSensitivityLevel.ApplicationSection)]
        [HttpGet("IsAnyCompletedSubs")]
        public async Task<bool> IsAnyCompletedSubs(long? portfolioId, long?
scenarioId)
        {
            if (!portfolioId.HasValue && !scenarioId.HasValue)
                throw new EmptyInputDataException();
            if(portfolioId.HasValue)
                await ValidatePortfolioBeforeAction(portfolioId.Value);

            return await
ServiceProvider.LogicProvider.PortfolioManager.IsAnyCompletedSubs(portfolioId,
scenarioId);
        }
    }
}
```

Додаток И

**ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ**

Назва роботи: *Комплексна магістерська кваліфікаційна робота «Клієнт-серверна система для передбачення поведінки фінансових інструментів Частина 2. Технічне проектування*

Тип роботи: кваліфікаційна робота

(кваліфікаційна робота, курсовий проект (робота), реферат, аналітичний огляд, інше – зазначити)

Підрозділ: кафедра АІІТ, ФКСА, 1ІСТ-21м

(кафедра, факультет, навчальна група)

Науковий керівник: Коцюбинський В.Ю., доц. каф. АІІТ

(прізвище, ініціали, посада)

Показники звіту подібності

| *Plagiat.pl (StrikePlagiarism)* | | *Unicheck* | |
|---|---|---|---|
| КП1 | - | Оригінальність | 87.5% |
| КП2 | - | | |
| Тривога/Білі знаки | / | Схожість | 12.5% |

Аналіз звіту подібності (відмітити потрібне)

**X Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

Заявляю, що ознайомлений (-на) з повним звітом подібності, який був згенерований Системою щодо роботи (додається)

Автор _____ Курніцький Д.П.

(підпис) (прізвище, ініціали)

Опис прийнятого рішення:

Допустити до захисту

Особа, відповідальна за перевірку _____ Маслій Р.В.