КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет автоматизації і інформаційних технологій
Кафедра управління проєктами

ПОЯСНЮВАЛЬНА ЗАПИСКА ДО КВАЛІФІКАЦІЙНОЇ РОБОТИ НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

на тему:

Управління проектом розробки програмного продукту для відстеження капіталу і прогнозування прибутку за допомогою ШІ

Душкін Андрій Андрійович

Київ 2024 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет: Автоматизації і інформаційних технологій
Випускова кафедра: Управління проєктами
Освітній рівень: Магістр за освітньо-професійною програмою
Галузь знань: 12 Інформаційні технології
Спеціальність: 126 Інформаційні системи та технології
Освітня програма: Штучний інтелект. Когнітивні технології

ЗАТВЕРДЖУЮ
Завідувач кафедри
Бушуєв С. Д.
“___” _____ 2024 року

ЗАВДАННЯ
ДО ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

Душкін Андрій Андрійович

1. Тема роботи: “Управління проектом розробки програмного продукту для відстеження капіталу і прогнозування прибутку за допомогою ШІ” затверджена наказом ректора КНУБА №1665/2 від 20.08.2024 року

2. Керівник роботи: Ільїн Олег Олександрович, д.т.н., проф

3. Строк подання студентом роботи до захисту: 21.11.2024

4. Зміст пояснювальної записки (перелік питань, які слід розробити):

а) теоретичний розділ: методи управління ІТ-проєктами, гібридні підходи в управлінні проєктами, архітектурні стилі ПЗ, реляційні та нереляційні бази даних, штучний інтелект для фінансового прогнозування;

б) дослідницько-аналітичний розділ: аналіз ринку фінансових програм для відстеження капіталу, SWOT-аналіз, визначення вимог, структура WBS, план управління проєктом, управління ризиками;

в) практичний розділ: проєктування бази даних, архітектура програмного продукту, інтеграція модуля штучного інтелекту, розробка і розгортання програмного продукту, використання Docker для контейнеризації.

5. Графічний матеріал за розділами: таблиці, малюнки, ER-діаграми бази даних, інтерфейс користувача, принцип роботи моделі ШІ.

6. Календарний план виконання роботи:

Види робіт та їх зміст	Дата виконання
Збір матеріалів обраного напряму роботи	05.08.2024 - 11.08.2024
Опрацювання та аналіз матеріалів роботи	12.08.2024 - 15.08.2024
Вступ	16.08.2024 - 01.09.2024
Розділ 1	02.09.2024 - 20.09.2024
Розділ 2	21.09.2024 - 15.10.2024
Розділ 3	16.10.2024 - 05.11.2024
Висновки	06.11.2024
Остаточне оформлення роботи	06.11.2024 - 10.11.2024
Перевірка роботи на плагіат	11.11.2024
Попередній захист роботи на кафедрі	13.11.2024
Направлення роботи на рецензування	13.11.2024

7. Консультанти розділів кваліфікаційної випускної роботи:

Розділ	ПІБ та посада консультанта	Перевірив	
		Дата	Підпис
Розділ 1			
Розділ 2			
Розділ 3			

8. Дата видачі завдання 15.08.2024

Зав. кафедри

_____ Бушуєв С.Д.

(підпис)

Керівник

_____ Ільїн О.О.

(підпис)

Студент

_____ Душкін А.А.

(підпис)

РЕЗЮМЕ (summary) до кваліфікаційної роботи магістра здобувача:		Душкін Андрій Андрійович	
ЗВО	Київський національний університет будівництва і архітектури		
Тема	Управління проектом розробки програмного продукту для відстеження капіталу і прогнозування прибутку за допомогою ШІ		
Освітній ступінь	Магістр за освітньо-професійною програмою навчання		
Факультет	Автоматизації і інформаційних технологій		
Кафедра	Управління проєктами		
Спеціальність	126 “Інформаційні системи та технології”		
Освітня програма	Штучний інтелект.Когнітивні технології		
Керівник	Ільїн Олег Олександрович, д.т.н, проф.		
Обсяг роботи:	пояснювальна записка, стор.	розділів	слайдів
	110	3	37
Розділ 1.	Розглянуто основні методи управління ІТ-проєктами та архітектури ПЗ, включно з предиктивними і гнучкими підходами, гібридними методологіями, перевагами і недоліками підходів Agile і Scrum. Проаналізовано архітектури, такі як монолітна, мікросервісна, клієнт-серверна та багатошарова. Визначено, що найоптимальнішим вибором для розробки є клієнт-серверна архітектура з елементами модульності з використанням Docker, Next.js і FastAPI. Обґрунтовано вибір моделі LSTM для прогнозування дохідності портфеля на основі часових рядів.		
Розділ 2.	Проведено аналіз ринку і конкурентного середовища, зокрема аналогічних рішень, таких як Snowball Analytics і Firekit. Визначено ключові функції, необхідні для розробки продукту, зокрема автоматизацію розрахунків прибутковості, інтеграцію з брокерами, можливість додавання кастомних активів та аналітичний модуль з елементами ШІ. За допомогою		

	SWOT-аналізу визначено сильні та слабкі сторони продукту, а також можливості та загрози ринку. Розроблено структуру WBS для організації робіт та управління ризиками.
Розділ 3.	Розглянуто етапи реалізації продукту для відстеження капіталу та прогнозування прибутковості. Проведено проєктування бази даних (концептуальне, логічне та фізичне проєктування), описано клієнт-серверну архітектуру з використанням FastAPI та Next.js, інтеграцію модуля LSTM для прогнозування, що забезпечує високу точність аналізу фінансових даних. Описано процес розгортання продукту із використанням Docker та управління інфраструктурою за допомогою Coolify.
Висновки по роботі:	Розроблено програмний продукт для відстеження капіталу та прогнозування його прибутковості з використанням штучного інтелекту, що відповідає сучасним вимогам ринку. Продукт забезпечує ефективне управління капіталом, аналіз і прогнозування фінансових показників. Прийняті рішення щодо технологій, архітектури, та методів прогнозування підвищують продуктивність, гнучкість і конкурентоспроможність розробленої системи.
Ключові слова: управління капіталом, штучний інтелект, LSTM, клієнт-серверна архітектура **Keywords:** capital management, artificial intelligence, LSTM, client-server architecture	

Студент: Андрій ДУШКІН

Керівник: Олег ІЛЬЇН

«____» листопада 2024 року

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет автоматизації і інформаційних технологій
Кафедра управління проєктами

ЗАТВЕРДЖУЮ
Завідувач кафедри
Бушуєв С. Д.
“___” _____ 2024 року

ПОЯСНЮВАЛЬНА ЗАПИСКА ДО КВАЛІФІКАЦІЙНОЇ РОБОТИ НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

Управління проектом розробки програмного продукту для відстеження капіталу і прогнозування прибутку за допомогою ШІ

Виконав студент групи:
Душкін Андрій Андрійович
Спеціальність: 126 Інформаційні системи та технології
Освітня програма: Штучний інтелект. Когнітивні технології.
Керівник: Ільїн О.О., д.т.н., проф.
Рецензент: Терентьєв О.О., д.т.н., проф.

ЗМІСТ

ВСТУП

З розвитком фінансових технологій (FinTech) та збільшенням кількості інвестиційних активів на ринку, виникає нагальна потреба в автоматизованих рішеннях для управління капіталом та прогнозування його прибутковості. Інвестори, як приватні, так і фінансові установи, потребують інструментів, які можуть аналізувати фінансові дані і надавати прогнози щодо майбутніх змін вартості активів. Використання штучного інтелекту (ШІ) у таких рішеннях дозволяє підвищити точність аналізу і зменшити ризики, пов'язані з інвестуванням, шляхом прогнозування на основі історичних даних.

Актуальність теми дослідження зумовлена зростаючим попитом на програмні продукти, які здатні не тільки відстежувати капітал, але й прогнозувати його дохідність. ШІ дозволяє оптимізувати процес прийняття рішень за рахунок аналізу великих обсягів даних і виявлення складних патернів у фінансових часових рядах. Розробка таких інструментів вимагає не лише високотехнологічного підходу, а й ефективного управління проєктом розробки програмного забезпечення.

Мета дослідження – розробити програмний продукт, що дозволяє відстежувати капітал та прогнозувати його прибутковість на основі історичних даних за допомогою технологій штучного інтелекту.

Завдання дослідження:

- Дослідити сучасні методи управління IT-проєктами.
- Провести аналіз ринку програмних продуктів для відстеження капіталу.
- Визначити архітектуру та технології для розробки програмного продукту.
- Розробити програмний продукт для аналізу капіталу.
- Розробити та інтегрувати модуль прогнозування прибутковості капіталу на основі ШІ.

Об'єкт дослідження – процес управління проєктом розробки програмного забезпечення для фінансового аналізу.

Предмет дослідження – технології штучного інтелекту для аналізу та прогнозування фінансових показників на основі часових рядів, а також методи управління проєктом у розробці програмного забезпечення.

Методи дослідження включають аналіз і синтез інформації, прогнозування на основі часових рядів, а також використання сучасних підходів до управління проєктами, таких як Scrum та Agile. Для прогнозування фінансових показників застосовувалися моделі LSTM та Temporal Fusion Transformer, які дозволяють обробляти складні часові ряди з великою кількістю даних. Окрім цього, використовувалися інструменти аналізу даних, такі як Pandas та scikit-learn, для ефективної обробки фінансових даних.

Робота складається з трьох основних розділів. Перший розділ присвячений теоретичним аспектам управління проєктами в IT-сфері та аналізу архітектури програмного забезпечення. Другий розділ охоплює аналіз ринку схожих програмних продуктів та проведення SWOT-аналізу. У третьому розділі детально розглядається процес реалізації програмного продукту, вибір технологій та інтеграція модуля штучного інтелекту для прогнозування прибутковості.

РОЗДІЛ 1. УПРАВЛІННЯ ПРОЄКТАМИ ТА АНАЛІЗ АРХІТЕКТУРИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Управління проєктами в IT-сфері

Управління проєктами в IT-сфері є складним і багатогранним процесом, який вимагає гнучкості, адаптивності та швидкої реакції на зміни. Різноманіття методологій, які використовуються для організації проєктної роботи в IT, дозволяють обирати підходи залежно від специфіки проєкту, його масштабів та вимог.

Одним з ключових аспектів управління IT-проєктами є **гібридний підхід**, який поєднує елементи предиктивного (традиційного) та адаптивного (гнучкого) підходів. Це дозволяє командами проєктів одночасно планувати етапи, які можуть бути чітко передбачені, і залишати місце для гнучкості у випадках, коли потрібна адаптація до нових умов. Як зазначено у **PMBOK (7-ме видання)**, гібридні методології дозволяють адаптуватися до змін завдяки поєднанню планових елементів та Agile-підходів [1, с. 36]. У тексті підкреслюється, що гібридний підхід корисний, коли проєкт працює в умовах невизначеності, наприклад, у швидко змінюваній IT-сфері.

Також важливим є **адаптивний підхід** до управління проєктами, який акцентує увагу на здатності команди швидко реагувати на зміни. Адаптивність передбачає, що команда проєкту здатна коригувати свої плани та дії відповідно до нових вимог або змін в оточенні проєкту. Як зазначено в **PMBOK**, адаптивність є ключовим аспектом сучасного управління проєктами, особливо в IT, де зміни в технологіях або ринкові умови можуть швидко змінювати вимоги до продукту або проєкту [1, с. 57].

Agile-методологія є найпоширенішою в IT-сфері, оскільки вона надає змогу швидко адаптуватися до змін і фокусуватися на цінності для клієнта. **Agile-маніфест**, опублікований у 2001 році, підкреслює важливість співпраці з клієнтами, гнучкості у процесі розробки та акценту на робочий продукт. Agile став невід'ємною частиною

IT-проєктів завдяки своїй здатності швидко адаптуватися до змін і забезпечувати безперервне вдосконалення продукту. Відповідно до **Agile-маніфесту**, "реакція на зміни важливіша за слідування плану", що робить цей підхід ідеальним для IT-проєктів, де вимоги часто змінюються під час виконання [2].

Scrum — це один з найпоширеніших фреймворків, який використовується в управлінні IT-проєктами в рамках **Agile**-методології. Його основними елементами є ітераційна розробка, яка дозволяє командам постійно адаптуватися до змін і забезпечувати регулярні поставки готових продуктів. Scrum-методика використовує регулярні короткі цикли, які називаються спринтами, для постійного вдосконалення проєкту. Згідно з **Scrum Guide**, основні зустрічі команди, такі як Daily Stand-ups, покликані забезпечити прозорість і підтримувати постійну комунікацію між членами команди для швидкого вирішення проблем і адаптації до змін [3].

Згідно з **Scrum Guide**, команда повинна працювати над досягненням результату в межах кожного спринту, що зазвичай триває 1-4 тижні. Такий підхід дозволяє зберігати гнучкість у розробці, що є необхідним для IT-проєктів з високою частотою змін.

Існує також необхідність використовувати **гібридні методології** в управлінні IT-проєктами. Наприклад, деякі великі IT-проєкти можуть поєднувати елементи **Waterfall** та **Agile**, щоб використовувати структурованість Waterfall для управління великими фазами, такими як архітектура системи, і гнучкість Agile для швидкого впровадження нових функцій і реагування на зміни. Це дозволяє отримати найкраще з обох світів, що особливо корисно для проєктів з високою невизначеністю та складністю [4].

На мою думку, в умовах сучасного IT-світу, де технології змінюються неймовірно швидко, гібридний підхід стає найоптимальнішим для ефективного управління. Поєднання структурованості традиційних методів і гнучкості Agile дозволяє не тільки реагувати на зміни, але й ефективно планувати довгострокові цілі.

Крім того, важливо, щоб команди вміли швидко адаптуватися до нових умов, працюючи в умовах високої невизначеності, що є характерним для ІТ-сфери.

1.2 Архітектура програмного забезпечення

Архітектура програмного забезпечення є основним аспектом, що визначає структуру системи, її компоненти та їхню взаємодію. Вона забезпечує як функціональність, так і масштабованість системи. Архітектура дозволяє досягати оптимального поєднання продуктивності, підтримуваності та гнучкості, що є важливим для складних ІТ-проектів.

Основними принципами архітектури ПЗ є **модульність**, **гнучкість** і **масштабованість**. Модульність передбачає розділення системи на незалежні компоненти (модулі), що дозволяє легше адаптувати та масштабувати систему. Це також підвищує стійкість системи до змін і спрощує її підтримку.

Гнучкість архітектури полягає в тому, що вона повинна мати здатність адаптуватися до змін вимог або середовища. Масштабованість, своєю чергою, дозволяє розширювати систему без необхідності її кардинальної перебудови [5, с. 12].

Існує кілька архітектурних стилів, які широко використовуються в розробці ПЗ:

1. **Монолітна архітектура**: У монолітних системах усі компоненти інтегровані в одну велику програму. Цей підхід спрощує початкову розробку, але з часом може ускладнювати підтримку і масштабування системи [6].
2. **Клієнт-серверна архітектура**: Цей стиль передбачає чітке розділення на клієнтську та серверну частини. Клієнт відповідає за взаємодію з користувачем, тоді як сервер обробляє запити і зберігає дані. Це дозволяє

розподіляти обчислювальні ресурси та збільшувати продуктивність системи [6].

3. **Мікросервісна архітектура**: Мікросервіси — це набір незалежних сервісів, кожен з яких відповідає за окрему функцію. Така архітектура дозволяє легше масштабувати систему і швидко вносити зміни. Кожен сервіс можна оновлювати і тестувати окремо, що значно підвищує гнучкість розробки [6].
4. **Багатошарова архітектура**: Цей стиль передбачає поділ функціональних частин системи на окремі шари, такі як шар інтерфейсу користувача, шар бізнес-логіки і шар доступу до даних. Це підвищує модульність системи та полегшує її підтримку [7].

В реальних проєктах часто використовується поєднання кількох архітектурних стилів. Як зазначається у конспекті лекцій, архітектура програмної системи рідко обмежується одним стилем. Наприклад, багатошарова архітектура може використовуватися для організації бізнес-логіки, тоді як мікросервіси забезпечують гнучкість і масштабованість на рівні окремих компонентів [5, с. 18].

Під час вибору архітектури програмного забезпечення слід враховувати низку факторів, зокрема вимоги до продуктивності, гнучкості та підтримуваності. Наприклад, мікросервісна архітектура краще підходить для великих проєктів, що потребують частих оновлень і високої масштабованості, тоді як клієнт-серверний підхід є оптимальним для веб-додатків, де необхідно обробляти великі обсяги даних у реальному часі [7].

Архітектура програмного забезпечення є основою для проектування сучасних ІТ-рішень. Вибір правильної архітектури визначає не лише продуктивність і гнучкість системи, але й можливість її подальшого розвитку. Використання комбінації архітектурних стилів дозволяє досягти оптимальних результатів у реалізації складних програмних продуктів.

1.3 Вибір технологій та інструментів для реалізації проєкту

У сучасній розробці програмних продуктів важливо не лише вибрати правильні архітектурні підходи, але й використовувати відповідні технології та інструменти для забезпечення ефективної реалізації проєкту. Нижче розглянуто основні технології, які були обрані для цього проєкту, та обґрунтовано їхній вибір.

Вибір технологій для клієнтської частини

Для реалізації клієнтської частини застосунку було обрано **React.js**, **Next.js** та **TypeScript**.

- **React.js** — це одна з найпопулярніших бібліотек для побудови інтерфейсів користувача. Вона забезпечує гнучкість, легкість в освоєнні та активну підтримку спільноти. React.js дозволяє створювати динамічні та масштабовані інтерфейси.
- **Next.js** доповнює React.js серверним рендерингом, що забезпечує кращу продуктивність і SEO-оптимізацію. Крім того, Next.js підтримує статичну генерацію сторінок, що робить його ідеальним для створення динамічних застосунків.
- **TypeScript** було обрано для забезпечення суворої типізації, що допомагає запобігати помилкам під час розробки та спрощує підтримку великого коду. Використання TypeScript у поєднанні з React.js та Next.js робить проєкт більш стабільним і надійним.

Фронтенд-платформа обрана через високу продуктивність і можливість швидкого рендерингу сторінок на стороні сервера, що позитивно впливає на швидкість завантаження та взаємодію з користувачем.

Вибір технологій для серверної частини

Серверна частина застосунку розгортається на базі **Python FastAPI**, що було обрано за свою легкість і гнучкість.

- **FastAPI** — це легкий веб-фреймворк для Python, який дозволяє створювати масштабовані серверні додатки. FastAPI легко інтегрується з базами даних та іншими сервісами, що робить його ідеальним вибором для цього проєкту.
- Основна функція бекенду — обробка запитів від клієнтів і взаємодія з базою даних та зовнішніми сервісами, зокрема з **Yahoo Finance API** для отримання історичних даних акцій. Окрім цього, у FastAPI буде інтегровано аналітичний модуль на основі штучного інтелекту для прогнозування фінансових результатів.

Цей вибір дозволяє реалізувати серверну частину з мінімальними витратами ресурсів і високою гнучкістю.

Вибір технологій для бази даних

Для зберігання даних використовується **PostgreSQL**, одна з найпотужніших реляційних баз даних з відкритим кодом. Для зручності налаштування та керування базою даних було обрано **Supabase** — платформу, яка надає інтеграцію з PostgreSQL і додаткові сервіси для управління даними.

- **PostgreSQL** забезпечує високу продуктивність і надійність у роботі з великими обсягами даних, що є важливим для обробки фінансових даних.
- **Supabase** дозволяє швидко налаштувати базу даних, реалізувати аутентифікацію та інтеграцію з бекендом за допомогою REST API.

Цей вибір забезпечує стабільність системи та спрощує роботу з даними, особливо у фінансовій сфері, де необхідно обробляти великий обсяг транзакцій.

Вибір додаткових інструментів

У процесі реалізації проєкту використовуються додаткові інструменти, зокрема **Docker**, **REST API**, та **S3** для зберігання файлів.

- **Docker** дозволяє контейнеризувати кожен компонент системи (фронтенд, бекенд, базу даних) і забезпечити їх незалежність один від одного. Це полегшує розгортання та підтримку системи, оскільки всі залежності ізольовані всередині контейнерів.
- **REST API** використовується для взаємодії між фронтендом і бекендом. Цей підхід дозволяє гнучко інтегрувати різні компоненти системи, що забезпечує масштабованість і зручність у підтримці.
- **S3** використовується для зберігання файлів. Це рішення забезпечує надійне зберігання даних, оскільки дозволяє масштабувати сховище файлів без обмежень у просторі.

Вибір хостингу

Одним із ключових рішень для цього проєкту стало вибір **VPS (Virtual Private Server)** для хостингу всіх компонентів системи. Вибір самостійного хостингу базувався на ряді факторів, що забезпечують гнучкість, контроль і економію ресурсів.

Альтернативою VPS могли б стати хмарні провайдери (AWS, Google Cloud, Microsoft Azure) або SaaS-платформи, такі як Heroku або Firebase. Ці рішення також популярні в сучасній розробці, оскільки забезпечують легке масштабування, просте налаштування і велику кількість вбудованих сервісів для розробки.

Основні переваги використання хмарних провайдерів та SaaS-платформ:

- **Автоматичне масштабування**: Хмарні провайдери автоматично збільшують ресурси у відповідь на зростання навантаження. Це робить їх привабливими для проєктів, де очікується різке збільшення кількості користувачів.
- **Інтеграція сервісів**: AWS, Google Cloud та інші провайдери надають широкий спектр додаткових сервісів, таких як бази даних, сховища

файлів, аналітика і машинне навчання, що спрощує розробку і управління проєктом.

- **Менеджмент інфраструктури**: SaaS-платформи дозволяють значно спростити управління сервером, оскільки все керується автоматично і не потребує технічної підтримки з боку розробників.

Проте ми вирішили використовувати самостійний хостинг на VPS з кількох важливих причин:

1. **Контроль та безпека**: Використовуючи VPS, команда має повний контроль над конфігурацією сервера, доступом до даних та управління безпекою. Це важливо для проєктів, де безпека є пріоритетом, і де потрібно забезпечити незалежність від сторонніх провайдерів.
2. **Економія витрат**: У порівнянні з хмарними провайдерами, які можуть бути дорожчими для проєктів з постійним навантаженням, VPS забезпечує фіксовану ціну за виділені ресурси. Це особливо вигідно для проєктів, що мають стабільний обсяг трафіку або для довгострокового використання.
3. **Гнучкість у конфігурації**: На відміну від SaaS-рішень, VPS дозволяє конфігурувати середовище під конкретні потреби проєкту. Це важливо, коли необхідно тонко налаштувати компоненти, такі як база даних, кешування або спеціальні скрипти для обробки даних.
4. **Легкість у масштабуванні**: Хоча хмарні провайдери автоматизують масштабування, VPS також надає можливість легко збільшити ресурси шляхом апгрейду конфігурації сервера або розгортання додаткових серверів. Це робить VPS гнучким рішенням для проєктів середнього та великого масштабу.

Отже, рішення використовувати VPS для цього проєкту дозволяє досягти балансу між повним контролем, безпекою та гнучкістю, що є важливими факторами для реалізації довгострокового продукту.

Вибір моделей ШІ для аналітичного модуля

Однією з ключових функцій проєкту є прогнозування дохідності портфеля користувача на основі історичних даних. Для реалізації цієї функції обиралися методи машинного навчання, що дозволяють працювати з часовими рядами, а також моделі, які підходять для аналізу фінансових даних.

Прогнозування дохідності акцій є складною задачею, оскільки фінансові ринки є нестабільними, і ціни акцій можуть різко змінюватися під впливом зовнішніх факторів. У зв'язку з цим для вирішення цієї задачі використовуються спеціалізовані моделі для прогнозування часових рядів, а саме:

1. **ARIMA (Autoregressive Integrated Moving Average)** — це класичний підхід до прогнозування часових рядів, який добре підходить для даних із короткими тимчасовими тенденціями. Однак ARIMA має обмеження, оскільки не справляється зі складними трендами та сезонними коливаннями, характерними для ринків акцій [9].
2. **Long Short-Term Memory (LSTM)** — це вид рекурентних нейронних мереж (RNN), який є популярним для аналізу часових рядів. LSTM добре працює з довгими часовими рядами і може запам'ятовувати важливі патерни в історичних даних, що робить його корисним для прогнозування цін на акції [8].
3. **Prophet** — бібліотека, розроблена компанією Facebook для роботи з часовими рядами, яку часто використовують для фінансових прогнозів. Prophet добре справляється з даними, що мають тенденції і сезонність, проте виявляється менш ефективним для ринків із сильною волатильністю, таких як ринки акцій [9].

4. **Temporal Fusion Transformer (TFT)** — одна з найсучасніших моделей для аналізу часових рядів, яка поєднує в собі можливості глибокого навчання та традиційних підходів для прогнозування. Ця модель забезпечує високу точність, оскільки може обробляти складні взаємодії між різними факторами, що впливають на фінансові дані [10].

Обґрунтування вибору моделі для прогнозування

На основі аналізу доступних джерел, таких як **Neptune.ai**, ми вирішили використовувати **LSTM** для прогнозування дохідності активів. **LSTM** вважається ефективною моделлю для фінансових часових рядів, оскільки здатна враховувати довготермінові залежності та обробляти складні часові ряди [8].

Модель **Prophet**, хоча й корисна для багатьох бізнес-застосувань, не є достатньо точною для фінансових даних через залежність від тенденцій і сезонності. Як зазначено в аналізі **Neptune.ai**, Prophet краще підходить для прогнозування даних з меншою волатильністю, таких як обсяги продажів, але менш ефективна для ринкових даних з високою волатильністю, таких як ціни на акції [9].

Temporal Fusion Transformer також була розглянута як модель для порівняння результатів, але через її складність і потребу в великих обсягах даних було вирішено зосередитися на **LSTM**, яка забезпечує кращий баланс між точністю і простотою впровадження [10].

Вибір моделей для прогнозування дохідності портфеля базувався на вимогах до точності та гнучкості. **LSTM** було обрано як основну модель для аналізу історичних даних і прогнозування цін на акції завдяки її здатності враховувати як короткотривалі зміни, так і довготермінові тенденції. Це рішення забезпечує високу точність прогнозування в умовах високої волатильності та складної структури ринку акцій.

ВИСНОВОК ДО РОЗДІЛУ 1

Управління проєктами та архітектура програмного забезпечення відіграють важливу роль у процесі створення сучасних IT-рішень. У цьому розділі було розглянуто основні методології управління проєктами, зокрема гібридні підходи, що поєднують традиційні предиктивні та гнучкі адаптивні методи. Було проаналізовано переваги та недоліки підходів Agile і Scrum, які є основними інструментами для керування IT-проєктами в умовах постійних змін.

З точки зору архітектури програмного забезпечення, було розглянуто різні архітектурні стилі, включно з монолітною, мікросервісною, клієнт-серверною та багатошаровою архітектурами. Кожна з них має свої переваги та недоліки, які слід враховувати під час вибору архітектури для конкретного проєкту.

Було зроблено висновок, що для розробки нашого продукту, найоптимальнішим рішенням є поєднання клієнт-серверної архітектури з елементами модульності, що дозволяє досягти необхідної гнучкості, масштабованості та продуктивності. Використання сучасних технологій, таких як Docker для контейнеризації, Next.js для фронтенду та Python FastAPI для бекенду, забезпечує стабільність та зручність у підтримці проєкту.

На основі аналізу доступних методів прогнозування дохідності було обрано модель LSTM, яка найбільш ефективно працює з фінансовими даними в умовах високої волатильності.

РОЗДІЛ 2. УПРАВЛІННЯ ПРОЄКТОМ РОЗРОБКИ ПРОГРАМНОГО ПРОДУКТУ ДЛЯ ВІДСТЕЖЕННЯ КАПІТАЛУ І ПРОГНОЗУВАННЯ ПРИБУТКУ

2.1 Аналіз ринку схожих програмних продуктів (Firekit, Snowball Analytics)

Для розробки ефективного програмного продукту для відстеження капіталу і прогнозування його прибутковості на основі штучного інтелекту важливо провести аналіз існуючих рішень на ринку. Розглянемо ключові функціональні можливості двох основних продуктів — **Snowball Analytics** і **Firekit**, які пропонують широкі можливості для управління інвестиціями.

Snowball Analytics

Snowball Analytics — це інвестиційний інструмент, орієнтований на управління великими портфелями активів та аналіз їхньої прибутковості. Він надає користувачам можливість вести одночасно кілька портфелів і проводити комплексний аналіз своїх інвестицій.

Основні функції **Snowball Analytics** включають:

- **Необмежена кількість портфелів**: Дозволяє користувачам керувати численними портфелями, що підходить для складних інвестиційних стратегій.
- **Необмежена кількість активів**: Можливість додавати необмежену кількість активів у портфелі, що забезпечує гнучкість у диверсифікації інвестицій.
- **Кастомні активи**: Підтримка додавання власних активів, включаючи депозити та нерухомість, що дозволяє вести облік не лише біржових інструментів.

- **Інструмент для ребалансування портфеля**: Повний набір функцій для автоматизованого та ручного ребалансування активів з метою підтримки оптимального співвідношення різних класів активів у портфелі.
- **Автоматизація відстеження дивідендів**: Система автоматично відстежує дивіденди, прогнозує їх надходження та дозволяє користувачам планувати отримання доходу від інвестицій.
- **Аналітичні інструменти**: Snowball Analytics надає детальний аналіз портфелів за такими показниками, як внутрішня норма рентабельності (IRR), диверсифікація активів за секторами, регіонами та класами.
- **Прогнозування прибутковості та бек-тестування**: Можливість тестування інвестиційних стратегій на основі історичних даних, що дозволяє користувачам виявляти найкращі стратегії для своїх портфелів.

Ці функції роблять Snowball Analytics потужним інструментом для інвесторів, які прагнуть отримати детальний аналіз своїх активів і автоматизувати процес відстеження доходу від інвестицій.

Firekit

Firekit також є інструментом для управління капіталом, який надає широкий набір функцій для інвесторів, що бажають ефективно керувати своїми активами та аналізувати їхню прибутковість.

Основні функції **Firekit** включають:

- **Просунута аналітика**: Firekit надає користувачам можливість аналізувати свої портфелі за різними показниками, такими як ефективність інвестицій, капіталізація, динаміка операцій, історія дивідендів та прогнози. Також доступні візуалізаційні інструменти, такі як теплові карти, для оцінки стану портфеля.

- **Модуль прогнозування**: Цей модуль дозволяє користувачам прогнозувати приріст капіталу та розраховувати можливі сценарії досягнення фінансових цілей, таких як вихід на фінансову свободу.
- **Відстеження різних активів**: Firekit підтримує облік великої кількості активів, включаючи готівку, депозити, акції, ETF, облігації, криптовалюту, бізнес та нерухомість. Це дозволяє ефективно керувати всіма видами інвестицій в одному місці.
- **Підтримка акцій та криптовалют**: Firekit пропонує розширені можливості для відстеження реальної вартості акцій, фондів і криптовалют та їхньої прибутковості. Користувачі отримують доступ до детальної інформації про ринкові інструменти, що дозволяє здійснювати більш обґрунтовані інвестиційні рішення.
- **Прихований режим**: Спеціальний режим дозволяє приховувати реальні показники портфеля під час роботи у публічних місцях, що забезпечує безпеку користувачів.
- **Прості операції з активами**: Firekit дозволяє легко фіксувати прибутки, надходження дивідендів або збитки, а також оновлювати баланс портфеля лише за кілька кліків.

Таким чином, **Firekit** забезпечує користувачів потужними інструментами для аналізу та прогнозування фінансових активів з акцентом на простоту використання і гнучкість керування.

Постановка вимог до проєкту

На основі аналізу існуючих рішень можна виділити основні функції, які будуть реалізовані в рамках розробки програмного продукту для відстеження капіталу і прогнозування його прибутковості:

1. **Можливість додавати портфелі**: Програмний продукт повинен дозволяти створювати та керувати кількома інвестиційними портфелями одночасно. Це дасть змогу користувачам ефективно управляти різними інвестиційними стратегіями.
2. **Можливість обліку готівки і планування бюджету**: Важливо забезпечити функціонал для ведення обліку готівкових коштів та бюджетування, щоб користувачі могли планувати свої фінансові операції.
3. **Показники дохідності**: Продукт має автоматично розраховувати показники прибутковості активів і портфелів, такі як внутрішня норма рентабельності (IRR), показники волатильності та інші ключові індикатори фінансової ефективності.
4. **Кастомні активи**: Система повинна підтримувати додавання користувацьких активів, таких як нерухомість, депозити або інші інвестиційні інструменти, що не відображаються на біржах.
5. **Калькулятор дохідності**: Окремий інструмент для розрахунку прогнозованої дохідності на основі історичних даних і очікуваних ринкових змін.
6. **Інструмент для ребалансування**: Функція автоматизованого або ручного ребалансування портфеля з метою підтримки оптимальної структури активів.
7. **Аналітичний модуль з елементом ШІ**: Основною відмінністю продукту стане аналітичний модуль, який буде використовувати алгоритми штучного інтелекту для прогнозування прибутковості активів на основі часових рядів та інших фінансових даних. Це дозволить користувачам отримувати більш точні прогнози щодо розвитку їхніх інвестицій.
8. **Інтеграція з брокерами:** Продукт повинен забезпечувати можливість інтеграції з популярними брокерами, такими як Interactive Brokers, для

автоматичного імпорту даних з існуючих портфелів користувачів. Це значно спростить процес налаштування системи та дозволить користувачам швидко отримати доступ до актуальних даних про свої інвестиції без необхідності ручного введення інформації.

Ці функції будуть визначати структуру та функціональність програмного продукту, забезпечуючи ефективне керування інвестиціями та підтримуючи користувачів у прийнятті фінансових рішень.

2.2 SWOT-аналіз програмного продукту

Для кращого розуміння позиціонування програмного продукту на ринку та визначення основних чинників, що можуть вплинути на його успіх, було проведено SWOT-аналіз. Цей інструмент дозволяє визначити сильні та слабкі сторони проєкту, а також можливості і загрози, які можуть виникнути в процесі його реалізації та подальшого розвитку.

Сильні сторони (Strengths)

1. **Інноваційний аналітичний модуль з елементом штучного інтелекту**: Програмний продукт використовує передові моделі штучного інтелекту для прогнозування прибутковості капіталу, що надає користувачам конкурентну перевагу в управлінні своїми інвестиціями. Прогнозування на основі історичних даних і часових рядів дозволить підвищити точність фінансових рішень.
2. **Гнучкість у додаванні кастомних активів**: Користувачі можуть додавати до системи свої унікальні активи, такі як нерухомість або приватні інвестиції, що робить продукт універсальним і придатним для різних категорій інвесторів. Це забезпечує більш комплексний облік капіталу, що важливо для великих інвесторів.

3. **Повний функціонал для управління інвестиціями**: Продукт включає широкий набір інструментів для управління портфелями, включаючи автоматизацію обліку дивідендів, інструмент для ребалансування портфеля, а також можливість обліку готівки та планування бюджету. Це дозволяє користувачам комплексно підходити до управління своїми фінансами.
4. **Простота використання та інтуїтивний інтерфейс**: Оскільки цільова аудиторія вже є інвесторами, система розроблена з урахуванням простоти і легкості у використанні. Користувачі отримують інтуїтивно зрозумілий інтерфейс, що робить продукт доступним навіть для тих, хто не має глибоких технічних знань.
5. **Автоматизація через інтеграцію з брокерами**: Підтримка інтеграції з популярними брокерами, такими як Interactive Brokers, дозволяє автоматично імпортувати дані про активи, що суттєво економить час користувачів і робить процес управління інвестиціями більш зручним.

Слабкі сторони (Weaknesses)

1. **Залежність від якості історичних даних**: Продукт використовує штучний інтелект для прогнозування, і його точність напряму залежить від якості та обсягу історичних даних, які будуть використовуватися. Наявність неповних або некоректних даних може негативно вплинути на точність прогнозів.
2. **Конкуренція з усталеними продуктами**: Продукти на кшталт Snowball Analytics і Firekit вже мають свою користувацьку базу та ринкові позиції. Продукт може зіткнутися з труднощами у завоюванні частки ринку, якщо не буде значних відмінностей у функціоналі або конкурентних переваг.

3. **Обмежені ресурси на старті:** Впровадження функцій, таких як інтеграція з криптовалютними біржами або розширена аналітика, потребує значних інвестицій.

Можливості (Opportunities)

1. **Розширення функціональності**: Можливість додавання нових функцій, таких як підтримка нових видів активів, розширені опції для інтеграції з іншими фінансовими сервісами, такими як криптовалютні біржі або платформи для управління бюджетом, дозволить розширити користувацьку базу та збільшити привабливість продукту.
2. **Зростання попиту на автоматизовані фінансові рішення**: З розвитком фінансових технологій та зростанням інтересу до автоматизації інвестиційних процесів попит на продукти, що пропонують прогнозування та аналітику на основі ШІ, зростатиме. Це відкриває можливості для розширення користувацької бази та проникнення на нові ринки.
3. **Масштабованість на міжнародні ринки**: Продукт можна адаптувати для різних фінансових ринків, що дозволить вийти на міжнародні ринки та залучити нових користувачів з інших країн. Враховуючи тенденцію глобалізації фінансових технологій, це відкриває додаткові можливості для масштабування.
4. **Тренд на фінансову грамотність:** Популярність інвестицій серед молодого покоління створює попит на подібні інструменти.

Загрози (Threats)

1. **Нестабільність фінансових ринків:** Швидкі та непередбачувані зміни на фінансових ринках, такі як економічні кризи або регуляторні зміни, можуть вплинути на ефективність прогнозування та загальну стабільність роботи продукту. Це може знизити рівень довіри користувачів до продукту.
2. **Підвищена конкуренція**: Ринок програм для управління інвестиціями є конкурентним, і нові гравці, а також удосконалені версії вже наявних продуктів можуть створити додатковий тиск на наш продукт. Постійне вдосконалення конкурентів вимагатиме значних інвестицій у розвиток продукту для збереження його конкурентних переваг.
3. **Технологічні ризики:** Можливі проблеми з інтеграцією або стабільністю роботи продукту.

Згідно з проведеним SWOT-аналізом, програмний продукт має значний потенціал завдяки своїм сильним сторонам, таким як використання штучного інтелекту для прогнозування прибутковості та підтримка кастомних активів. Інтеграція з брокерами та аналітичний модуль додають йому конкурентних переваг на ринку. Слабкі сторони, такі як залежність від якості даних та високі вимоги до обчислювальних ресурсів, можуть бути компенсовані шляхом удосконалення алгоритмів та оптимізації продуктивності.

Можливості для розвитку включають розширення функціональності та інтеграцію з додатковими сервісами, що підвищить привабливість продукту для користувачів. Однак важливо враховувати загрози, пов'язані з нестабільністю фінансових ринків та конкуренцією, що може вимагати швидкої адаптації продукту до змін ринкових умов і регуляторних вимог. В цілому, сильні сторони і можливості продукту створюють перспективи для успішного виходу на ринок, за умови врахування і пом'якшення можливих загроз і слабких сторін.

2.3 План управління проєктом та його етапи

Планування проєкту є першим і одним із найважливіших етапів, який визначає стратегію та підхід до реалізації всіх завдань. У рамках цього підпункту відбувається визначення основних етапів, їхня послідовність, розподіл ресурсів та часові рамки для кожного етапу. Правильне планування допомагає уникнути проблем із термінами, бюджетом та ресурсами, а також забезпечує чітке бачення всіх етапів розробки.

Основні етапи, що включаються у планування:

1. Підготовчий етап (аналіз вимог та складання технічного завдання)

Цей етап включає в себе збір інформації, аналіз вимог користувачів та визначення технічних специфікацій. Підготовка технічного завдання (ТЗ) є важливою частиною цього етапу, оскільки в ньому фіксуються всі функціональні вимоги, архітектурні рішення та основні цілі проєкту. Складання ТЗ дозволяє команді чітко розуміти, що потрібно реалізувати на кожному етапі, та мінімізує ризики непорозумінь.

Основні завдання цього етапу:

- Визначення ключових функцій продукту, таких як облік портфелів, аналітика, інтеграція з брокерами, модуль штучного інтелекту для прогнозування.
- Аналіз ринку та вивчення конкурентних рішень для визначення унікальних переваг продукту.
- Визначення технологій для розробки: фронтенд (React.js, Next.js), бекенд (FastAPI, Python), база даних (PostgreSQL), аналітичні моделі (LSTM, інші алгоритми для прогнозування).

2. Розробка MVP (Minimum Viable Product)

MVP — це мінімально життєздатна версія продукту, яка включає лише ключові функції, необхідні для того, щоб продукт міг бути використаний першими користувачами. Основна мета MVP — швидко запустити базовий продукт, зібрати фідбек від користувачів і виявити слабкі місця та можливі покращення на ранніх етапах.

Основні кроки розробки MVP:

- **Клієнтська частина**: Розробка інтерфейсу користувача для обліку портфелів і взаємодії з системою. Це включає дизайн інтуїтивного та зручного інтерфейсу, який дозволить користувачам додавати свої активи, переглядати аналітику і здійснювати інтеграцію з брокерами.
- **Серверна частина**: Створення бекенду, який оброблятиме дані, запити від користувачів, виконуватиме аналітичні обчислення, та підтримуватиме взаємодію з брокерськими платформами через API.
- **База даних**: Налаштування бази даних для зберігання інформації про користувачів, їхні портфелі, транзакції та результати аналітичних обчислень.
- **Інтеграція з брокерами**: Включає початкову інтеграцію з одним або кількома брокерськими платформами, такими як Interactive Brokers, щоб користувачі могли автоматично імпортувати свої дані.
- **Тестування MVP**: Після розробки базової версії продукту проводиться тестування на перших користувачах для збору фідбеку. Це допоможе визначити ключові проблеми та виявити можливості для покращення перед впровадженням нових функцій.

Основні функції MVP:

- Облік інвестиційних портфелів.

- Показники дохідності.
- Базовий калькулятор дохідності.
- Інтеграція з брокерами для імпорту даних.

3. Інтеграція аналітичного модуля ШІ

На цьому етапі відбувається розробка та інтеграція модуля штучного інтелекту, який буде використовуватися для прогнозування прибутковості активів на основі історичних даних. Моделі ШІ допоможуть користувачам приймати більш обґрунтовані інвестиційні рішення.

Основні етапи інтеграції ШІ:

- **Вибір моделі ШІ**: Для аналізу часових рядів і прогнозування було вирішено використовувати **LSTM** (Long Short-Term Memory), яка добре підходить для роботи з фінансовими даними. Також можуть використовуватися інші моделі, наприклад, Prophet чи GRU, залежно від вимог.
- **Навчання моделі**: Після вибору моделі проводиться навчання на реальних фінансових даних для досягнення максимальної точності прогнозування.
- **Інтеграція з продуктом**: Моделі ШІ інтегруються з бекендом для обробки запитів від користувачів і надання прогнозів по їхніх портфелях.
- **Тестування модуля ШІ**: Тестування моделей на точність та ефективність у реальних умовах, з використанням історичних даних користувачів.

Цей етап передбачає детальне налаштування моделі ШІ, щоб вона могла правильно працювати з фінансовими даними користувачів і надавати точні прогнози.

4. Тестування та налагодження

Тестування продукту є одним із ключових етапів для забезпечення його стабільної роботи та виявлення помилок до запуску на ринок. Під час тестування перевіряється функціональність всіх модулів системи, включаючи роботу портфелів, інтеграцію з брокерами, точність прогнозів ШІ та загальну продуктивність.

Основні завдання тестування:

- **Функціональне тестування**: Перевірка роботи всіх функцій продукту, зокрема додавання та керування портфелями, облік готівки, інтеграція з брокерами та аналітичні модулі.
- **Навантажувальне тестування**: Перевірка здатності системи працювати під значним навантаженням (багато користувачів одночасно, великий обсяг даних).
- **Тестування прогнозів ШІ**: Оцінка точності та надійності прогнозів, які надає модель ШІ на основі історичних фінансових даних.
- **Збір фідбеку від користувачів**: Залучення перших користувачів до тестування продукту для збору їхньої думки про зручність інтерфейсу, ефективність функцій та можливі вдосконалення.

5. Запуск та впровадження продукту

Після завершення всіх попередніх етапів продукт готовий до фінального запуску. Цей етап передбачає не лише технічний запуск, але й підготовку маркетингових кампаній, організацію підтримки користувачів та подальше масштабування системи.

Основні завдання:

- **Розгортання продукту**: Публічний запуск продукту, доступність на веб- або мобільній платформі.

- **Підтримка та моніторинг**: Забезпечення технічної підтримки користувачів, моніторинг роботи продукту для виявлення та усунення можливих помилок.
- **Масштабування**: Оптимізація серверної інфраструктури та системи для підтримки великої кількості користувачів.

Планування є основою для успішного виконання всіх наступних етапів. У цьому підпункті ми визначили ключові етапи розробки продукту: починаючи від збору вимог і складання ТЗ, через розробку MVP та інтеграцію ШІ, до тестування та фінального запуску продукту на ринок. Кожен етап має свої специфічні завдання, і їх правильне планування дозволить мінімізувати ризики та забезпечити своєчасне виконання проєкту.

2.4 Розподіл ресурсів: робота команди та структура WBS

Однією з найважливіших фаз управління проєктом є планування, яке визначає чіткі етапи роботи, розподіл відповідальностей та оптимальне використання ресурсів. Ефективне планування дозволяє мінімізувати ризики, забезпечити високу якість розробки продукту та дотримання встановлених термінів. У рамках розробки програмного продукту для відстеження капіталу та прогнозування прибутковості були визначені ключові етапи, які послідовно реалізуються в ході виконання проєкту.

Робота над програмним продуктом розділена на кілька основних фаз, що включають початкове дослідження, проєктування, розробку, тестування та впровадження продукту. Кожна з цих фаз включає конкретні підетапи та завдання, що дозволяють забезпечити поступове досягнення поставлених цілей.

Для ефективного виконання завдань розробки програмного продукту була створена чітка структура команди, що складається зі спеціалістів, кожен з яких відповідає за певний напрямок робіт. Розподіл ролей та обов’язків всередині команди

дозволяє оптимізувати процес розробки та забезпечити своєчасне виконання поставлених завдань.

Організаційна структура команди

1. **Проєктний менеджер (PM)**:
    - Відповідає за загальне управління проєктом, контроль за виконанням завдань, координацію дій команди та забезпечення дотримання графіку робіт. PM також займається вирішенням проблем, що виникають у процесі реалізації проєкту, та забезпечує комунікацію між командою та стейкхолдерами.
2. **Бізнес аналітик**:
    - Займається збором вимог до продукту, аналізом ринку та визначенням ключових функціональних можливостей продукту. Business Analyst взаємодіє з користувачами для уточнення їхніх потреб та формує бізнес-вимоги для команди розробників.
3. **Системний архітектор**:
    - Відповідальний за технічне проєктування архітектури продукту, вибір технологій, що будуть використовуватися для розробки бекенду, фронтенду, бази даних та інтеграцій з брокерами.
4. **Front-End розробник**:
    - Розробляє інтерфейс користувача, забезпечує взаємодію користувача з системою через веб-браузер або мобільний додаток. Відповідальний за реалізацію функцій взаємодії з портфелями та аналітичними даними.
5. **Back-End розробник**:
    - Розробляє серверну частину продукту, відповідає за обробку даних, забезпечує взаємодію між базою даних та користувацьким

інтерфейсом. Також відповідає за реалізацію функцій обліку активів та взаємодію з брокерськими платформами через API.

6. **Data Scientist**:
    - Розробляє моделі штучного інтелекту для прогнозування прибутковості капіталу на основі часових рядів. Data Scientist також займається аналізом даних, які використовуються для моделювання та прогнозування.
7. **QA (Тестувальник)**:
    - Забезпечує тестування продукту на всіх етапах розробки. Тестувальник проводить функціональне, навантажувальне тестування, перевіряє роботу аналітичних модулів та тестує взаємодію з брокерами.
8. **DevOps Інженер**:
    - Відповідає за налаштування інфраструктури та автоматизацію процесів розгортання системи. DevOps Інженер також підтримує безперервну інтеграцію (CI/CD), моніторить стабільність системи та займається її масштабуванням.

WBS (Work Breakdown Structure)

Для того, щоб структурувати всі етапи роботи над проєктом та забезпечити чіткий контроль за виконанням завдань, використовується система WBS (Work Breakdown Structure). WBS — це важливий інструмент планування, який пов’язує цілі з ресурсами та діяльністю в логічну структуру, сприяючи управлінню програмним проектом [13]. WBS дозволяє розбити весь проєкт на окремі завдання, які легко контролювати та вимірювати. Кожен рівень WBS відображає послідовність робіт, що необхідні для досягнення кінцевої мети.

Фаза 1: Ініціалізація

1.1 Визначення цілей і обсягу проєкту

1.1.1 Формування загальних цілей продукту

1.1.2 Визначення функціональних вимог

1.1.3 Розробка технічного завдання (ТЗ)

1.2 Ідентифікація зацікавлених сторін (стейкхолдерів)

1.2.1 Визначення основних стейкхолдерів

1.2.2 Визначення бізнес-вимог від користувачів

1.3 Формування команди проєкту

1.3.1 Найм спеціалістів для розробки продукту

1.3.2 Розподіл ролей та обов’язків у команді

1.4 Організація Kick-off зустрічі

1.4.1 Підготовка презентаційних матеріалів

1.4.2 Проведення зустрічі з усіма членами команди

Фаза 2: Планування та підготовка

2.1 Аналіз ринку та збір вимог

2.1.1 Проведення маркетингового дослідження

2.1.2 Збір бізнес-вимог користувачів

2.1.3 Визначення технічних вимог до продукту

2.2 Підготовка юридичної та ліцензійної документації

2.2.1 Підготовка необхідних юридичних документів для ведення підприємницької діяльності

2.3 Планування команди та ресурсів

2.3.1 Формування детального плану розробки програмного продукту

2.3.2 Планування ресурсів для реалізації проєкту (включаючи програмні та апаратні засоби)

2.4 Розробка маркетингової стратегії

2.4.1 Аналіз цільової аудиторії та конкурентів

2.4.2 Підготовка рекламних матеріалів для просування продукту

Фаза 3: Розробка

3.1 Розробка клієнтської частини (Frontend)

3.1.1 Проєктування архітектури інтерфейсу користувача

3.1.2 Реалізація інтерфейсу користувача для управління портфелями

3.1.3 Тестування клієнтської частини продукту

3.2 Розробка серверної частини (Backend)

3.2.1 Проєктування архітектури серверної частини

3.2.2 Розробка основного функціоналу для обробки даних

3.2.3 Інтеграція з брокерами через API

3.3 Налаштування інфраструктури та DevOps

3.3.1 Налаштування середовища розробки

3.3.2 Конфігурація серверів та баз даних

3.3.3 Впровадження процесів CI/CD для автоматичного розгортання

3.4 Інтеграція аналітичного модуля ШІ

3.4.1 Розробка моделей ШІ для прогнозування прибутковості

3.4.2 Інтеграція ШІ в загальну архітектуру продукту

3.4.3 Тестування прогнозів і точності ШІ

Фаза 4: Забезпечення якості та тестування

4.1 Забезпечення якості (QA)

4.1.1 Розробка тестових сценаріїв для перевірки функціональності

4.1.2 Проведення функціонального тестування продукту

4.1.3 Проведення навантажувального тестування для оцінки продуктивності

4.2 Приймання продукту користувачами

4.2.1 Проведення тестування на реальних користувачах

4.2.2 Збір відгуків і внесення коректив у продукт

У рамках описаного плану управління проєктом та структури WBS, чітко визначені основні етапи виконання робіт, що дозволяє здійснювати контроль за виконанням завдань та забезпечити своєчасне досягнення цілей проєкту.

2.5 Ризики та виклики

Після побудови плану виконання проєкту (WBS) та визначення основних етапів розробки важливо оцінити можливі ризики, які можуть вплинути на виконання завдань, дотримання строків та ресурсів. SWOT-аналіз, проведений раніше, частково відобразив деякі ризики на рівні загальних загроз для продукту, проте тепер слід більш детально розглянути виклики, що можуть виникати на кожному етапі виконання проєкту. Це дозволить краще підготуватися до можливих затримок або перешкод і забезпечити своєчасне реагування на них.

Ризики можна поділити на кілька категорій залежно від їх природи: технічні, операційні, фінансові та конкурентні. Вони безпосередньо стосуються виконання завдань, передбачених у WBS, а також можуть впливати на загальну ефективність управління проєктом.

Технічні ризики

1. **Інтеграція з брокерами**. Одним із ключових технічних завдань є налагодження стабільної інтеграції з брокерами, такими як Interactive Brokers. Враховуючи, що API брокерів можуть змінюватися, це створює ризик невідповідності між системами.
    - **Управління ризиком**: Постійний моніторинг змін в API брокерів та впровадження гнучкої системи оновлення інтеграцій.

2. **Масштабованість системи**. Збільшення кількості користувачів може створити проблеми з продуктивністю системи, особливо на етапі, коли обробляються великі обсяги історичних даних для аналітики та прогнозування.
    - **Управління ризиком**: Передбачення горизонтального масштабування серверів та оптимізація архітектури системи на ранніх етапах розробки.

Операційні ризики

1. **Якість і доступність даних**. Модулі штучного інтелекту, які використовуються для прогнозування прибутковості, залежать від якісних історичних даних. Проблеми з отриманням або обробкою таких даних можуть суттєво знизити точність прогнозів.
    - **Управління ризиком**: Важливо забезпечити надійні джерела даних, а також створити алгоритми, які будуть перевіряти якість даних перед їх обробкою.
2. **Нестабільність роботи команди**. Збої в роботі або комунікації між членами команди можуть спричинити затримки або зниження якості виконання завдань.
    - **Управління ризиком**: Регулярні внутрішні зустрічі та налагоджена комунікація допоможуть своєчасно виявляти проблеми та вирішувати їх. Також важливо мати плани на випадок заміни ключових спеціалістів.

Фінансові ризики

1. **Перевищення бюджету**. У разі затримки на окремих етапах може виникнути потреба в додаткових фінансових витратах, що негативно вплине на загальний бюджет проєкту.
    - **Управління ризиком**: Запасний фінансовий резерв, контроль за витратами на кожному етапі та постійний моніторинг бюджету допоможуть уникнути перевитрат.

Конкурентні ризики

1. **Конкуренція на ринку**. Продукти, подібні до Snowball Analytics і Firekit, вже мають велику базу користувачів і сильні ринкові позиції. Наш продукт може зіткнутися з проблемою швидкого залучення користувачів.
    - **Управління ризиком**: Виділення ключових унікальних функцій, таких як модуль ШІ, гнучкість у додаванні кастомних активів і інтеграція з брокерами, може підвищити конкурентоздатність продукту. Також варто розробити стратегію маркетингу для просування унікальних функцій продукту.

Таким чином, ризики можна ідентифікувати на різних етапах виконання проєкту, і важливо заздалегідь підготувати стратегії їх мінімізації. Такий підхід дозволить команді вчасно реагувати на виклики та забезпечити своєчасне виконання завдань, що вказані у WBS.

ВИСНОВОК ДО РОЗДІЛУ 2

У другому розділі було проведено детальний аналіз процесу управління проєктом розробки програмного продукту для відстеження капіталу та прогнозування його прибутковості. На основі вивчення схожих рішень, таких як

Snowball Analytics і Firekit, були сформовані вимоги до продукту, що базуються на найкращих практиках ринку та технологічних можливостях. У процесі аналізу ринку були виділені ключові функціональні елементи, серед яких інтеграція з брокерами, можливість додавання кастомних активів, автоматизація розрахунків дохідності та впровадження аналітичного модуля з використанням ШІ.

На основі SWOT-аналізу було визначено сильні та слабкі сторони продукту, можливості для розвитку та загрози, з якими він може зіткнутися на ринку. Це дозволило сформувати бачення майбутньої конкурентоспроможності продукту та виявити критичні аспекти, які потребують уваги на етапах розробки і виходу на ринок.

У розділі також було представлено план управління проєктом, включаючи розподіл ресурсів та визначення ключових етапів реалізації. Структура WBS допомогла розбити процес розробки на конкретні завдання та підетапи, що дозволяє забезпечити чіткий контроль за виконанням робіт та своєчасне досягнення цілей проєкту. Крім того, детально розглянуто можливі технічні, операційні, фінансові та конкурентні ризики, а також шляхи їх мінімізації.

У підсумку, розробка програмного продукту для відстеження капіталу та прогнозування прибутковості є складним процесом, який потребує ретельного планування та управління. Чітке визначення вимог до продукту, аналіз ринку, правильна структура команди, управління ризиками та використання сучасних технологій, таких як штучний інтелект, забезпечать успішну реалізацію проєкту.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ ДЛЯ ВІДСТЕЖЕННЯ КАПІТАЛУ ТА ПРОГНОЗУВАННЯ ПРИБУТКУ ЗА ДОПОМОГОЮ ШІ

3.1 Проєктування бази даних

У сучасному програмуванні використовуються різні типи баз даних, кожен з яких має свої особливості, переваги та недоліки, що робить їх придатними для різних завдань і сфер застосування. Основними типами є реляційні та нереляційні бази даних.

Реляційні бази даних (РБД) є одним із найпоширеніших типів систем управління базами даних, основною характеристикою яких є організація даних у вигляді таблиць, між якими існують чіткі зв’язки. Таблиці складаються з рядків і стовпців, де кожен рядок відповідає запису (екземпляру сутності), а кожен стовпець — атрибуту. Дані у реляційних базах зберігаються у формалізованій структурі, що дозволяє забезпечити цілісність та узгодженість даних.

Переваги реляційних баз даних:

- Чітка структура даних: Завдяки нормалізації даних забезпечується уникнення дублювання та надлишковості.
- Підтримка складних запитів: РБД підтримують SQL (Structured Query Language), що дозволяє виконувати складні запити для обробки даних.
- Цілісність даних: Підтримка транзакцій і механізмів забезпечення цілісності даних (ACID-транзакції).

Недоліки реляційних баз даних:

- Обмежена масштабованість: Складність розподілу даних між декількома серверами, що може бути проблемою для дуже великих систем.

- Складність модифікації структури: Внесення змін до структури таблиць може вимагати значних ресурсів та планування.

Нереляційні бази даних (NoSQL) відрізняються від реляційних тим, що не використовують фіксовану схему для зберігання даних. Ці системи призначені для роботи з великими обсягами даних та високою швидкістю обробки. Дані можуть зберігатися у вигляді документів, графів, пар «ключ-значення» або колонок.

Переваги нереляційних баз даних:

- Висока масштабованість: Завдяки горизонтальному масштабуванню можна легко розподіляти дані між багатьма серверами.
- Гнучкість у структурі даних: Відсутність чіткої схеми дозволяє зберігати дані різного формату та змінювати їх структуру без значних витрат часу.

Недоліки нереляційних баз даних:

- Відсутність підтримки складних транзакцій: Не всі NoSQL бази підтримують транзакції, що може бути важливо для систем, де критично необхідно зберігати цілісність даних.
- Обмежена можливість складних запитів: Не всі системи підтримують складні операції над даними, що можуть виконуватися в реляційних системах.

Вибір бази даних для проєкту

Для розробки програмного продукту для відстеження капіталу та прогнозування прибутковості було обрано реляційну базу даних PostgreSQL. Це потужна система управління базами даних з відкритим кодом, яка поєднує переваги

реляційної моделі зі здатністю працювати з нереляційними даними. Основними причинами вибору PostgreSQL для цього проєкту є:

- Цілісність даних та підтримка транзакцій: PostgreSQL відповідає вимогам фінансових систем завдяки підтримці ACID-транзакцій, що гарантує збереження точності та узгодженості даних.
- Розширені можливості запитів: Підтримка складних SQL-запитів дозволяє здійснювати аналітичну обробку даних і виконувати складні операції з об’єднанням кількох таблиць.
- Гнучкість: PostgreSQL підтримує типи даних JSON та JSONB, що дозволяє працювати з нереляційними даними, зберігаючи гнучкість у структурі даних.
- Масштабованість: Система підтримує вертикальне та горизонтальне масштабування, що є важливим для збільшення обсягів даних при розвитку системи.

Проєктування бази даних — це багатоступеневий процес, що включає ряд взаємопов’язаних етапів, кожен з яких має вирішальне значення для успішного створення функціональної та надійної бази даних. Ці етапи допомагають визначити вимоги до системи, структуру даних, взаємозв’язки між ними та спосіб їх зберігання. Основними етапами проєктування бази даних є:

1. Системний аналіз предметної області

Системний аналіз є першим і найважливішим етапом, оскільки він визначає загальне розуміння предметної області та вимог користувачів. На цьому етапі:

- Визначаються реальні об'єкти, які підлягають моделюванню (наприклад, транзакції, активи, портфелі).

- Аналізуються взаємозв'язки між об'єктами та їхні характеристики.
- Формуються початкові вимоги до бази даних на основі аналізу функціональних потреб користувачів і системи загалом.

Цей етап передбачає детальне вивчення технічного завдання, виявлення основних процесів і даних, які будуть зберігатися та оброблятися в системі. Результати системного аналізу стають основою для наступних етапів проєктування.

2. Концептуальне проєктування

Концептуальне проєктування полягає у створенні концептуальної моделі бази даних, яка надає високорівневий огляд даних і зв’язків між ними без прив'язки до конкретної СУБД. На цьому етапі:

- Визначаються основні сутності та їхні атрибути (наприклад, користувачі, активи, транзакції).
- Встановлюються зв’язки між сутностями з урахуванням їхніх характеристик.
- Будується ER-діаграма (Entity-Relationship Diagram), яка ілюструє зв’язки між сутностями і допомагає зрозуміти логічну структуру бази даних.
- Метою цього етапу є створення чіткої інфологічної моделі, що забезпечить правильне відображення предметної області та її взаємозв’язків.

3. Логічне проєктування

Логічне проєктування ґрунтується на концептуальній моделі та включає розробку логічної структури бази даних, що буде реалізована у вибраній СУБД (у нашому випадку, PostgreSQL). На цьому етапі:

- Проводиться нормалізація даних для уникнення дублювання та забезпечення цілісності даних.
- Створюється логічна модель таблиць, визначаються їхні ключі, атрибути та зв’язки.
- Перевіряється відповідність моделі вимогам транзакцій і безпеки даних.

Цей етап дозволяє структурувати дані таким чином, щоб забезпечити ефективне зберігання та обробку інформації.

4. Фізичне проєктування

Фізичне проєктування визначає засоби фізичної реалізації логічної моделі. На цьому етапі:

- Визначаються типи даних, які використовуватимуться для атрибутів таблиць.
- Розробляється план розміщення даних на фізичних носіях для забезпечення оптимальної швидкості доступу до даних.
- Впроваджуються механізми індексації для прискорення виконання запитів.
- Налаштовуються параметри збереження даних, такі як розмір буферів, механізми кешування та оптимізація використання дискового простору.

Фізичне проєктування завершується створенням конкретних SQL-скриптів для створення таблиць, індексів і зовнішніх ключів, що забезпечують цілісність даних і ефективну обробку запитів.

5. Тестування та оптимізація

Після реалізації фізичної моделі проводиться тестування бази даних. Основні завдання на цьому етапі:

- Перевірка цілісності даних і правильності відображення зв’язків між сутностями.
- Тестування продуктивності бази даних при високих навантаженнях.
- Оптимізація запитів і структури даних для підвищення швидкодії.
- Цей етап забезпечує підготовку бази даних до експлуатації, перевірку її стійкості до навантажень і відповідності заданим вимогам.

Концептуальне проєктування бази даних

Мета концептуального проєктування бази даних полягає у створенні моделі, яка відображає сутності предметної області, їх атрибути та зв'язки між ними у найбільш природній для розуміння формі. Ця модель стає основою для подальших етапів логічного і фізичного проєктування. В основі концептуального моделювання лежить створення ER-діаграм (діаграм «Сутність – Зв’язок»), які візуально відображають структуру даних і зв’язки між елементами.

Основні елементи концептуальної моделі

- Сутність — це унікальний об'єкт предметної області, інформацію про який потрібно зберігати в базі даних.
- Атрибут — характеристика сутності, яка визначає її властивості.
- Ключ — мінімальний набір атрибутів, що однозначно ідентифікує сутність.
- Зв'язок між сутностями — функціональні залежності, які відображають взаємодію між сутностями. Зв'язки бувають трьох типів: один-до-одного (1:1), один-до-багатьох (1), багато-до-багатьох (N).

На основі аналізу предметної області і даних, представлених у вихідному файлі з описом структури бази даних, було визначено наступні основні сутності та побудовано ER-діаграму:

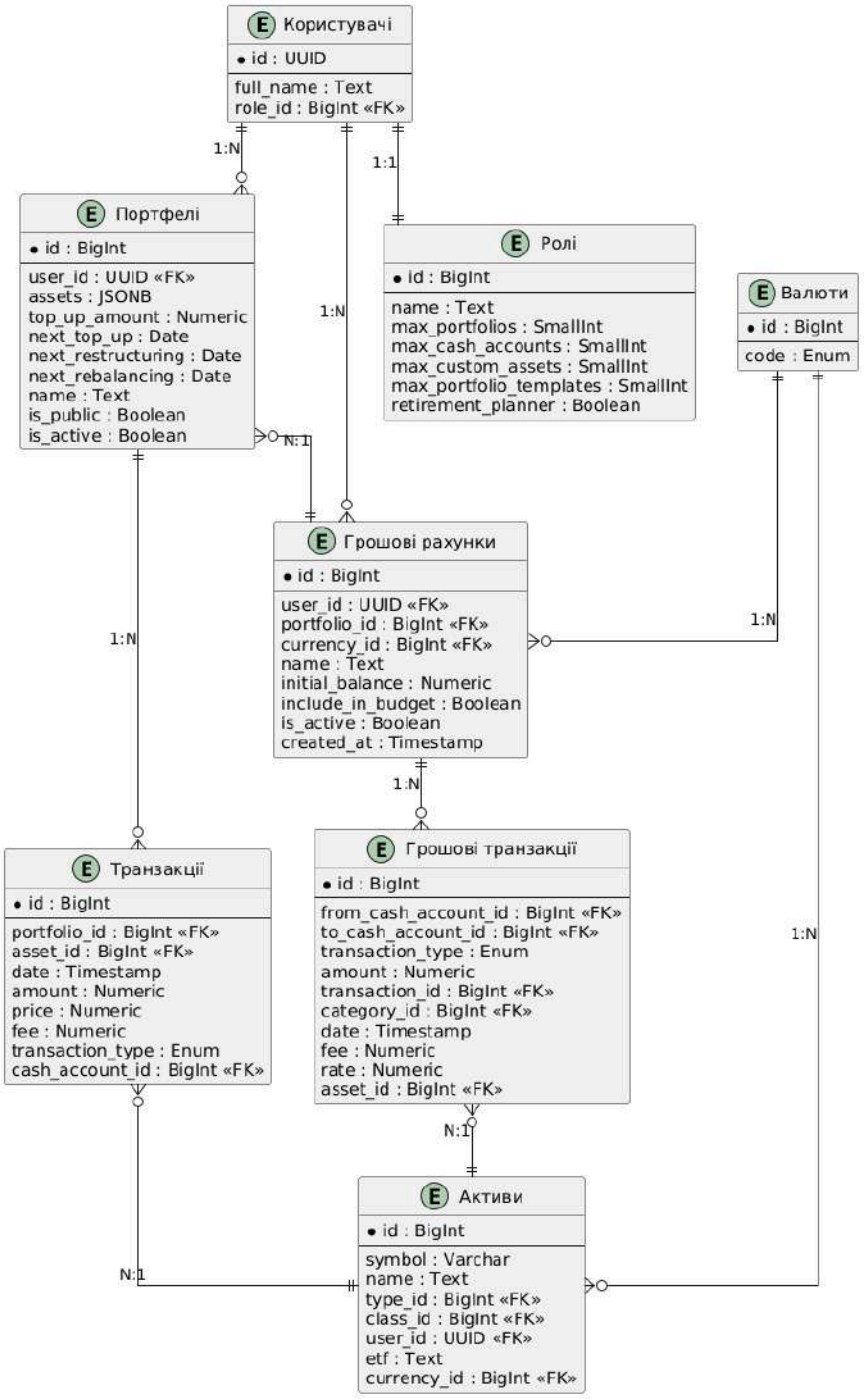


Рисунок 3.1 ER-діаграма бази даних

Сутність «Користувач» (users) — є центральною для управління доступом та персоналізованими функціями в системі. Вона зберігає інформацію про зареєстрованих користувачів і включає такі атрибути, як унікальний ідентифікатор, повне ім'я користувача тощо. Основною метою цієї сутності є забезпечення збереження інформації про користувачів, які взаємодіють із системою. Зв'язки цієї сутності з іншими сутностями, такими як «Портфелі» та «Підписки користувачів», дозволяють інтегрувати її в загальну структуру бази даних.

Сутність «Портфелі» (portfolios) — представляє інвестиційні портфелі, що належать користувачам. Вона містить такі атрибути, як унікальний ідентифікатор портфеля, ім'я, ідентифікатор користувача, активи та інформацію про дати поповнення та реструктуризації портфеля. Основна мета цієї сутності — зберігати та відстежувати інвестиційні стратегії користувачів, а також взаємодіяти з іншими сутностями, такими як «Транзакції» та «Грошові рахунки».

Сутність «Транзакції» (transactions) — використовується для відображення фінансових операцій, що здійснюються в межах портфелів. Вона містить дані про суму, дату, тип транзакції, а також інформацію про активи, що беруть участь у транзакціях. Ця сутність є критичною для моніторингу та обліку фінансових операцій користувачів і має зв'язки з такими сутностями, як «Активи» та «Портфелі».

Сутність «Активи» (assets) — зберігає інформацію про різні активи, які можуть бути додані до портфелів користувачів. Вона включає атрибути, як-от унікальний ідентифікатор, символ, назва, тип і клас активу. Зв'язки цієї сутності з іншими таблицями, такими як «Типи активів» та «Історичні ціни активів», дозволяють відстежувати зміни у вартості активів і керувати інвестиційними портфелями.

Сутність «Історичні ціни активів» (assets_history_prices) — містить інформацію про зміну вартості активів з плином часу. Вона допомагає у створенні аналітичних

звітів і прогнозуванні фінансових трендів. Основні атрибути включають дату запису, ідентифікатор активу, ціну та відсоткові зміни. Сутність пов'язана з таблицею «Активи», що дозволяє проводити комплексний аналіз вартості активів.

Сутність «Грошові рахунки» (cash_accounts) — відповідає за збереження даних про грошові активи користувачів, їхні баланси, валюти та асоційовані портфелі. Основні атрибути включають ідентифікатор рахунку, баланс, ім'я рахунку та валюту. Ця сутність зв'язана з «Портфелями» і «Користувачами» для комплексного управління фінансовими ресурсами користувачів.

Сутність «Валюти» (currencies) — містить інформацію про всі валюти, що підтримуються системою. Її атрибути включають ідентифікатор та код валюти. Сутність використовується для забезпечення точності фінансових операцій і підтримки багатовалютних портфелів, зв'язуючи дані з іншими сутностями, такими як «Грошові рахунки» та «Транзакції».

Сутність «Категорії грошових транзакцій» (cash_transactions_categories) — містить інформацію про типи грошових операцій та їх доступність для різних функцій у системі. Вона має атрибути, такі як ідентифікатор, назва та тип транзакції.

Сутність «Грошові транзакції» містить записи про фінансові операції між різними грошовими рахунками користувачів. Вона включає такі атрибути, як ідентифікатор транзакції, сума, дата, тип транзакції, комісії та рахунки, з яких і на які переводяться кошти. Основна мета цієї сутності — відстеження всіх рухів фінансів, забезпечення прозорості та точності фінансових операцій. Зв’язки включають посилання на сутності «Грошові рахунки» та «Активи» для забезпечення зв’язку з відповідними фінансовими об'єктами.

Сутність «Ролі» (roles) — містить інформацію про різні доступні ролі в системі, їхні обмеження та функціональні можливості. Ключові атрибути включають ідентифікатор ролі, назву ролі та обмеження (наприклад, максимальна кількість портфелів або грошових рахунків). Ця сутність важлива для управління доступом і правами користувачів у системі та має зв'язки із сутністю «Підписки користувачів».

Сутність «Публічні портфелі» (public_portfolios) — зберігають дані про портфелі, доступні для перегляду іншими користувачами або загальнодоступні. Основні атрибути включають ідентифікатор портфеля та дату створення. Ця сутність допомагає користувачам ділитися своїми інвестиційними стратегіями та аналізувати портфелі інших. Зв'язки із сутністю «Портфелі» дозволяють асоціювати кожен запис із відповідним портфелем.

Даталогічне проєктування бази даних

Даталогічне проєктування бази даних – це етап, на якому створюється логічна структура бази даних на основі інфологічної моделі. Логічна структура орієнтована на конкретну систему управління базами даних (СУБД) і враховує її особливості та обмеження. Основною метою даталогічного проєктування є забезпечення зручного й ефективного збереження та обробки даних у системі.

Даталогічна модель відображає сукупність таблиць і зв’язків між ними, а також включає визначення первинних та зовнішніх ключів, які забезпечують логічні зв’язки між таблицями. Для реляційних баз даних ці зв'язки реалізуються за допомогою зовнішніх ключів. Кожна таблиця містить набір атрибутів, серед яких виділяються первинні ключі (PK), що забезпечують унікальність записів, та зовнішні ключі (FK), які забезпечують зв'язки з іншими таблицями.

Переваги даталогічної моделі:

- Відображення логічних зв'язків між сутностями.
- Забезпечення унікальності записів через використання первинних ключів.
- Можливість створення складених ключів для складніших зв'язків.
- Підтримка цілісності даних через визначення зовнішніх ключів.

Перетворення інфологічної моделі в даталогічну:

- Зв'язки 1:1 – кожна сутність перетворюється на окрему таблицю, а зв'язок реалізується через зовнішній ключ.
- Зв'язки 1 – головна сутність перетворюється на таблицю, до якої додається зовнішній ключ у підлеглій таблиці.
- Зв'язки М – додається зв’язувальна таблиця, яка містить первинні ключі обох пов'язаних таблиць і може включати додаткові атрибути.

Даталогічна модель дозволяє чітко визначити структуру бази даних і підготувати її до реалізації на фізичному рівні, адаптуючи модель до можливостей вибраної СУБД. У цьому розділі представлено опис основних сутностей бази даних та їхні атрибути з вказанням ключів, типів даних та обмежень (табл. 3.1).

Таблиця 3.1 Склад та характеристики атрибутів таблиць

Атрибут	Ключ	Обов’язкове значення	Тип даних	Обмеження
Таблиця «Користувачі» (Users)				
id	PK	так	UUID	Унікальний
full_name	-	так	Text	До 255 символів
role_id	FK	так	BigInt	Ідентифікатор з таблиці «Ролі»
Таблиця «Портфелі» (Portfolios)				

Атрибут	Ключ	Обов’язкове значення	Тип даних	Обмеження
id	PK	так	BigInt	Унікальний
user_id	FK	так	UUID	Ідентифікатор з таблиці «Користувачі»
assets	-	ні	JSONB	Немає
top_up_amount	-	ні	Numeric	Позитивне значення
next_top_up	-	ні	Date	Дата у майбутньому
next_restructuring	-	ні	Date	Дата у майбутньому
next_rebalancing	-	ні	Date	Дата у майбутньому
name	-	так	Text	До 255 символів
is_public	-	так	Boolean	true/false
is_active	-	так	Boolean	true/false
Таблиця «Транзакції» (Transactions)				
id	PK	так	BigInt	Унікальний
portfolio_id	FK	так	BigInt	Ідентифікатор з таблиці «Портфелі»
asset_id	FK	так	BigInt	Ідентифікатор з таблиці «Активи»
date	-	так	Timestamp	Дата та час транзакції
amount	-	так	Numeric	Позитивне значення
price	-	так	Numeric	Позитивне значення
fee	-	ні	Numeric	Позитивне значення
transaction_type	-	так	Enum	Тип транзакції
cash_account_id	FK	ні	BigInt	Ідентифікатор з таблиці «Грошові рахунки»
Таблиця «Грошові рахунки» (CashAccounts)				
id	PK	так	BigInt	Унікальний
user_id	FK	так	UUID	Ідентифікатор з таблиці «Користувачі»

Атрибут	Ключ	Обов’язкове значення	Тип даних	Обмеження
portfolio_id	FK	ні	BigInt	Ідентифікатор з таблиці «Портфелі»
currency_id	FK	так	BigInt	Ідентифікатор з таблиці «Валюти»
name	-	так	Text	До 255 символів
initial_balance	-	так	Numeric	Позитивне значення
include_in_budget	-	так	Boolean	true/false
is_active	-	так	Boolean	true/false
created_at	-	так	Timestamp	Дата створення
Таблиця «Грошові транзакції» (CashTransactions)				
id	PK	так	BigInt	Унікальний
from_cash_account_id	FK	так	BigInt	Ідентифікатор з таблиці «Грошові рахунки»
to_cash_account_id	FK	так	BigInt	Ідентифікатор з таблиці «Грошові рахунки»
transaction_type	-	так	Enum	Тип транзакції
amount	-	так	Numeric	Позитивне значення
transaction_id	FK	ні	BigInt	Ідентифікатор з таблиці «Транзакції»
category_id	FK	ні	BigInt	Ідентифікатор з таблиці «Категорії транзакцій»
date	-	так	Timestamp	Дата транзакції
fee	-	ні	Numeric	Позитивне значення
rate	-	ні	Numeric	Позитивне значення
asset_id	FK	ні	BigInt	Ідентифікатор з таблиці «Активи»
Таблиця «Валюти» (Currencies)				
id	PK	так	BigInt	Унікальний
code	-	так	Enum	Код валюти

Атрибут	Ключ	Обов’язкове значення	Тип даних	Обмеження
Таблиця «Активи» (Assets)				
id	PK	так	BigInt	Унікальний
symbol	-	так	Varchar	До 50 символів
name	-	так	Text	До 255 символів
type_id	FK	так	BigInt	Ідентифікатор з таблиці «Типи активів»
class_id	FK	так	BigInt	Ідентифікатор з таблиці «Класи активів»
user_id	FK	ні	UUID	Ідентифікатор з таблиці «Користувачі»
etf	-	ні	Text	Інформація про ETF
currency_id	FK	так	BigInt	Ідентифікатор з таблиці «Валюти»
Таблиця «Ролі» (Roles)				
id	PK	так	BigInt	Унікальний
name	-	так	Text	До 100 символів
max_portfolios	-	так	SmallInt	Позитивне значення
max_cash_accounts	-	так	SmallInt	Позитивне значення
max_custom_assets	-	так	SmallInt	Позитивне значення
max_portfolio_templates	-	так	SmallInt	Позитивне значення
retirement_planner	-	так	Boolean	true/false

Фізичне проєктування бази даних

Фізичне проєктування бази даних – це ключовий етап, на якому створюється конкретна структура бази даних для обраної системи управління базами даних (СУБД), зокрема PostgreSQL. Цей етап передбачає перетворення логічної моделі у

фізичну, адаптуючи її до вимог та можливостей СУБД для ефективного збереження, обробки та управління даними.

Одним з важливих завдань фізичного проєктування є вибір відповідних типів даних для атрибутів, що зберігаються у базі даних. PostgreSQL підтримує широкий спектр типів даних, таких як `INTEGER`, `BIGINT`, `TEXT`, `VARCHAR`, `BOOLEAN`, `DATE` та `TIMESTAMP`. Зокрема, для унікальної ідентифікації записів використовується тип `UUID`, а для збереження чисел із високою точністю – тип `NUMERIC`. Спеціалізовані типи, як-от `JSONB`, дозволяють зберігати структуровані дані у форматі JSON, забезпечуючи гнучкість і можливість роботи з неструктурованою інформацією. Типи `ENUM` використовуються для обмеження значень певними списками, що зручно для атрибутів, які мають фіксовані варіанти, наприклад, типи транзакцій.

З метою підтримки цілісності даних використовуються ключі. Первинні ключі (PK) гарантують унікальність записів у таблиці, а зовнішні ключі (FK) забезпечують зв’язки між таблицями. Це допомагає створити чітку систему логічних зв’язків, що підтримує узгодженість даних. Наприклад, у таблиці «Портфелі» атрибут `user_id` є зовнішнім ключем, який вказує на користувача з таблиці «Користувачі».

Одним з важливих аспектів фізичного проєктування є створення індексів для підвищення швидкості доступу до даних. PostgreSQL підтримує різні типи індексів, такі як B-tree (стандартний тип для загальних завдань), а також GIN та GiST для індексації полів `JSONB` і повнотекстового пошуку. Використання індексів з умовами дозволяє обмежити їх застосування до певних значень, що оптимізує виконання запитів і знижує навантаження на систему.

Для створення представлень даних у зручному вигляді використовуються представлення (views). Представлення у PostgreSQL – це віртуальні таблиці, які

дозволяють створювати складні запити і представляти дані у спрощеній формі. Матеріалізовані представлення зберігають результат запиту фізично, що значно підвищує швидкість доступу до великих обсягів даних.

Автоматизація процесів у базі даних можлива завдяки використанню тригерів і збережених функцій. Тригери виконують задані дії при певних подіях, наприклад, при вставці, оновленні або видаленні записів. Це дозволяє автоматизувати такі операції, як обчислення нових значень або оновлення зв’язаних записів. Збережені функції, що пишуться на мові `PL/pgSQL`, забезпечують виконання складних бізнес-логічних операцій на рівні бази даних, знижуючи потребу в обробці даних на стороні клієнта.

Важливим аспектом фізичного проєктування є партиціонування великих таблиць, що сприяє підвищенню продуктивності при роботі з великими обсягами даних. PostgreSQL підтримує різні типи партиціонування, зокрема за діапазоном значень, списком і хеш-функціями.

Таким чином, фізичне проєктування бази даних завершує процес переходу від логічної моделі до реальної реалізації системи. Завдяки використанню таких інструментів, як індекси, представлення, тригери та партиціонування, забезпечується висока продуктивність та надійність бази даних, що відповідає вимогам і особливостям обраної СУБД.

3.2 Архітектура програмного продукту та взаємодія компонентів

Архітектура програмного продукту є основою для забезпечення стабільної роботи системи, її ефективного масштабування та адаптивності до змін. У цьому проєкті було обрано клієнт-серверну архітектуру з модульним підходом, що дозволяє оптимізувати структуру системи під сучасні вимоги до продуктивності та гнучкості.

Обґрунтування вибору модульної клієнт-серверної архітектури

Вибір клієнт-серверної архітектури був зумовлений необхідністю чіткого розподілу функцій між клієнтською та серверною частинами, що дає змогу створити систему з високою продуктивністю та швидким відгуком. Така архітектура підтримує чітку взаємодію між компонентами через стандартизовані протоколи, як-от HTTP, забезпечуючи стабільність і безпеку передачі даних. Модульний підхід обрано для зручності управління системою, легкості розширення функціоналу та можливості незалежного оновлення окремих компонентів без впливу на загальну роботу системи.

Модульність дозволяє створювати окремі блоки з чітко визначеними функціями, що спрощує процеси розробки, тестування та масштабування. Цей підхід також дає можливість різним командам працювати над різними компонентами системи паралельно, що скорочує час на впровадження нових функцій. Модульна архітектура також сприяє швидкому реагуванню на зміни вимог бізнесу та технічних аспектів.

Опис архітектури

Клієнтська частина системи реалізована за допомогою фреймворку Next.js, який є потужним інструментом для створення веб-застосунків на основі JavaScript і React. Next.js забезпечує переваги у вигляді серверного рендерингу, що покращує продуктивність застосунку та підтримує SEO-оптимізацію. Завдяки підтримці статичного та динамічного рендерингу сторінок Next.js дозволяє створювати веб-додатки з високою швидкістю завантаження та плавною взаємодією з користувачем.

Фронтенд взаємодіє з бекендом через RESTful API, надсилаючи HTTP-запити для отримання даних про користувачів, портфелі, транзакції та інші ресурси. Окрім

основної бібліотеки Next.js, застосунок використовує додаткові бібліотеки для управління станом, обробки форм, а також для забезпечення маршрутизації та захисту маршрутів.

Серверна частина системи побудована на базі FastAPI, фреймворку для розробки веб-додатків та RESTful API на мові Python. FastAPI вирізняється своєю високою продуктивністю завдяки підтримці асинхронного програмування, що дозволяє ефективно обробляти одночасні запити. Інтуїтивно зрозумілий синтаксис та автоматичне генерування документації на основі OpenAPI роблять цей фреймворк зручним для розробки й підтримки API. Серверна частина забезпечує обробку бізнес-логіки, звертається до бази даних для виконання операцій CRUD і повертає результати клієнту у зручному форматі.

Система використовує Supabase як основний інструмент для управління базою даних PostgreSQL та забезпечення додаткових функцій, таких як автентифікація користувачів і робота з даними в реальному часі. Supabase надає зручні бібліотеки для взаємодії як з фронтендом (JavaScript), так і з бекендом (Python), що спрощує інтеграцію та управління даними. Це дозволяє забезпечити безшовний доступ до бази даних, обробляючи дані з мінімальною затримкою.

Авторизація та безпека

Для забезпечення безпеки використовується механізм автентифікації через Google OAuth та передача JWT-токенів для підтвердження прав доступу користувачів. Це гарантує, що всі запити до серверної частини та бази даних супроводжуються перевіркою автентифікації, підвищуючи рівень захисту системи.

Загальна взаємодія компонентів

Архітектура системи включає взаємодію між клієнтською частиною, серверною частиною та базою даних через стандартизовані протоколи. Клієнтська частина надсилає запити до бекенду для отримання або зміни даних. Серверна частина, обробляючи запити, може взаємодіяти з базою даних Supabase напряму, забезпечуючи ефективне управління даними. Збереження та обробка інформації відбуваються швидко завдяки використанню індексів і оптимізованих запитів.

Детальний огляд взаємодії компонентів системи представлений на діаграмі компонентів архітектури (рис. 3.2). Така модульна клієнт-серверна архітектура забезпечує надійність системи, полегшує її розширення та масштабування, дозволяючи адаптувати її під нові виклики і потреби бізнесу.

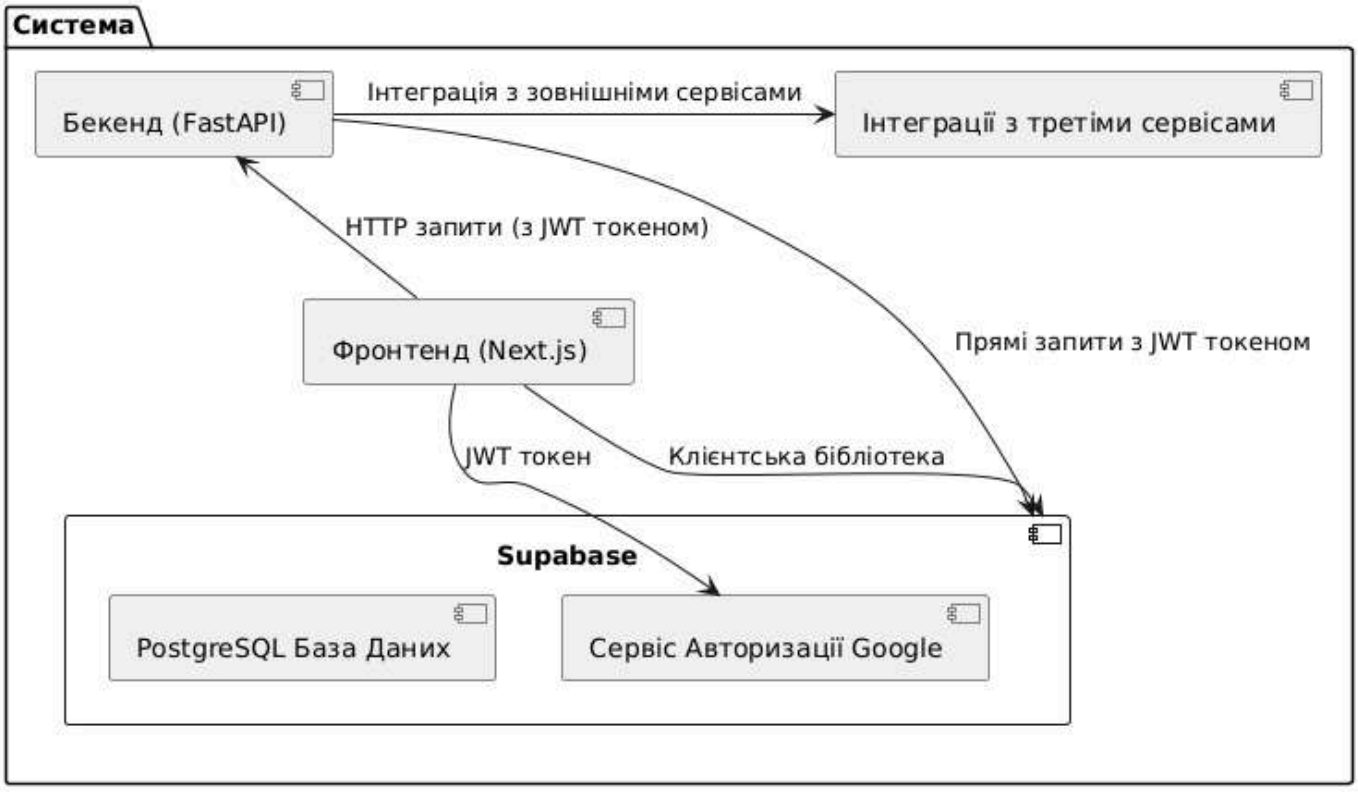


Рисунок 3.2 Діаграма взаємодії компонентів архітектури

3.3 Інтеграція модуля штучного інтелекту для прогнозування прибутковості

Під час аналізу конкурентів у розділі 2 було виявлено, що більшість сучасних інвестиційних платформ використовують модулі аналітики для допомоги користувачам у складанні оптимальних інвестиційних портфелів. Такі модулі часто надають рекомендації на основі історичних даних та стандартних методів аналізу. Нами було прийнято рішення реалізувати схожий модуль, проте з удосконаленням, що полягає у застосуванні методів штучного інтелекту для прогнозування прибутковості портфелів, зокрема використовуючи моделі глибокого навчання.

Вибір моделі штучного інтелекту та її переваги

Прогнозування дохідності портфеля акцій є одним з основних завдань, що стоять перед інвесторами та фінансовими аналітиками. Складність цієї задачі полягає у волатильності ринків, численних зовнішніх факторах і залежностях у часових рядах, які важко передбачити традиційними методами. Саме тому ми обрали використання моделі Long Short-Term Memory (LSTM), що є підвидом рекурентних нейронних мереж (RNN). Завдяки унікальній архітектурі, LSTM має здатність запам’ятовувати довгострокові залежності в даних, що дозволяє їй успішно обробляти складні послідовності, характерні для фінансових часових рядів [14].

LSTM демонструє велику ефективність порівняно з традиційними методами, такими як ARIMA та регресійні підходи, за рахунок можливості обробляти нелінійні взаємозв’язки та залежності. Дослідження підтверджують, що LSTM може покращувати показники ризику та дохідності портфелів, перевершуючи за цими критеріями індекс S&P 500 і інші еталонні стратегії [16], [17]. Наприклад, моделі LSTM дозволяють оптимізувати портфелі шляхом прогнозування цін на акції, що у

свою чергу допомагає створювати портфелі з оптимальним співвідношенням ризику та дохідності [15].

Порівняно з іншими підходами, такими як випадкові ліси (Random Forest) і підтримуючі векторні регресії (SVR), LSTM показує кращі результати в прогнозуванні часових рядів. У дослідженнях зазначено, що LSTM забезпечує більш точні прогнози завдяки своїй здатності обробляти часові залежності та тренди, що характерні для фінансових ринків [19]. Це робить LSTM відмінним інструментом для моделювання ринкових змін і забезпечення гнучкого підходу до аналізу великих обсягів історичних даних.

Особливо цінним є поєднання LSTM з іншими методами, такими як глибокі нейронні мережі (DNN) або механізмами уваги, що підвищують її здатність виділяти найважливіші характеристики у даних [20]. Використання таких гібридних підходів дає змогу інтегрувати фундаментальні дані та аналіз настроїв ринку, що підвищує точність прогнозування. Це дозволяє LSTM не лише ефективно обробляти часові ряди, але й адаптуватися до змінних умов ринку, зберігаючи свою конкурентну перевагу.

Таким чином, вибір LSTM для нашого аналітичного модуля є обґрунтованим і базується на численних перевагах цієї моделі, зокрема її здатності враховувати довготермінові взаємозв’язки, адаптуватися до складних патернів даних та забезпечувати високу точність прогнозів.

Навчання моделі та джерела даних

Навчання моделі LSTM (Long Short-Term Memory) для прогнозування фінансових часових рядів є складним процесом, який включає використання історичних даних для формування прогнозів. Однією з ключових переваг моделі

LSTM є здатність обробляти довготривалі залежності та виявляти важливі закономірності у даних завдяки особливій структурі, яка включає стан комірки (Ct). Цей стан дозволяє зберігати та оновлювати інформацію протягом тривалого часу, що є важливим для моделювання залежностей у часових рядах.

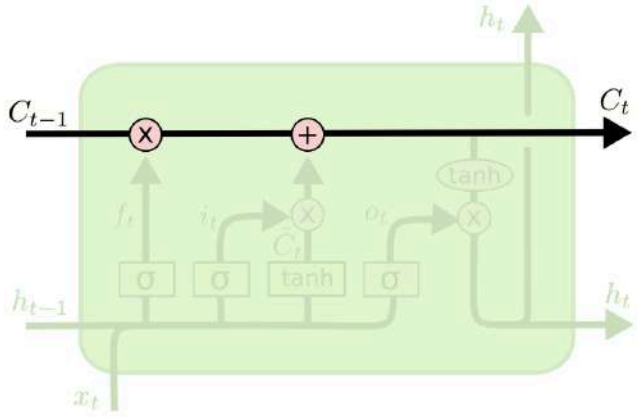
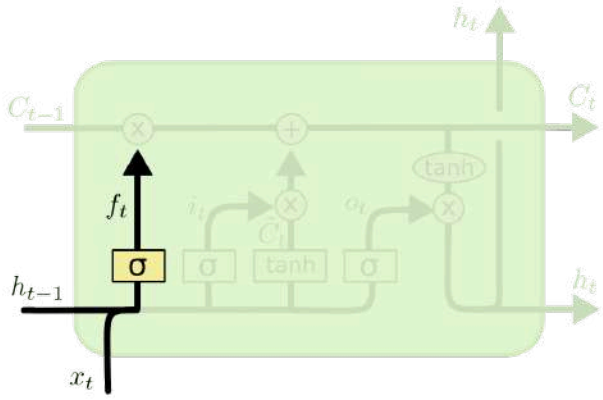


Рисунок 3.3 LSTM. Стан комірки.

LSTM складається з трьох основних шлюзів (воріт), які регулюють потік інформації [21]:

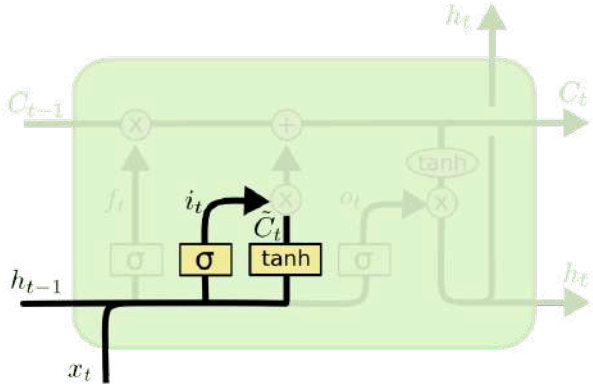
- Шлюз забуття (рис. 3.4) відповідає за те, які частини інформації необхідно видалити зі стану комірки. Для цього застосовується сигмоїдна функція, яка повертає значення у діапазоні від 0 до 1, де 1 означає повне збереження інформації, а 0 — її повне відкидання.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Рисунок 3.4 LSTM. Шлюз забуття

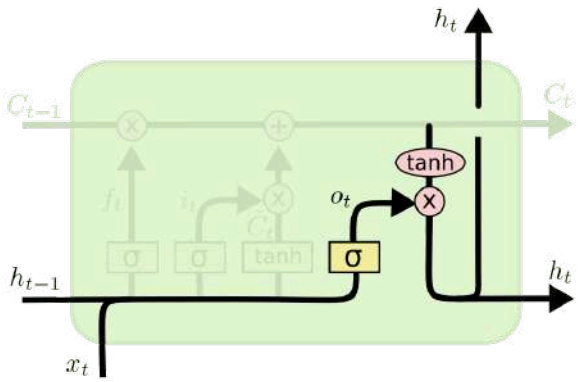
- Вхідний шлюз (рис. 3.5) визначає, які нові дані додати до стану комірки. На цьому етапі застосовується сигмоїдна функція для зважування значень у вхідному векторі, а потім функція tanh обмежує значення між -1 та 1, забезпечуючи баланс під час оновлення стану комірки.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Рисунок 3.5 LSTM. Вхідний шлюз

- Вихідний шлюз (рис. 3.6) контролює, які дані будуть передані далі з поточної комірки до наступної. Як і у вхідному шлюзі, тут використовується сигмоїдна функція разом із функцією tanh, щоб обмежити вихідні значення.



$$o_t = \sigma\left(W_o \; [\, h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Рисунок 3.6 LSTM. Вихідний шлюз

Під час навчання LSTM використовує метод зворотного поширення похибки через час (BPTT) для корекції ваг мережі. Ця методика дозволяє моделі враховувати похибки, що накопичуються протягом багатьох часових кроків, і налаштовувати ваги для підвищення точності прогнозу.

Для навчання моделі використовуються масштабовані дані про ціни акцій, наприклад, отримані з Yahoo Finance. Дані перед тренуванням моделі нормалізуються за допомогою StandardScaler, щоб зменшити вплив великих розбіжностей у значеннях. Вхідні послідовності складаються з векторів, що містять попередні значення цін на акції (наприклад, за останні 50 днів), і використовуються для прогнозування наступного значення.

Процес навчання включає оптимізацію гіперпараметрів моделі, таких як кількість нейронів у прихованих шарах, розмір пакету даних та кількість епох. Використання оптимізатора Adam забезпечує ефективну адаптацію ваг для швидшого сходження.

Після навчання модель тестується на відкладеній вибірці для оцінки ефективності. Показники, такі як корінь середньоквадратичної похибки (RMSE) та середня абсолютна процентна похибка (MAPE), використовуються для оцінки точності прогнозів. Наприклад, модель LSTM у дослідженні показала високий рівень точності з RMSE, значно меншим за результати традиційних методів, що підкреслює її ефективність для прогнозування цін акцій [15].

Результати навчання та оцінка ефективності моделі

Після завершення процесу навчання моделі LSTM результати показали високу точність у прогнозуванні цін на акції, що підтвердило обґрунтованість використання саме цієї архітектури для завдання фінансового аналізу. Основною метою навчання

було створення моделі, яка зможе передбачати зміни цін на основі історичних даних з урахуванням часових залежностей.

Процес навчання моделі включав налаштування декількох ключових параметрів:

- Кількість нейронів у прихованих шарах: У кожному прихованому шарі було використано 50 нейронів, що забезпечує баланс між складністю моделі та її здатністю до узагальнення даних.
- Оптимізатор: Використовувався оптимізатор Adam, який є популярним вибором для моделей глибокого навчання через його здатність адаптивно змінювати коефіцієнт навчання. Цей оптимізатор дозволяє швидше досягати мінімуму функції втрат, ніж класичні алгоритми, такі як стохастичний градієнтний спуск (SGD). Adam поєднує в собі переваги двох інших оптимізаторів — AdaGrad і RMSProp, що робить його ефективним для роботи з великими даними та складними нейронними мережами.
- Кількість епох: Модель навчалася протягом 15 епох, що забезпечило достатню кількість ітерацій для стабільної конвергенції.
- Розмір пакету даних (batch size): Використовувався розмір пакету 20, що дозволяє оновлювати ваги моделі частіше, ніж при використанні великих пакетів, і водночас знижує обчислювальні витрати.

Для ефективного навчання модель потребує правильного масштабування даних. У цьому проєкті використовувався StandardScaler для нормалізації цін акцій, що знижує вплив великих відмінностей у значеннях і забезпечує стабільність навчання. Було обрано StandardScaler замість MinMaxScaler через непередбачувані коливання цін акцій, які не мають постійного мінімуму або максимуму.

Вхідні дані включали послідовності, що містили історичні значення за попередні 50 днів, які використовувалися для прогнозування ціни на наступний день. Цей підхід дозволяє моделі виявляти патерни та залежності у часових рядах.

Після навчання модель була протестована на відкладеній вибірці даних, що не використовувалася під час тренування. Для оцінки ефективності застосовувалися такі метрики:

- Корінь середньоквадратичної похибки (RMSE): Значення RMSE у 12.58 підтвердило, що середня похибка прогнозу є досить низькою, що є суттєвим показником для фінансових прогнозів.
- Середня абсолютна процентна похибка (MAPE): Значення MAPE у 2% свідчить про те, що модель допускає лише незначні відхилення від реальних значень у середньому, що підтверджує її точність.

Графічний аналіз показав майже ідеальне накладення передбачених значень на фактичні дані тестової вибірки (рис. 3.7). Це доводить, що модель LSTM здатна точно передбачати тренди і коливання ринку. Візуалізація трендів була проведена за допомогою побудови графіків порівняння фактичних і прогнозованих цін, що демонструє високу кореляцію між ними.

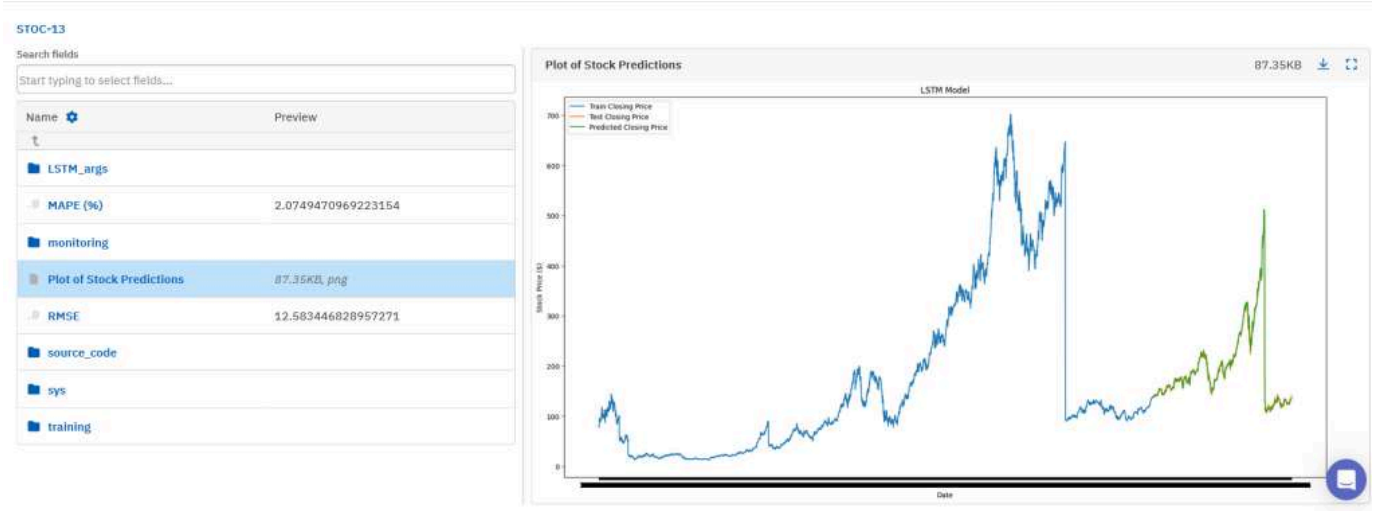


Рисунок 3.7 Результат навчання моделі

Результати навчання показали, що модель LSTM є потужним інструментом для прогнозування фінансових часових рядів, забезпечуючи високий рівень точності. Використання оптимізатора Adam сприяло швидкому та ефективному навчання моделі, що дозволило підвищити її продуктивність у порівнянні з традиційними методами, такими як моделі ковзних середніх (SMA, EMA). Модель демонструє потенціал для інтеграції у реальні фінансові системи з метою підтримки прийняття рішень та підвищення ефективності інвестиційних стратегій.

Розгортання та використання моделі

Модель LSTM буде інтегрована як елемент модуля аналітики в системі та підключена як одна з функцій серверу на базі FastAPI, що працює на Python. Завдяки цьому підходу забезпечується гнучка та швидка обробка запитів користувачів у реальному часі, що є важливим для фінансових застосунків.

Завдяки розгортанню моделі LSTM у застосунку, користувачі отримають змогу доступу до точних прогнозів ринкових трендів, які вони можуть використовувати для створення інвестиційних стратегій. Ця інтеграція надасть наступні можливості:

- **Автоматизоване прогнозування** на основі обраних інвестиційних активів, що дозволить оцінити потенційну прибутковість і ризики як окремих активів, так і їх комбінацій.
- **Інтерактивний інтерфейс** для введення параметрів і даних, які користувач бажає аналізувати, та для отримання результатів у зручному для перегляду форматі.
- **Історичний аналіз і порівняння** фактичних і прогнозованих значень для оцінки ефективності моделі. Це дасть змогу користувачам здійснювати більш інформовані рішення щодо своїх інвестиційних дій.
- **Допомога у створенні інвестиційного портфеля**: на основі автоматизованих прогнозів користувачі зможуть обирати оптимальні активи та їх співвідношення, що відповідають їхній фінансовій стратегії та профілю ризику.

Таким чином, модель LSTM стане ключовим компонентом нашого аналітичного модуля, що підтримує користувачів у прийнятті обґрунтованих інвестиційних рішень та підвищенні ефективності їхніх фінансових стратегій. Інтеграція цього модуля дозволить платформі виділитися серед конкурентів, надаючи користувачам інноваційний інструмент для аналізу та планування інвестицій.

3.4 Процес розгортання програмного продукту

При розгортанні програмного продукту важливим аспектом є вибір платформи для його розміщення. Серед численних варіантів — спільний хостинг, хмарні рішення (наприклад, AWS, Azure, Google Cloud) та віртуальні приватні сервери (VPS). Кожен із цих варіантів має свої переваги та недоліки, але для нашого проєкту було обрано VPS, зважаючи на кілька ключових чинників.

- **Контроль та налаштування середовища** Однією з головних переваг використання VPS є можливість повного контролю над середовищем. Це дозволяє налаштовувати сервер згідно з індивідуальними вимогами, встановлювати потрібні програми та змінювати конфігурації системи. Інші варіанти, такі як спільний хостинг, часто обмежують доступ до системних налаштувань, що може створити труднощі при інтеграції спеціалізованих сервісів або при розгортанні складних додатків.
- **Продуктивність та виділені ресурси** VPS надає виділені ресурси, включаючи оперативну пам'ять, обчислювальну потужність і дисковий простір. Це забезпечує стабільну продуктивність програми навіть при високих навантаженнях. У порівнянні зі спільним хостингом, де ресурси розподіляються між багатьма користувачами, VPS пропонує значно кращу продуктивність та надійність.
- **Безпека** Оскільки VPS є ізольованим середовищем, він забезпечує вищий рівень безпеки, ніж спільний хостинг. Доступ до VPS можна налаштувати через захищені канали, наприклад, використовуючи SSH, що мінімізує ризики несанкціонованого доступу. Крім того, адміністратор сервера має можливість самостійно налаштовувати параметри безпеки, застосовувати правила брандмауера та впроваджувати системи моніторингу для виявлення потенційних загроз.
- **Гнучкість і масштабованість** VPS забезпечує можливість швидкої адаптації до змін у потребах системи. Наприклад, якщо обсяги оброблюваних даних або кількість користувачів значно зростає, ресурсні потужності VPS можуть бути масштабовані шляхом зміни конфігурації сервера. Хмарні рішення також пропонують масштабованість, але вони зазвичай вимагають більших витрат та певного рівня експертизи для налаштування і управління.

- **Вартість** З точки зору вартості, VPS є більш доступним рішенням у порівнянні з хмарними платформами, такими як AWS чи Azure, які можуть вимагати більших витрат на підтримку високої продуктивності та додаткові сервіси. VPS дозволяє оптимізувати витрати, забезпечуючи при цьому необхідний рівень функціональності та продуктивності.

Чому не обрано інші варіанти

- **Спільний хостинг**: недостатньо ресурсів і обмежені можливості налаштування середовища.
- **Хмарні платформи**: висока складність налаштування та більші витрати при використанні додаткових сервісів, таких як автоматичне масштабування або резервне копіювання.
- **Виділені фізичні сервери**: забезпечують максимальний контроль, але значно дорожчі та складніші в управлінні, що може бути надмірним для даного проєкту.

Враховуючи ці аспекти, вибір VPS був логічним рішенням для забезпечення ефективного та стабільного розгортання нашого програмного продукту.

Використання системи управління інфраструктурою

Після вибору VPS для розгортання програмного продукту важливим аспектом є ефективне управління інфраструктурою та спрощення процесу розгортання. Для цієї мети було обрано платформу Coolify, яка забезпечує зручне управління хостингом і контейнеризацією додатків, а також автоматизацію ключових процесів.

Coolify — це сучасна платформа з відкритим вихідним кодом, призначена для автоматизації процесів розгортання та управління інфраструктурою. Завдяки інтеграції з Docker, Coolify дозволяє створювати ізольовані середовища для виконання програмних компонентів, що сприяє стандартизації та спрощенню управління розгортанням. Платформа підтримує роботу з різними типами додатків і сервісів, що робить її універсальним інструментом для проєктів будь-якої складності.

Переваги використання Coolify

- Інтеграція з Docker: Coolify використовує Docker для розгортання додатків, що забезпечує надійне ізольоване середовище виконання. Це підвищує стабільність і передбачуваність роботи програмного забезпечення на всіх етапах розробки, тестування та розгортання.
- Зручний інтерфейс: Платформа пропонує інтуїтивно зрозумілий веб-інтерфейс для управління серверами та розгорнутими додатками. Це знижує потребу в застосуванні командного рядка та дозволяє адміністраторам швидко налаштовувати середовище.
- Автоматизація процесів: Coolify автоматизує розгортання, оновлення та резервне копіювання, що зменшує трудомісткість управління інфраструктурою та підвищує ефективність командної роботи.
- Підтримка різних сервісів: Coolify забезпечує підтримку розгортання не лише веб-додатків і серверних сервісів, але й баз даних (наприклад, PostgreSQL, MySQL, MongoDB), що робить її універсальним рішенням для розробки комплексних систем.
- Моніторинг та безпека: Платформа надає вбудовані інструменти для моніторингу використання ресурсів і контролю стану серверів. Це дозволяє своєчасно виявляти можливі проблеми та забезпечувати

безперебійну роботу системи. Також підтримується інтеграція з Let’s Encrypt для автоматичного налаштування SSL-сертифікатів.

Процес використання Coolify для розгортання

- Інсталяція на сервері: Coolify встановлюється на VPS за допомогою простого скрипта, який автоматично налаштовує Docker та інші необхідні компоненти. Це забезпечує швидке налаштування середовища для розгортання.
- Контейнеризація компонентів: За допомогою Coolify створюються Docker-контейнери для різних компонентів програмного продукту, таких як серверна частина, клієнтська частина та база даних. Це спрощує управління та підтримку всіх елементів системи.
- Налаштування доменів та HTTPS: Інтеграція з Let’s Encrypt забезпечує автоматичне отримання та оновлення SSL-сертифікатів, що підвищує рівень безпеки та надає можливість безпечного доступу до системи через HTTPS.
- Автоматизація резервного копіювання та оновлення: Coolify дозволяє налаштовувати автоматичні оновлення контейнерів і резервне копіювання даних, що сприяє підвищенню надійності та збереженню даних у разі непередбачених ситуацій.

Контейнеризація є ключовою технологією для сучасного розгортання програмного забезпечення, що забезпечує стандартизоване середовище виконання для додатків. У межах даного проєкту контейнеризація здійснюється за допомогою платформи Docker, що надає численні переваги в управлінні програмними компонентами, їх розгортанні та підтримці.

Основні переваги контейнеризації

- Ізольоване середовище виконання: Docker дозволяє запускати додатки у відокремлених контейнерах, кожен з яких містить усі необхідні залежності та конфігурації. Це гарантує, що додаток працює однаково в будь-якому середовищі, від локального розробника до сервера розгортання.
- Швидке розгортання та масштабування: Завдяки контейнерам можна розгортати нові версії програмного забезпечення та масштабувати систему за лічені хвилини. Контейнеризація полегшує запуск додаткових екземплярів сервісів для обробки збільшених навантажень.
- Легкість у підтримці та оновленні: Контейнери дозволяють легко оновлювати та підтримувати додатки, оскільки всі зміни зосереджені у відповідних Docker-образах. Це спрощує процес впровадження змін та забезпечує безперервну інтеграцію та доставку (CI/CD).
- Оптимальне використання ресурсів: Контейнери використовують ресурси системи ефективніше, ніж віртуальні машини, оскільки вони працюють на спільному ядрі операційної системи та мають менший обсяг.

Створення контейнерів для програмних компонентів

Для розгортання проєкту використовуються окремі Docker-файли, які описують конфігурацію контейнерів для кожного компоненту системи: серверної частини (FastAPI), клієнтської частини (Next.js) та бази даних.

У Docker-файлі вказуються основні інструкції для створення контейнера:

- Вибір базового образу: Наприклад, для серверної частини на Python обирається базовий образ python:3.10.
- Встановлення залежностей: Через файл requirements.txt для Python або package.json для Next.js встановлюються необхідні бібліотеки.
- Копіювання вихідного коду: У Docker-контейнер додається вихідний код додатка.
- Налаштування запуску: Вказується команда для запуску додатка, наприклад uvicorn app:app --host 0.0.0.0 --port 8000 для FastAPI.

Приклад базового Docker-файлу для FastAPI:

.dockerfile

```
FROM python:3.10-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
CMD ["uvicorn", "app:app", "--host", "0.0.0.0", "--port", "8000"]
```

Контейнери створюються та запускаються за допомогою команд Docker:

- Створення образу: docker build.
- Запуск контейнера: docker run

Для розгортання всіх компонентів системи використовується Docker Compose, що дозволяє одночасно запускати кілька контейнерів та налаштовувати взаємодію між ними.

Інтеграція проксі-сервера

Для маршрутизації запитів до відповідних контейнерів використовується Caddy, який виступає як проксі-сервер та забезпечує автоматичне налаштування HTTPS. Caddy легко інтегрується з Docker, що дозволяє автоматично підключати контейнери до відповідних маршрутів. Це спрощує управління сертифікатами SSL і гарантує безпечний доступ до системи.

Ключовий приклад конфігурації Caddy:

```
myapp.example.com {
   reverse_proxy myapp:8000
}
```

Використання Caddy у поєднанні з Docker підвищує безпеку, забезпечує масштабованість і спрощує процеси розгортання та підтримки додатків.

3.5 Приклад роботи програмного продукту

Для демонстрації роботи програмного продукту розглянемо основні етапи взаємодії користувача з програмою, включаючи реєстрацію, перегляд дашборду, аналіз портфелів, додавання активів, а також використання функцій прогнозування прибутковості.

Сторінка реєстрації

На рис. 3.8 показано сторінку реєстрації, де новий користувач має можливість створити обліковий запис, ввівши електронну адресу та пароль. Реєстрація дає доступ до всіх функцій програми, зокрема до управління портфелями, прогнозування та інших інструментів.

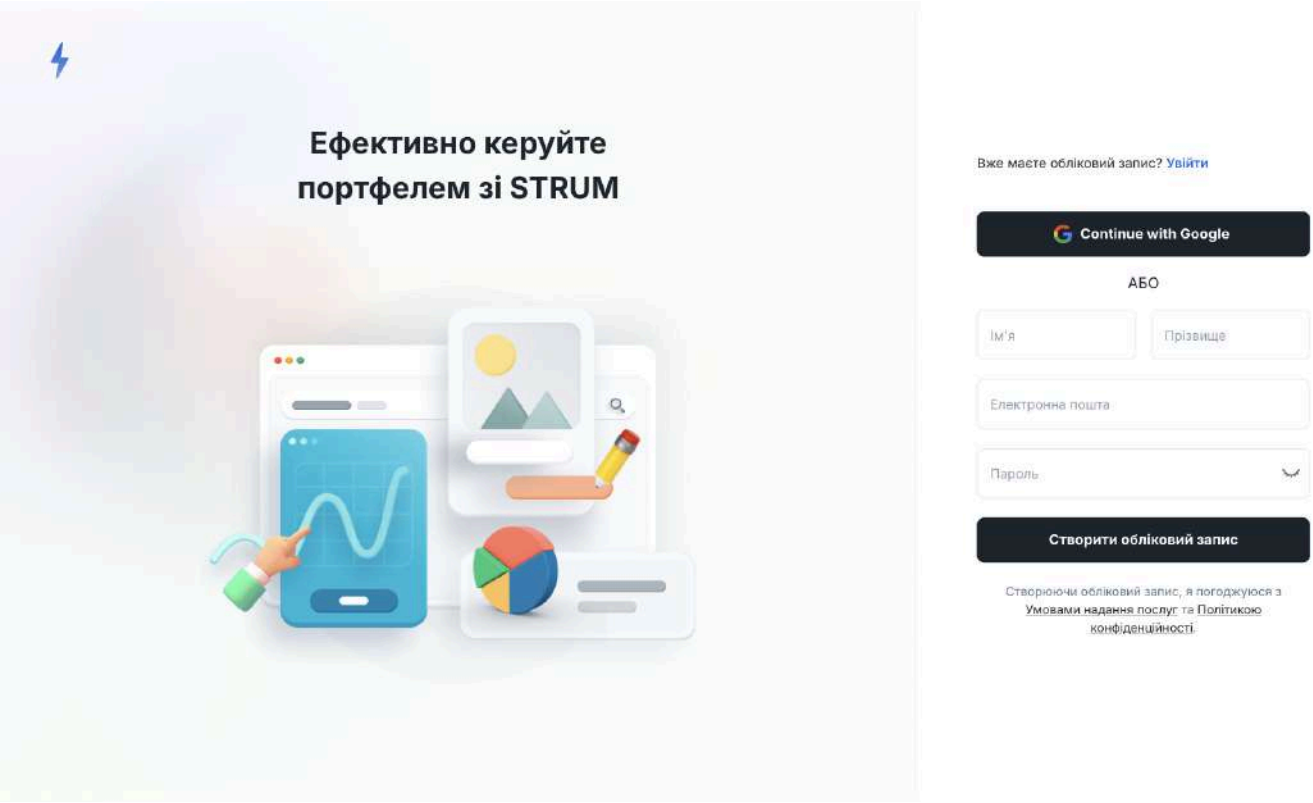


Рисунок 3.8 Сторінка реєстрації користувача.

Головний екран

На рис. 3.9 зображено головний екран програми. Ця сторінка є центром аналітики для інвестора: вона містить графіки дохідності для кожного з портфелів, а також показник загального капіталу. Крім того, зображено розподіл активів по класам, що допомагає користувачеві оцінити структуру інвестицій.

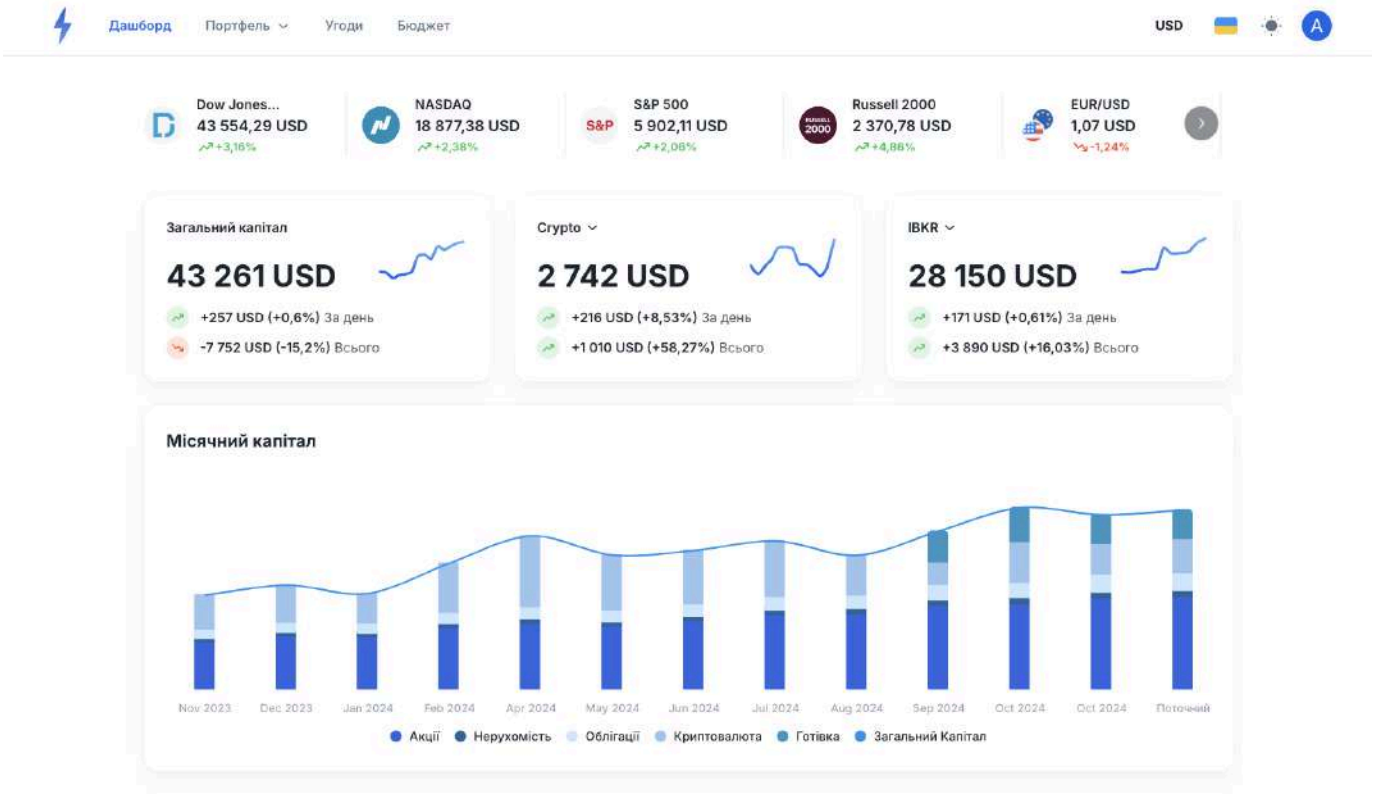


Рисунок 3.9 Головний екран програми (Dashboard) з графіками дохідності та розподілом активів по класам.

Сторінка огляду портфелів

На рис. 3.10 представлено сторінку Summary, де користувач може вибрати один або кілька портфелів для детального аналізу. Після вибору портфелів відображається:

- Історичний графік дохідності з опціями перегляду денних, місячних або річних даних.
- Графік, який показує розподіл активів по категоріям.
- Таблиця зі списком активів, де вказано такі показники, як поточна ціна, позиція, назва, логотип, прибуток або збиток (PnL), алокація, середня ціна покупки та нереалізовані прибутки.

- Користувач також має можливість додати нові транзакції купівлі або продажу активів, змінити ціну кастомного активу, здійснити поповнення або зняття з портфеля, а також перейти на сторінку імпорту транзакцій від брокера.

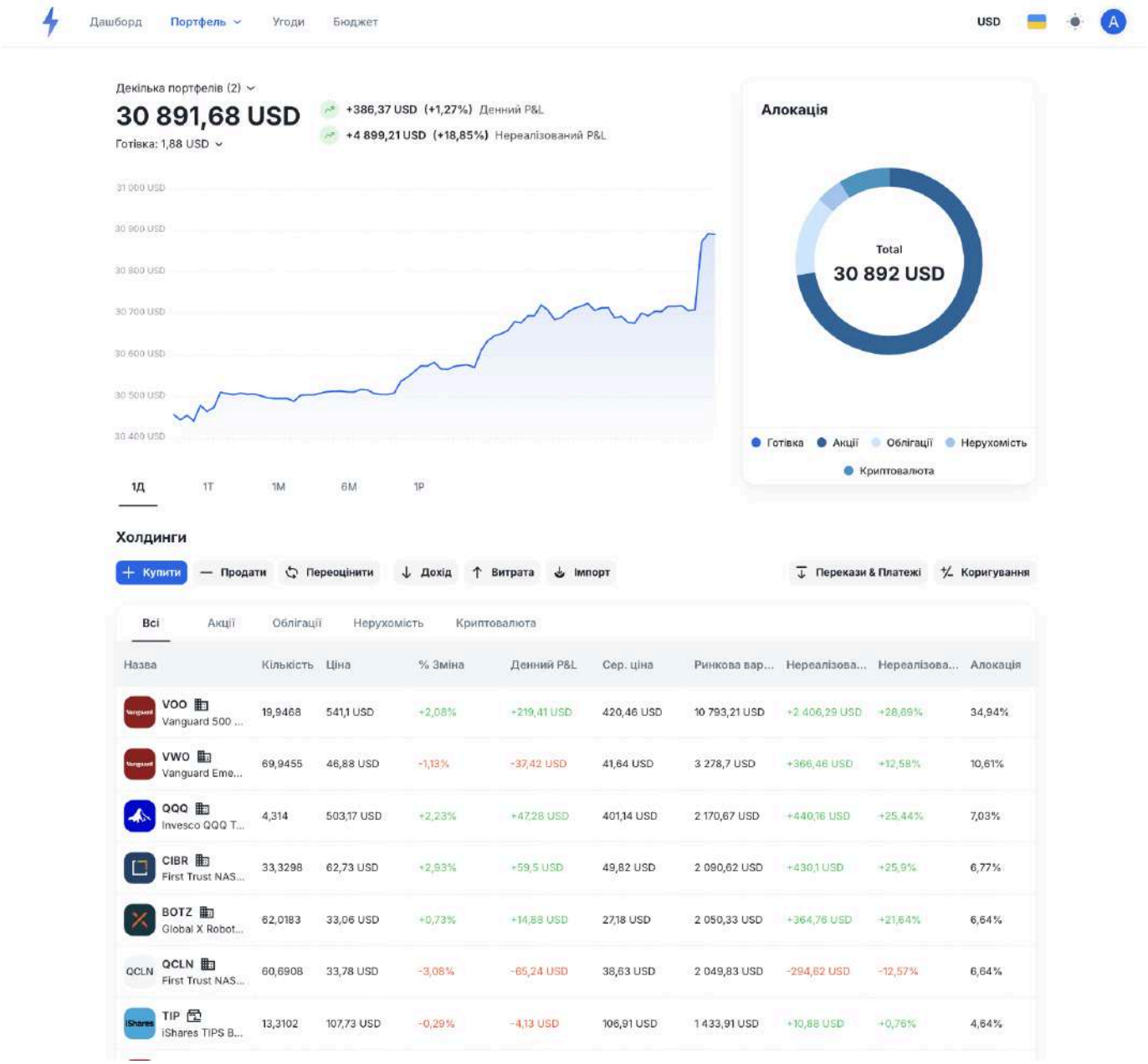


Рисунок 3.10 Сторінка огляду портфелів із графіками та таблицею активів.

Сторінка угод

На рис. 3.11 зображено сторінку з усіма фінансовими угодами користувача. Тут відображені всі транзакції з активами, включаючи купівлю, продаж, поповнення портфелів тощо. Фільтри дозволяють відсортувати транзакції за портфелем, активом і типом операції, що спрощує навігацію.

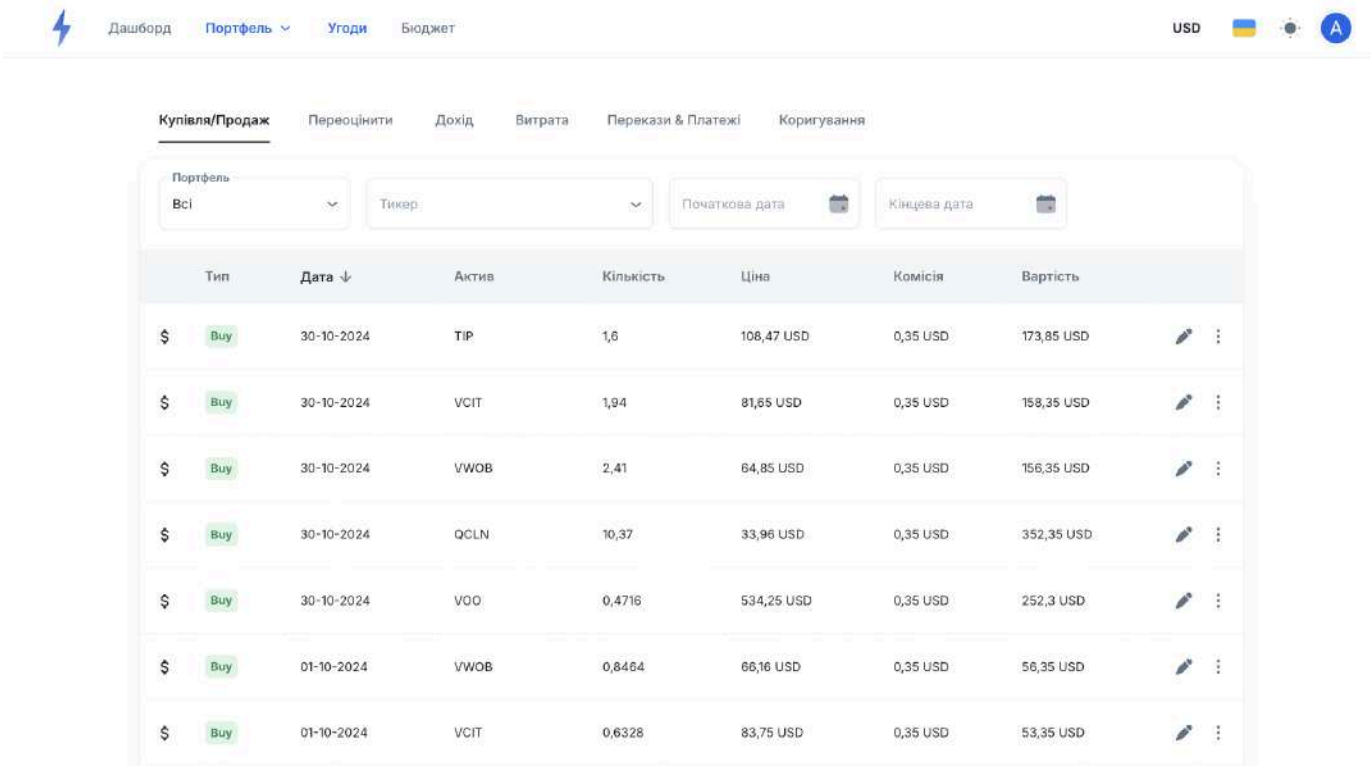


Рисунок 3.11 Сторінка транзакцій з фільтрацією за типом операцій.

Сторінка бюджету

На рис. 3.12 показана сторінка бюджету, що дозволяє користувачеві керувати грошовими операціями. Вона містить наступні розділи:

- Створення готівкових рахунків та додавання операцій поповнення, зняття чи переказу між рахунками.

- Віджет з історичною статистикою доходів, витрат та інвестованих коштів, а також планування бюджету.
- Віджет для планування доходів та витрат за місяцями, що допомагає користувачеві бачити цілі на майбутні періоди.

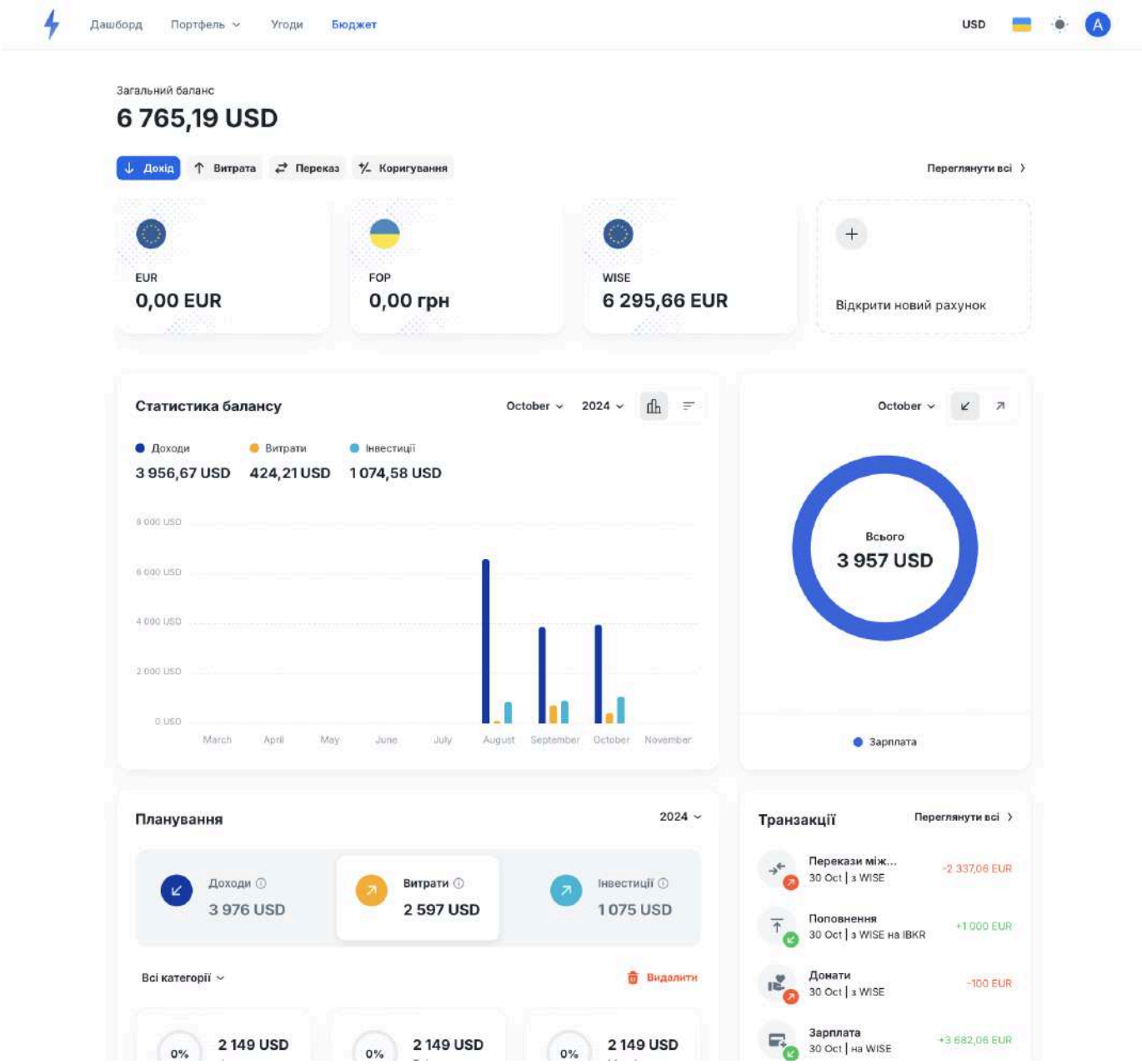


Рис. 3.12 Сторінка управління бюджетом з можливістю планування.

Сторінка додавання користувацьких активів

На рис. 3.13 представлена сторінка додавання кастомних активів. Користувач може ввести дані для таких активів, як нерухомість, бізнес або інші інвестиційні інструменти, які не є біржовими, і додати їх до портфеля.

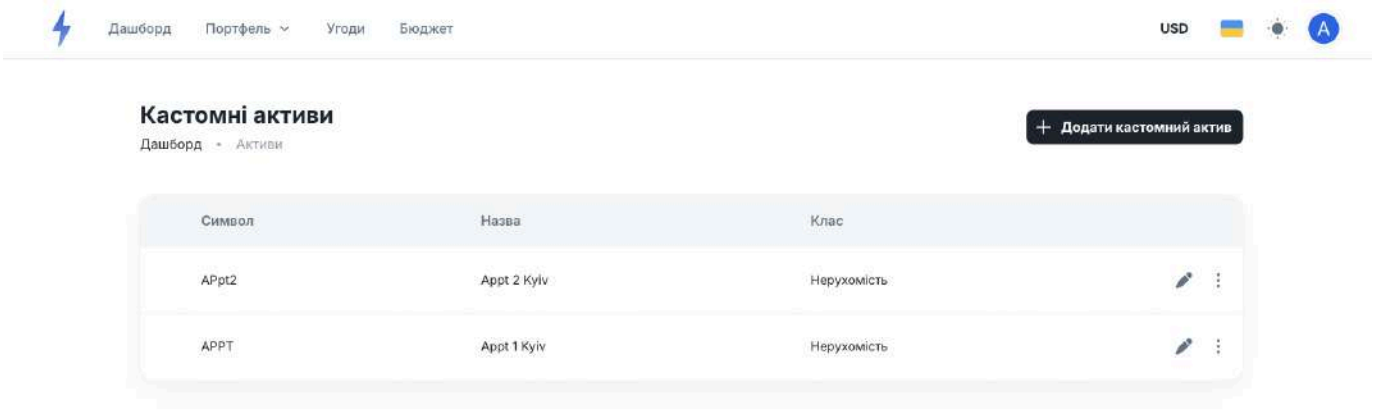


Рис. 3.13 Сторінка додавання користувацьких активів.

Сторінка управління портфелем

На рис. 3.14 показано інтерфейс для управління портфелем, де користувач може створити новий портфель, задати цільові алокації для активів, а також створити портфельний шаблон. Використовуючи цей шаблон, користувач може провести

ребалансування або заплановану докупівлю активів у портфель (рис. 3.15).

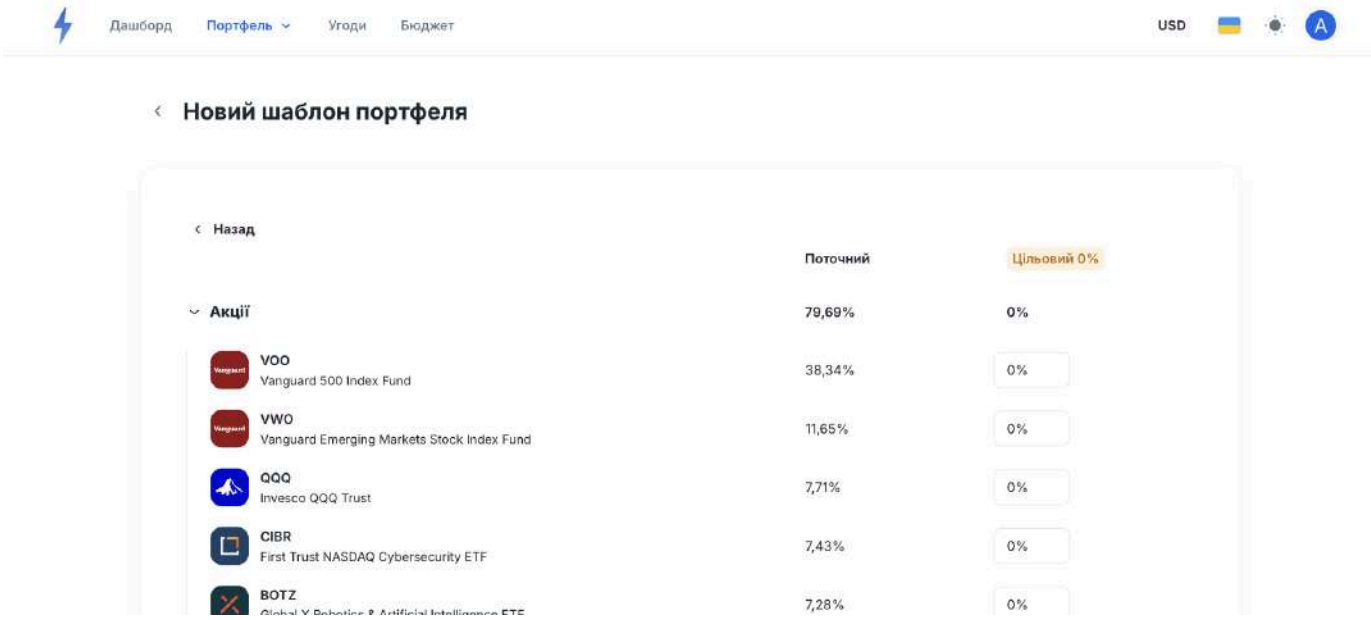


Рис. 3.14 Сторінка управління портфелем з можливістю створення шаблонів.

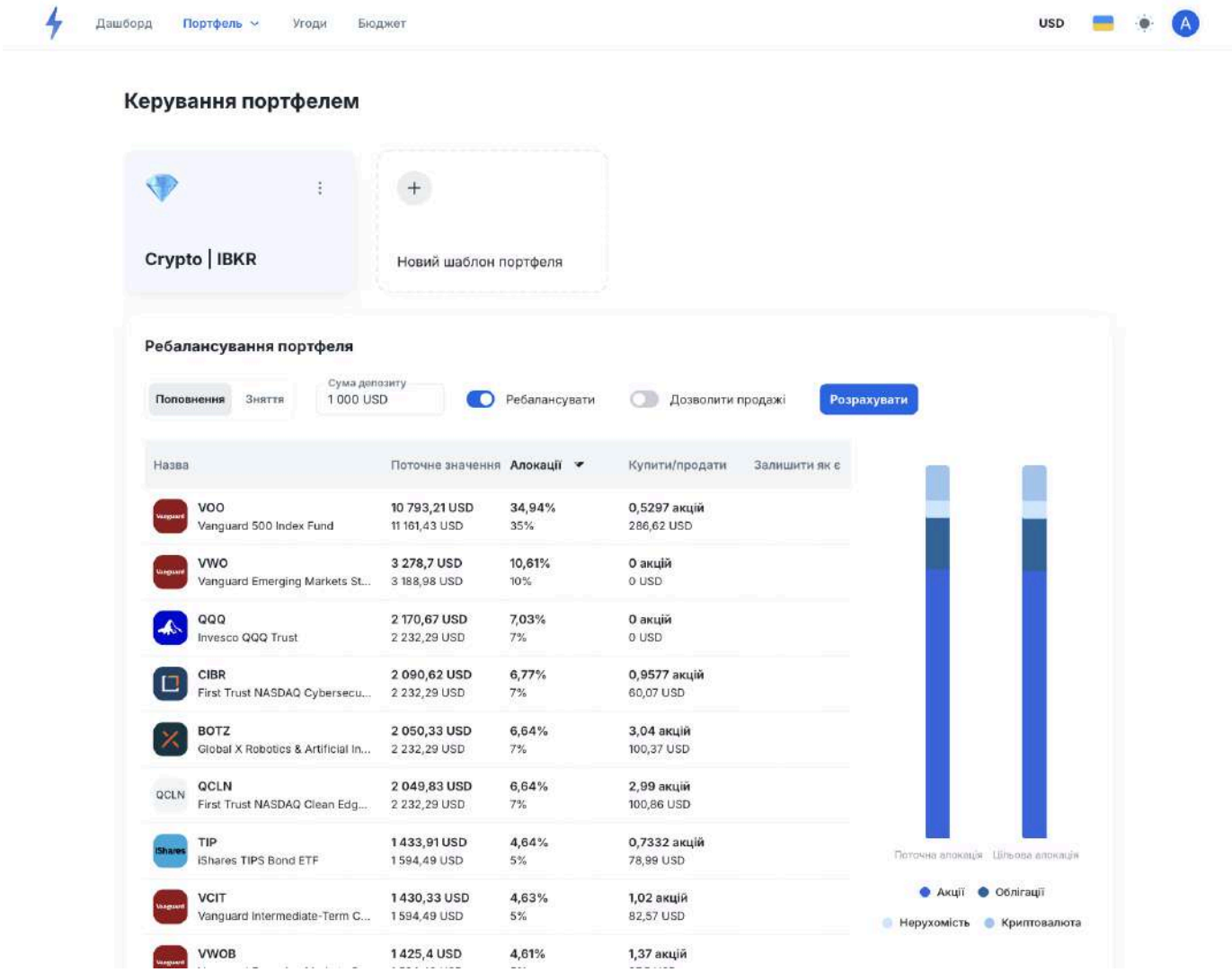


Рис. 3.15 Сторінка управління портфелем з можливістю ребалансування.

Ця функція дозволяє автоматизувати процес управління інвестиціями, підтримуючи цільові співвідношення активів.

ВИСНОВОК ДО РОЗДІЛУ 3

У цьому розділі було розглянуто процес реалізації програмного продукту для відстеження капіталу та прогнозування прибутковості за допомогою штучного

інтелекту. Було описано основні етапи проєктування бази даних, включно з концептуальним, логічним і фізичним проєктуванням, які забезпечили створення надійної та масштабованої бази даних.

Архітектура програмного продукту була розроблена на основі модульної клієнт-серверної моделі, що сприяє ефективності й зручності в управлінні системою. Використання фреймворку FastAPI для серверної частини й Next.js для клієнтської частини забезпечило високу продуктивність і гнучкість застосунку.

Інтеграція модуля штучного інтелекту на базі моделі LSTM підвищила можливості прогнозування прибутковості портфелів, забезпечуючи точні результати аналізу історичних фінансових даних. Завдяки адаптації сучасних алгоритмів і глибокому навчання, модель демонструє високу ефективність і точність.

Процес розгортання продукту реалізовано із використанням контейнеризації через Docker та управління інфраструктурою за допомогою Coolify, що спрощує розгортання і забезпечує стабільність та безпеку системи.

Таким чином, створений програмний продукт відповідає поставленим вимогам і забезпечує користувачам можливість ефективного управління капіталом та прогнозування фінансових показників. Він поєднує сучасні технології та інструменти, що дозволяє йому легко масштабуватися та вдосконалюватися в майбутньому.

ВИСНОВОК

Проведене дослідження дозволило успішно досягти мети розробки програмного продукту для відстеження капіталу та прогнозування його прибутковості на основі історичних даних із використанням технологій штучного інтелекту. Застосування сучасних методів управління проєктами, аналіз ринку, вибір оптимальної архітектури та технологій забезпечили ефективність розробки та її адаптацію до сучасних потреб інвесторів.

У першому розділі було здійснено детальний огляд методологій управління проєктами, що дозволило обрати найбільш відповідні підходи для розробки даного програмного продукту. Було визначено, що гібридний підхід, який поєднує традиційні та гнучкі методи управління, є оптимальним для IT-проєктів з високою невизначеністю. Також було досліджено різні архітектурні стилі та вибрано клієнт-серверну архітектуру з елементами модульності, що забезпечує високу продуктивність, масштабованість і зручність управління системою.

Другий розділ був присвячений аналізу ринку та конкурентного середовища програмних продуктів для відстеження капіталу, що дозволило визначити ключові функціональні особливості продукту та сформувати вимоги до нього. На основі SWOT-аналізу були виявлені сильні сторони, можливості для розвитку, а також ризики, які слід було врахувати під час розробки. Ці висновки допомогли визначити стратегічні цілі та плани щодо розробки програмного забезпечення, що підвищило конкурентоспроможність продукту.

У третьому розділі докладно описано процес реалізації програмного продукту, що включав етапи проєктування бази даних, розробку архітектури та інтеграцію модуля штучного інтелекту. Основним досягненням стало використання моделі LSTM для прогнозування прибутковості, що продемонструвала високу ефективність у роботі з фінансовими часовими рядами. Використання таких інструментів, як

Docker для контейнеризації та Coolify для управління інфраструктурою, дозволило забезпечити стабільне та безпечне розгортання системи.

Результати роботи підтвердили, що застосування штучного інтелекту значно покращує процес аналізу фінансових даних, підвищуючи точність прогнозів і зменшуючи ризики інвесторів. Проєкт відповідає сучасним вимогам ринку, надаючи користувачам інноваційний інструмент для управління капіталом та підтримки прийняття рішень. Це забезпечує ефективну адаптацію продукту до змін у ринкових умовах та його можливість подальшого розвитку й вдосконалення.

Загалом, поставлені завдання були виконані, що підтверджує досягнення мети дослідження. Програмний продукт поєднує сучасні технології, аналітичні можливості та гнучкий підхід до управління проєктами, що забезпечує його стійкість та ефективність на ринку фінансових технологій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Project Management Institute. Стандарт з управління проєктами та Настанова до зводу знань з управління проєктами (Настанова PMBOK). – 7-е видання. – Ньютаун-сквер, Пенсильванія: Project Management Institute, Inc., 2021. – 400 с.
2. Agile Manifesto. Agile маніфест. – [Електронний ресурс]. – Режим доступу: https://agilemanifesto.org/iso/uk/manifesto.html.
3. Schwaber, K., Sutherland, J. Scrum Guide / K. Schwaber, J. Sutherland. – [Електронний ресурс]. – Режим доступу: https://scrumguides.org/scrum-guide.html.
4. Бушуєв Д. А., Козир Б. Ю. Гібридні методології управління інфраструктурними проєктами / Д. А. Бушуєв, Б. Ю. Козир // Сучасний стан наукових досліджень та технологій в промисловості. – 2020. – № 1 (11). – С. 35–43. – DOI: https://doi.org/10.30837/2522-9818.2020.11.035.
5. Поплавський О. А. Основи архітектури програмного забезпечення: конспект лекцій. – КНУБА, Київ, 2023. – 45 с.
6. Understanding Software Architecture: A Complete Guide [Електронний ресурс] // Medium. URL: https://sarrahpitaliya.medium.com/understanding-software-architecture-a-complete-guide-cb8f05900603 (дата звернення: 01.06.2024).
7. 12 Common Software Architecture Styles Essential for Architects [Електронний ресурс] // Medium. URL: https://medium.com/@hubian/12-common-software-architecture-styles-essential-for-architects-c22d19471980 (дата звернення: 01.06.2024).
8. Predicting Stock Prices Using Machine Learning [Електронний ресурс] // Neptune.ai. URL: https://neptune.ai/blog/predicting-stock-prices-using-machine-learning (дата звернення: 01.06.2024).

9. Selecting a Model for Time Series Prediction [Електронний ресурс] // Neptune.ai. URL: https://neptune.ai/blog/select-model-for-time-series-prediction-task (дата звернення: 01.06.2024).
10. Multivariate Time Series Forecasting: Predicting Future Stock Returns Using Deep Learning [Електронний ресурс] // Medium. URL: https://theaiquant.medium.com/multivariate-time-series-forecasting-predicting-future-stock-returns-using-deep-learning-b43ed320340d (дата звернення: 01.06.2024).
11. Ghosh, Pushpendu, Neufeld, Ariel, Sahoo, Jajati K. Forecasting directional movements of stock prices for intraday trading using LSTM and random forests // arXiv. URL: https://arxiv.org/pdf/2004.10178v2.pdf (дата звернення: 01.06.2024).
12. Liu, Chongda, Wang, Jihua, Xiao, Di, Liang, Qi. Forecasting S&P 500 Stock Index Using Statistical Learning Models // Semantic Scholar. URL: https://pdfs.semanticscholar.org/8a20/9a264c16b7826bac3a234d1bc839c82396d3.pdf (дата звернення: 01.06.2024).
13. Tausworthe, R., 1984. The work breakdown structure in software project management. J. Syst. Softw., 1, pp. 181-186. https://doi.org/10.1016/0164-1212(79)90018-9.
14. Adila, P., Saepudin, D., & Ihsan, A. Prediction of Stocks Return in the LQ45 Index with Long-Short-Term-Memory (LSTM) and Its Application for Portfolio Selection // 2022 10th International Conference on Information and Communication Technology (ICoICT), 194-199. URL: https://doi.org/10.1109/ICoICT55009.2022.9914825 (дата звернення: 01.06.2024).
15. Sen, J., Dutta, A., & Mehtab, S. Stock Portfolio Optimization Using a Deep Learning LSTM Model // 2021 IEEE Mysore Sub Section International Conference (MysuruCon), 263-271. URL: https://doi.org/10.1109/MysuruCon52639.2021.9641662 (дата звернення: 01.06.2024).

16. Ma, Y., Han, R., & Wang, W. Portfolio optimization with return prediction using deep learning and machine learning // Expert Syst. Appl., 165, 113973. URL: https://doi.org/10.1016/j.eswa.2020.113973 (дата звернення: 01.06.2024).
17. Peng, Y. Portfolio Optimization Based on LSTM for 5 Stocks // Advances in Economics, Management and Political Sciences. URL: https://doi.org/10.54254/2754-1169/25/20230504 (дата звернення: 01.06.2024).
18. Gaur, Y. Stock Market Price Prediction Using LSTM // International Journal for Research in Applied Science and Engineering Technology. URL: https://doi.org/10.22214/ijraset.2023.57673 (дата звернення: 01.06.2024).
19. Yan, Y., Nie, X., Wang, M., & Chen, Y. LSTM-based Stock Price Prediction Model using News Sentiments // Advances in Economics and Management Research. URL: https://doi.org/10.56028/aemr.6.1.57.2023 (дата звернення: 01.06.2024).
20. Liu, Z., & Yang, B. Enhancing Portfolio Allocation by LSTM Model // Advances in Economics, Management and Political Sciences. URL: https://doi.org/10.54254/2754-1169/48/20230432 (дата звернення: 01.06.2024).
21. Olah, C. Understanding LSTM Networks // Colah's Blog. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (дата звернення: 01.06.2024).

ДОДАТКИ

Додаток 1

Презентація роботи в MS PowerPoint

Рисунок Д1.1

Рисунок Д1.2

Рисунок Д1.3

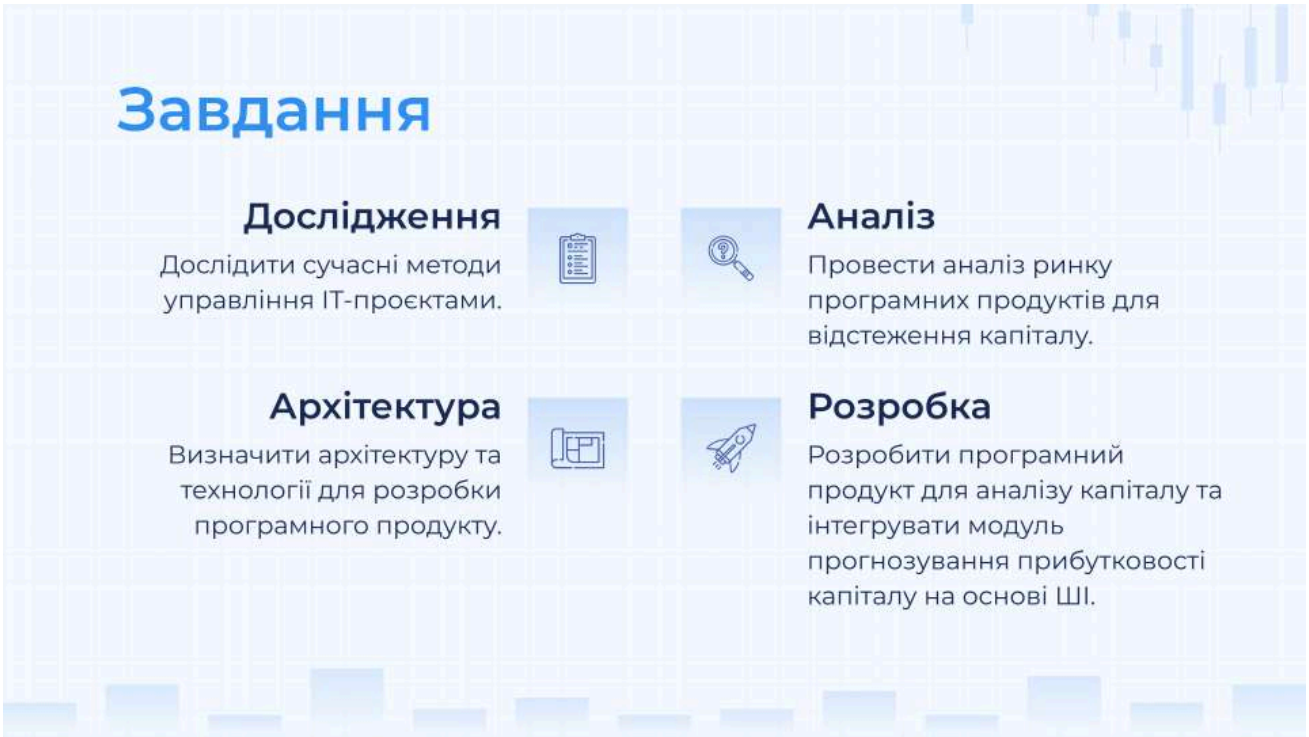


Рисунок Д1.4

Рисунок Д1.5

Рисунок Д1.6

Рисунок Д1.7

Рисунок Д1.8

Рисунок Д1.9

Рисунок Д1.10

Рисунок Д1.11

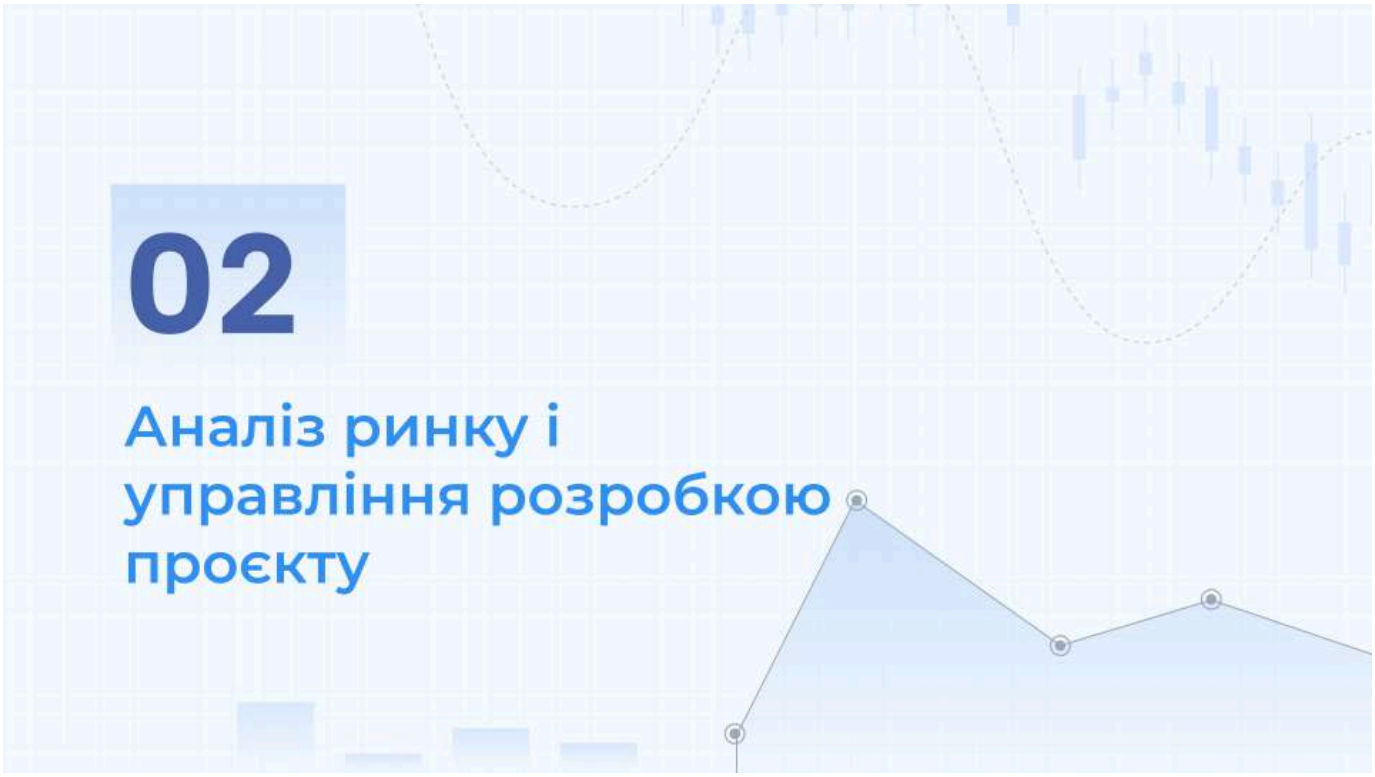


Рисунок Д1.12

Аналіз конкурентів

Snowball Analytics

- Необмежена кількість портфелів і активів.
- Користувацькі активи
- Ребалансування, автоматизація дивідендів.
- Аналітика, прогнозування прибутковості.

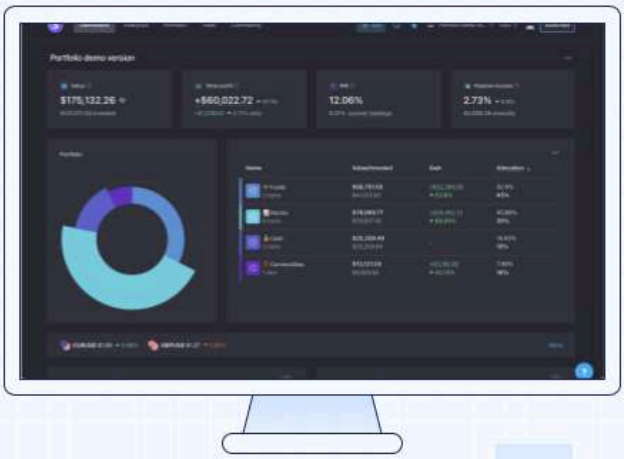


Рисунок Д1.13

Аналіз конкурентів

Firekit

- Просунута аналітика, теплові карти.
- Модуль прогнозування, підтримка акцій та криптовалют.
- Простота використання, прихований режим.
- Операції з активами у кілька кліків.

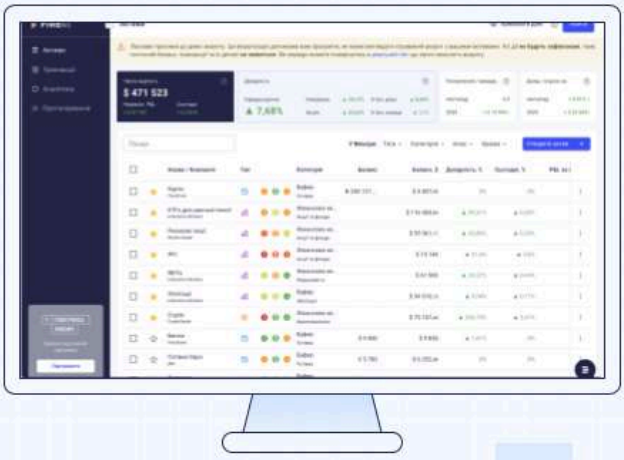


Рисунок Д1.14

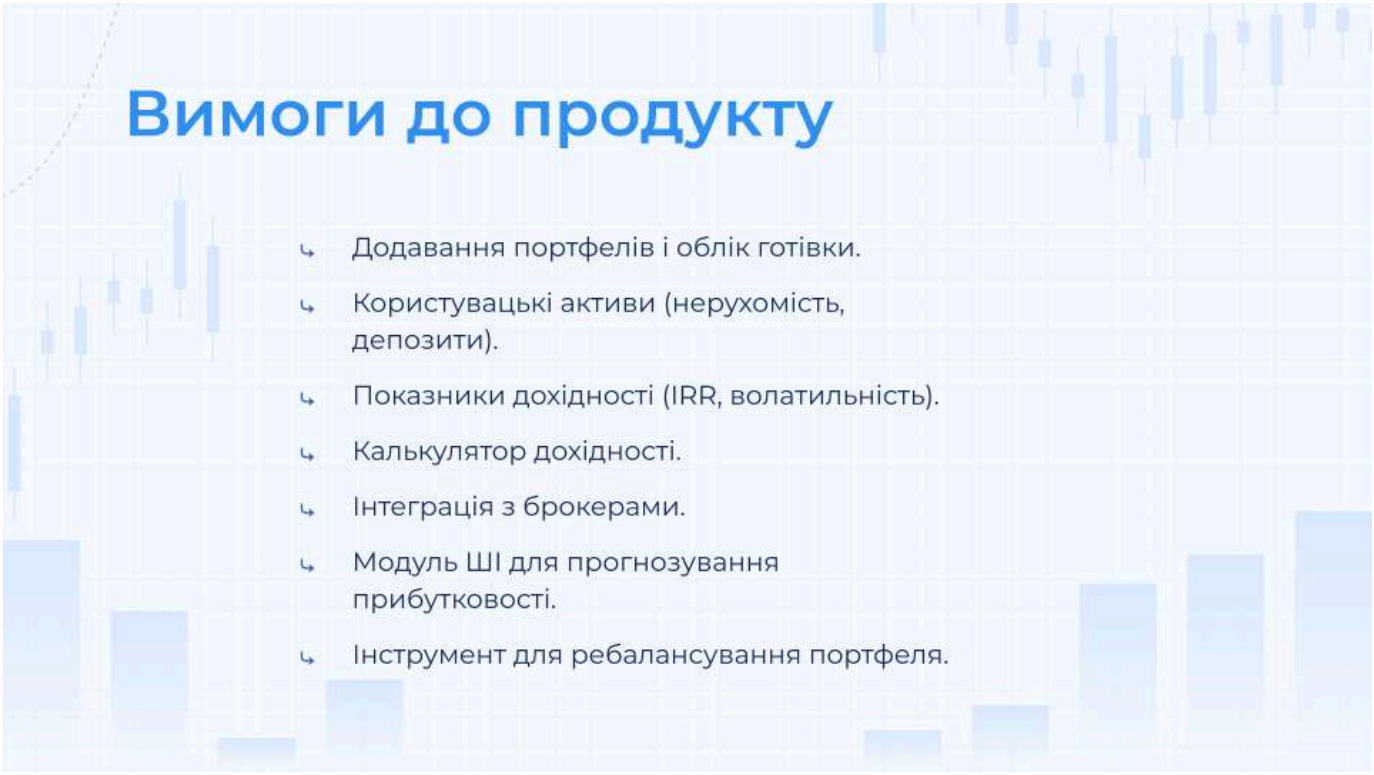


Рисунок Д1.15

Рисунок Д1.16

Рисунок Д1.17

Рисунок Д1.18

Рисунок Д1.19

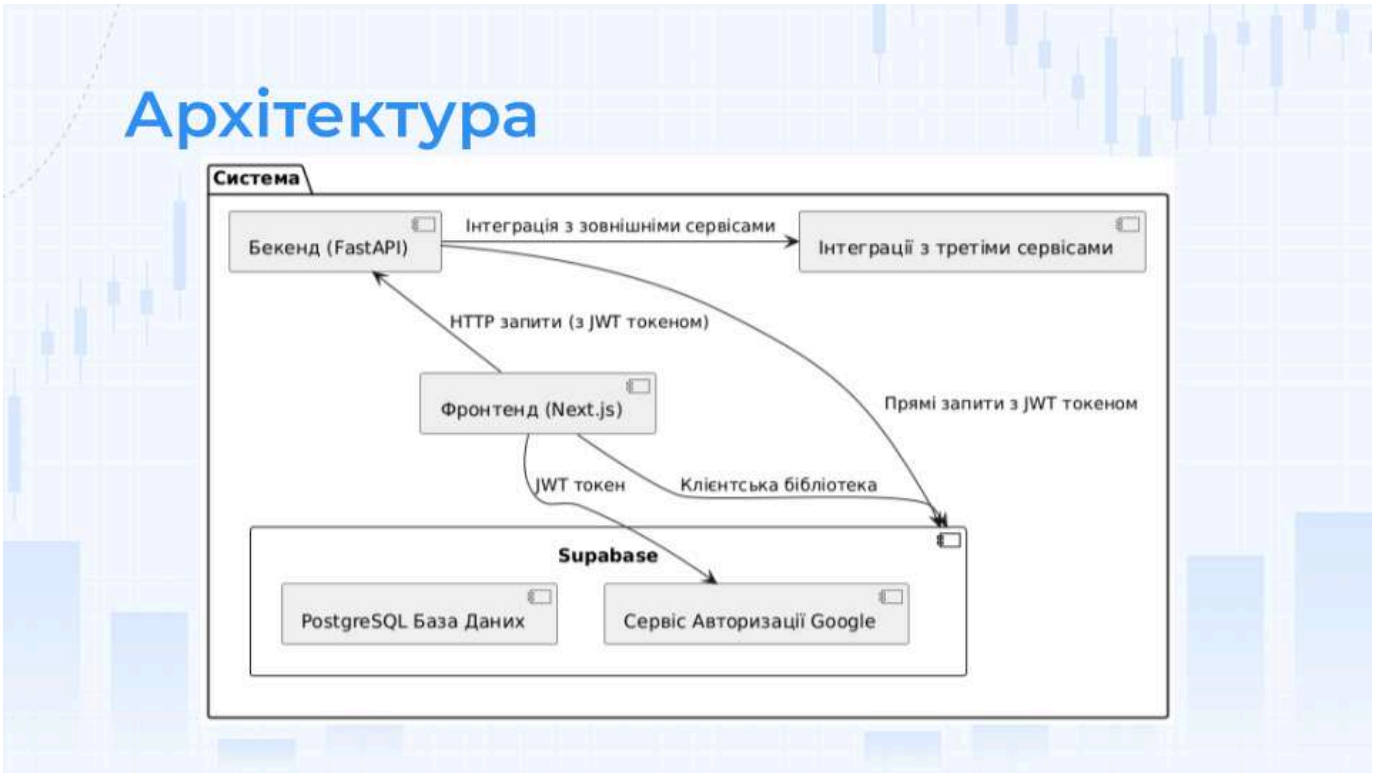


Рисунок Д1.20

База даних

Обрана технологія

- **Реляційна СУБД PostgreSQL**

Переваги

- Відкритий код
- Масштабування
- Наявність зручних API

Основні етапи проєктування БД

- **Системний аналіз**: аналіз вимог і предметної області.
- **Концептуальне проєктування**: створення ER-діаграми для визначення сутностей і зв'язків.
- **Логічне проєктування**: нормалізація даних, визначення ключів.
- **Фізичне проєктування**: типи даних, індексація, оптимізація запитів.

Рисунок Д1.21

База даних

Основні сутності

- **Користувач**
- **Роль**
- **Портфель**
- **Транзакція**
- **Актив**
- **Грошовий рахунок**
- **Грошова транзакція**
- **Валюта**

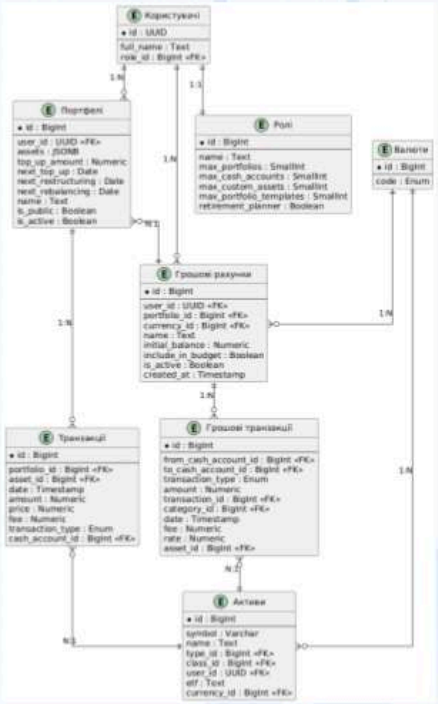


Рисунок Д1.22

Клієнтська та серверна частини системи

Клієнтська частина (Frontend):

- **Next.js** (React.js).
- Серверний рендеринг (SEO-оптимізація).
- Взаємодія через **REST API**.

Серверна частина (Backend):

- **FastAPI** (Python).
- Асинхронне програмування для обробки запитів.
- Інтеграція з базою даних (Supabase).

Управління базою даних:

- PostgreSQL через **Supabase**.
- Автентифікація користувачів.
- Робота з даними в реальному часі.

Рисунок Д1.23

Модуль аналітики з ШІ

Архітектура інтеграції:

- Сервер: FastAPI.
- Інтерфейс: інтуїтивно зрозумілий UI для введення параметрів та перегляду прогнозів.

Функції модуля:

- Прогнозування трендів і прибутковості активів.
- Порівняння прогнозованих та реальних значень.
- Підтримка прийняття інвестиційних рішень.

Переваги:

- Автоматизація створення портфелів.
- Інтерактивний інтерфейс для аналізу.
- Конкурентна перевага завдяки інноваційному підходу.

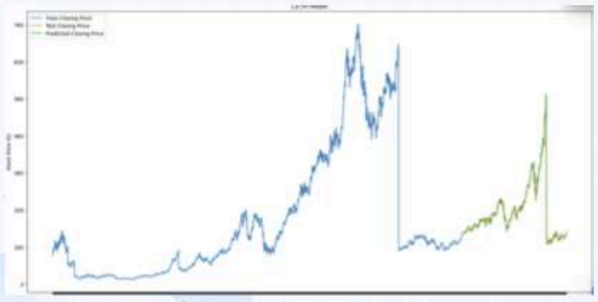


Рисунок Д1.24

Рисунок Д1.25

Рисунок Д1.26

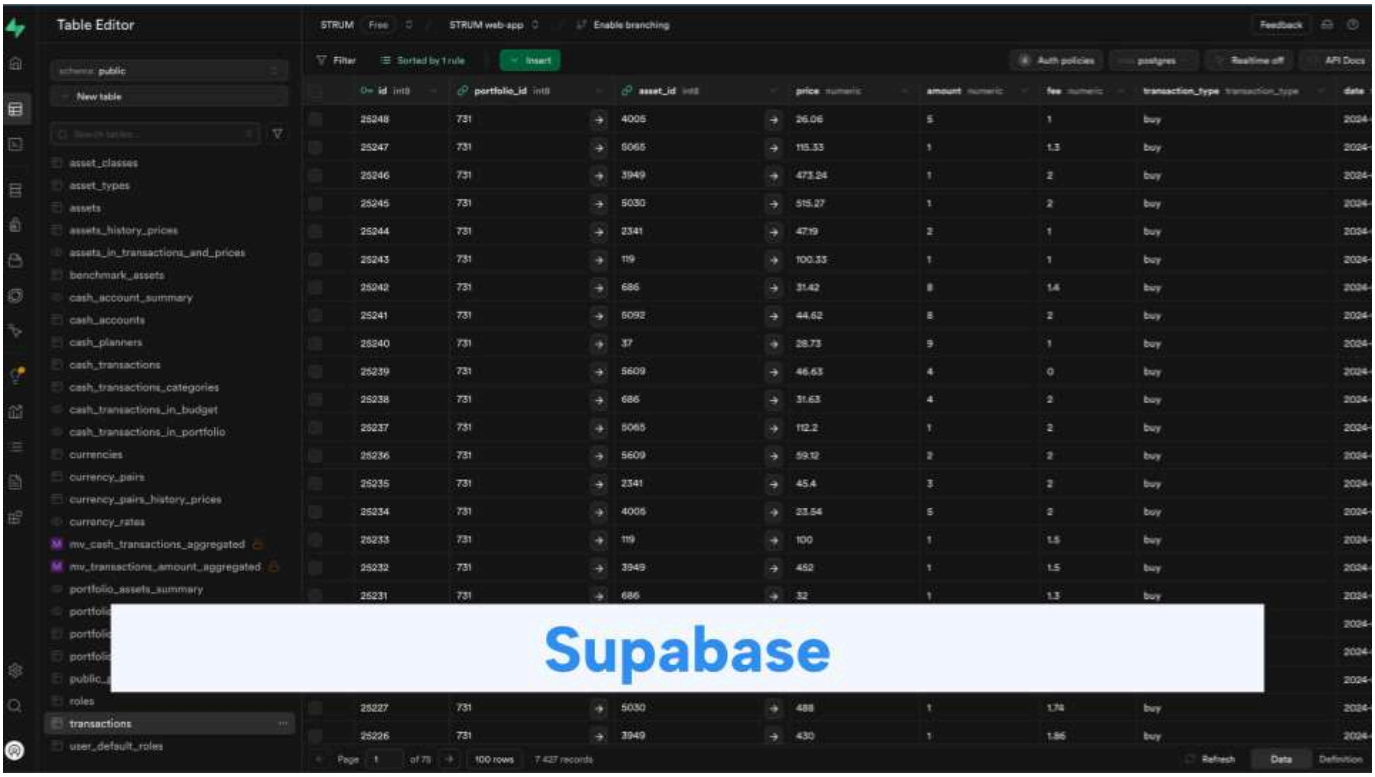


Рисунок Д1.27

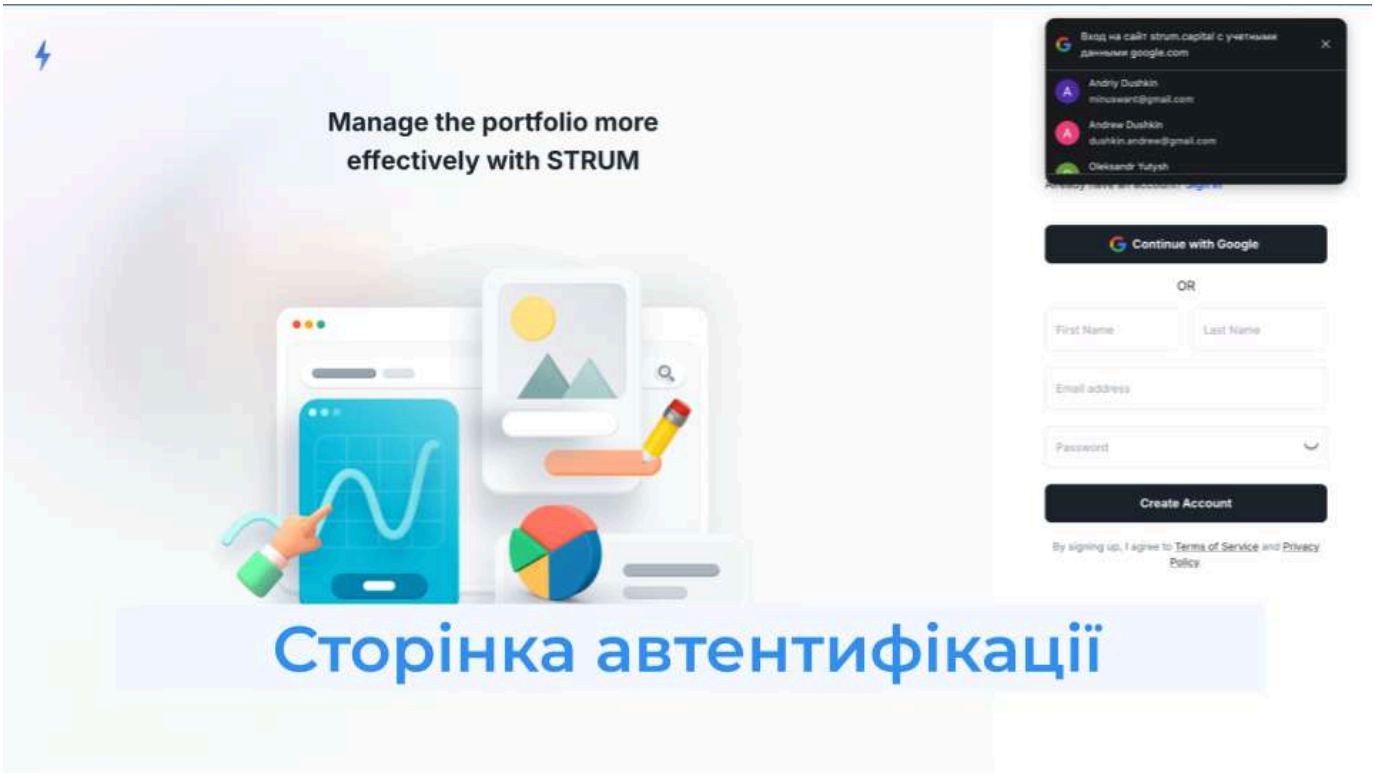


Рисунок Д1.28

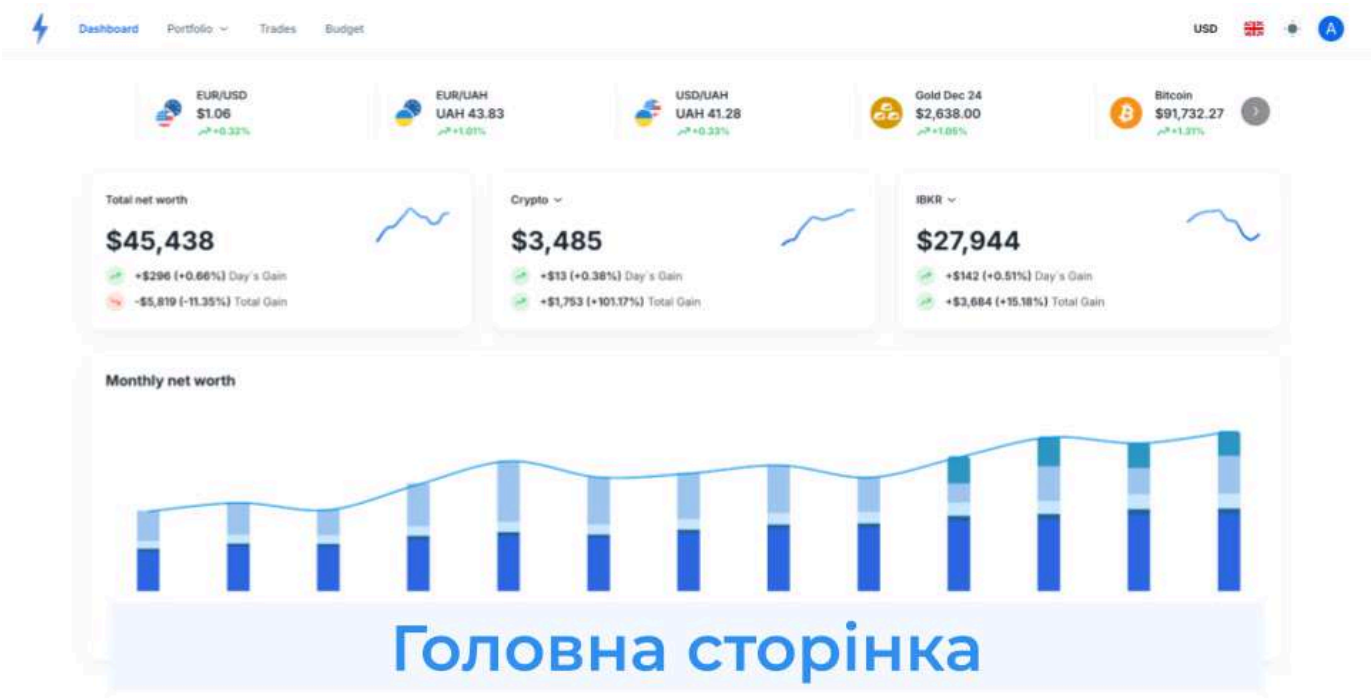


Рисунок Д1.29

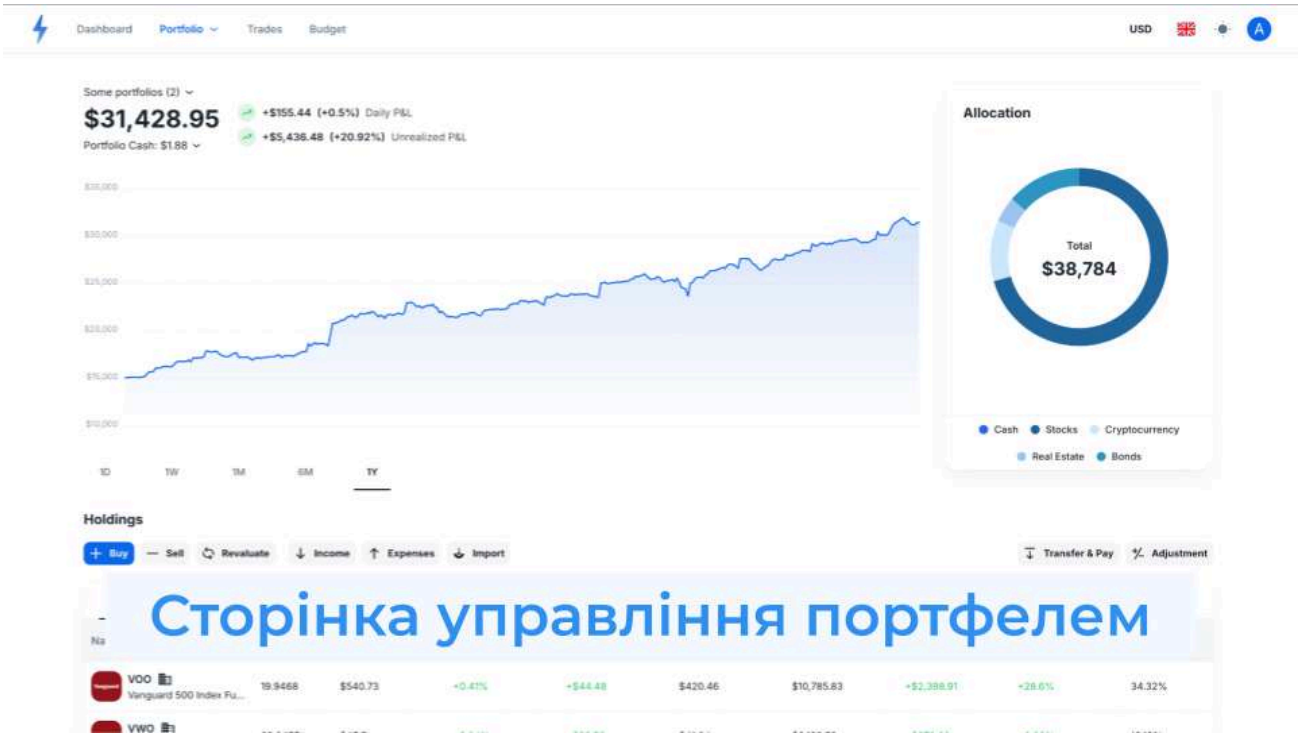


Рисунок Д1.30

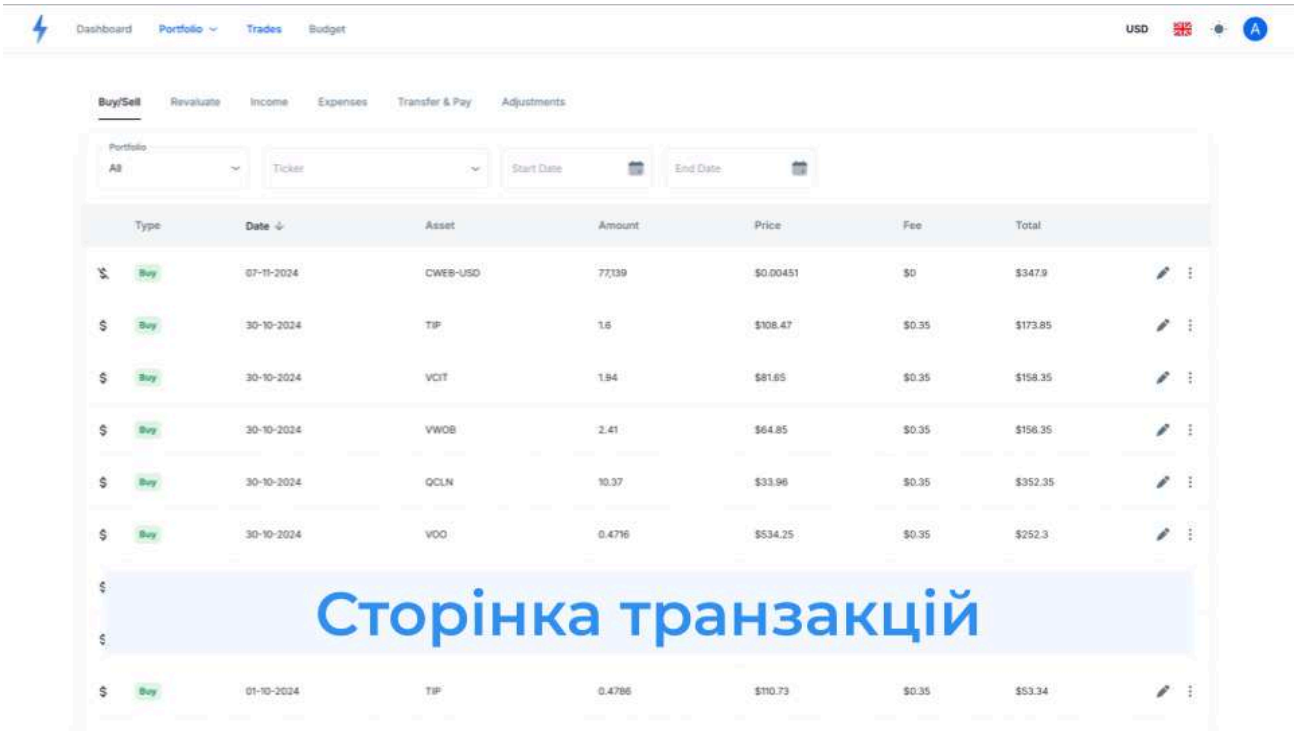


Рисунок Д1.31

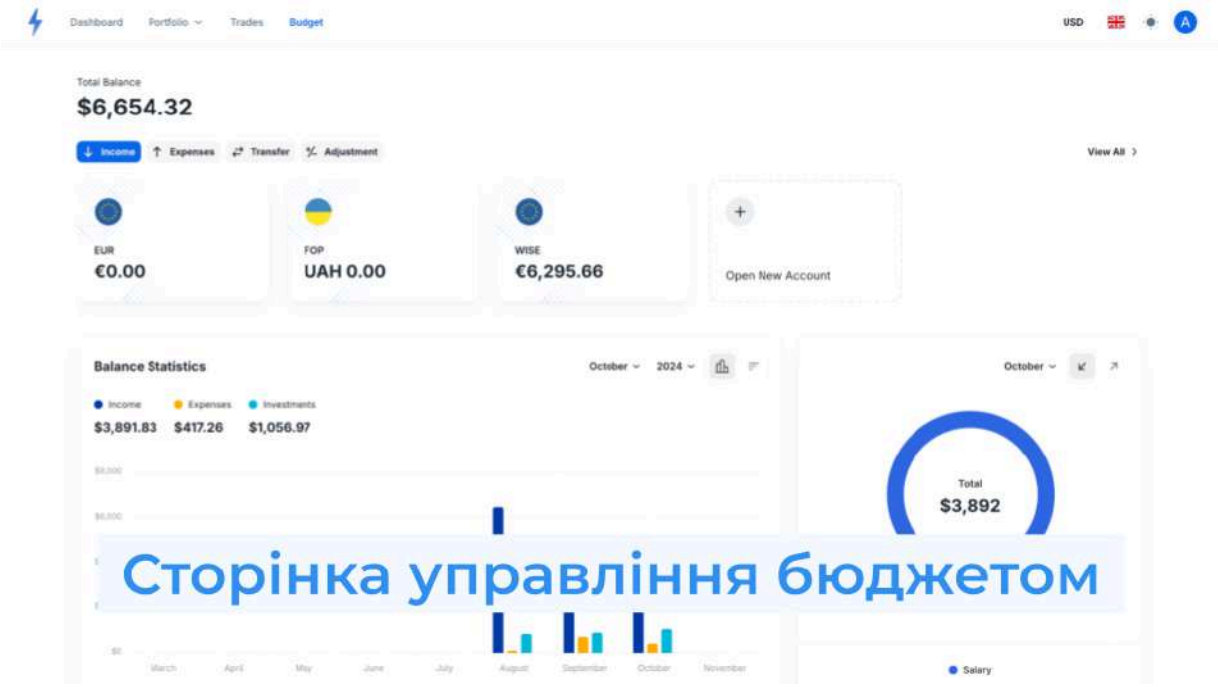


Рисунок Д1.32

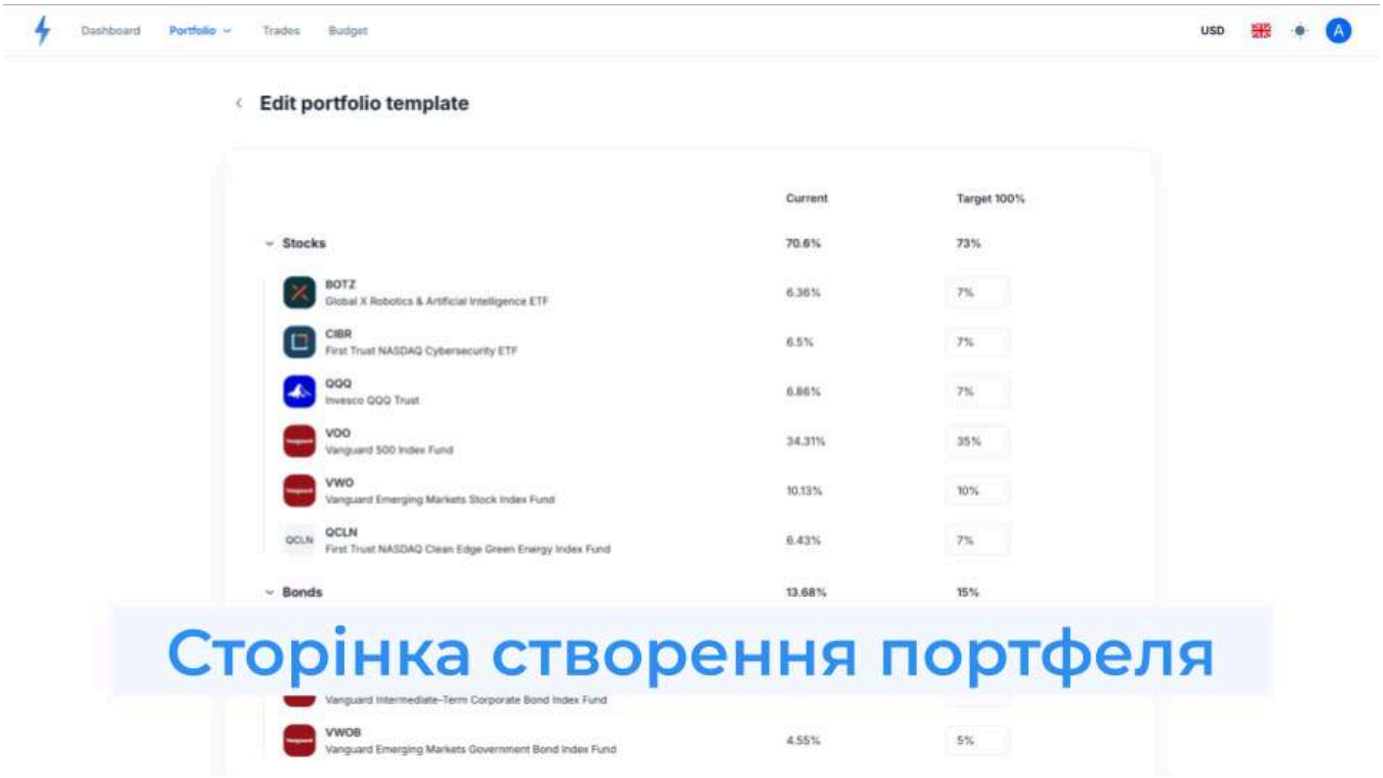


Рисунок Д1.33

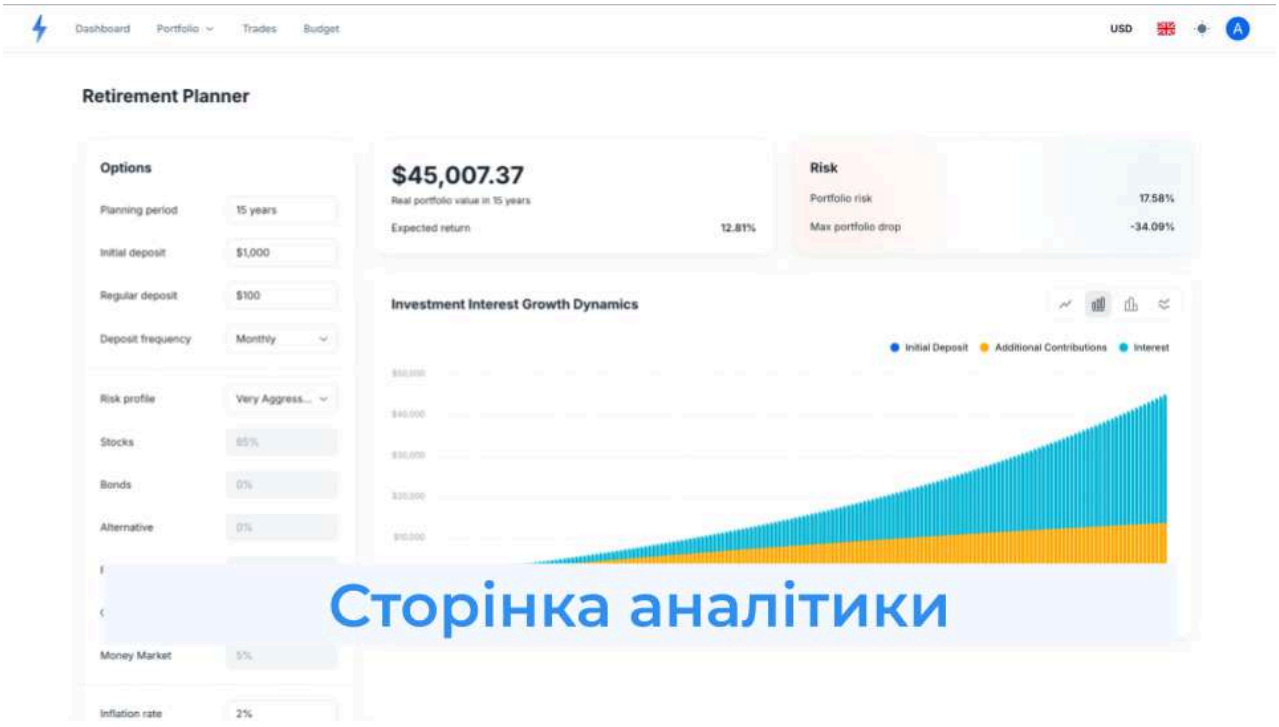


Рисунок Д1.34

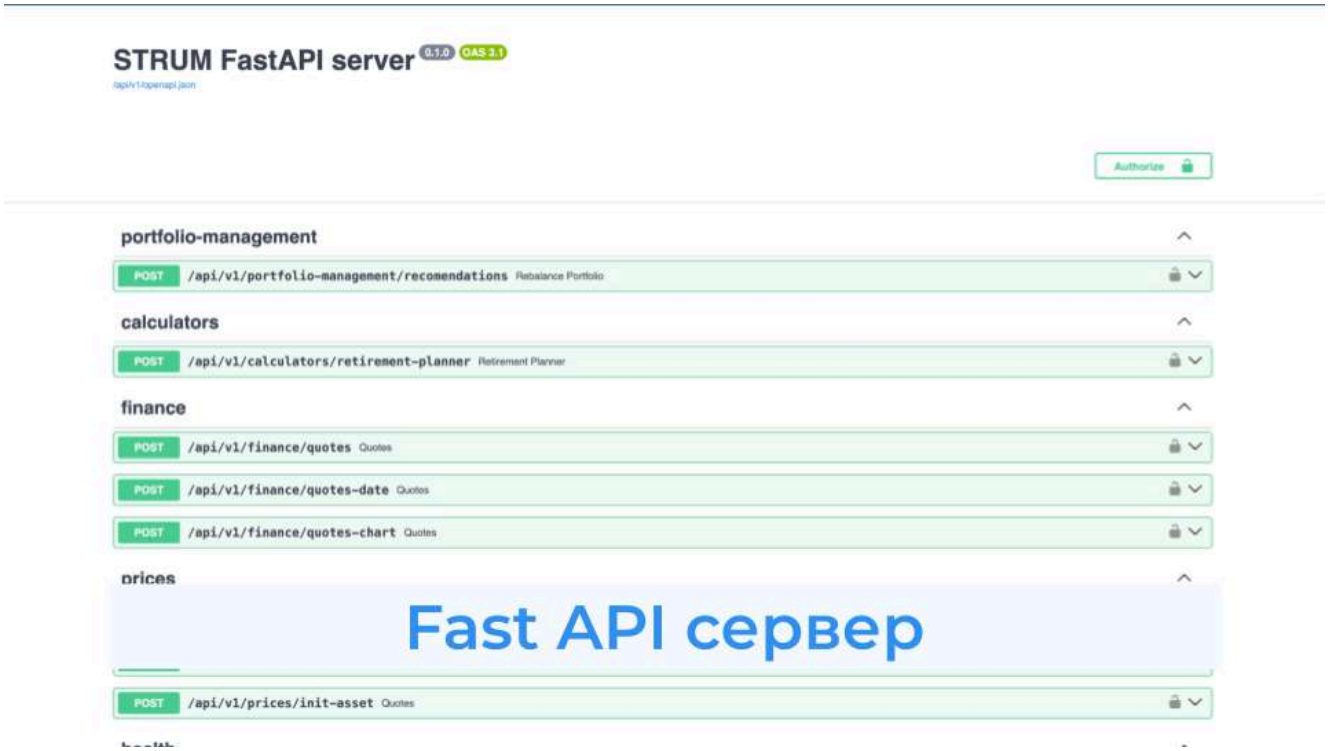


Рисунок Д1.35

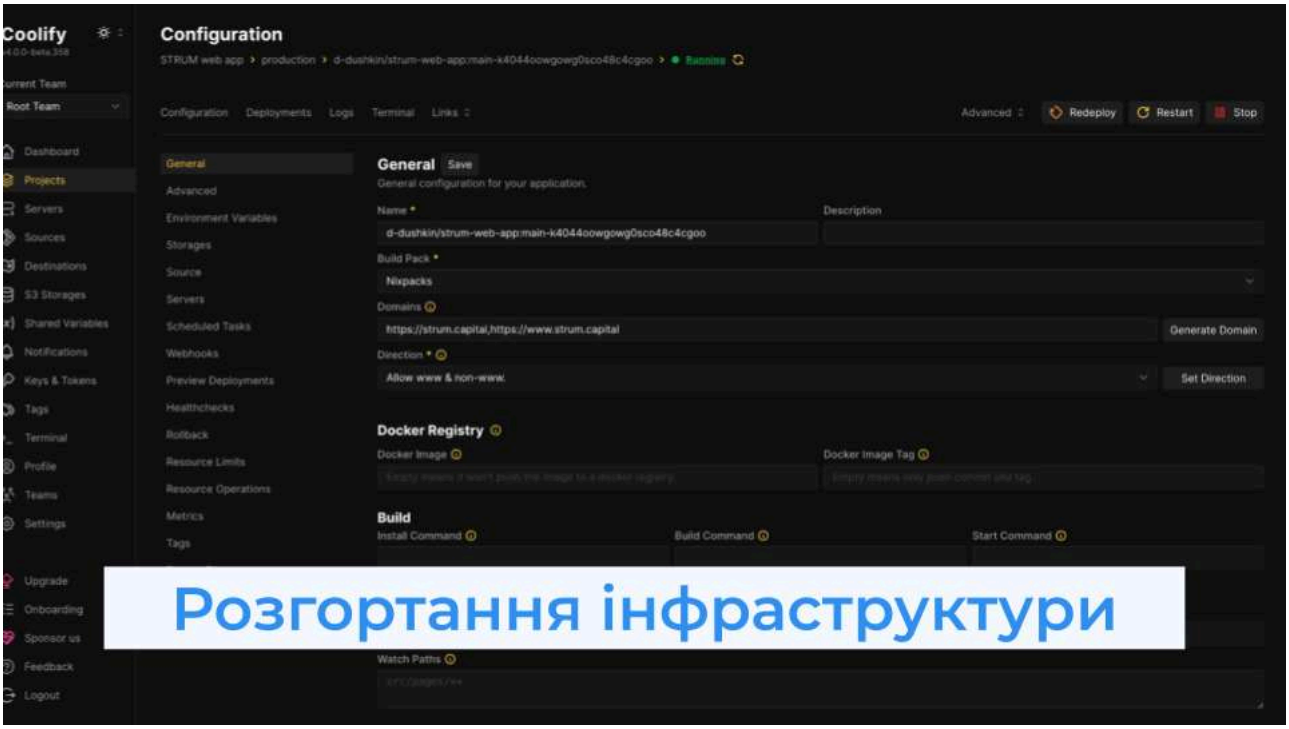


Рисунок Д1.36

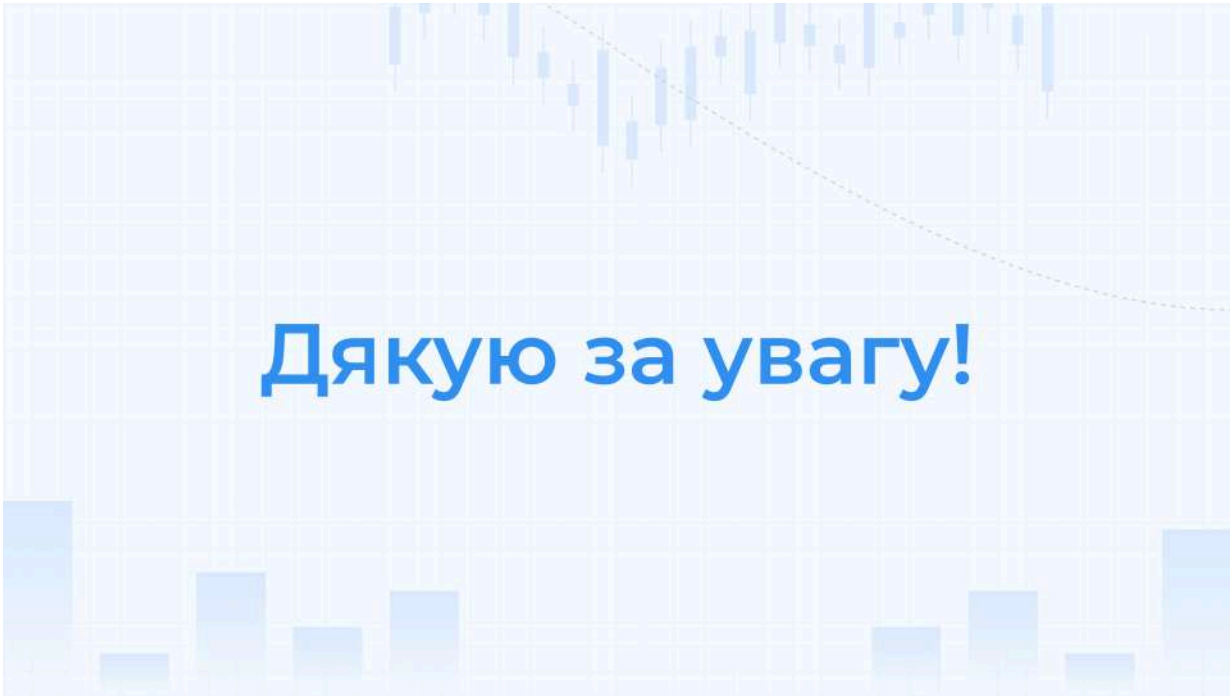


Рисунок Д1.37