

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Вебплатформа для організації діяльності приватного трейдера»

Виконав студент групи ІПЗ-22-4

ТУШАК Віталій Сергійович

Керівник роботи:

ЧИЖМОТРИЯ Олексій Володимирович

Рецензент:

САВІЦЬКИЙ Роман Святославович

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. каф. інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«27» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ТУШАК Віталій Сергійович**

Керівник роботи: **ЧИЖМОТРА Олексій Володимирович**

Тема роботи: **«Вебплатформа для організації діяльності приватного трейдера»,**

затверджена Наказом закладу вищої освіти від **«27» лютого 2026 р., №92/с**

Вихідні дані для роботи: **розробка вебплатформи для організації діяльності приватного трейдера, що забезпечує структурування знань та централізовану організацію робочого процесу трейдера.**

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Чижмотря О.В.	01.03.2026р.	01.03.2026р.
2	Чижмотря О.В.	01.03.2026р.	01.03.2026р.
3	Чижмотря О.В.	01.03.2026р.	01.03.2026р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з програмного продукту у вигляді вебплатформи для організації діяльності приватного трейдера та пояснювальної записки. Робота складається зі вступу, 3-х розділів, висновків, списку використаних джерел із 33 найменувань (3 сторінки), 3 додатки (15 сторінок), 43 таблиці та 34 ілюстрації по тексту. Повний обсяг робіт становить 133 сторінки, з них 117 сторінок основного тексту.

Метою роботи є створення спеціалізованої інформаційної системи, яка дозволяє користувачу впорядковувати власну торгову діяльність, фіксувати результати угод, аналізувати їх та накопичувати знання у межах єдиного робочого середовища.

У роботі визначено ключові завдання, пов'язані з розробкою програмного продукту, проведено аналіз наявних рішень, а також наведено основні сценарії використання системи. Представлено загальну структуру платформи, описано принципи взаємодії між її компонентами та логіку організації даних у вигляді ієрархічної моделі.

У межах роботи було досліджено підходи до організації даних і побудови архітектури застосунку, що дозволило забезпечити ефективну роботу системи та її відповідність типовим задачам користувача.

КЛЮЧОВІ СЛОВА: ВЕБПЛАТФОРМА, ПРИВАТНИЙ ТРЕЙДЕР, ДЕРИВАТИВИ, ІНФОРМАЦІЙНА СИСТЕМА, ТОРГОВИЙ ЖУРНАЛ, БАЗА ДАНИХ, АНАЛІЗ ДАНИХ, РОБОЧИЙ ПРОСТІР, АВТОМАТИЗАЦІЯ.

					ІПЗ.КР.Б – 121 – 26 – ПЗ				
Змн.	Арк.	№ докум.	Підпис	Дата					
Розроб.	Тушак В.С.				«Вебплатформа для організації діяльності приватного трейдера» Пояснювальна записка	Літ.	Арк.	Аркушів	
Керівник	Чижмотря О.В.						3	133	
Рецензент	Савіцький Р.С.					Житомирська політехніка, група ІПЗ-22-4			
Зав. каф.	Вакалюк Т.А.								

ABSTRACT

The bachelor's qualification thesis consists of a software product in the form of a web platform for organizing the activities of a retail trader and an explanatory note. The thesis consists of an introduction, three chapters, conclusions, a list of 33 references (3 pages), and 3 appendices (15 pages). It contains 43 tables and 34 figures throughout the text. The total volume of the thesis is 133 pages, including 117 pages of the main text.

The purpose of the work is to develop a specialized information system that enables the user to structure trading activities, record trade results, analyze them, and accumulate knowledge within a unified workspace.

The work defines the key tasks related to the development of the software product, includes an analysis of existing solutions, and describes the main usage scenarios of the system. An overview of the platform architecture is presented, along with the principles of interaction between its components and the logic of data organization in the form of a hierarchical model.

Within the scope of the work, approaches to data organization and application architecture design were studied, which made it possible to ensure efficient system operation and its alignment with typical user tasks.

KEYWORDS: WEB PLATFORM, PRIVATE TRADER, DERIVATIVES, INFORMATION SYSTEM, TRADING JOURNAL, DATABASE, DATA ANALYSIS, WORKSPACE, AUTOMATION.

		Тушак В.С.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Чижевотря О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		4

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	7
ВСТУП	9
РОЗДІЛ 1. АНАЛІЗ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ...	11
1.1 Характеристика предметної області	11
1.2 Аналіз наявних підходів	14
1.3 Постановка задачі дослідження та розробки	20
Висновки до першого розділу	24
РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБПЛАТФОРМИ ДЛЯ ОРГАНІЗАЦІЇ ДІЯЛЬНОСТІ ПРИВАТНОГО ТРЕЙДЕРА	26
2.1 Визначення вимог до системи	26
2.2 Проєктування моделей даних системи	35
2.3 Вибір архітектурного підходу та обґрунтування інструментальних засобів	66
2.4 Проєктування алгоритмів функціонування системи	74
2.5 Реалізація ключових компонентів системи	81
Висновки до другого розділу	84
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБПЛАТФОРМОЮ ДЛЯ ОРГАНІЗАЦІЇ ДІЯЛЬНОСТІ ПРИВАТНОГО ТРЕЙДЕРА	86
3.1 Розгортання та конфігурація середовища	86
3.2 Порядок роботи з платформою	89
3.3 Забезпечення безпеки та надійності	99
3.4 Тестування системи	102
Висновки до третього розділу	109
ВИСНОВКИ	112
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	115
ДОДАТКИ	118

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		5

Додаток А	119
Додаток Б	127
Додаток В	133

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		6

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

CFD – це фінансовий інструмент, який дозволяє отримувати прибуток або збиток від зміни ціни базового активу без реального володіння ним.

CFD-трейдер – це користувач системи, який торгує CFD та використовує платформу для організації, фіксації та аналізу своєї діяльності.

API – це набір правил і механізмів, за допомогою яких різні частини системи обмінюються даними.

JWT – компактний URL-безпечний спосіб передачі даних між сторонами, який використовується для автентифікації.

ORM – технологія, що дозволяє працювати з базою даних через об'єкти мови програмування, а не писати SQL-запити вручну.

AST – абстрактне синтаксичне дерево, структура даних, що використовується для представлення синтаксичної структури вихідного коду або формул у вигляді дерева вузлів.

CI/CD – практики безперервної інтеграції та безперервної доставки коду, які автоматизують процеси збірки, тестування та розгортання програмного забезпечення.

DTO – об'єкт передачі даних, патерн проектування, що використовується для передачі даних між підсистемами додатку, часто з метою валідації та типізації.

JSONB – бінарний формат зберігання даних JSON у базі даних PostgreSQL, який підтримує індексацію та швидкий пошук по вкладених структурах.

P&L – фінансовий показник прибутку або збитку (Profit and Loss) від закритої угоди або за певний період часу.

REST – архітектурний стиль взаємодії компонентів розподіленої системи в мережі, що базується на використанні стандартних методів протоколу HTTP.

R-multiple – метрика ризику, яка показує, скільки одиниць ризику було зароблено або втрачено в конкретній угоді відносно початкового стоп-лосу.

SQL – структурована мова запитів, стандартизована мова програмування для створення, модифікації та керування даними в реляційних базах даних

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		7

UI/UX – сукупність понять інтерфейсу користувача (User Interface) та досвіду взаємодії з ним (User Experience), що визначають зручність та ефективність роботи з платформою.

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		8

ВСТУП

Актуальність теми полягає в тому, що ринок роздрібної торгівлі деривативами сьогодні росте швидше, ніж будь-коли: відкрити рахунок у брокера й почати торгувати можна за кілька хвилин, без спеціальної освіти чи значного капіталу. Разом із цим зростає і кількість людей, які щодня стикаються з однією й тією самою проблемою – дані про власну торгівлю розкидані по різних застосунках, і зібрати з них цілісну картину важко.

Попри те, що сама проблема не нова, ринок досі не запропонував її системного вирішення. Кожен трейдер вимушений самотійно складати робочий процес із засобів, не призначених для цього, і щоразу заново винаходити власну систему обліку. Готового інструменту, орієнтованого саме на трейдера й здатного поєднати все необхідне в одному місці, на ринку практично немає.

Тому потреба у спеціалізованій платформі, яка закриває цю прогалину, стає дедалі відчутнішою - не як абстрактна теоретична проблема, а як цілком конкретний запит зростаючої аудиторії приватних трейдерів. Саме ця потреба і стала підставою для цієї роботи.

Метою є розробка вебплатформи для трейдера, яка забезпечить централізоване зберігання, структурування та аналіз торгових даних. Встановлена мета обумовлює наступні завдання:

- проведення аналізу наявних підходів до організації торгової діяльності та ведення журналів угод;
- визначення архітектури та узагальненої структури системи;
- обґрунтування та вибір засобів реалізації платформи;
- розробка моделі даних і структури їх взаємодії;
- реалізація основних функціональних можливостей платформи для фіксації, зберігання та аналізу даних.

Об'єктом дослідження є процес організації діяльності приватного трейдера в умовах роботи з різними інформаційними даними.

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				9
Змн.	Арк.	№ докум.	Підпис	Дата		

Предметом дослідження є архітектура та інформаційна модель спеціалізованої вебплатформи для подолання фрагментації даних у діяльності приватного трейдера.

Методи дослідження. В роботі було використано методи об'єктно-орієнтованого програмування, проєктування архітектури інформаційних систем та розробки користувацького інтерфейсу.

Практичне значення отриманих результатів. Розроблений програмний продукт може бути використаний трейдерами для фіксації торгової діяльності, аналізу результатів та систематизації власного досвіду в межах єдиного робочого середовища.

Апробація. За темою кваліфікаційної роботи було опубліковано 2 тези конференцій [1, 2].

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				10
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. АНАЛІЗ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

1.1 Характеристика предметної області

Приватний трейдер, що працює з CFD та ф'ючерсними інструментами, має умови, при яких його результат залежить тільки від його якісних рішень. Відповідно без чітко налагодженої системи записів весь цей щоденний досвід просто зникає, як і можливість стати кращим [3].

Протягом дня трейдер накопичує різну інформацію, яку можна умовно поділити на декілька типів.

Угоди – це ядро торгової активності. Кожна угода фіксує фактичні параметри операції, такі як базовий актив, напрямок, обсяг, ціна відкриття та закриття, часові мітки, фінансовий результат (P&L), комісії і так далі [4]. Фактично це структуровані дані для первинного статистичного аналізу. Але окрім сухих чисел важливо також розуміти що саме супроводжувало таке рішення про відкриття угоди та її подальший менеджмент. Без цього неможливо зрозуміти чому угода мала саме такий результат [5].

Аналіз дня – це вже саме той аспект, що надає конкретику здійсненим угодам і дозволяє впливати на їх результат. Сюди записується що було на графіку перед відкриттям позиції, як графік рухався далі і чим все завершилося. Також трейдер часто записує тут свій хід думок та емоції, що дають додаткову інформації для формування закономірностей та використання їх на свою користь [5].

Нотатки – ці дані формують якісну складову торгового досвіду. Якщо потрібно зібрати дані з вільною структурою, то використовують саме нотатки. Цей масив даних здебільшого складається з технічних аспектів торгівлі, і менше емоційних. Причому тематика цих нотаток може бути різною, але це невід'ємна та досить важлива частина даних трейдера [6].

Помилки – це також досить важливий розділ. Вони відіграють особливу роль, оскільки помилка це не просто про збиткову угоду, це про відхилення або

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				11
Змн.	Арк.	№ докум.	Підпис	Дата		

від заздалегідь прописаних правил в торговій стратегії, або від плану на день, ну або ж емоційне відхилення. Мінімальна структура цього типу даних, а саме тип помилки, тригер та складність дозволяє виявити повторювані поведінкові патерни, які мають бути цілеспрямовано скоректовані [5].

Статистика – також один з ключових елементів. Трейдер повинен мати інструменти об'єктивної самооцінки для того, щоб зрозуміти в якому напрямку відбувається рух. До таких метрик відносять % прибуткових угод, математичне очікування, коефіцієнт прибутковості, середнє співвідношення ризику до прибутку. Ці показники формують на основі накопичених даних і дають змогу об'єктивно оцінити ефективність торгової стратегії, і самого трейдера у виконанні [5].

Як вже згадувалося вище, сучасний ринок не пропонує рішення, який би покривав всі аспекти в одному місці, а трейдер змушений використовувати набір розрізнених інструментів, закриваючи лише частину потреб.

В результаті цього формується стан, який можна визначити як інформаційна фрагментація [6], тобто критично важливі дані розподілені між багатьма непов'язаними середовищами, між якими відсутня взаємодія. Відповідно, така організація робочого процесу, окрім збільшення операційного навантаження, також унеможлиблює комплексний аналіз.

Сама ж інформаційна фрагментація має безпосередній вплив на ефективність торгової діяльності. Перш за все варто зазначити, що трейдинг вимагає повного включення в роботу, максимальну концентрацію під час прийняття рішень. А сам факт використання декількох додатків створюють ще більше когнітивного навантаження. Відповідно, має негативний вплив в момент, коли потрібно швидко прийняти рішення, через що трейдер може отримати збиток, або ж не той результат який він хотів. Можна зробити висновок, що втрата концентрації стає прямим наслідком розпорошення даних між непов'язаними інструментами [3].

Окрім миттєвих наслідків, фрагментація суттєво ускладнює ретроспективний аналіз торгової діяльності. Щоб виявити закономірності у

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				12
Змн.	Арк.	№ докум.	Підпис	Дата		

торгівлі потрібно мати можливість переглядати історичні дані в різних зрізах: за активами, за часовими проміжками, за типами входів, виходів, доставки та все інше. А коли дані розподілені між непов'язаними інструментами, такий аналіз вимагає ручного збирання інформації з кількох джерел, що робить його надзвичайно трудомістким і, як наслідок, він рідко буде проводитися на практиці, оскільки мозок не любить зайве працювати. Трейдер фактично позбавляється можливості регулярно звіряти свої дії з накопиченим досвідом, оскільки витрати часу на підготовку даних перевищують потенційну користь від аналізу [6].

Ще одним критичним наслідком є неможливість систематичного виявлення закономірностей, які є основою когнітивних упереджень. Такі проблеми як надмірна впевненість, уникнення збитків, або ж імпульсивне реагування на невдачі та спроба їх так би мовити “відіграти” проявляються не в окремих угодах, а саме в систематичних відхиленнях від написаних правил [7; 12]. Їхнє виявлення вимагає порівняння запланованих параметрів угод із фактичними результатами, а також розуміння кореляції цих відхилень з суб'єктивними факторами типу емоційного стану, втоми або проблем поза трейдингом. І маючи фрагментованість встановлення зв'язків між такими аспектами даних стає практично нездійсненним [6].

Організація інформаційного простору трейдера безпосередньо впливає на якість прийняття рішень через зменшення когнітивного навантаження та формування ефективного зворотного зв'язку. Коли всі необхідні дані доступні в єдиному інтерфейсі, трейдер витрачає значно менше ментальних ресурсів на пошук інформації та більше на її аналіз, що є критичним в умовах невизначеності фінансових ринків. Разом з тим, структуроване накопичення даних дозволяє автоматизувати розрахунок статистичних показників та візуалізувати динаміку ключових метрик, створюючи механізм зворотного зв'язку, який допомагає трейдеру швидко коригувати поведінку на основі об'єктивних даних, а не суб'єктивних відчуттів [5].

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				13
Змн.	Арк.	№ докум.	Підпис	Дата		

Коли всі потрібні дані зібрані в одному місці і доступ до них швидкий, у трейдера вивільняється той самий ресурс уваги, який важливий під час торгівлі. Замість того щоб згадувати, в якому файлі лежать дані, наприклад, вчорашнього дня, оскільки ситуація на ринку схожа на попередню, він може зосередитися на графіку й ухвалити рішення швидше. Крім того, коли система сама рахує ключові метрики й одразу показує, де план розійшовся з фактом, з'являється швидкий зворотний зв'язок.

Другий важливий момент це боротьба із самообманом, на який усі так чи інакше схильні. Прибуткові угоди пам'ятаються яскраво, а от збиткові хочеться швидше забути або списати на вину ринку. Автоматизована статистика, побудована на повному масиві даних, працює як неупереджене дзеркало: вона показує реальну картину, а не ту, яку нам хочеться бачити. А маючи під рукою завжди чіткі правила власної торгової системи й перелік найдорожчих минулих помилок, набагато легше втриматися від імпульсивного вчинку [5].

1.2 Аналіз наявних підходів

На поточний момент у процесі організації діяльності приватного трейдера використовуються різні підходи до зберігання та обробки інформації, що відрізняються за рівнем спеціалізації, гнучкістю та функціональними можливостями. Вибір інструменту зазвичай визначається персональними потребами користувача, але як зазначалося, жоден із підходів не забезпечує повного охоплення всіх аспектів торгової діяльності.

Наявні рішення можна поділити на три категорії: табличні інструменти, які досить прості у використанні, але потребують значної адаптації під потреби трейдингу; універсальні системи управління даними – досить гнучкі за структурою, проте також не орієнтовані на специфіку торгової діяльності; спеціалізовані торгові журнали розроблені безпосередньо для фіксації та аналізу угод, однак мають обмежені можливості для налаштування під індивідуальні потреби користувача.

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				14
Змн.	Арк.	№ докум.	Підпис	Дата		

Можна зробити висновок, що користувач змушений або витратити значний час на адаптацію універсальних систем під власні потреби, або використовувати спеціалізовані інструменти, що не покривають усі аспекти діяльності. У результаті інформація ділиться між різними середовищами, що ускладнює її узгоджене використання та подальший аналіз.

Один із найпопулярніших інструментів для ведення торгового журналу серед трейдерів-початківців – Google Sheets [8] (рис. 1.1). Тут можна швидко створити таблицю, додати потрібні колонки, прописати формули для статистики. Хмарне зберігання дозволяє мати доступ до даних з будь-якого пристрою, а спільний доступ дає змогу аналізувати результати разом із наставником або колегами.

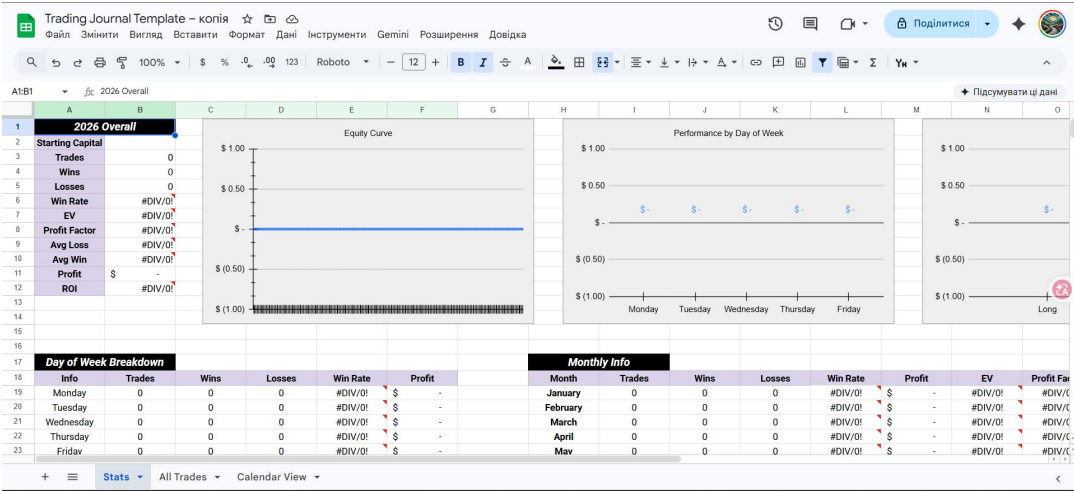


Рис. 1.1. Реалізована система в Google Sheets

Основна перевага це повна свобода у налаштуванні структури. Трейдер сам вирішує, які параметри угод фіксувати. Однак ця свобода має зворотний бік: усю відповідальність за розрахунки, цілісність даних і підтримку журналу трейдер бере на себе. Фактично доводиться самостійно проєктувати систему аналізу власної торгівлі.

Крім того, Google Sheets не вирішує проблему розпорошення інформації. Як мінімум для збереження знімків графіків потрібен буде Google Drive, а для ведення великих нотаток може знадобитися текстовий редактор. У результаті

контекст роботи розбитий між кількома сервісами, і підтримувати цілісну картину стає дедалі важче.

Автоматизація в таблицях теж обмежена. Хоча є можливість писати скрипти на Google Apps Script, налаштування автоматичного імпорту угод вимагає додаткових технічних знань і часу. А якщо записів стає багато, а формули складними, продуктивність таблиці помітно знижується.

Таким чином, Google Sheets дає високу гнучкість і доступність, але не централізує процес роботи трейдера і не позбавляє від рутини ручного введення даних.

Notion [9] – універсальна платформа для управління інформацією, яка поєднує бази даних, текстові нотатки, таблиці та зв'язки між ними в одному робочому просторі (рис. 1.2). Трейдери часто використовують її для ведення журналу угод, аналітичних матеріалів, планування торгових сесій та створення особистої бази знань.

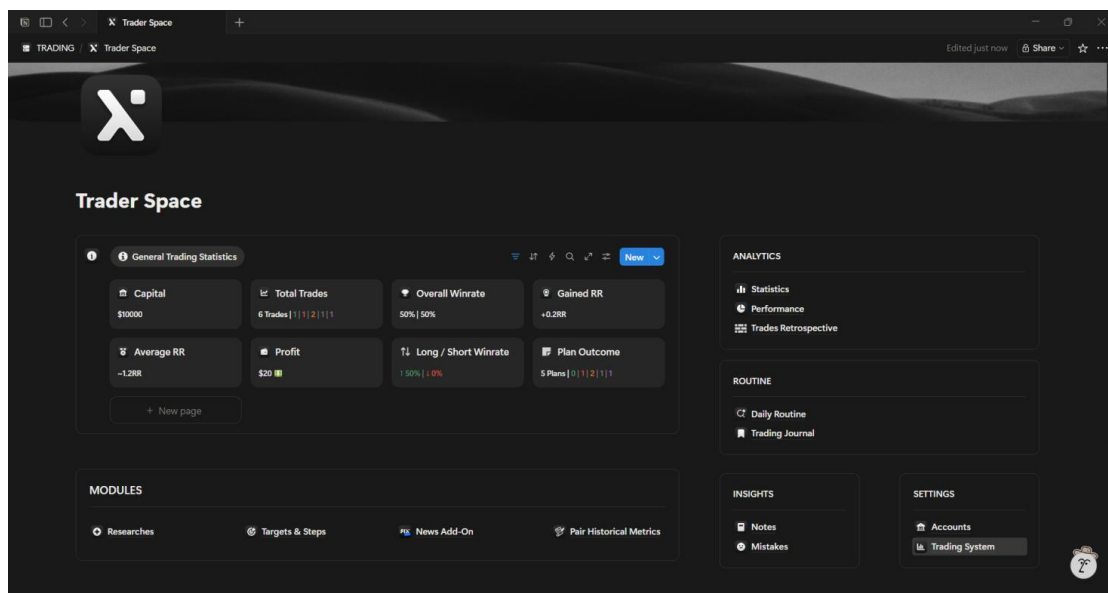


Рис. 1.2. Реалізована система в Notion

Головна перевага Notion перед Google Sheets це можливість будувати зв'язки між різними записами. Ви можете пов'язати угоду з конкретною стратегією, додати нотатки про емоційний стан, прикріпити аналіз ринку. Усе

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				16
Змн.	Арк.	№ докум.	Підпис	Дата		

це зберігається в єдиному середовищі, що дає вищий рівень централізації порівняно з таблицями.

Однак Notion не є спеціалізованим трейдинговим інструментом. Автоматичного імпорту угод із брокера тут немає, або він реалізується через складні інтеграції.

Найбільша проблема – це початкове налаштування. Щоб створити дійсно зручний торговий журнал, потрібно добре розібратися в логіці Notion, продумати структуру баз даних, зв'язки між ними та способи відображення інформації. На це йде багато часу. І навіть після цього в системі немає вбудованих інструментів для автоматичного аналізу поведінкових помилок або розрахунку професійних торгових метрик.

В результаті Notion дозволяє створити досить централізоване середовище для роботи трейдера, але вимагає значних зусиль на адаптацію і не пропонує готових аналітичних механізмів.

TraderSync [10] – платформа, створену виключно для ведення торгового журналу та аналізу результатів (рис. 1.3). На відміну від універсальних Google Sheets або Notion, тут усе вже налаштовано під трейдинг: готова структура даних, професійна статистика, графіки ефективності.

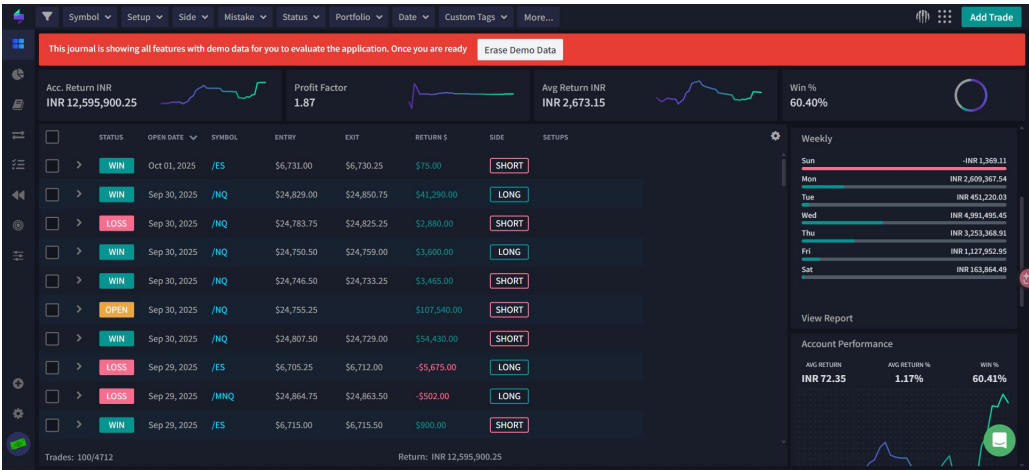


Рис. 1.3. TraderSync

Головна перевага це висока автоматизація. TraderSync підтримує прямий імпорт угод із багатьох брокерів і терміналів. Трейдеру не доводиться вручну

вводити кожну угоду. Система одразу показує ключові показники, аналіз стратегій, відстеження повторюваних помилок та навіть оцінку впливу психологічних факторів на результат (у розширених тарифах).

Однак спеціалізація має зворотний бік і це обмежена гнучкість. Трейдер працює в межах тієї моделі даних, яку пропонує платформа, і не можете суттєво її змінити під свої потреби. Якщо він хоче вести довгострокове планування, зберігати навчальні матеріали або детальні текстові рефлексії, TraderSync для цього підходить погано. Доводиться знову використовувати Notion, Obsidian або інші інструменти. Таким чином, навіть спеціалізований журнал не покриває всіх аспектів діяльності трейдера в одному середовищі.

TraderSync ідеальний для глибокого аналізу угод, якщо не потрібна гнучка структура даних. Але він не замінює повноцінного робочого простору для планування та навчання.

Таблиця 1.1

Порівняльна характеристика наявних інструментів

Характеристика	Google Sheets	Notion	TraderSync
Ведення журналу угод	+	+	+
Налаштування власної структури даних	+	+	+/-
Зв'язки між записами та сутностями	-	+	+/-
Збереження знімків графіків	+/-	+	+
Ведення текстових нотаток	+/-	+	+/-
Планування торгових сесій	-	+	-
Зберігання навчальних матеріалів	+/-	+	-
Автоматичний імпорт угод	-	-	+

продовж. табл. 1.1

Вбудована торгова аналітика	-	+/-	+
Аналіз поведінкових помилок	-	+/-	+
Візуалізація статистики	+/-	+/-	+
Робота з різними типами даних	+	+	-
Централізоване інформаційне середовище	-	+/-	+/-
Підтримка цілісного процесу діяльності трейдера	+/-	+/-	+/-
Гнучкість моделі даних	+	+	-
Масштабованість системи	+	+	-
Автоматизація процесів	+/-	-	+
Необхідність використання додаткових систем	+	+	+
Кількість середовищ у типовому робочому процесі	-	+/-	+/-
Рівень інформаційної фрагментації	-	+/-	+/-
Зв'язність даних між елементами системи	-	+	+/-
Спеціалізація під трейдинг	-	-	+
Можливість адаптації під індивідуальний стиль роботи	+	+	-

Умовні позначення:

“+” – функціональність реалізована повністю та доступна у базовій версії;

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		19

“+/-” – функціональність доступна з обмеженнями (потребує додаткових налаштувань / платного тарифу / сторонніх інтеграцій);

“-” – функціональність відсутня або реалізована на рівні, недостатньому для повноцінного використання.

1.3 Постановка задачі дослідження та розробки

Проведений аналіз наявних інструментів чітко показує: універсального рішення для трейдера досі не існує. Кожен із розглянутих підходів частково вирішує проблему, але разом із тим несе власний набір обмежень, і саме ці обмеження визначають, чому жоден із них не став стандартом у спільноті.

Електронні таблиці дають трейдеру повну свободу дій: можна налаштувати будь-яку структуру, додати формули, побудувати графіки, змінити логіку в будь-який момент. Але ця свобода має ціну. З часом таблиця перетворюється на складний, погано документований артефакт, у якому трейдер витрачає більше часу на підтримку структури, ніж на саму торгівлю й аналіз результатів. Виникають дублювання даних, непослідовності у форматуванні, залежності між аркушами, які важко відстежити. Будь-яка зміна логіки потребує ручного перероблення великих масивів даних. Крім того, таблиці не мають вбудованого розуміння торгових концепцій таких як поняття угоди, активу, ризику чи торгової сесії потрібно описувати власноруч і кожного разу заново.

Notion вирішує частину цих проблем. Це гнучкий простір, де можна поєднати текстові нотатки, бази даних і зв'язки між ними. Трейдер може зберігати торговий журнал поруч із нотатками про ринок, торговими правилами та аналізом помилок. Але Notion це інструмент загального призначення, і він не розуміє специфіки трейдингу. Немає жодних вбудованих шаблонів для угод, немає автоматичного розрахунку прибутковості, немає логіки управління ризиком. Усе це доводиться будувати самостійно, і початковий етап налаштування нерідко стає серйозним бар'єром, особливо для трейдерів без

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				20
Змн.	Арк.	№ докум.	Підпис	Дата		

технічного досвіду. Більше того, коли система вже побудована, її модифікація стає ризикованою: легко зламати зв'язки між базами даних чи втратити частину логіки.

TraderSync обирає протилежний підхід. Платформа з першого дня готова до роботи: зрозумілий інтерфейс, автоматичний імпорт угод, аналітичні панелі, звіти. Але саме ця готовність і є обмеженням. Структура системи жорстко зафіксована розробниками, і трейдер не може суттєво її змінити. Не можна додати власну властивість до угоди, не можна ввести нові категорії аналізу, не можна адаптувати систему під специфічну торгову стратегію чи нестандартний тип активів. Трейдер змушений підлаштовуватися під платформу, а не навпаки.

Спільна проблема всіх трьох підходів це відсутність балансу між гнучкістю та предметною орієнтацією. Інструменти або занадто відкриті й вимагають великих зусиль на побудову структури, або занадто закриті й не дають трейдеру адаптувати систему під свої потреби. Ця проблема не є технічною в чистому вигляді, вона концептуальна. Потрібна система, яка від початку розуміє предметну область трейдингу, але водночас залишає достатній простір для гнучкості й персоналізації.

З огляду на виявлені обмеження, постає конкретна потреба: трейдеру потрібна платформа, яка зберігає всі торгові дані в одному місці, структурує їх і дає змогу аналізувати без зайвих зусиль на налаштування. При цьому важлива не лише централізація інформації, а й її зв'язність. Система повинна не просто накопичувати записи, а відображати залежності між ними, щоб з окремих угод можна було виводити закономірності, бачити динаміку, розуміти, які рішення призводять до збитків, а які до стабільного результату.

Таким чином, задача розробки полягає у проєктуванні та реалізації вебплатформи, орієнтованої на підтримку типових сценаріїв діяльності трейдера, із можливістю гнучкого налаштування структури даних без порушення цілісності та узгодженості інформації.

Досягнення цієї мети вимагає послідовної роботи від аналізу предметної області до перевірки готового продукту.

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				21
Змн.	Арк.	№ докум.	Підпис	Дата		

Для позначення розроблюваної системи в межах роботи надалі використовується назва «FIX Space». Назва походить від ідеї усунення (fix) інформаційної фрагментації та створення єдиного інформаційного простору (space), у межах якого трейдер може зберігати, структурувати та аналізувати всі аспекти власної діяльності. Надалі терміни «платформа» та «FIX Space» використовуються як взаємозамінні позначення розроблюваного програмного продукту.

Перш ніж проектувати систему, необхідно добре розуміти, як трейдер насправді працює із даними. Це передбачає дослідження наявних підходів до ведення торгових журналів і організації інформації не лише для порівняння інструментів, а й для того, щоб побачити, які патерни роботи повторюються незалежно від платформи. Потрібно визначити типові сценарії діяльності трейдера: що він фіксує перед угодою, що після, які дані аналізує на тижневому огляді, як зберігає торгові правила, як відстежує власний психологічний стан. На основі цього аналізу формуються конкретні функціональні вимоги до платформи. Поряд із функціональними вимогами визначаються й нефункціональні: надійність зберігання даних, швидкість відгуку інтерфейсу, захист облікових записів і приватність торгової інформації.

На етапі проектування системи визначається загальна архітектура застосунку та принципи взаємодії його компонентів. Вибір архітектурного підходу безпосередньо впливає на те, наскільки легко буде підтримувати й розширювати платформу в майбутньому. Архітектура повинна чітко розділяти відповідальності між шарами, підтримувати незалежне масштабування та не створювати жорстких залежностей між частинами системи.

Паралельно розробляється модель даних. Це один із найвідповідальніших кроків, оскільки саме структура бази даних визначає, які запити можна реалізувати ефективно, а які ні. Модель має враховувати природу торгових даних: ієрархію сутностей, гнучкість типів полів, зв'язки між записами в різних базах даних. Важливо, щоб схема дозволяла трейдеру змінювати структуру своїх даних без втрати вже введеної інформації.

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				22
Змн.	Арк.	№ докум.	Підпис	Дата		

На рівні проєктування також визначається логіка взаємодії користувача з інтерфейсом: де розміщуються ключові елементи, як влаштована навігація між робочими просторами, яким чином трейдер редагує властивості запису, як виглядають аналітичні подання. Проєктування інтерфейсу ведеться паралельно з проєктуванням бекенду, щоб уникнути ситуацій, коли інтерфейс вимагає від API даних, які складно або неефективно надати.

Центральна частина роботи це програмна реалізація спроектованої системи. Реалізація охоплює кілька взаємопов'язаних блоків. По-перше, механізми для роботи з робочими просторами та базами даних: створення, редагування, видалення структур, управління доступом, навігація між рівнями ієрархії. По-друге, система запису та зберігання властивостей: кожен запис у базі даних може мати різні типи полів і система повинна коректно обробляти кожен із них. По-третє, інструментарій для аналізу: фільтри, сортування, агрегація, формули та автоматизації, які дозволяють трейдеру отримувати з накопичених даних корисні висновки, а не просто список записів.

Важливо, що реалізація ведеться з урахуванням цілісності даних: будь-яка операція редагування структури не повинна пошкоджувати або втрачати вже внесену інформацію. Це технічно нетривіальна вимога, яка потребує уважного проєктування міграцій і обробки граничних випадків.

Завершальний етап роботи це перевірка того, що система функціонує коректно і відповідає сформованим вимогам. Тестування охоплює кілька рівнів: перевірка окремих функцій на рівні юніт-тестів, перевірка взаємодії між компонентами на рівні інтеграційних тестів, а також оцінка поведінки системи в повних користувацьких сценаріях.

Поряд із функціональним тестуванням проводиться оцінка продуктивності: наскільки швидко система відповідає на запити, як поводить себе при збільшенні кількості записів, чи є вузькі місця у роботі з базою даних. Результати тестування порівнюються з раніше визначеними критеріями якості, і на їх основі ухвалюються рішення щодо подальших доопрацювань.

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		23

Висновки до першого розділу

У першому розділі було розглянуто, з чим саме щодня стикається приватний трейдер, коли намагається розібратися у власних результатах. Виявилося, що одних лише записів про угоди з їхніми цифрами недостатньо, оскільки для повноцінного аналізу потрібен ще й контекст. Йдеться не тільки про параметри угод, а й про емоційний стан, нотатки, поведінкові помилки та власні спостереження. Саме поєднання цих шарів інформації дозволяє сформувати більш об'єктивне розуміння власної торгової діяльності.

На практиці ж дані зазвичай розподілені між різними файлами, таблицями та сервісами, які між собою не пов'язані. Такий стан організації визначається як інформаційна фрагментація. Її наслідками стають постійні перемикання між середовищами, зростання когнітивного навантаження та ускладнення ретроспективного аналізу. У результаті трейдер витрачає значну частину уваги не на аналіз ринку, а на пошук і збирання інформації.

У межах розділу також було проаналізовано існуючі підходи до організації торгової діяльності. Розглянуті рішення показали, що сучасні інструменти або забезпечують гнучкість, але потребують значної ручної організації, або пропонують готову структуру, проте суттєво обмежують адаптацію під індивідуальний стиль роботи. Через це жоден із підходів не забезпечує одночасно централізацію даних, гнучкість структури та підтримку всіх аспектів діяльності трейдера в межах єдиного середовища.

Окремо було сформовано методологію оцінювання ефективності майбутньої системи. Було визначено основні користувацькі та технічні метрики, критерії оцінювання якості функціонування платформи, а також методи збору й аналізу даних. Запропонований підхід дозволяє оцінювати не лише функціональні можливості системи, а й її вплив на процес роботи трейдера, рівень централізації інформації, швидкість роботи з даними та стабільність використання системи.

Проведений аналіз показав, що централізація інформації в межах єдиного середовища потенційно дозволяє знизити когнітивне навантаження та

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		24

спростити роботу з торговими даними. Коли всі необхідні матеріали доступні в одному місці, трейдер витрачає менше ресурсів на пошук контексту й більше на аналіз та прийняття рішень. Саме це формує основу для подальшого проєктування та реалізації спеціалізованої вебплатформи, орієнтованої на подолання інформаційної фрагментації у діяльності приватного трейдера.

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		25

РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБПЛАТФОРМИ ДЛЯ ОРГАНІЗАЦІЇ ДІЯЛЬНОСТІ ПРИВАТНОГО ТРЕЙДЕРА

2.1 Визначення вимог до системи

Головною метою даної платформи є допомогти трейдеру краще розуміти власні дії, бачити помилки та приймати більш зважені рішення. Відповідно, система має стати робочим середовищем, яке зменшить хаос і вплив суб'єктивних чинників на торгівлю. На основі даної ідеї можна сформулювати наступні бізнес-вимоги.

Система повинна об'єднувати всі основні типи даних трейдера в єдиному просторі з можливістю встановлювати логічні зв'язки між ними, щоб уникнути фрагментації інформації та сформувати цілісне уявлення про власну діяльність.

Накопичення досвіду має бути структурованим: користувач повинен мати змогу повертатися до минулих даних, аналізувати їх та виявляти повторювані закономірності, при цьому система забезпечує сталість базової структури з можливістю її адаптації під індивідуальні потреби.

Платформа має виступати інструментом об'єктивної оцінки, автоматично розраховуючи ключові показники та виявляючи розбіжності між запланованими діями та фактичними результатами, щоб зменшити вплив когнітивних викривлень на процес прийняття рішень.

Користувач не повинен витрачати значний час на початкове налаштування, тому система має пропонувати готову предметно-орієнтовану структуру, яку водночас можна гнучко змінювати під власний стиль торгівлі без залучення розробника.

Архітектура системи повинна допускати подальше розширення функціоналу, додавання нових типів даних і сценаріїв використання без порушення вже реалізованої логіки, що забезпечує довгострокову життєздатність продукту.

Оскільки платформа працює з персональними даними та результатами фінансової діяльності, вона має гарантувати цілісність, збереження та захист

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		26

інформації від несанкціонованого доступу, а також бути передбачуваною та зрозумілою в роботі для формування довіри користувача.

Проектування функціональної архітектури платформи виходить із необхідності чітко окреслити межі системи та сценарії взаємодії з нею, для чого було обрано методологію варіантів використання. Відповідно можна зосередитися на реальних потребах користувача, абстрагуючись від технічних деталей реалізації на ранніх етапах розробки, і сформуванати набір функцій, що стане фундаментом для подальшого проектування бази даних та API.

У системі визначено дві ключові ролі учасників, кожна з яких має свій специфічний набір прав доступу та сценаріїв поведінки. **Неавторизований користувач**, або гість, є відвідувачем платформи, який ще не пройшов процедуру ідентифікації. Його можливості обмежені ознайомленням із загальнодоступною інформацією про сервіс, а також реєстрацією нового облікового запису або відновленням пароля. На цьому етапі користувач не має доступу до персональних даних, оскільки система ще не ідентифікувала його особу.

Авторизований користувач, або трейдер, є основним актором системи, який успішно пройшов автентифікацію та отримав доступ до персонального робочого простору. Для цієї ролі реалізовано повний спектр функціональних можливостей платформи. Трейдер може створювати та редагувати записи про угоди, керувати структурою баз даних, налаштовувати автоматизації, переглядати детальну статистику ефективності торгівлі.

Для повного та систематизованого опису функціональних можливостей платформи доцільно деталізувати кожен варіант використання у вигляді структурованих таблиць (табл. 2.1-2.8). Першу групу – сценарії автентифікації неавторизованого користувача та первинного доступу до системи - наведено в табл. 2.1.

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижмотря О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		27

Таблиця 2.1

Опис прецедентів автентифікації та управління простором

Назва	Актори	Опис
Зареєструватися	Неавторизований користувач	Створення облікового запису з підтвердженням через email
Увійти до системи	Неавторизований користувач	Авторизація через пароль або Google OAuth для доступу до даних
Відновити пароль	Неавторизований користувач	Скидання пароля за допомогою посилання на електронну пошту

Після завершення реєстрації користувачу необхідно підтвердити свою електронну адресу, після чого він отримує доступ до базових механізмів організації робочого середовища – створення та налаштування робочих просторів і секцій бокової панелі. Відповідні прецеденти наведено в табл. 2.2.

Таблиця 2.2

Опис прецедентів верифікації та організації робочого середовища

Назва	Актори	Опис
Підтвердити пошту	Неавторизований користувач	Верифікація акаунту через унікальне посилання для активації
Керувати робочим простором	Авторизований користувач	Створення, редагування та видалення. Налаштування робочих просторів, локальних параметрів простору.
Керувати секціями	Авторизований користувач	Створення, редагування та видалення. Організація структури бокової панелі за допомогою секцій

Наступна група прецедентів стосується безпосередньо управління структурою даних – створення баз, налаштування їхніх властивостей та роботи із записами, що є фундаментом для подальшої фіксації торгової інформації. Ці сценарії описано в табл. 2.3.

Таблиця 2.3

Опис прецедентів управління структурою даних

Назва	Актори	Опис
-------	--------	------

продовж. табл. 2.3

Керувати базами даних	Авторизований користувач	Створення, редагування та видалення, дублювання та блокування структури
Керувати та налаштовувати властивості	Авторизований користувач	Створення, редагування та видалення. Вибір типів даних (числа, дати, формули) для колонок у базі
Керувати записами	Авторизований користувач	Додавання, редагування, копіювання та видалення даних у таблицях

Окрему групу формують прецеденти роботи з вмістом і шаблонами, які дозволяють користувачу наповнювати записи аналітичним текстом та прискорювати рутинні дії за допомогою готових структур, а також гнучко налаштовувати подання даних. Опис цих прецедентів наведено в табл. 2.4.

Таблиця 2.4

Опис прецедентів роботи з вмістом та шаблонами

Назва	Актори	Опис
Редагувати вміст	Авторизований користувач	Написання аналітики до угод з використанням блоків та компонентів
Керувати шаблонами	Авторизований користувач	Створення, редагування та видалення. Використання готових структур для швидкого заповнення запису
Налаштування подання	Авторизований користувач	Створення кастомних фільтрів, сортувань, групувань для даних, показ властивостей

Прецеденти пошуку та обміну даними, наведені в табл. 2.5, забезпечують швидкий доступ до інформації в межах простору, а також перенесення історії угод із зовнішніх джерел.

Таблиця 2.5

Опис прецедентів пошуку та обміну даними

Назва	Актори	Опис
Шукати записи	Авторизований користувач	Швидкий пошук інформації по всьому простору або окремій базі
Імпортувати та експортувати дані	Авторизований користувач	Завантаження історії торгів із CSV-файлів зовнішніх платформ. Вивантаження даних із бази у форматі CSV

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				29
Змн.	Арк.	№ докум.	Підпис	Дата		

Ключову роль для аналітичної складової платформи відіграють прецеденти, описані в табл. 2.6: вони охоплюють перегляд статистики й торгових метрик, налаштування формул для розрахунку складних показників та керування автоматизаціями, що змінюють дані за заданими тригерами.

Таблиця 2.6

Опис прецедентів аналітики та автоматизації

Назва	Актори	Опис
Аналізувати статистику	Авторизований користувач	Створення та редагування статистики. Перегляд графіків та торгових метрик
Керувати формулами	Авторизований користувач	Створення, редагування та видалення. Налаштування автоматичного розрахунку складних показників
Керувати автоматизаціями	Авторизований користувач	Створення правил для автоматичної зміни даних за тригерами, редагування та видалення.

Прецеденти інтеграції та адміністрування, наведені в табл. 2.7, описують взаємодію системи із зовнішніми біржами та брокерами, а також інформування користувача про стан цих з'єднань.

Таблиця 2.7

Опис прецедентів інтеграції та адміністрування

Назва	Актори	Опис
Керувати підключеннями	Авторизований користувач	Створення, редагування та видалення. Налаштування API-зв'язків із біржами та платформами
Синхронізувати угоди	Авторизований користувач	Автоматичне отримання закритих позицій з біржових рахунків
Переглядати сповіщення	Авторизований користувач	Отримання системних повідомлень про роботу сервісів

Завершує перелік прецедент управління профілем користувача, наведений у табл. 2.8, що охоплює зміну особистих даних та налаштувань безпеки облікового запису.

Опис прецеденту управління профілем

Назва	Актори	Опис
Керувати акаунтом	Авторизований користувач	Зміна особистих даних, аватара, дати та часу та налаштувань безпеки, видалення акаунту

Для візуалізації зв'язків між акторами та функціональними модулями системи розроблено діаграму варіантів використання (рис. 2.1).

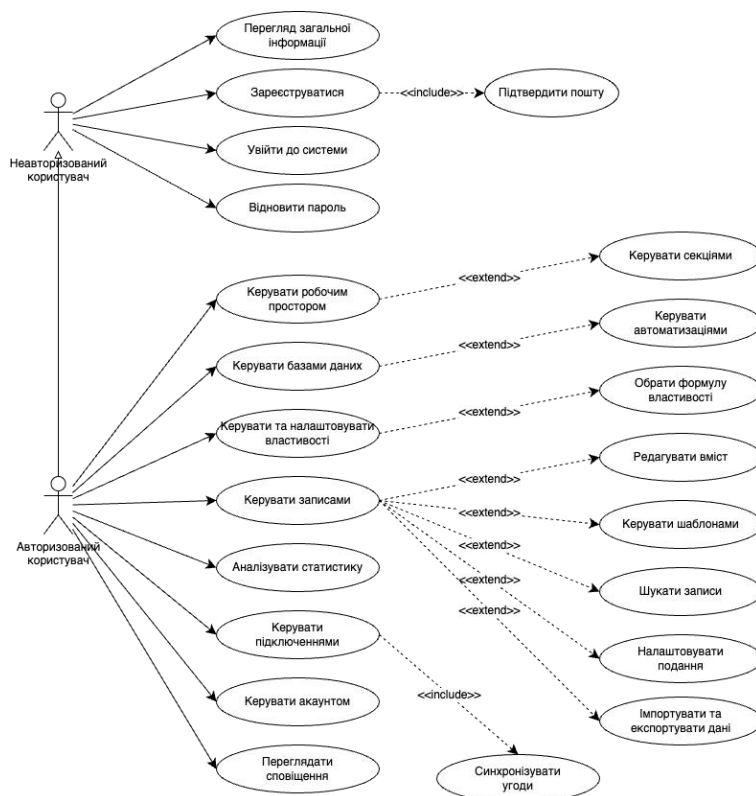


Рис. 2.1. Діаграма варіантів використання

Система буде побудована навколо логіки єдиного простору, де кожна дія відбувається без необхідності перемикається між різними програмами. Нижче наведено узагальнений опис ключових можливостей платформи, сформований на основі визначених сценаріїв використання.

Система забезпечуватиме надійний вхід та управління обліковим записом. Користувач зможе зареєструватися за допомогою email або увійти через Google акаунт, при цьому автоматично підтверджуватиме пошту та створюватиме захищену сесію. Платформа дозволить змінювати пароль та, за необхідності,

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				31
Змн.	Арк.	№ докум.	Підпис	Дата		

повністю видаляти свій профіль разом із усіма даними, гарантуючи повний контроль користувача над своєю інформацією.

Основою роботи стане гнучкий робочий простір. При реєстрації система автоматично створить персональний простір із готовими шаблонами баз даних, адаптованими під потреби трейдера. Користувач зможе створювати додаткові простори, дублювати їх структуру для тестування нових стратегій та швидко перемикатися між ними.

Всередині простору дані будуть організовуватися за допомогою секцій, тобто іменованих груп у бічній панелі, порядок та вигляд яких можна буде легко налаштовувати під себе.

Платформа надаватиме можливості для роботи з базами даних. Трейдери зможуть створювати власні таблиці, додавати до них різноманітні властивості та налаштовувати їх відображення.

Система підтримуватиме створення кількох подань для однієї бази даних, що дозволить переглядати одні й ті самі дані під різними кутами: сортувати, групувати, вибирати які властивості показувати, а які ні, або фільтрувати.

Створення нового запису відбуватиметься миттєво з використанням попередньо налаштованих шаблонів, які автоматично заповнюють стандартні поля. Кожен запис матиме розширену контентну область, де трейдер зможе писати детальні нотатки, додавати скріншоти графіків та формувати структуровані блоки тексту. Система автоматично зберігатиме всі зміни та створюватиме історичні знімки, щоб у разі помилки можна було повернутися до попередньої версії тексту.

Для глибшого аналізу система пропонуватиме вбудовані формули та автоматизації. Користувач зможе обрати потрібен йому тип формули для отримання конкретної інформації та використати її.

Автоматизації дозволять системі самостійно виконувати рутинні дії: наприклад, створювати запис у журналі помилок, якщо угода закрилася зі збитком, або надсилати сповіщення про важливі події. Інтеграція з біржами стане важливим кроком до повної автоматизації збору даних.

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		32

Платформа дозволить підключати акаунти Binance та MetaTrader 5 у режимі лише читання. Система автоматично синхронізуватиме історію угод, зіставляючи їх із внутрішніми записами, що позбавить трейдера від необхідності вручну вносити кожну операцію. Усі конфіденційні ключі доступу зберігатимуться у зашифрованому вигляді. В майбутньому список доступних платформ можна буде збільшити.

Аналітика та пошук допоможуть швидко знаходити відповіді на важливі питання. Глобальний пошук працюватиме миттєво по всьому простору, дозволяючи знайти будь-яку угоду чи нотатку. Вбудована статистика автоматично розраховуватиме ключові метрики ефективності та візуалізуватиме їх у вигляді зрозумілих графіків, даючи змогу порівнювати результати за різні періоди часу.

Налаштування інтерфейсу забезпечать комфортне щоденне використання. Користувач зможе обрати світлу або темну тему, змінити мову інтерфейсу, налаштувати формат дат та часовий пояс.

Система також підтримуватиме імпорт та експорт даних у форматі CSV, що дасть змогу перенести дані з минулого робочого простору.

Нефункціональні вимоги визначатимуть якісні характеристики платформи, що забезпечуватимуть її ефективну та надійну роботу в умовах щоденного використання трейдером. Оцінювання цих характеристик базуватиметься на міжнародному стандарті ISO/IEC 25010 [11; 13], який дозволяє структуровано підійти до таких аспектів, як швидкість роботи, безпека даних та зручність інтерфейсу.

Вимоги до продуктивності формуватимуться з урахуванням необхідності миттєвої реакції системи. Час відкриття робочого простору, переходу між базами даних або застосування фільтрів не перевищуватиме 300 мілісекунд. Створення нового запису про угоду відбуватиметься миттєво з точки зору сприйняття користувачем, а перерахунок значень властивостей типу FORMULA після зміни залежних полів завершуватиметься протягом 1 секунди. Завантаження сторінки статистики з графіками триватиме не більше 2 секунд

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		33

навіть при великому обсязі даних. Важливою умовою буде стабільність швидкодії: продуктивність пошуку не деградуватиме зі зростанням кількості записів, а виконання автоматизацій відбуватиметься у фоновому режимі, не впливаючи на чутливість інтерфейсу.

Надійність та відмовостійкість гарантуватимуть цілісність накопиченого досвіду. Усі операції з базою даних виконуватимуться в межах транзакцій, що відповідають вимогам ACID, завдяки чому дані користувача зберігатимуться навіть у разі технічних збоїв. Ініціалізація робочого простору при реєстрації здійснюватиметься в єдиній транзакції, що унеможлиблює часткове створення структури. Для запобігання втраті текстової інформації реалізовуватиметься механізм автоматичного збереження контентної області записів кожні п'ять хвилин. Формули та автоматизації завжди повертатимуть однаковий результат за однакових вхідних даних. Для уникнення циклічних залежностей у формулах та нескінченних спрацювань автоматизацій впроваджуватимуться алгоритми захисту.

Безпека платформи будуватиметься на принципах ізоляції даних та криптографічного захисту. Дані кожного користувача повністю ізолюватимуться на рівні робочого простору, а доступ до ресурсів буде можливим виключно в межах автентифікованої сесії. Паролі зберігатимуться у вигляді хешів, сформованих алгоритмом. Механізм автентифікації базуватиметься на токенах доступу, що передаватимуться через захищені файли cookie, що убезпечить їх від перехоплення клієнтським кодом. Токен доступу діятиме 15 хвилин, а токен оновлення 7 днів, причому при зміні пароля всі активні сесії автоматично анулюватимуться. Підтримка входу через сторонні сервіси здійснюватиметься без передачі паролів системі. API-ключі та облікові дані інтеграцій зберігатимуться в зашифрованому вигляді, а ключі з правами на торгівлю або виведення коштів відхилятимуться на етапі підключення. Для захисту від атак реалізовуватиметься обмеження частоти запитів: не більше 5 спроб входу за хвилину з однієї IP-адреси та не більше 200 запитів за хвилину

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижемотря О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		34

до всіх API-ендпоінтів. Валідація вхідних даних на сервері здійснюватиметься суворо за схемою, відкидаючи всі зайві поля.

Зручність використання буде пріоритетом проєктування інтерфейсу. Будь-яка типова операція щоденного сценарію, така як фіксація угоди або перегляд статистики, виконуватиметься не більше ніж за 3 кліки від поточного місця в інтерфейсі. Глобальний пошук буде доступний з будь-якої сторінки. Повідомлення про помилки формуватимуться зрозумілою мовою, міститимуть опис причини виникнення проблеми та рекомендації щодо її усунення, уникаючи технічних кодів. Перед виконанням незворотних дій, таких як видалення бази даних, система вимагатиме явного підтвердження від користувача через діалогове вікно.

Доступність системи забезпечуватиметься підтримкою останніх двох версій найпопулярніших веббраузерів. Функціонування платформи вимагатиме стабільного підключення до мережі Інтернет. Масштабованість архітектури досягатиметься завдяки архітектурі на серверній частині, що дозволить додавати новий функціонал та збільшувати навантаження без необхідності модифікації існуючого коду чи порушення цілісності вже реалізованих модулів.

Стабільність зв'язків між сутностями та швидкість обробки запитів зберігатимуться навіть при значному зростанні обсягу накопичених торгових даних. Конвеєр безперервної інтеграції та розгортання (CI/CD) забезпечуватиме автоматичне тестування, перевірку безпеки та збірку артефактів при кожному оновленні коду, гарантуючи стабільність продукції та можливість швидкого виправлення помилок.

2.2 Проєктування моделей даних системи

Від спроектованих моделей даних безпосередньо залежать швидкість роботи та гнучкість налаштувань. Головне завдання полягає в тому, щоб поєднати чітку структуру, необхідну для точних статистичних розрахунків, зі свободою користувача створювати власні поля та зв'язки. Тому система не

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		35

обмежуватиме трейдера жорсткими таблицями, а реалізує гібридний підхід: реляційна цілісність для системних сутностей і гнучкий формат JSON для користувацьких даних.

Щоб зробити систему дійсно гнучкою, необхідно надати користувачеві інструменти для опису будь-якої інформації. Замість того щоб примушувати трейдера підлаштовуватися під технічні обмеження жорстких таблиць, система пропонуватиме набір універсальних типів даних, які адаптуватимуться під специфіку торгових задач. Це будуть не просто стандартні поля для введення тексту чи чисел, а спеціалізовані інструменти, що дозволять структурувати дані на всіх рівнях. Кожен тип матиме свої власні налаштування та правила валідації.

Система пропонуватиме трейдеру 11 універсальних типів властивостей, кожен з яких матиме свої специфічні налаштування та правила валідації. Такий різноманітний набір інструментів дозволить описати будь-яку інформацію. Детальний опис конфігурації кожного типу наведено нижче.

Властивість типу **ТЕКСТ** слугуватиме основою для зберігання текстової інформації, включаючи прості рядки або форматований контент. Вона використовуватиметься для написання нотаток, описів торгових стратегій, назв активів та будь-якого іншого текстового контенту.

Для роботи з числовими даними призначено тип **ЧИСЛО**, який підтримуватиме гнучке форматування: від цілих чисел та дробових значень до валют і відсотків. Цей тип потрібен для фіксації цін входу, обсягів позицій, комісій та інших фінансових метрик. Конфігурація включатиме `format` для вибору способу відображення, а також опціональні параметри `decimalPlaces`, `currencySymbol` символ валюти, `prefix` та `suffix` для додавання тексту перед або після числа.

Тип **ДАТА** забезпечуватиме зберігання часових міток у форматі ISO 8601, що критично важливо для фіксації дат відкриття та закриття угод, термінів виконання завдань. Користувач зможе налаштовувати формат відображення дати через параметр `format`, вмикати або вимикати відображення часу та обирати його формат.

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		36

Для бінарних станів, таких як виконано чи не виконано використовуватиметься тип **ЧЕКБОКС**. Він зберігатиме булеве значення та часто застосовуватиметься в чеклистах підготовки до торгівлі або для позначення дотримання правил ризик-менеджменту.

Вимірювання тривалості процесів, наприклад, часу утримання позиції або загального часу торгівлі за день, реалізовуватиметься через тип **ТРИВАЛІСТЬ**. Значення зберігатиметься як ціле число секунд, але відображатиметься у зручному для користувача форматі, визначеному параметром format.

Для категоризації даних за фіксованими наборами значень слугуватиме тип **ВИБІР**. Він дозволить обирати один або кілька варіантів із згрупованого списку, що ідеально підходить для позначення типів помилок, торгових сесій або категорій активів. Конфігурація включатиме прапорець isMultiSelect для дозволу множинного вибору та масив categories, де кожна категорія матиме назву label та список опцій options з власними значеннями value, кольорами color та іконками icon.

Тип **СТАТУС** спеціалізуватиметься на відстеженні прогресу виконання задач або стадій угоди, використовуючи три семантичні категорії: todo (не почато), inProgress (в роботі) та complete (завершено).

Для побудови складних зв'язків між різними базами даних, наприклад, для прив'язки угоди до конкретного торгового рахунку або стратегії, використовуватиметься тип **ЗВ'ЯЗОК**. Він зберігатиме посилання на записи іншої сутності. Конфігурація включатиме relatedEntityId, тобто унікальний ідентифікатор бази даних, на яку посилається поле, та прапорець multiple, який визначатиме, чи можна обрати лише один запис, чи кілька одночасно.

Найпотужнішим інструментом для автоматичних розрахунків стане тип **ФОРМУЛА**. Це поле лише для читання, яке обчислюватиме своє значення на основі інших властивостей запису. Воно використовуватиметься для розрахунку R-Multiple, співвідношення ризику до прибутку RR, відсотка ризику від балансу тощо. Конфігурація включатиме type, presetName для вибору

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		37

одного з десяти вбудованих шаблонів, скомпільований вираз `expression`, тип результату `resultType` та технічний стан інтерфейсу `uiState`.

Для суб'єктивної оцінки якості угоди, рівня впевненості або складності помилки призначено тип **РЕЙТИНГ**. Він відображатиметься у вигляді зірок і дозволить швидко візуалізувати рейтинг. Налаштування включатимуть *maxStars* (максимальна кількість, від 1 до 10) та прапорець `allowHalf`, який дозволить використовувати половинчасті значення, наприклад, 3.5 зірки.

Візуалізація прогресу досягнення цілей, наприклад, виконання плану по прибутку або заповнення профілю, реалізовуватиметься через тип **ПРОГРЕС**. Він відображатиметься як кольоровий бар і працюватиме в заданому діапазоні. Конфігурація міститиме `minValue` та `maxValue` для меж діапазону, `step` для кроку зміни, `showLabel` для відображення числа на барі, а також масив `thresholds`, який дозволить змінювати колір бару залежно від поточного значення (наприклад, червоний для низьких показників, зелений для високих).

Окрім структурованих даних у вигляді колонок, трейдеру потрібен простір для вільного викладу думок, аналізу графіків та деталізації своїх рішень. Для цього система надаватиме потужний редактор контентної області, який працюватиме за принципом конструктора. Користувач зможе формувати візуально зрозумілі структури з текстових фрагментів, медіафайлів та інших елементів, групуючи їх у логічні блоки з власним оформленням.

В основі організації контенту лежатиме чотирирівнева ієрархічна модель, що поєднує гнучкість розмітки з візуальною цілісністю. Верхнім рівнем виступатимуть рядки, які фактично будуть контейнерами, які будуть ділитися на колонки до 5 штук, ширина яких регулюватиметься перетягуванням меж. Всередині колонок розміщуватимуться компоненти.

Для забезпечення безпеки даних та зручності редагування система реалізовуватиме механізми автоматичного збереження та керування історією змін. Усі дії користувача фіксуватимуться на сервері, що надаватиме доступ до функцій скасування та повторення дій навіть після перезавантаження сторінки. Окрім того, платформа створюватиме знімки контентної області кожні п'ять

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		38

хвилин. Зберігатиметься до п'ятдесяти останніх знімків для кожного запису, і у разі досягнення ліміту найстаріший із них автоматично видалятиметься.

Компонент типу **ТЕКСТ** виступає фундаментальним елементом для фіксації розгорнутих торгових рефлексій, детальних описів ринкових ситуацій та емоційного стану трейдера. Його здатність підтримувати базове форматування дозволяє акцентувати увагу на ключових висновках за допомогою жирного шрифту чи курсиву, перетворюючи хаотичний потік думок на структурований, легко читабельний матеріал безпосередньо всередині торгового щоденника.

Для подолання візуального хаосу у великих масивах аналітичних даних застосовується компонент **ЗАГОЛОВОК**, який забезпечує чітку ієрархічну структуру контенту. Створення заголовків різних рівнів дозволяє трейдеру миттєво орієнтуватися у просторі нотаток, швидко знаходячи потрібний розділ, наприклад, аналіз конкретної сесії чи висновки щодо стратегії, без виснажливого прокручування всього документа.

Компонент **ВИНОСКА** слугує потужним інструментом візуального акцентування критично важливої інформації, такої як правила ризик-менеджменту, попередження про типові поведінкові помилки або ключові інсайти. Завдяки можливості налаштування типу стилю, кольору та іконки, цей блок створює миттєвий візуальний якір, який не дозволяє важливим настановам загубитися в загальному потоці тексту, сприяючи дисципліні під час прийняття рішень.

Для забезпечення цілісності інформаційного простору без необхідності покидати поточний контекст використовується компонент **БАЗА ДАНИХ**, який інтегрує таблиці з інших баз даних прямо в тіло запису. Це дає змогу трейдеру бачити пов'язані сутності, наприклад, історію всіх угод конкретного дня в записі щоденного аналізу або перелік помилок, прив'язаних до певної стратегії, формуючи таким чином повну та зв'язну картину торговельної активності.

Візуалізація об'єктивних метрик ефективності реалізується через компонент **ГРАФІК**, який трансформує сухі статистичні дані у зрозумілі

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		39

графічні образи. На основі властивостей запису або агрегованих даних пов'язаних баз цей інструмент будує лінійні, стовпчикові або кругові діаграми, дозволяючи миттєво оцінити динаміку прибутковості, математичне очікування або співвідношення ризику до прибутку, що є критичним для боротьби з когнітивними упередженнями.

Компонент **ЧЕКЛИСТ** впроваджує елемент системної дисципліни у торговий процес, перетворюючи абстрактні правила на конкретний список дій. Він ідеально підходить для створення чеклистів входу в угоду або планів на торгову сесію, де кожен пункт може бути відмічений як виконаний, що мінімізує ризик імпульсивних рішень та допомагає трейдеру діяти виключно в межах своєї торгової системи.

Найважливішим інструментом для візуального підтвердження торгових ідей залишається компонент **МЕДІА**, який дозволяє безпосередньо вбудовувати скріншоти графіків у тіло аналітичного запису. Підтримка основних форматів зображень із чітким обмеженням розміру гарантує, що технічна картина ринку на момент відкриття чи закриття позиції буде надійно зафіксована для подальшого детального розбору без уповільнення роботи платформи.

Для усунення візуального шуму та логічного розмежування різномірних блоків контенту застосовується компонент **РОЗДІЛЬНИК**. Додавання горизонтальної лінії дозволяє чітко відокремити, наприклад, сухі фактичні параметри угоди від глибокої емоційної рефлексії, покращуючи загальне сприйняття структури документа та знижуючи когнітивне навантаження під час читання.

Коли необхідно компактно перелічити параметри, кроки алгоритму дій або фактори, що вплинули на ринкове рішення, на допомогу приходить компонент **СПИСОК**. Він дозволяє створювати як марковані переліки для неупорядкованих наборів думок, так і нумеровані послідовності для чітких інструкцій, забезпечуючи лаконічне та структуроване подання інформації без зайвого тексту.

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				40
Змн.	Арк.	№ докум.	Підпис	Дата		

Для швидкого порівняння статичних даних, які не вимагають складної логіки повноцінної бази даних, використовується компонент **ТАБЛИЦЯ**. Він дає змогу формувати прості сітки даних у межах одного блоку, що ідеально підходить для зіставлення характеристик кількох активів або зведення ключових показників за торговий день у зручному для сприйняття форматі.

Перше, з чим стикнеться користувач після реєстрації це вже готовий до роботи інструмент. Система автоматично ініціалізуватиме робочий простір із набором попередньо налаштованих баз даних, які закривають ключові потреби трейдера. Це дозволить почати працювати миттєво, залишаючи при цьому можливість адаптувати ці шаблони під свій унікальний стиль торгівлі. Детальний опис полів та їхніх типів для кожної бази наведено у відповідних таблицях нижче.

У секції **Routine** розміщено ядро операційної діяльності трейдера. Центральною базою є **Trading Journal**, де кожен запис відповідає одній угоді. Тут фіксуються всі технічні параметри: валютна пара, напрямок, ціни входу та виходу, обсяг позиції, рівні. Особливістю цієї бази є наявність автоматичних формул для розрахунку. Також тут відображаються суб'єктивні фактори: таймфрейм, модель входу, торгова сесія, рівень впевненості, емоційний стан та дотримання плану. Ця база має численні зв'язки з іншими модулями системи, формуючи єдиний контекст угоди. Структура властивостей цієї бази детально описана в таблиці 2.9.

Таблиця 2.9

Властивості бази даних Trading Journal

Назва	Тип властивості	Опис
Name	TEXT	Короткий ідентифікатор угоди
Status	SELECT	Відображає життєвий цикл угоди
Pair	SELECT	Конкретний актив для аналізу інструментів
Direction	SELECT	Напрямок позиції
Entry Price	NUMBER	Точна ціна виконання ордеру

Exit Price	NUMBER	Середня ціна виходу
Quantity	NUMBER	Об'єм позиції
Entry Date	DATE	Час входу в угоду
Exit Date	DATE	Час виходу з угоди
Outcome	SELECT	Фінальний результат угоди
Initial SL	NUMBER	Ціна, за якої ідея вважається невалідною
Initial TP	NUMBER	Цільова ціна виходу
Risk Amount (\$)	FORMULA	Сума збитку при досягненні Stop Loss
Risk %	FORMULA	Відсоток від капіталу під загрозою
Planned R	FORMULA	Потенційна винагорода відносно ризику
Actual R	FORMULA	Фактичне R-кратне
Gross P&L	FORMULA	Результат угоди без урахування витрат
Fees	NUMBER	Сумарні витрати на комісії та свопи
Net P&L	FORMULA	Реальний чистий прибуток
Narrative Timeframe	SELECT	Основний аналітичний таймфрейм
Execution Timeframe	SELECT	Таймфрейм точки входу
Point A (Origin)	TEXT	Логічна точка старту цінового імпульсу
Point B (Target)	TEXT	Ринкова ціль ціни
Entry Model	SELECT	Конкретний механічний сетап
Session / Time	SELECT	Торговий період
Confidence	RATING	Внутрішня впевненість у сетапі в момент входу
Setup Quality	RATING	Об'єктивна якість сетапу по чеклисту
Plan Adherence	SELECT	Наскільки точно виконано торговий план
Emotion	SELECT (multi)	Емоційний фон під час торгівлі
MFE (Max Fav)	NUMBER	Максимальна ціна на користь угоди
MAE (Max Adv)	NUMBER	Найгірша ціна проти угоди
Exit Efficiency	FORMULA	Наскільки вчасно вийшли
Hold Time	FORMULA	Тривалість угоди
Trading System	RELATION	Прив'язка до конкретної стратегії
Account	RELATION	Рахунок, на якому відбувалася торгівля

Daily Routine	RELATION	Контекст дня (аналіз до сесії)
Notes	RELATION	Уроки та спостереження з угоди
Mistakes	RELATION	Допущені помилки
Is Win	FORMULA	Допоміжний прапорець для агрегації Win Rate

Доповнює журнал база **Daily Routine**, призначена для логування торгових днів. Вона містить поля для фіксації дати, основної торгової пари, ринкового нарративу, логіки, ключових новин та каталізаторів. За допомогою формул система автоматично розраховує точність нарративу, порівнюючи очікування з фактичним результатом дня. Кожен запис у цій базі пов'язаний із конкретними угодами, нотатками та помилками, що дозволяє бачити загальну картину дня, а не лише окремі операції. Опис властивостей наведено в таблиці 2.10.

Таблиця 2.10

Властивості бази даних Daily Routine

Назва	Тип властивості	Опис
Name	TEXT	Назва торгової дня
Date	DATE	Дата сесії для відстеження активності
Pair	SELECT	Головний актив дня
Narrative	SELECT	Очікуваний напрямок ринку на основі HTF
Narrative Logic	TEXT	Детальне обґрунтування біасу
Key Catalyst	SELECT (multi)	Новини/події, що спричиняють волатильність
Narrative Outcome	SELECT	Фактичний напрямок ринку під час сесії
Narrative Accuracy	FORMULA	Автоматична оцінка якості аналізу
Key Levels	TEXT	Ключові зони з пре-маркет аналізу
Session P&L	FORMULA	Сумарний P&L всіх угод за сесію
Trade Count	FORMULA	Кількість угод за сесію
Account	RELATION	Основний рахунок сесії
Trades	RELATION	Усі угоди протягом сесії
Notes	RELATION	Спостереження за ціною в реальному часі
Mistakes	RELATION	Помилки, зафіксовані в цей день

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		43

Для стандартизації підготовки до торгівлі слугує база **Routine Library**. Вона діє як бібліотека. Тут трейдер може фіксувати свій стан перед сесією: якість сну, ментальну готовність, наявність відволікань та план на день. Властивості цієї бази представлені в таблиці 2.11.

Таблиця 2.11

Властивості бази даних Routine Library

Назва	Тип властивості	Опис
Name	TEXT	Назва шаблону
Date	DATE	Дата створення
Sleep Quality	RATING	Фізіологічний базис дисципліни
Pre-Market State	SELECT (multi)	Емоційний фон до відкриття ринку
Post-Market State	SELECT (multi)	Психологічний стан після сесії
Plan Adherence	RATING	Рівень дотримання торгового плану
Distractions	SELECT (multi)	Зовнішні тригери втрати концентрації

Секція **Insight** призначена для накопичення знань та аналізу помилок. База **Notes** виступає сховищем аналітичних матеріалів, структурованих за категоріями: підготовка, виконання, менеджмент, вихід та психологія. Кожен інсайт має рівень достовірності та джерело походження. Завдяки зв'язкам із Trading Journal, будь-яка нотатка може бути безпосередньо прив'язана до конкретної угоди або сесії, створюючи глибокий контекст прийняття рішень. Перелік властивостей наведено в таблиці 2.12.

Таблиця 2.12

Властивості бази даних Notes

Назва	Тип властивості	Опис
Name	TEXT	Ідентифікатор та стисла суть нотатки
Date	DATE	Час фіксації ідеї
Category	SELECT (multi)	Тематична приналежність для фільтрації
Status	SELECT	Поточний етап життєвого циклу нотатки

Confidence	SELECT	Рівень емпіричної або теоретичної валідації
Source	SELECT (multi)	Першоджерело виникнення
Market Regime	SELECT (multi)	Ринкові умови, де патерн є релевантним
Rating	RATING	Суб'єктивна вага та важливість правила
Times Applied	FORMULA	Кількість реальних застосувань в угодах
Last Used	FORMULA	Час останньої інтеграції в торговий процес
Pair	SELECT (multi)	Специфіка інструментів для застосування
Trading Journal	RELATION	Інтеграція з фактичними угодами
Mistakes	RELATION	Зв'язок із помилками, що породили інсайт
Daily Routines	RELATION	Контекст сесії, де було зроблено відкриття

База **Mistakes** реалізує механізм системного трекінгу помилок. Вона класифікує порушення за тими ж п'ятьма категоріями, що й нотатки, та автоматично розраховує їхню серйозність на основі частоти повторень та загальних збитків пов'язаних угод. Ключовою особливістю є інтеграція з автоматизаціями: якщо в журналі угод поле Plan Adherence позначено як Broken, система автоматично створює запис у базі помилок, змушуючи трейдера усвідомити та задокументувати причини відхилення від стратегії. Структура властивостей описана в таблиці 2.13.

Таблиця 2.13

Властивості бази даних Mistakes

Назва	Тип властивості	Опис
Name	TEXT	Ідентифікатор та стисла суть поведінкового збою
Date	DATE	Час першої фіксації відхилення від системи
Category	SELECT (multi)	Етап торгового процесу, де стався збій
Trigger	SELECT (multi)	Емоційний або ринковий каталізатор помилки
Impact Type	SELECT	Характер завданої шкоди (фінансова, психологічна тощо)

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		
						45

Prevention Rule	TEXT	Алгоритм або обмеження для блокування рецидиву
Status	SELECT	Поточний етап роботи над усуненням проблеми
Resolved Date	DATE	Маркер успішної корекції поведінкового патерну
Severity	FORMULA	Рівень загрози на основі частоти повторень
Recurrence Count	FORMULA	Кількість проявів помилки в реальних умовах
Total Cost	FORMULA	Пряма фінансова ціна порушення дисципліни
Last Used	FORMULA	Час останнього прояву слабкості
Trading Journal	RELATION	Інтеграція з конкретними збитковими угодами
Daily Routines	RELATION	Зв'язок із сесіями, що передували помилці

Завершує секцію база **Performance Review**, призначена для періодичного огляду результатів (тижневого, місячного або квартального). Вона агрегує дані з Trading Journal та Mistakes, автоматично розраховуючи кількість угод, Win Rate, чистий P&L та середній R-Multiple за обраний період. Це дозволяє трейдеру виділяти ключові помилки, формулювати уроки та ставити цілі на наступний етап, спираючись на об'єктивні цифри, а не суб'єктивні відчуття. Властивості цієї бази наведено в таблиці 2.14.

Таблиця 2.14

Властивості бази даних Performance Review

Назва	Тип властивості	Опис
Name	TEXT	Ідентифікатор
Date	DATE	Час фіксації підсумків огляду
Period	SELECT	Масштаб аналітичного зрізу (тиждень, місяць, квартал)
Period Start	DATE	Початок аналізованого відрізка часу
Period End	DATE	Кінець аналізованого відрізка часу
Grade	SELECT	Підсумкова суб'єктивна оцінка роботи за період
Net P&L	FORMULA	Чистий фінансовий результат після всіх витрат
Gross P&L	FORMULA	Валовий результат до вирахування комісій

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				46
Змн.	Арк.	№ докум.	Підпис	Дата		

Trade Count	FORMULA	Загальна кількість закритих угод
Win Rate	FORMULA	Відсоток прибуткових угод (базовий індикатор точності)
Total R	FORMULA	Сумарна віддача в R-кратних за період
Avg R	FORMULA	Математичне очікування однієї угоди
Max R	FORMULA	Найкращий показник ефективності входу/виходу
Min R	FORMULA	Найгірший показник, маркер критичних збоїв
Profit Factor	NUMBER	Співвідношення валового прибутку до збитку
Max Drawdown %	NUMBER	Максимальне просідання капіталу (маркер ризику)
Discipline Score	RATING	Рівень дотримання правил та торгового плану
Psychology Score	RATING	Оцінка емоційної стійкості та контролю
Process Score	RATING	Якість підготовки, аналізу та ведення журналу
Trades	RELATION	Інтеграція з мікрорівнем конкретних угод
Mistakes	RELATION	Зв'язок із системними помилками періоду
Daily Routines	RELATION	Контекст щоденної роботи та підготовки
Trading Systems	RELATION	Оцінка ефективності застосованих стратегій
Notes	RELATION	Висновки за період

Секція **Settings** містить інфраструктурні бази даних. База **Accounts** використовується для обліку торгових рахунків різних типів. Для кожного рахунку фіксуються стартовий баланс, валюта, дати активності та правила ризик-менеджменту. Для проп-рахунків додатково враховуються специфічні умови провайдерів, такі як фаза випробовування, цілі по прибутку та правила консистентності. Ця база тісно пов'язана з Trading Journal, дозволяючи відстежувати результати по кожному рахунку окремо. Опис властивостей наведено в таблиці 2.15.

Таблица 2.15

Властивості бази даних Accounts

Назва	Тип властивості	Опис
-------	-----------------	------

Name	TEXT	Унікальний ідентифікатор рахунку
Account Type	SELECT	Визначення режиму роботи (Prop, Live, Demo)
Account ID	TEXT	Технічний ідентифікатор для синхронізації
Currency	SELECT	Базова валюта розрахунків
Starting Balance	NUMBER	Початковий розмір капіталу
Status	SELECT	Поточний життєвий цикл рахунку
Start Date	DATE	Дата початку використання
End Date	DATE	Дата закриття або втрати
Max Overall Drawdown	NUMBER	Абсолютний ліміт просадки капіталу
Drawdown Type	SELECT	Механіка розрахунку просадки
Daily Loss Limit	NUMBER	Добовий ліміт збитків
Max Position Size	NUMBER	Максимальний допустимий об'єм позиції
Max Open Trades	NUMBER	Ліміт одночасно відкритих угод
News Trading	SELECT	Дозвіл на торгівлю під час новин
Weekend Holding	SELECT	Дозвіл на перенесення позицій на вихідні
Hard Stop Loss	CHECKBOX	Вимога обов'язкового фізичного стоп-лосу
Provider	SELECT	Назва проп-компанії
Phase	SELECT	Поточний етап оцінки або роботи
Profit Target	NUMBER	Цільовий показник прибутковості
Consistency Rule	NUMBER	Вимога до рівномірності прибутку
Min Trading Days	NUMBER	Мінімальна кількість активних днів
Profit Split	NUMBER	Частка прибутку трейдера
Challenge Fee	NUMBER	Вартість участі у челенджі
Broker	TEXT	Назва брокера (для Live/Demo)
Platform	SELECT	Використовуваний торговий термінал
Leverage	SELECT	Розмір наданого кредитного плеча
Operations	RELATION	Інтеграція з фінансовими транзакціями
Trades	RELATION	Зв'язок із усіма виконаними угодами

База **Operations** веде облік фінансових потоків, не пов'язаних із безпосередньою торгівлею. Тут фіксуються депозити, виведення коштів та виплати. Кожен запис прив'язаний до конкретного рахунку в базі Accounts, що дозволяє будувати повну картину руху капіталу та реальної прибутковості з урахуванням всіх витрат та надходжень. Структура властивостей представлена в таблиці 2.16.

Таблиця 2.16

Властивості бази даних Operations

Назва	Тип властивості	Опис
Name	TEXT	Опис операції для навігації
Type	SELECT	Класифікація руху капіталу (введення, виведення, комісії)
Status	SELECT	Відстеження життєвого циклу транзакції
Date	DATE	Час ініціації фінансової операції
Settlement Date	DATE	Дата фактичного зарахування або списання коштів
Account	RELATION	Прив'язка транзакції до конкретного торгового рахунку
Payment Method	SELECT	Ідентифікація платіжного каналу або шлюзу
Reference	TEXT	Унікальний ідентифікатор для верифікації у зовнішніх системах
Notes	TEXT	Додатковий контекст або коментар до операції
Amount	NUMBER	Валовий обсяг коштів, що підлягають руху
Fee	NUMBER	Фіксація супутніх витрат на обробку транзакції
Net Amount	FORMULA	Розрахунок чистого руху капіталу після вирахування комісій

База **Trading System** слугує реєстром торгових стратегій. На поточному етапі вона містить назви стратегій та дати їх створення, а також властивості для агрегації статистики щодо ефективності даної стратегії. Зв'язок цієї бази з Trading Journal дозволяє трейдеру чітко розрізняти, яка саме модель входу була використана в угоді, та аналізувати ефективність кожної стратегії ізольовано одна від одної. Властивості цієї бази описані в таблиці 2.17.

Властивості бази даних Trading System

Назва	Тип властивості	Опис
Name	TEXT	Ідентифікатор та стисла суть стратегії
Date	DATE	Час створення або ініціалізації гіпотези
Status	SELECT	Поточний етап життєвого циклу системи
Category	SELECT	Класифікація базового підходу до ринку
Market Type	SELECT (multi)	Ринкові умови, де система є релевантною
Pairs	SELECT (multi)	Специфіка інструментів для застосування
Timeframes	SELECT (multi)	Часові проміжки для аналізу та входу
Sessions	SELECT (multi)	Торгові періоди активності системи
Risk Per Trade %	NUMBER	Жорсткий ліміт капіталу на одну операцію
Max Daily Loss %	NUMBER	Ліміт втрати на день
Max Simultaneous Trades	NUMBER	Обмеження розпорошення уваги та ризику
Min Sample Size	NUMBER	Статистичний поріг для оцінки ефективності
Trades	RELATION	Інтеграція з фактичними виконаними угодами
Total Trades	FORMULA	Загальна кількість протестованих сетапів
Total R	FORMULA	Сумарна віддача в R-кратних
Avg R	FORMULA	Математичне очікування однієї угоди
Win Rate	FORMULA	Відсоток прибуткових реалізацій
Profit Factor	NUMBER	Співвідношення валового прибутку до збитку
Expectancy	NUMBER	Середній очікуваний результат на операцію
Max Consecutive Losses	NUMBER	Маркер максимального психологічного навантаження
Confidence	RATING	Суб'єктивна оцінка надійності системи
Last Updated	DATE	Час останньої валідації або корекції правил

Логічна взаємодія між усіма дев'ятьма базами даних предметної області візуалізована на діаграмі зв'язків (рис. 2.2).

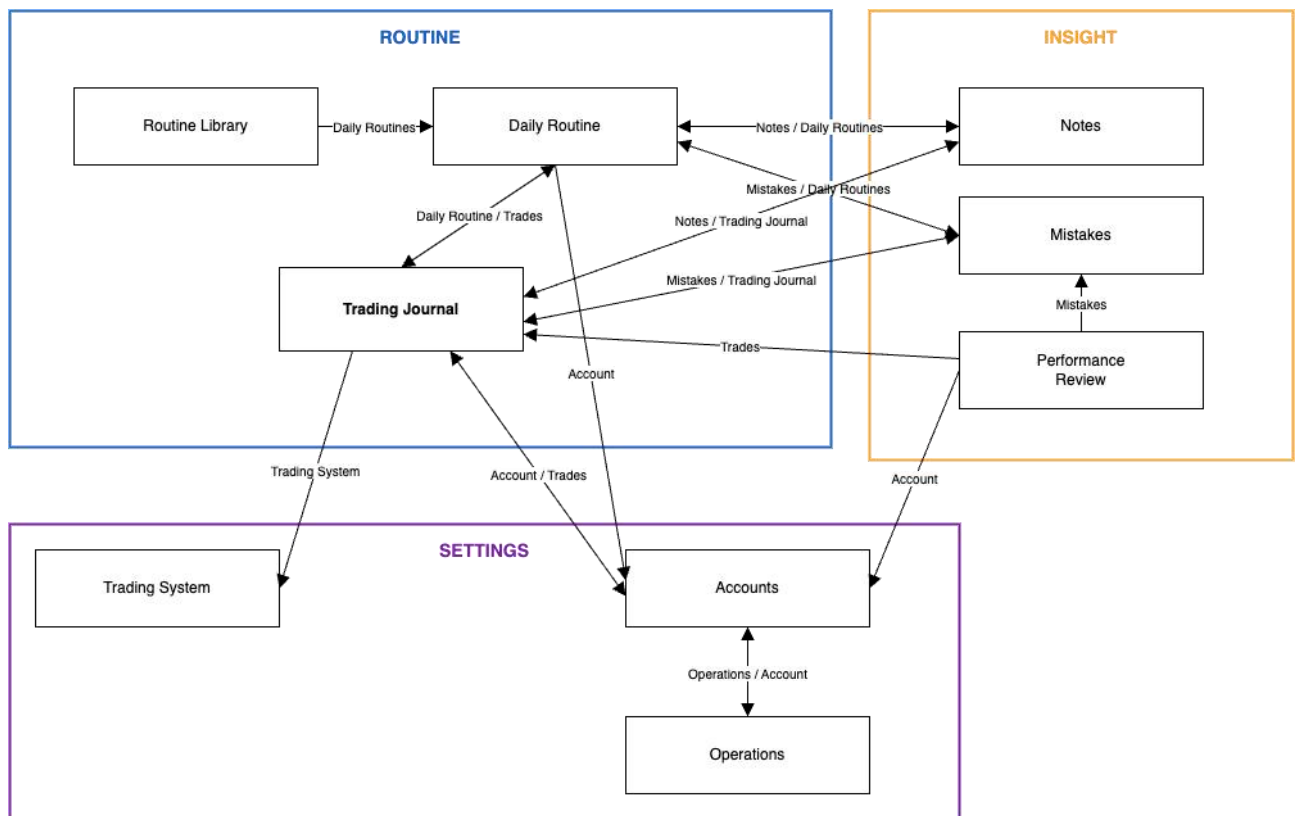


Рис. 2.2. Діаграма зв'язків баз даних предметної області

В основі цієї гнучкості лежатиме чітка структура бази даних. Свідомо уникається надмірна складність, але водночас забезпечуватимемо надійність зв'язків між сутностями. Кожна таблиця виконуватиме свою конкретну роль: від зберігання даних користувачів і їхніх робочих просторів до детального логування кожної дії автоматизацій. Особливу увагу приділено таблицям, які разом формуватимуть динамічне ядро системи. Саме тут дані зберігатимуться у форматі JSONB, що дозволить додавати нові поля без зміни схеми бази даних, зберігаючи при цьому високу швидкість пошуку та фільтрації завдяки індексам PostgreSQL.

Модуль автентифікації та управління акаунтом базується на таблиці **User**, яка є центральною сутністю платформи. Вона містить базові дані користувача: електронну пошту, унікальне ім'я, хеш пароля, сформований алгоритмом bcrypt, а також іконку профілю. Саме цей запис виступає власником усіх інших даних у системі, включаючи робочі простори, налаштування та історію дій. Детальна структура цієї таблиці представлена в таблиці 2.18.

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		51

Таблиця 2.18

Поля таблиці User

Поле	Тип	Опис
id	String	Унікальний ідентифікатор користувача (UUID)
email	String	Унікальна електронна адреса для комунікації та входу
username	String	Унікальне ім'я користувача для відображення в системі
passwordHash	String	Криптографічний хеш пароля для безпечної автентифікації
icon	String	Посилання на аватар або іконку профілю
isVerified	Boolean	Прапорець підтвердження електронної пошти
createdAt	DateTime	Час реєстрації акаунта

Для безпечного керування сесіями використовується таблиця **RefreshToken**, де зберігаються хешовані токени оновлення з чітко визначеним терміном дії тридцять днів та можливістю миттєвого анулювання. Детальна структура цієї таблиці представлена в таблиці 2.19.

Таблиця 2.19

Поля таблиці RefreshToken

Поле	Тип	Опис
id	String	Унікальний ідентифікатор токена
userId	String	Зовнішній ключ, що пов'язує токен з користувачем
tokenHash	String	Хеш токена оновлення сесії для безпеки
expiresAt	DateTime	Час завершення терміну дії токена
createdAt	DateTime	Час створення токена
revokedAt	DateTime	Час примусового відкликання токена (вихід з системи)

Одноразові токени для підтвердження електронної пошти зберігаються у таблиці **EmailVerificationToken**, яка має обмежений час життя та фіксує момент використання. Детальна структура цієї таблиці представлена в таблиці 2.20.

Поля таблиці EmailVerificationToken

Поле	Тип	Опис
id	String	Унікальний ідентифікатор токена
userId	String	Зовнішній ключ до користувача
tokenHash	String	Хеш токена для підтвердження пошти
expiresAt	DateTime	Час завершення дії токена верифікації
usedAt	DateTime	Час успішного використання токена
createdAt	DateTime	Час генерації токена

Одноразові токени для скидання пароля зберігаються у таблиці **PasswordResetToken**, яка також має обмежений час життя та фіксує момент використання. Детальна структура цієї таблиці представлена в таблиці 2.21.

Таблиця 2.21

Поля таблиці PasswordResetToken

Поле	Тип	Опис
id	String	Унікальний ідентифікатор токена
userId	String	Зовнішній ключ до користувача
tokenHash	String	Хеш токена для скидання пароля
expiresAt	DateTime	Час завершення дії токена скидання
usedAt	DateTime	Час успішного використання для зміни пароля
createdAt	DateTime	Час генерації запиту на скидання

Інтеграція зі сторонніми провайдерами реалізована через таблицю **GoogleAccount**, яка пов'язує Google ID з локальним профілем користувача та зберігає необхідні метадані для OAuth-автентифікації. Детальна структура цієї таблиці представлена в таблиці 2.22.

Таблиця 2.22

Поля таблиці GoogleAccount

Поле	Тип	Опис
id	String	Унікальний ідентифікатор запису OAuth

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		
						53

userId	String	Зовнішній ключ до локального користувача
googleId	String	Унікальний ідентифікатор акаунта в екосистемі Google
email	String	Електронна адреса, прив'язана до Google акаунта
accessToken	String	Токен доступу для інтеграції з сервісами Google
refreshToken	String	Токен для оновлення сесії OAuth
tokenExpiresAt	DateTime	Час закінчення дії токена доступу

Для зберігання персональних налаштувань призначено таблицю **Settings**. Вона реалізує гнучкий механізм збереження параметрів інтерфейсу та поведінки системи у форматі ключ-значення, згрупованих за категоріями (профіль, безпека, вигляд тощо), що дозволяє легко розширювати конфігурацію без зміни схеми БД. Детальна структура цієї таблиці представлена в таблиці 2.23.

Таблиця 2.23

Поля таблиці Settings

Поле	Тип	Опис
id	String	Унікальний ідентифікатор налаштування
userId	String	Зовнішній ключ до користувача
key	String	Технічний ключ параметра (наприклад, 'theme' або 'timezone')
value	Json	Гнучке значення налаштування у форматі JSON
category	String	Група або категорія для логічного об'єднання налаштувань

Таблиця **Notification** відповідає за внутрішньосистемні сповіщення, які класифікуються за типами: інформаційні, помилки, автоматизації, інтеграції. Система автоматично обмежує кількість збережених сповіщень до п'ятдесяти на користувача, видаляючи найстаріші при перевищенні ліміту. Детальна структура цієї таблиці представлена в таблиці 2.24.

Таблиця 2.24

Поля таблиці Notification

Поле	Тип	Опис
id	String	Унікальний ідентифікатор сповіщення
userId	String	Зовнішній ключ до отримувача
type	NotificationType	Категорія події (INFO, ERROR, AUTOMATION, INTEGRATION)
text	String	Текстовий контент сповіщення для користувача
isRead	Boolean	Статус прочитання для фільтрації неактивних подій
link	String	Опціональне посилання для переходу до джерела події
createdAt	DateTime	Час генерації сповіщення

Інтеграція із зовнішніми брокерськими сервісами, такими як Binance реалізована через таблицю **IntegrationConnection**. Тут зберігаються зашифровані облікові дані, API-ключі, паролі інвестора, статус підключення та метадані синхронізації. Статуси підключень та типи сервісів визначаються відповідними enum-типами, що забезпечує сувору валідацію даних на рівні бази. Ця таблиця дозволяє системі автоматично імпортувати історію угод та синхронізувати баланси рахунків. Детальна структура цієї таблиці представлена в таблиці 2.25.

Таблиця 2.25

Поля таблиці IntegrationConnection

Поле	Тип	Опис
id	String	Унікальний ідентифікатор підключення
userId	String	Зовнішній ключ до власника інтеграції
spaceId	String	Опціональна прив'язка до конкретного робочого простору
service	IntegrationService	Тип зовнішнього сервісу (Binance, MetaTrader5)
name	String	Зрозуміле ім'я підключення для інтерфейсу
credentials	Json	Зашифровані ключі API та токени доступу
status	IntegrationStatus	Поточний стан синхронізації (ACTIVE, INACTIVE, ERROR)

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		55

syncInterval	Int	Інтервал між автоматичними циклами синхронізації (у хвиликах)
marketType	String	Тип ринку для специфічних налаштувань парсингу
externalAccountId	String	Ідентифікатор акаунта на стороні зовнішнього сервісу
lastSyncAt	DateTime	Час останнього успішного циклу обміну даними
lastSyncError	String	Текст помилки останнього невдалого циклу
consecutiveFailures	Int	Лічильник послідовних помилок для автоматичного відключення
createdAt	DateTime	Час створення інтеграції
updatedAt	DateTime	Час останньої модифікації налаштувань

Ядром організації даних користувача є модуль робочих просторів, що складається з таблиць Space, Section та Database. Таблиця **Space** представляє ізольований робочий простір, який має назву та іконку. Кожен користувач може мати до п'яти таких просторів. Детальна структура цієї таблиці представлена в таблиці 2.26.

Таблиця 2.26

Поля таблиці Space

Поле	Тип	Опис
id	String	Унікальний ідентифікатор простору
ownerId	String	Зовнішній ключ до власника простору
name	String	Назва робочого простору
icon	String	Візуальний символ простору
isDefault	Boolean	Прапорець простору за замовчуванням для нових записів
createdAt	DateTime	Час створення простору

Всередині простору дані групуються за допомогою таблиці **Section**, яка слугує навігаційною папкою у бічній панелі, не містячи власне даних, а лише

визначаючи порядок відображення баз. Детальна структура цієї таблиці представлена в таблиці 2.27.

Таблиця 2.27

Поля таблиці Section

Поле	Тип	Опис
id	String	Унікальний ідентифікатор розділу
spaceId	String	Зовнішній ключ до простору, що містить розділ
name	String	Назва розділу (наприклад, 'Аналітика', 'Рутини')
key	String	Технічний ключ для програмного доступу до розділу
position	Int	Порядок сортування розділів у бічній панелі
icon	String	Іконка розділу для швидкої візуальної ідентифікації
color	String	Колірна схема для візуального групування
createdAt	DateTime	Час створення розділу
updatedAt	DateTime	Час останнього оновлення метаданих

Ключовою сутністю є **Database**. Це структурована колекція записів із власним набором властивостей. Вона має конфігураційне поле для додаткових налаштувань. Детальна структура цієї таблиці представлена в таблиці 2.28.

Таблиця 2.28

Поля таблиці Database

Поле	Тип	Опис
id	String	Унікальний ідентифікатор бази даних
spaceId	String	Зовнішній ключ до простору, якому належить база
sectionId	String	Опціональна прив'язка до розділу для структурування
name	String	Технічна назва або ключ бази даних
title	String	Відображувана назва бази даних в інтерфейсі
type	String	Тип сутності (наприклад, 'trading-journal', 'mistakes')
key	String	Унікальний системний ключ для ідентифікації шаблонів
icon	String	Іконка бази даних

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		57

position	Int	Порядок сортування баз даних всередині розділу
isKey	Boolean	Позначка системної бази даних, яку не можна видалити
isLocked	Boolean	Захист від випадкового видалення або зміни структури
createdAt	DateTime	Час створення бази даних
updatedAt	DateTime	Час останньої зміни конфігурації

Гнучкість структури даних забезпечується таблицями PropertyGroup та Property. Таблиця **Property** визначає колонки бази даних, їхні типи, позицію та специфічну конфігурацію у форматі JSONB, яка відрізняється залежно від типу властивості. Наприклад, для типу ВИБІР тут зберігається список доступних опцій, а для ЧИСЛА формат відображення валюти чи відсотків. Детальна структура цієї таблиці представлена в таблиці 2.29.

Таблиця 2.29

Поля таблиці Property

Поле	Тип	Опис
id	String	Унікальний ідентифікатор властивості (колонки)
databaseId	String	Зовнішній ключ до бази даних
groupId	String	Опціональна прив'язка до групи властивостей
name	String	Відображувана назва властивості
type	PropertyType	Тип даних (TEXT, NUMBER, SELECT, FORMULA тощо)
position	Int	Порядок сортування колонок у таблиці
icon	String	Іконка властивості
hint	String	Підказка або опис для користувача
group	String	Додаткова категоризація для внутрішньої логіки
isVisible	Boolean	Прапорець видимості колонки за замовчуванням
isProtected	Boolean	Захист системних властивостей від видалення
integrationKey	String	Ключ для мапінгу зовнішніх даних (наприклад, 'entryPrice')
createdAt	DateTime	Час створення властивості

updatedAt	DateTime	Час модифікації налаштувань властивості
config	Json	Специфічні налаштування (опції SELECT, формат NUMBER)
visibilityCondition	Json	Логіка умовного відображення властивості

Таблиця **PropertyGroup** дозволяє логічно об'єднувати властивості в картці запису та налаштовувати умови їхньої видимості. Детальна структура цієї таблиці представлена в таблиці 2.30.

Таблиця 2.30

Поля таблиці

Поле	Тип	Опис
id	String	Унікальний ідентифікатор групи властивостей
databaseId	String	Зовнішній ключ до бази даних
name	String	Назва групи (наприклад, 'Ризик-менеджмент', 'Психологія')
position	Int	Порядок відображення групи у формі запису
visibility	Json	Умови відображення групи залежно від стану запису
createdAt	DateTime	Час створення групи
updatedAt	DateTime	Час оновлення налаштувань групи

Основні дані користувача зберігаються у таблицях Record, PropertyValue, RecordContent та RecordContentSnapshot. Таблиця **Record** представляє окремий рядок у базі даних, наприклад одну угоду або нотатку, і містить метайнформацію, таку як назва, іконка та посилання на шаблон. Детальна структура цієї таблиці представлена в таблиці 2.31.

Таблиця 2.31

Поля таблиці Record

Поле	Тип	Опис
id	String	Унікальний ідентифікатор запису (рядка)
databaseId	String	Зовнішній ключ до бази даних, якій належить запис

templateId	String	Опціональна прив'язка до шаблону, з якого створено запис
sourceIntegrationId	String	Ключ інтеграції, якщо запис імпортовано автоматично
sourceLabel	String	Мітка джерела для відстеження походження даних
sourcePositionId	String	Ідентифікатор позиції у зовнішній системі (наприклад, ID ордера)
sourceCurrency	String	Валюта, у якій запис імпортовано ззовні
name	String	Назва запису (заголовок рядка)
icon	String	Іконка або емодзі для візуального маркування
deletedAt	DateTime	Час м'якого видалення (для відновлення)
createdAt	DateTime	Час створення запису
updatedAt	DateTime	Час останньої модифікації метаданих

Безпосередньо значення комірок зберігаються у таблиці **PropertyValue**, де кожен запис є унікальною парою «ідентифікатор запису – ідентифікатор властивості», а саме значення зберігається у форматі JSONB. Детальна структура цієї таблиці представлена в таблиці 2.32.

Таблиця 2.32

Поля таблиці PropertyValue

Поле	Тип	Опис
id	String	Унікальний ідентифікатор значення
recordId	String	Зовнішній ключ до запису
propertyId	String	Зовнішній ключ до властивості
value	Json	Універсальне сховище даних будь-якого типу у форматі JSON
computed	Boolean	Прапорець, що вказує на обчислюване (формульне) значення

Розширений текстовий контент та медіафайли зберігаються у **RecordContent** у вигляді ієрархічного дерева. Детальна структура цієї таблиці представлена в таблиці 2.33.

Таблиця 2.33

Поля таблиці RecordContent

Поле	Тип	Опис
id	String	Унікальний ідентифікатор контенту
recordId	String	Зовнішній ключ до запису (один до одного)
content	Json	Тіло запису (текст, блоки, зображення) у форматі JSON
lastEditedAt	DateTime	Час останнього редагування контенту

Історія змін цього контенту фіксується у **RecordContentSnapshot** для можливості відновлення попередніх версій. Детальна структура цієї таблиці представлена в таблиці 2.34.

Таблиця 2.34

Поля таблиці RecordContentSnapshot

Поле	Тип	Опис
id	String	Унікальний ідентифікатор знімка
recordContentId	String	Зовнішній ключ до основного контенту запису
content	Json	Повна копія тіла запису на момент створення знімка
createdAt	DateTime	Час створення резервної копії

Для прискорення введення даних система використовує механізм шаблонів, реалізований через таблиці **Template** та **TemplatePropertyValue**. Таблиця **Template** зберігає загальну структуру шаблону, включаючи патерн для автоматичної генерації назви та початковий стан контентної області. Детальна структура цієї таблиці представлена в таблиці 2.35.

Таблиця 2.35

Поля таблиці Template

Поле	Тип	Опис
id	String	Унікальний ідентифікатор шаблону
databaseId	String	Зовнішній ключ до бази даних, для якої створено шаблон

name	String	Назва шаблону
description	String	Опис призначення шаблону
icon	String	Іконка шаблону
namePattern	String	Патерн для автоматичної генерації назви нового запису
content	Json	Структура тіла запису за замовчуванням
isDefault	Boolean	Прапорець шаблону, що застосовується автоматично
position	Int	Порядок сортування шаблонів у списку вибору
createdAt	DateTime	Час створення шаблону
updatedAt	DateTime	Час останнього оновлення

Конкретні попередньо заповнені значення властивостей для цього шаблону зберігаються у `TemplatePropertyValue`, що дозволяє створювати складні пресети для різних типів угод або нотаток. Детальна структура цієї таблиці представлена в таблиці 2.36.

Таблиця 2.36

Поля таблиці `TemplatePropertyValue`

Поле	Тип	Опис
id	String	Унікальний ідентифікатор значення шаблону
templateId	String	Зовнішній ключ до шаблону
propertyId	String	Зовнішній ключ до властивості
value	Json	Значення за замовчуванням для цієї властивості при створенні

Налаштування відображення даних та автоматизація процесів реалізуються через таблиці `View`, `Automation` та `AutomationLog`. Таблиця **View** зберігає конфігурацію подання бази даних: фільтри з логікою AND/OR, сортування, групування та видимість колонок. Детальна структура цієї таблиці представлена в таблиці 2.37.

Поля таблиці View

Поле	Тип	Опис
id	String	Унікальний ідентифікатор подання (вигляду таблиці)
databaseId	String	Зовнішній ключ до бази даних
name	String	Назва подання
icon	String	Іконка подання
position	Int	Порядок сортування подань
isDefault	Boolean	Прапорець подання за замовчуванням
isLocked	Boolean	Захист системного подання від видалення
pageSize	Int	Кількість записів на одній сторінці
recordLimit	Int	Максимальна загальна кількість записів для відображення
useDefaultTemplate	Boolean	Використовувати стандартний шаблон при створенні звідси
defaultTemplateId	String	ID конкретного шаблону для цього подання
filters	Json	Масив умов фільтрації записів
filterLogic	FilterLogic	Логіка поєднання фільтрів (AND або OR)
sort	Json	Правила сортування колонок
groupBy	String	Ключ властивості для групування рядків
hiddenColumns	String[]	Масив ID колонок, прихованих у цьому поданні
columnWidths	Json	Налаштування ширини колонок для збереження UI
textWrap	Boolean	Перенесення тексту всередині комірок
searchQuery	String	Збережений пошуковий запит
config	Json	Специфічні налаштування відображення (канбан, галерея)
createdAt	DateTime	Час створення подання
updatedAt	DateTime	Час оновлення конфігурації подання

Таблиця **Automation** описує правила автоматизації, що складаються з тригера (створення запису, зміна поля або розклад), умов виконання та масиву дій. Детальна структура цієї таблиці представлена в таблиці 2.38.

Таблиця 2.38

Поля таблиці Automation

Поле	Тип	Опис
id	String	Унікальний ідентифікатор автоматизації
databaseId	String	Зовнішній ключ до бази даних, де спрацьовує тригер
name	String	Назва сценарію автоматизації
trigger	AutomationTrigger	Подія-ініціатор (створення, зміна поля, розклад)
actions	Json	Масив дій, що виконуються (зміна полів, створення записів)
active	Boolean	Прапорець активації/деактивації сценарію
createdAt	DateTime	Час створення автоматизації
updatedAt	DateTime	Час останньої зміни логіки
config	Json	Додаткові параметри виконання

Журнал виконання цих правил фіксується у **AutomationLog**, де зберігається статус успішності, пов'язаний запис та текст помилок у разі невдачі. Детальна структура цієї таблиці представлена в таблиці 2.39.

Таблиця 2.39

Поля таблиці AutomationLog

Поле	Тип	Опис
id	String	Унікальний ідентифікатор логу виконання
automationId	String	Зовнішній ключ до сценарію автоматизації
sourceRecordId	String	ID запису, який ініціював спрацювання
status	AutomationStatus	Результат виконання (SUCCESS, FAILURE, SKIPPED)
result	String	Детальний опис результату або текст помилки
createdAt	DateTime	Час виконання сценарію

Для роботи з імпортом даних із зовнішніх файлів використовуються таблиці ImportMapping та ImportHistory. **ImportMapping** зберігає правила зіставлення стовпців CSV-файлу з властивостями бази даних у форматі JSONB,

що дозволяє користувачеві не налаштовувати маппінг щоразу заново. Детальна структура цієї таблиці представлена в таблиці 2.40.

Таблиця 2.40

Поля таблиці ImportMapping

Поле	Тип	Опис
id	String	Унікальний ідентифікатор конфігурації імпорту
databaseId	String	Зовнішній ключ до цільової бази даних
name	String	Назва профілю імпорту
sourceType	String	Тип вихідного файлу (CSV)
mappingRules	Json	Правила зіставлення колонок файлу з властивостями бази
createdAt	DateTime	Час створення профілю
updatedAt	DateTime	Час останньої зміни правил мапінгу

ImportHistory веде облік всіх операцій імпорту, фіксуючи їхній статус (в обробці, завершено, помилка), кількість оброблених записів та логи помилок. Детальна структура цієї таблиці представлена в таблиці 2.41.

Таблиця 2.41

Поля таблиці ImportHistory

Поле	Тип	Опис
id	String	Унікальний ідентифікатор запису в історії імпорту
importMappingId	String	Зовнішній ключ до використаного профілю імпорту
status	ImportStatus	Стан обробки (PENDING, PROCESSING, COMPLETED, FAILED)
recordsCreated	Int	Кількість успішно створених записів
recordsFailed	Int	Кількість записів, які не вдалося імпортувати
errorLog	Json	Детальний лог помилок для діагностики
sourceFileInfo	Json	Метадані вихідного файлу (розмір, назва, хеш)
createdAt	DateTime	Час початку імпорту
completedAt	DateTime	Час завершення обробки

Аутентифікація

- User**
 - id: String «PK»
 - email: String «UK»
 - username: String «UK»
 - passwordHash: String
 - icon: String
 - isVerified: Boolean
 - createdAt: DateTime
- RefreshToken**
 - id: String «PK»
 - userId: String «FK»
 - tokenHash: String
 - expiresAt: DateTime
 - revokedAt: DateTime?
 - createdAt: DateTime
- EmailVerificationToken**
 - id: String «PK»
 - userId: String «FK»
 - tokenHash: String
 - expiresAt: DateTime
 - usedAt: DateTime?
 - createdAt: DateTime
- PasswordResetToken**
 - id: String «PK»
 - userId: String «FK»
 - tokenHash: String
 - expiresAt: DateTime
 - usedAt: DateTime?
 - createdAt: DateTime
- GoogleAccount**
 - id: String «PK»
 - userId: String «FK»
 - googleId: String «UK»
 - email: String
 - accessToken: String
 - refreshToken: String
 - tokenExpiresAt: DateTime?

Дані сервісу

- Settings**
 - id: String «PK»
 - userId: String «FK»
 - key: String
 - value: Json
 - category: String
 - «UK: userId key»
- Notification**
 - id: String «PK»
 - userId: String «FK»
 - type: NotificationType
 - text: String
 - isRead: Boolean
 - link: String?
 - createdAt: DateTime
- IntegrationConnection**
 - id: String «PK»
 - userId: String «FK»
 - spaceId: String «FK»
 - service: IntegrationService
 - name: String
 - credentials: Json
 - status: IntegrationStatus
 - isManual: Boolean
 - marketType: String
 - externalAccountId: String?
 - lastSyncAt: DateTime?
 - lastSyncError: String?
 - connectionFailures: Int
 - createdAt: DateTime
 - updatedAt: DateTime

Простір

- Space**
 - id: String «PK»
 - spaceId: String «FK»
 - name: String
 - icon: String?
 - isDefault: Boolean
 - position: Int
 - key: String?
 - color: String?
 - createdAt: DateTime
 - updatedAt: DateTime
 - «UK: ownerId, name»
- Section**
 - id: String «PK»
 - spaceId: String «FK»
 - name: String
 - key: String?
 - position: Int
 - color: String?
 - createdAt: DateTime
 - updatedAt: DateTime
 - «UK: spaceId, name»

База даних

- Database**
 - id: String «PK»
 - spaceId: String «FK»
 - sectionId: String «FK»
 - name: String
 - title: String
 - type: String?
 - key: String?
 - icon: String?
 - position: Int
 - isKey: Boolean
 - isLocked: Boolean
 - createdAt: DateTime
 - updatedAt: DateTime
 - «UK: spaceId, name»
- Record**
 - id: String «PK»
 - databaseId: String «FK»
 - templatedId: String «FK»
 - sourceLabel: String
 - sourcePattern: String?
 - sourceCurrency: String?
 - name: String
 - icon: String?
 - createdAt: DateTime
 - detectedAt: DateTime?
 - updatedAt: DateTime
- PropertyGroup**
 - id: String «PK»
 - databaseId: String «FK»
 - name: String
 - position: Int
 - visibility: Boolean?
 - createdAt: DateTime
 - updatedAt: DateTime
- Property**
 - id: String «PK»
 - groupId: String «FK»
 - name: String
 - type: PropertyType
 - position: Int
 - icon: String?
 - hint: String?
 - isVisible: Boolean
 - isProtected: Boolean
 - integrationKey: String?
 - config: Json?
 - visibilityCondition: Json?
 - createdAt: DateTime
 - updatedAt: DateTime
- RecordContent**
 - id: String «PK»
 - recordId: String «FK»
 - content: Json
 - lastifiedAt: DateTime
- PropertyValue**
 - id: String «PK»
 - recordId: String «FK»
 - propertyId: String «FK»
 - value: Json?
 - computed: Boolean
 - «UK: recordId, propertyId»

Шаблони

- Template**
 - id: String «PK»
 - databaseId: String «FK»
 - name: String
 - description: String «FK»
 - icon: String?
 - namePattern: String?
 - content: Json
 - isDefault: Boolean
 - position: Int
 - hiddenColumns: String?
 - groupBy: String?
 - columnFields: Json?
 - textWrap: Boolean
 - searchQuery: String?
 - config: Json?
 - createdAt: DateTime
 - updatedAt: DateTime
 - «UK: databaseId, name»
- TemplatePropertyValue**
 - id: String «PK»
 - templatedId: String «FK»
 - propertyId: String «FK»
 - value: Json?
 - «UK: templatedId, propertyId»

Перегляди та автоматизації

- View**
 - id: String «PK»
 - databaseId: String «FK»
 - name: String
 - icon: String?
 - isDefault: Boolean
 - position: Int
 - pageSize: Int
 - recordLimit: Int?
 - useDefaultTemplate: Boolean
 - defaultTemplateId: String?
 - filters: Json?
 - filterLogic: FilterLogic
 - sort: Json?
 - createdAt: DateTime
 - updatedAt: DateTime
 - «UK: databaseId, name»
- Automation**
 - id: String «PK»
 - databaseId: String «FK»
 - name: String
 - trigger: AutomationTrigger
 - condition: Json?
 - actions: Json
 - isActive: Boolean
 - config: Json
 - createdAt: DateTime
 - updatedAt: DateTime
- AutomationLog**
 - id: String «PK»
 - automationId: String «FK»
 - sourceRecordId: String?
 - status: AutomationStatus
 - result: String?
 - createdAt: DateTime
 - updatedAt: DateTime

Імпорт

- ImportMapping**
 - id: String «PK»
 - status: ImportStatus
 - sourceType: String
 - mapProperties: Json
 - createdAt: DateTime
 - updatedAt: DateTime
- ImportHistory**
 - id: String «PK»
 - importMappingId: String «FK»
 - recordCreated: Int
 - recordFailed: Int
 - errors: Json?
 - sourceFields: Json
 - createdAt: DateTime
 - completedAt: DateTime?

Регіональна

- RecordContentSnapshot**
 - id: String «PK»
 - recordContentId: String «FK»
 - content: Json
 - createdAt: DateTime

2.3 Вибір архітектурного підходу та обґрунтування інструментальних засобів

		Тушак В.С.			ІІЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чиждотря О.В.				66
Змн.	Арк.	№ докум.	Підпис	Дата		

обрана архітектура дозволяє вносити зміни передбачувано, не ризикуючи зламати те, що вже працює [26].

На найвищому рівні система побудована за клієнт-серверною моделлю. Її ключова ідея полягає не просто у фізичному розділенні двох частин, а у чіткому розмежуванні відповідальності між ними. Клієнт відповідає за відображення даних і взаємодію з користувачем, сервер у свою чергу за бізнес-логіку, обробку запитів і доступ до бази даних. Між ними існує єдина точка взаємодії у вигляді API, через яку проходить уся комунікація. Такий поділ дає практичну перевагу: інтерфейс можна повністю переробити без жодних змін у серверній логіці, а сервер може еволюціонувати незалежно від способу відображення інформації. Для даної системи ця модель є єдиним обґрунтованим вибором на верхньому рівні, оскільки вона забезпечує необхідну ізоляцію між шарами та природно відображає реальний розподіл між браузером і сервером.

Клієнтська частина побудована за компонентною архітектурою. На відміну від класичного MVC, де логіка, дані і відображення розділені між трьома окремими шарами, компонентний підхід групує їх навколо конкретних частин інтерфейсу, де кожен компонент відповідає за свій фрагмент відображення і містить лише ту логіку, яка безпосередньо до нього відноситься. Компоненти організовані за доменами системи: окремі групи для роботи з базами даних, записами, властивостями, навігацією та налаштуваннями.

Спільний стан застосунку управляється через контекст, що забезпечує доступ до поточного простору, активної бази даних і даних користувача з будь-якої точки інтерфейсу. Логіка взаємодії з API виокремлена в окремий шар, що дозволяє повторно використовувати її в різних компонентах і тримати самі компоненти чистими від деталей реалізації запитів.

Всередині серверної частини постає окреме питання: “Як організувати саму логіку?”. Тут найчастіше розглядають монолітну та мікросервісну архітектуру як два протилежні підходи [14; 15].

Монолітна архітектура розміщує всю логіку в межах одного застосунку. Всі модулі системи існують в одному процесі, звертаються одне до одного

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				67
Змн.	Арк.	№ докум.	Підпис	Дата		

напрямую і розгортаються разом. Відповідно, немає потреби у міжсервісній взаємодії, немає мережевих затримок між компонентами, немає складної інфраструктури оркестрації. Налаштування і тестування також значно простіші, оскільки вся поведінка системи знаходиться в одному місці. Моноліт добре справляється з задачами обмеженого масштабу, де функціональність зосереджена навколо єдиного домену і не потребує незалежного масштабування окремих частин.

Мікросервісна архітектура будується на протилежному принципі: система декомпозується на незалежні сервіси, кожен із яких відповідає за вузьку функціональність, має власну базу даних і може розгортатися та масштабуватися окремо. Ця модель добре проявляє себе у великих продуктах з високим навантаженням і складною доменною логікою, де різні частини системи зростають нерівномірно і командам потрібна технологічна незалежність. Проте разом із цим мікросервіси несуть значну операційну складність: з'являються потреби у міжсервісній взаємодії через мережу, механізмах синхронізації даних, трасуванні запитів між сервісами та інфраструктурі для їх розгортання і моніторингу. Навіть проста операція може вимагати послідовних викликів до кількох сервісів, що ускладнює як розробку, так і пошук помилок.

Для даної системи обрано монолітну архітектуру, і це рішення продиктоване конкретними характеристиками задачі, а не звичкою чи зручністю. Система орієнтована на одного типу користувача, а саме на індивідуального трейдера, відповідно, і вся її логіка зосереджена навколо єдиного домену: просторів, баз даних, записів і властивостей. Тут немає підстав для незалежного масштабування окремих компонентів. Мікросервісна архітектура в таких умовах не вирішує жодної реальної проблеми, але створює цілий набір нових: складну інфраструктуру, мережеву взаємодію між сервісами та значно вищий поріг входу в розробку.

Натомість монолітний підхід дозволяє зосередитися на логіці самого продукту. Варто уточнити, що обраний моноліт організований за принципом

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		68

модульного моноліту: кожен домен системи такі як простори, бази даних, записи, властивості, шаблони – виокремлений у самостійний модуль із власними контролером, сервісом, провайдерами і репозиторієм. Модулі взаємодіють одне з одним через чіткі інтерфейси, а не через прямі залежності між реалізаціями. Така організація дає переваги у розгортанні та налагодженні, і не дозволяє системі перетворитися на заплутану кодову базу зі зростанням функціональності.

Таким чином, архітектурне рішення відповідає масштабу задачі і залишає простір для контрольованого зростання системи без необхідності її повного переосмислення [14; 17].

При розробці вебплатформи одним із ключових рішень є вибір набору інструментів, фреймворків і мов програмування, на яких будується система. Система постійно працює зі складними структурованими даними, які передаються туди й назад між клієнтом та сервером. Тому найважливіше тут наскільки моделі даних узгоджені між собою і наскільки передбачувано поводить ся система. Технології не можна обирати наосліп або просто тому, що вони популярні. Вибір має виходити з самої задачі.

Для обох частин системи, тобто для клієнта і сервера, варто взяти одну мову програмування. Дані постійно передаються: записи, їхні властивості, значення. Одна й та сама модель використовується з обох боків. Якщо сервер віддає одне, а клієнт чекає іншого, виникають помилки.

Можна піти різними шляхами. Наприклад, узяти для сервера C# або Java, а для клієнта JavaScript. Сьогодні для цього вже є готові інструменти на кшталт генерації клієнтського коду з OpenAPI-специфікації, які знімають проблему ручної синхронізації типів, або GraphQL із власною системою типів. Але тоді моделі даних усе одно доведеться описувати двічі, хай навіть автоматизовано.

Інший шлях це використати чистий JavaScript на обох сторонах. Це прибирає дублювання, але не дає жодних гарантій щодо типів: що сервер віддав, те клієнт і прийняв, навіть якщо структура не та. TypeScript розв'язує обидві проблеми. Мова залишається одна, а компілятор перевіряє типи ще до запуску

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				69
Змн.	Арк.	№ докум.	Підпис	Дата		

[22]. Серверна частина системи займається бізнес-логікою, яка сама собою розпадається на окремі домени: користувачі, простори, бази даних, записи та властивості. Кожен з цих елементів має свою відповідальність і внутрішню будову, але при цьому вони взаємодіють один з одним. Крім самої логіки, потрібно забезпечити базові інфраструктурні речі: автентифікацію, контроль доступу, обмеження запитів і централізовану обробку помилок.

Тут є різні варіанти. Фреймворки на зразок Express або Fastify, що не нав'язують жодної структури і дають повну свободу, але всю відповідальність за архітектуру перекладають на розробника. У підсумку багато часу йде не на реалізацію функціоналу, а на те, щоб підтримувати код узгодженим. Інший варіант: взяти важкі платформи як Spring Boot або ASP.NET, які одразу задають жорстку архітектуру, але тоді розривається єдиний стек з клієнтською частиною.

Тому доцільно використати NestJS. Він зберігає екосистему TypeScript, але водночас пропонує модульну структуру, механізм ін'єкції залежностей і готові інструменти для вирішення інфраструктурних задач. Це дозволяє зосередитися на логіці доменів, не витрачаючи сили на побудову базового каркасу [16].

Клієнтська частина системи працює з інтерфейсом, який має глибоку ієрархію: простір, база даних, запис, внутрішній вміст запису. Кожен рівень цієї структури має свій стан і повинен відображатися в маршруті. Тому потрібен інструмент, який вміє природно працювати з вкладеною маршрутизацією.

Один з варіантів: зв'язка React з Vite і React Router. Це дає високу гнучкість і швидку збірку, але всю систему маршрутів треба налаштовувати вручну. Інший підхід: використати Vue або Nuxt, які пропонують файлову маршрутизацію і дозволяють швидко почати розробку без купи налаштувань. Але при детальнішому розгляді важлива не лише швидкість старту, а й контроль над структурою проєкту, коли він стає складнішим.

У цьому сенсі Next.js виглядає збалансованіше. Він поєднує гнучкість React з вбудованою файловою маршрутизацією. Це дозволяє природно

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				70
Змн.	Арк.	№ докум.	Підпис	Дата		

відобразити складну ієрархію інтерфейсу без зайвого коду. Саме те, що він одночасно дає контроль і зменшує кількість допоміжних рішень, і стало причиною вибору [25].

Після того як визначилися з клієнтом і сервером, постає питання, як вони будуть спілкуватися. На цьому рівні система може використовувати різні підходи до організації API. Найпоширеніші це REST та GraphQL.

GraphQL дозволяє клієнту самому вирішувати, які дані йому потрібні, що зручно, коли багато різних запитів і складна агрегація даних. Але це вимагає додаткового шару контролю складності запитів і складнішої логіки кешування. Тобто серверна частина стає важчою.

REST натомість будується на чітко визначених шляхах і стандартних HTTP методах. Це робить його більш передбачуваним і простим у реалізації. Для системи з чіткою доменною моделлю і невеликою кількістю сценаріїв взаємодії така передбачуваність є плюсом. Вона дозволяє зменшити складність і уникнути зайвих абстракцій. Тому для цього проєкту краще обрати REST, як більш простий і достатній варіант [19].

Оформлення інтерфейсу теж потребує окремого підходу. Система складається з багатьох дрібних компонентів. Кожен компонент має власні стилі, і важливо, щоб вони не конфліктували між собою. Класичні глобальні стилі погано масштабуються, бо зміна в одному місці може поламати інше.

CSS Modules частково вирішують цю проблему, але вимагають тримати структуру і стилі в різних файлах. CSS in JS рішення, наприклад styled components, вбудовують стилі прямо в код, але додають зайвий рівень абстракції. Tailwind CSS пропонує інший підхід: використовувати утилітарні класи прямо в розмітці. Це скорочує відстань між логікою і зовнішнім виглядом. У підсумку код компонента залишається компактным і самодостатнім. Саме це і стало головною причиною вибрати Tailwind.

Щодо зберігання даних, система має справу з чіткою структурою і великою кількістю зв'язків між сутностями. Зазвичай запити отримують пов'язані дані за один раз. Тому критично важливо підтримувати реляційну

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				71
Змн.	Арк.	№ докум.	Підпис	Дата		

модель і цілісність даних. Документо-орієнтовані бази на кшталт MongoDB краще підходять для гнучких або нечітких даних. Але в цьому випадку вони призвели б до дублювання і ускладнили б запити.

Серед реляційних СУБД можна дивитися на різні варіанти, наприклад MySQL. Однак PostgreSQL дає більше можливостей для роботи зі складними запитами і типами даних. Саме тому він краще підходить для задач, де структура даних відома заздалегідь і активно використовуються зв'язки між сутностями [23; 24].

Робота з базою даних на рівні коду теж потребує окремого рішення. Пряме використання SQL дає максимальний контроль, але ускладнює підтримку і збільшує ризик помилок. ORM рішення на зразок TypeORM або Sequelize трохи спрощують життя, але можуть мати проблеми з типізацією або бути складними в налаштуванні. Prisma виступає компромісом. Він дозволяє генерувати типобезпечний клієнт на основі схеми бази даних і автоматизує міграції. Це забезпечує узгодженість між моделлю даних і кодом і зменшує ручну роботу.

Окремо варто сказати про валідацію даних. На сервері вона реалізована через class-validator. Це дозволяє описувати правила перевірки прямо в структурах даних і автоматично застосовувати їх при обробці запитів. На клієнті краще використовувати інструменти, які перевіряють дані ще до відправки на сервер. У цьому випадку Zod підходить добре, бо він дає типобезпечну валідацію і сумісний із TypeScript.

У клієнтській частині системи доводиться мати справу з двома різними видами стану. Перший це локальний стан інтерфейсу, наприклад вибрані елементи або те, яка сторінка зараз відкрита. Другий це серверний стан, тобто дані, які приходять через API і можуть змінюватися незалежно від клієнта. Головне завдання зробити так, щоб дані були узгодженими, запити не дублювалися, а оновлення стану після змін не вимагало купи ручної роботи.

Простий варіант з React Context API. Він дозволяє зберігати дані в одному місці, але не вміє кешувати, інвалідувати або синхронізуватися з сервером.

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				72
Змн.	Арк.	№ докум.	Підпис	Дата		

Інша можливість це глобальні менеджери стану, як Redux або Zustand. Вони добре працюють з локальним станом, але теж не вирішують проблеми серверних даних. Тому окремо було розглянуто спеціалізований інструмент для серверного стану, а саме TanStack Query. Він дає готові механізми для кешування, повторного використання запитів і автоматичного оновлення. У підсумку обрано комбінацію: TanStack Query для серверного стану, а Context API для локального. Так відповідальність розділяється, і жоден інструмент не доводиться завантажувати тим, для чого він не призначений.

Системі потрібен механізм автентифікації, який безпечно пускає користувачів, підтримує сесії і дозволяє контролювати, чи дійсні вони ще. Додатково треба мати змогу відкликати доступ і керувати тим, як довго живуть токени.

Розглядалося кілька підходів. Класичний варіант із серверними сесіями, коли стан користувача зберігається на сервері. Це спрощує контроль, але ускладнює масштабування. Інший варіант взяти JWT і зберігати токени в localStorage. Це просто в реалізації, але створює ризики для безпеки, бо токен стає доступним для клієнтського JavaScript коду. Також є готові рішення, наприклад NextAuth.js, які автоматизують багато процесів, але обмежують гнучкість і контроль над внутрішньою логікою.

У підсумку обрано підхід з JWT у поєднанні з HTTP only куками. Access token використовується для авторизації запитів, а refresh token зберігається в базі даних у вигляді хешу. Відповідно так можна поєднати безпеку (токени недоступні для клієнтського коду) і контроль (можна відкликати сесії та обмежувати їх) [20].

Організація кодової бази теж впливає на зручність розробки. У системі варто використовувати монорепозіторій. Це дозволяє тримати клієнт, сервер і спільні пакети в одному місці. Так простіше перевикористовувати код і забезпечувати узгодженість моделей. Альтернативи: розділити на окремі репозіторії або взяти складніший інструмент як Nx. Але перший варіант ускладнює взаємодію між частинами, а другий потребує багато налаштувань.

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		73

Turborepo разом з npm дозволяє організувати збірку так, щоб не дублювати залежності і прискорити розробку.

Розгортання системи це окремий етап, який напряду впливає на доступність, стабільність і подальшу підтримку. Оскільки система має клієнтську і серверну частини, їх краще розгортати окремо, щоб оновлення і масштабування були незалежними. Клієнтська частина на Next.js орієнтована на платформу Vercel. Причина проста: тісна інтеграція з фреймворком, що надає автоматичне збирання, оптимізацію статички і підтримку серверних функцій.

Для розміщення бази даних та API-сервера використовуватиметься платформа Railway. Цей вибір обумовлений її здатністю автоматизувати процеси деплою та масштабування, що дозволить зосередитися на розробці функціоналу, а не на налаштуванні серверів. Railway забезпечуватиме стабільне з'єднання між контейнером додатка та керованою базою даних PostgreSQL, автоматично керуючи пулом з'єднань та резервним копіюванням.

2.4 Проєктування алгоритмів функціонування системи

Функціональна повнота вебплатформи не обмежується лише можливістю зберігати дані. Її цінність для користувача полягає в здатності системи самостійно обробляти інформацію та перетворювати прості записи на структуроване знання. На цьому етапі проєктування увага зміщується з архітектурних рішень на деталізацію внутрішньої логіки роботи ключових модулів, які забезпечують інтелектуальну складову платформи.

Розроблені алгоритми покликані вирішити три головні завдання. Це автоматизація аналітичних розрахунків без втручання користувача, гнучке управління складною ієрархією контенту та надійна інтеграція із зовнішніми джерелами даних. Кожен із цих процесів має чітко визначену послідовність дій, умови виконання та механізми обробки виняткових ситуацій. Нижче наведено детальний опис алгоритмів функціонування системи, які лежать в основі

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		74

реалізації динамічних властивостей, структуризації контенту та імпортих операцій.

Алгоритм обчислення динамічних є одним із ключових механізмів системи та забезпечує підтримку динамічних властивостей, реалізованих через поля типу FORMULA. Такі властивості автоматично перераховуються при зміні залежних даних і використовуються для розрахунку аналітичних показників, зокрема чистого прибутку, співвідношення ризику до прибутку та інших похідних метрик. Загальний процес функціонування цього механізму наведено на рисунку 2.4.

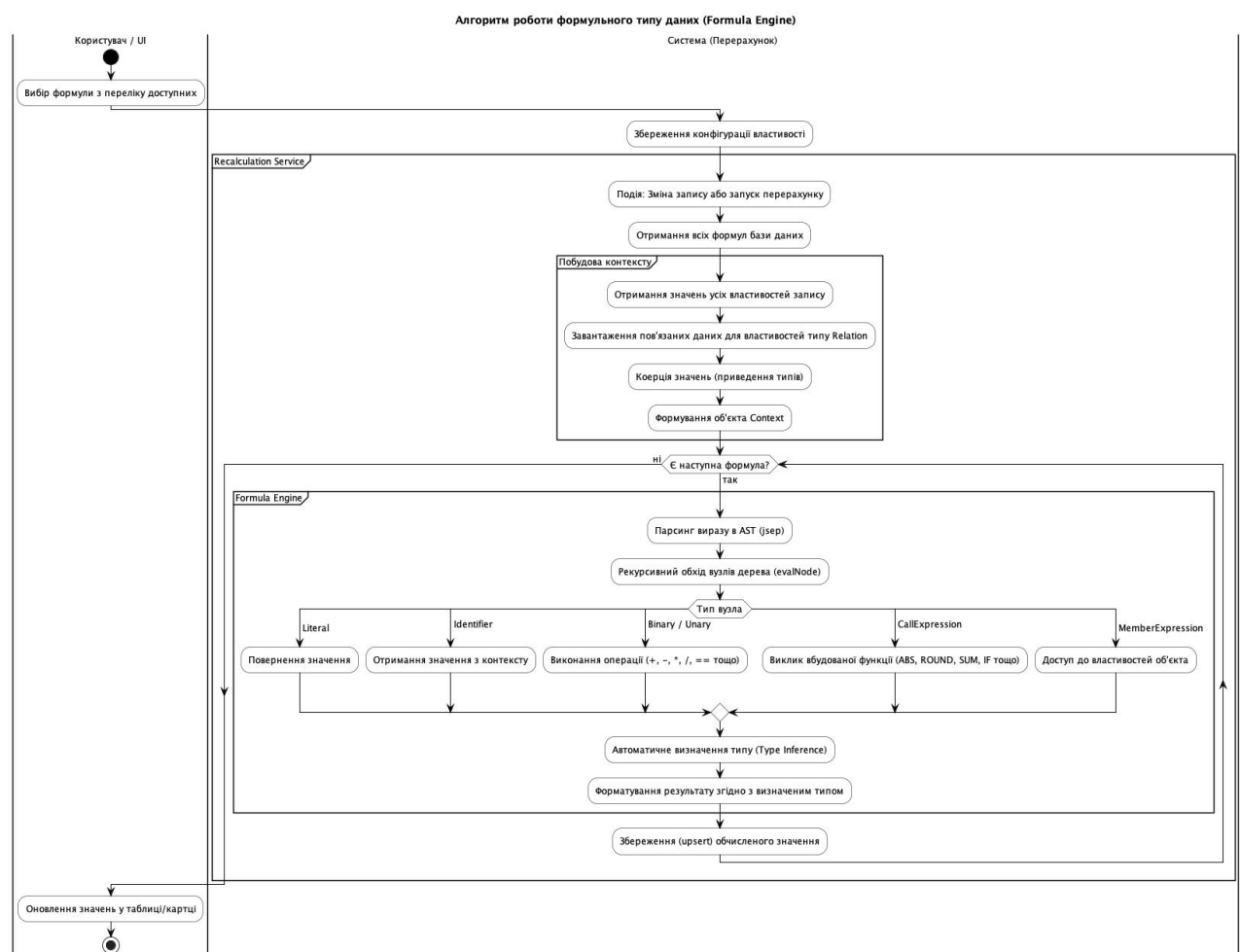


Рис. 2.4. Діаграма активності для алгоритму обчислювальних властивостей

На першому етапі система виконує перевірку коректності введеного виразу. Для цього використовується спеціальна бібліотека jsep, яка аналізує

текст формули та намагається побудувати його синтаксичну структуру. Якщо вираз містить помилки, наприклад, незбалансовані дужки або невідомі оператори, система негайно повідомляє про це користувача і не дозволяє зберегти налаштування. Це запобігає появі непрацюючих формул у базі даних. Якщо ж синтаксис є правильним, конфігурація властивості зберігається, і система переходить до фази обчислень.

Обчислення запускається реактивно: щоразу, коли змінюється будь-яке значення в записі, або коли користувач примусово ініціює перерахунок. Система спочатку збирає всі формули, які належать до поточної бази даних. Після цього формується контекст, тобто повний набір даних, необхідних для розрахунків. Сюди входять значення всіх інших полів цього запису. Якщо формула посилається на дані з пов'язаних записів, система додатково завантажує ці зовнішні дані. Усі отримані значення приводяться до потрібних типів: текстові рядки перетворюються на числа, а дати на стандартний формат, щоб уникнути помилок під час обчислень.

Далі вступає в дію ядро обчислень Formula Engine. Воно проходить по кожній формулі окремо. Текст формули знову перетворюється на абстрактне синтаксичне дерево, яке представляє собою набір пов'язаних між собою операцій [18]. Система обходить це дерево вузол за вузлом. Якщо вузол є просто числом або константою, вона повертається як є. Якщо це посилання на інше поле, система бере відповідне значення з підготовленого контексту.

Під час обходу дерева виконуються всі необхідні дії. Це можуть бути прості арифметичні операції, такі як додавання чи ділення, або більш складні логічні порівняння. Якщо у формулі використано спеціальні функції, наприклад, округлення, абсолютне значення або умовний оператор, система викликає відповідний вбудований алгоритм для їх обробки.

Після того як усі вузли дерева оброблені, система отримує фінальний результат. Вона автоматично визначає тип цього результату: чи є він числом, булевим значенням чи текстом. Отримане значення форматується відповідно до

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		76

налаштувань поля і зберігається в базі даних. Цей процес оновлення відбувається дуже швидко і прозоро для користувача.

На завершальному етапі інтерфейс системи миттєво оновлюється. Користувач бачить нові, актуальні значення у таблиці або картці запису.

Алгоритм управління ієрархічним контентом та знімками стану забезпечує структуроване зберігання текстової та медійної інформації у вигляді блочної моделі та підтримує механізм автоматичного версіонування змін. Це дозволяє зберігати історію редагувань і відновлювати попередні стани документа у випадку помилкових змін. Загальний процес роботи алгоритму наведено на рисунку 2.5.

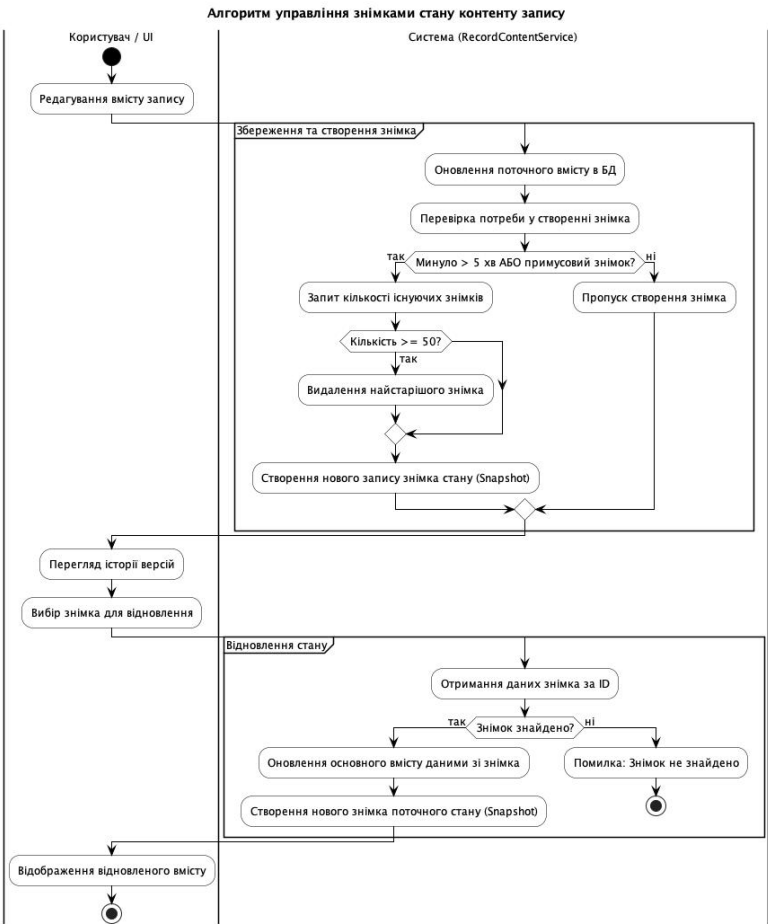


Рис. 2.5. Діаграма активності для алгоритму управління ієрархічним контентом та знімками стану

На першому етапі система фіксує новий стан контенту, оновлюючи основний запис у базі даних. Це гарантує, що актуальна інформація завжди

доступна для перегляду. Одночасно запускається механізм перевірки необхідності створення знімка стану. Система аналізує час, що минув з моменту останнього збереження версії. Якщо пройшло більше п'яти хвилин або якщо користувач спеціально ініціював примусове збереження, алгоритм переходить до наступного кроку. Якщо ж зміни відбуваються дуже часто, система пропускає створення нової копії, щоб уникнути засмічення сховища дублікатами.

Коли умова для створення знімка виконується, система підраховує кількість вже наявних версій для цього конкретного запису. Для підтримки оптимальної продуктивності та економії місця встановлено ліміт у 50 знімків на один запис. Якщо цей ліміт перевищено, алгоритм автоматично знаходить і видаляє найстаріший знімок, звільняючи місце для нового. Після цього створюється нова копія поточного стану контенту, яка зберігається в окремій таблиці з прив'язкою до часу створення. Такий підхід забезпечує постійну доступність історії змін без необхідності ручного очищення старих даних.

Окрім автоматичного збереження, алгоритм передбачає можливість ручного відновлення даних. Коли користувач вирішує повернутися до попередньої версії, він обирає потрібний знімок зі списку історії. Система отримує дані обраної версії за її унікальним ідентифікатором. Якщо знімок успішно знайдено, основний вміст запису повністю замінюється даними з цієї старої копії. Важливою особливістю є те, що після відновлення система миттєво створює новий знімок поточного стану. Це діє як додаткова страховка: якщо відновлення було помилковим, користувач зможе легко скасувати цю дію, повернувшись до стану, який був безпосередньо перед відновленням.

У разі, якщо обраний знімок не знайдено (наприклад, через технічну помилку або пошкодження даних), система повідомляє користувача про помилку і перериває процес відновлення, залишаючи поточний вміст без змін. Після успішного завершення всіх операцій інтерфейс платформи оновлюється, відображаючи відновлений текст або структуру блоків.

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		78

Алгоритм парсингу та мапінгу зовнішніх даних відповідає за трансформацію сирих торгових даних із форматів CSV або API-відповідей бірж у стандартизовані внутрішні об'єкти платформи, усуваючи потребу в ручному копіюванні параметрів угод. Його головне завдання це надійна інтеграція різнорідних джерел інформації шляхом валідації типів даних, конвертації форматів дат та чисел, а також зіставлення зовнішніх колонок із властивостями бази даних відповідно до налаштувань користувача. Запуск алгоритму відбувається за ініціативою трейдера під час імпорту файлів або за розкладом для синхронізації з брокером, забезпечуючи масове створення записів із детальною звітністю про успішні операції та виявлені помилки форматування (рис. 2.6).

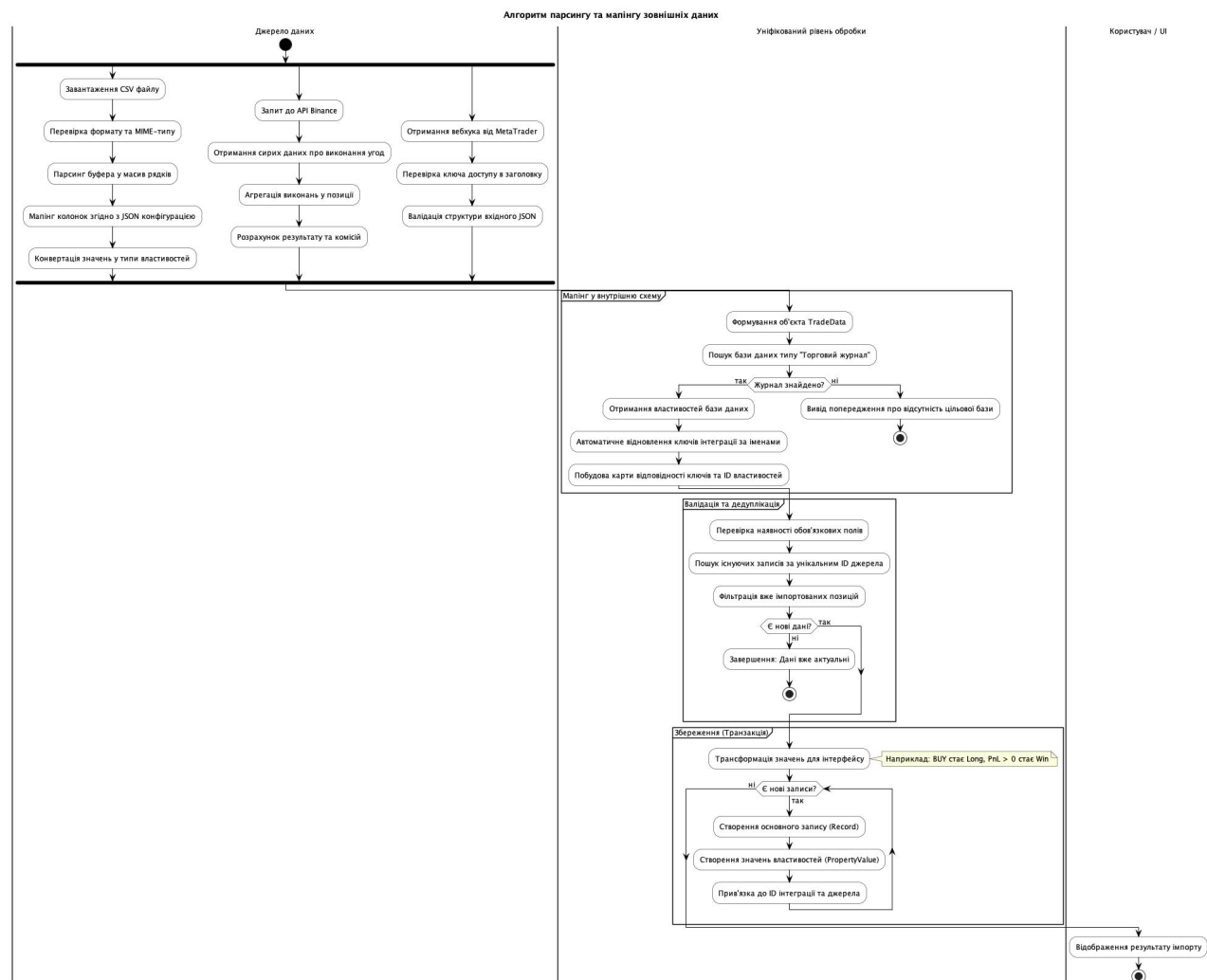


Рис. 2.6. Діаграма активності для алгоритму парсингу та мапінгу зовнішніх даних

На початковому етапі система одночасно обробляє дані з кількох можливих каналів надходження. Якщо користувач обирає файловий імпорт, алгоритм перевіряє формат та MIME-тип завантаженого CSV-файлу, після чого парсить його бінарний буфер у масив текстових рядків. У разі роботи з API, наприклад Binance, система виконує запити для отримання сирих даних про виконання ордерів, агрегує їх у цілісні позиції та розраховує фінальний фінансовий результат з урахуванням комісій. Також передбачена обробка вебхуків від терміналів типу MetaTrader, де спочатку перевіряється ключ доступу в заголовку запиту та валідується структура вхідного JSON-об'єкта. Такий паралельний підхід дозволяє платформі бути гнучкою та підтримувати різноманітні сценарії інтеграції.

Після отримання сирих даних вони передаються на уніфікований рівень обробки для мапінгу у внутрішню схему системи. Алгоритм формує стандартизований об'єкт даних та шукає в робочому просторі користувача базу даних типу Trading Journal. Якщо цільова база знайдена, система отримує список її властивостей та автоматично відновлює ключі інтеграції за їхніми назвами, будуючи карту відповідності між зовнішніми колонками чи полями API та внутрішніми ідентифікаторами властивостей. Якщо ж відповідної бази не виявлено, процес переривається з виведенням попередження, що запобігає втраті даних у невизначеному середовищі.

Наступним критичним кроком є валідація та дедуплікація отриманої інформації. Система перевіряє наявність усіх обов'язкових полів у сформованих об'єктах та здійснює пошук вже наявних записів за унікальним ідентифікатором джерела. Це дозволяє відфільтрувати вже імпортовані позиції та уникнути створення дублікатів у журналі. Якщо нових унікальних даних не виявлено, алгоритм завершує роботу, повідомляючи про те, що інформація вже актуальна.

За наявності нових валідних даних запускається процес транзакційного збереження. Перед записом значення трансформуються для зручності інтерфейсу: наприклад, напрямок угоди BUY перетворюється на мітку Long, а

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				80
Змн.	Арк.	№ докум.	Підпис	Дата		

додатне значення прибутку маркується як Win. Далі в циклі для кожної нової угоди створюється основний запис та пов'язані з ним значення властивостей, які прив'язуються до ID інтеграції та конкретного джерела даних. Використання транзакцій гарантує, що навіть у разі збою всі дані будуть збережені цілісно або не будуть збережені взагалі, захищаючи статистику від пошкодження.

На завершальному етапі інтерфейс користувача оновлюється, відображаючи результати успішного імпорту. Трейдер бачить нові записи у своєму журналі, які вже готові до аналізу та пов'язані з відповідною статистикою.

2.5 Реалізація ключових компонентів системи

Одним із найскладніших технічних завдань стала реалізація безпечного обчислювального ядра для динамічних властивостей. Сервіс FormulaEngine відповідає за інтерпретацію користувацьких формул у реальному часі. На відміну від стандартних підходів, що використовують функцію eval(), яка є вразливою до виконання шкідливого коду, у системі застосовано бібліотеку jsep [32]. Вона парсить текстовий вираз у абстрактне синтаксичне дерево AST, яке потім рекурсивно обходиться спеціальним інтерпретатором.

```
evaluate(config: FormulaPropertyConfig, context: Record<string, unknown>): unknown {
  try {
    const { expression } = config;
    if (!expression) return null;

    // Парсинг виразу в дерево AST для подальшого рекурсивного обходу
    const ast = jsep(expression);

    // Виклик внутрішнього рекурсивного методу для обчислення вузлів
    const result = this.evalNode(ast, context);

    // Технічне визначення типу результату для коректного мапінгу в JSONB
    const inferredType = this.inferType(result);

    return this.formatResult(result, inferredType);
  } catch (error) {
    this.logger.warn("Formula evaluation failed", { error: (error as Error).message });
  }
}
```

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				81
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    return null;
  }
}

```

Після отримання результату обчислень система автоматично визначає його тип та форматує відповідно до налаштувань поля (наприклад, перетворює число на валютний формат або дату на ISO-рядок). У разі виникнення помилок, наприклад, ділення на нуль, алгоритм повертає null, не перериваючи загальний процес оновлення запису, що забезпечує стабільність інтерфейсу.

Другим критичним компонентом є RecordContentService, який керує ієрархічним контентом записів. Оскільки дані зберігаються у форматі JSONB у PostgreSQL, важливим аспектом реалізації стало забезпечення транзакційної цілісності при одночасному оновленні основного вмісту та створенні його версій. Для оптимізації використання дискового простору впроваджено механізм ротації за принципом FIFO, коли кількість знімків перевищує ліміт у 50 одиниць, система автоматично видаляє найстаріший запис перед створенням нового.

```

async update(recordId: string, dto: UpdateRecordContentDto):
Promise<RecordContentResponseDto> {
  return await this.recordContentRepo.transaction(async (tx) => {
    // Атомарне оновлення контенту в PostgreSQL
    await this.recordContentRepo.update(recordId, { content: dto.content }, tx);

    const now = new Date();
    const lastSnapshot = await this.recordContentRepo.findLastSnapshot(recordContentId, tx);

    // Логіка створення знімка на основі часового інтервалу (5 хв)
    if (!lastSnapshot || now.getTime() - lastSnapshot.createdAt.getTime() >
this.SNAPSHOT_INTERVAL_MS) {
      const count = await this.recordContentRepo.countSnapshots(recordContentId, tx);

      // Технічне обмеження кількості записів у таблиці Snapshots
      if (count >= this.MAX_SNAPSHOTS) {
        await this.recordContentRepo.deleteOldestSnapshot(recordContentId, tx);
      }
      await this.recordContentRepo.createSnapshot(recordContentId, dto.content, tx);
    }
    return this.findByRecordId(recordId, tx);
  });
}

```

Такий підхід гарантує, що користувач завжди має доступ до історії змін без ризику перевантаження бази даних зайвими дублікатами великих JSON-об'єктів.

Для вирішення проблеми інформаційної фрагментації та усунення необхідності ручного введення даних реалізовано модуль інтеграції з торговими терміналами. Зокрема, для платформи MetaTrader 5 розроблено кастомний радник FixSpaceSync.mq5 на мові MQL5 [33]. Цей скрипт моніторить закриття позицій, формує структурований JSON-об'єкт з параметрами угоди (ціна входу/виходу, комісії, P&L) та надсилає його через захищений Webhook на API-ендпоінт платформи.

```
string tradeObj = StringFormat(
    "{ \"sourcePositionId\": \"%I64d\", \"symbol\": \"%s\", \"direction\": \"%s\", \"entryPrice\": %.5f, \"exitPrice\": %.5f, \"quantity\": %.2f, \"grossPnL\": %.5f, \"fees\": %.5f, \"netPnL\": %.5f, \"openTime\": \"%s\", \"closeTime\": \"%s\", \"currency\": \"%s\" }",
    positionId, StringEscape(symbol), direction, entryPrice, exitPrice, volume, grossPnL, fees, netPnL,
    FormatDate(openTime), FormatDate(closeTime), StringEscape(currency)
);

string payload = StringFormat("{ \"connectionId\": \"%s\", \"trades\": %s }",
    StringEscape(InpConnectionId), tradesJson);
SendWebhook(payload); // Виконує WebRequest із заголовком x-api-key
```

На стороні сервера реалізовано механізм дедуплікації: кожна отримана угода перевіряється на наявність унікального ідентифікатора джерела. Якщо такий запис вже існує в базі даних, імпорт пропускається, що запобігає появі дублікатів при повторній синхронізації.

Клієнтська частина системи побудована з урахуванням вимог до високої інтерактивності. Для управління серверним станом використано бібліотеку TanStack Query, яка реалізує патерн Optimistic Updates. Коли користувач змінює дані, інтерфейс миттєво оновлюється локально, не чекаючи відповіді від сервера. Успішне завершення мутації призводить до інвалідації відповідних ключів кешу, що запускає фонове оновлення даних та синхронізацію з актуальним станом бази даних.

```
export function useUpdateRecord(databaseId: string) {
```

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижемотря О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		83

```

const queryClient = useQueryClient();
return useMutation({
  mutationFn: ({ id, data }: { id: string; data: UpdateRecordDto }) => updateRecord(id, data),
  onSuccess: (updated) => {

    // Технічна інвалідація кешу для колекції та конкретного об'єкта
    queryClient.invalidateQueries({ queryKey: queryKeys.records.all(databaseId) });
    queryClient.setQueryData(queryKeys.records.detail(databaseId, updated.id), updated);
  },
});
}

```

Взаємодія всіх описаних компонентів візуалізована на діаграмі класів (рис. 2.7). Вона демонструє ієрархію залежностей та принципи інкапсуляції бізнес-логіки всередині модульної архітектури платформи.

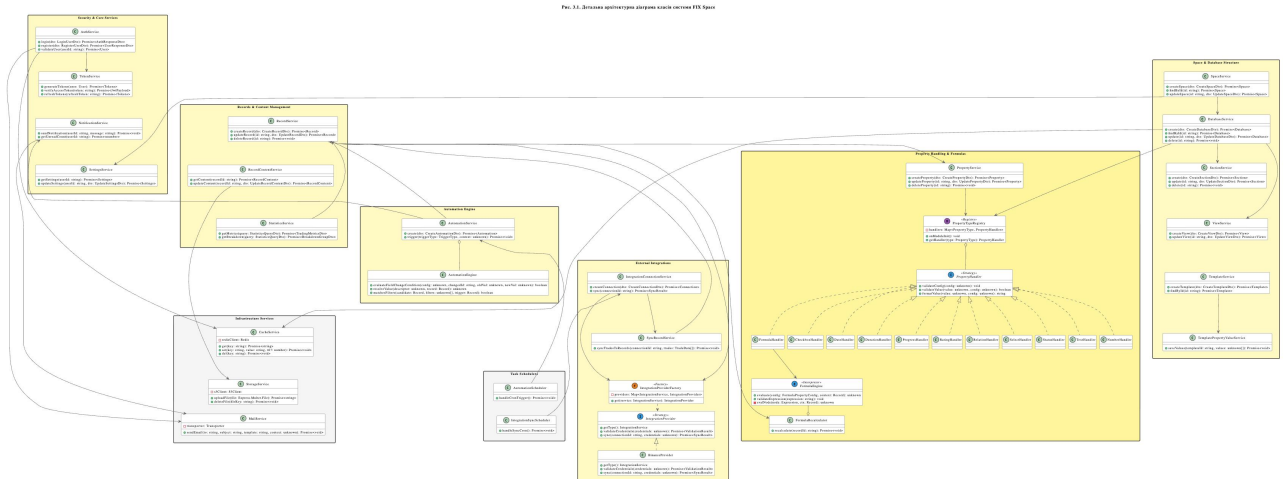


Рис. 2.7. Діаграма класів основних сервісів серверної частини

Висновки до другого розділу

У другому розділі було сформовано комплекс вимог, який базується на потребі усунення інформаційної фрагментації та мінімізації впливу когнітивних викривлень на процес прийняття торгових рішень. Визначено ключові бізнес-вимоги, спрямовані на створення єдиного структурованого середовища для накопичення досвіду, а також деталізовано функціональні можливості системи через призму варіантів використання для двох основних ролей: неавторизованого користувача та трейдера. Окремо обґрунтовано нефункціональні вимоги щодо продуктивності, надійності, безпеки даних та зручності інтерфейсу відповідно до стандарту ISO/IEC 25010.

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				84
Змн.	Арк.	№ докум.	Підпис	Дата		

Розроблено гібридну модель даних, яка поєднує суворі обмеження цілісності для системних сутностей із гнучкою структурою для користувацьких властивостей і контенту. Детально описано дев'ять предметних баз даних, які автоматично ініціалізуються при створенні простору та забезпечують всебічне охоплення торгових процесів. Визначено підходи до зберігання даних користувачів, налаштувань інтеграцій, шаблонів та історії змін, що дозволяє системі адаптуватися під індивідуальні потреби трейдера без зміни загальної архітектури.

Обґрунтовано вибір модульної монолітної архітектури як оптимального рішення для системи, орієнтованої на індивідуального користувача. Такий підхід дозволяє уникнути зайвої операційної складності при збереженні чіткого розмежування відповідальності між модулями. Технологічний стек забезпечує узгодженість між клієнтською та серверною частинами, мінімізує ризики помилок при передачі даних і створює передумови для швидкої розробки та подальшого масштабування продукту.

Спроектовано алгоритми функціонування ключових модулів, які забезпечують автоматизацію рутинних процесів і підтримують інтелектуальну складову системи. Описано логіку обчислення динамічних властивостей, механізм версіонування контентної області записів та алгоритм парсингу й мапінгу зовнішніх торгових звітів. Візуалізація цих процесів у вигляді діаграм активності демонструє послідовність дій, умови прийняття рішень та методи обробки виняткових ситуацій.

Реалізовано ключові компоненти системи, які разом формують її функціональну основу. Сервіс керування контентом записів забезпечує автоматичне версіонування без перевантаження сховища. Модуль інтеграції з торговими терміналами усуває необхідність ручного введення угод, а клієнтська частина реагує на дії користувача миттєво, не чекаючи підтвердження від сервера. Взаємодія цих компонентів підтверджує, що обрана архітектура здатна підтримувати повний цикл роботи трейдера в межах єдиної платформи.

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		85

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБПЛАТФОРМОЮ ДЛЯ ОРГАНІЗАЦІЇ ДІЯЛЬНОСТІ ПРИВАТНОГО ТРЕЙДЕРА

3.1 Розгортання та конфігурація середовища

Стабільність, безпека та передбачуваність функціонування значною мірою залежать від правильно організованої інфраструктури. У сучасній розробці програмного забезпечення критично важливо усунути розбіжності між середовищами розробки, тестування та експлуатації. Для вирішення цього завдання було обрано підхід, що поєднує контейнеризацію сервісів, автоматизацію процесів доставки коду (CI/CD) та використання спеціалізованих хмарних платформ для хостингу. Відповідно можна зменшити вплив людського фактора при налаштуванні серверів та гарантує ідентичну поведінку системи на будь-якому етапі її життєвого циклу.

Основою інфраструктурної частини є контейнеризація за допомогою технології Docker [27]. Серверна частина платформи, написана на NestJS, пакується у легковаговий образ, який містить лише необхідні для виконання бінарні файли та залежності Node.js. Для оптимізації розміру фінального образу та підвищення безпеки використано багатоетапне збирання. На перших етапах відбувається встановлення залежностей монорепозиторію, компіляція TypeScript-коду та видалення зайвих пакетів для розробки. Фінальний етап формує мінімальний образ на базі Alpine Linux, що суттєво зменшує площу потенційних атак та прискорює деплой.

Для локальної розробки та швидкого розгортання повного стеку технологій використовується оркестратор docker-compose. Конфігураційний файл docker-compose.yml описує взаємодію трьох ключових компонентів: API-сервера, реляційної бази даних PostgreSQL та кеш-сховища Redis. Особливу увагу приділено порядку запуску сервісів: контейнер з базою даних має механізм перевірки готовності, який періодично опитує порт СУБД. API-сервер налаштований так, щоб починати свою роботу лише після того, як база даних

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		86

повідомить про свою доступність, що запобігає помилкам підключення при старті системи.

services:

db:

image: postgres:16

environment:

POSTGRES_DB: fixspace

POSTGRES_USER: postgres

POSTGRES_PASSWORD: postgres

volumes:

- pgdata:/var/lib/postgresql/data

healthcheck:

test: ["CMD-SHELL", "pg_isready -U postgres -d fixspace"]

interval: 5s

timeout: 5s

retries: 5

api:

build:

context: .

dockerfile: apps/api/Dockerfile

environment:

- DATABASE_URL=postgresql://postgres:postgres@db:5432/fixspace

- JWT_SECRET=\${JWT_SECRET}

- REDIS_HOST=redis

depends_on:

db:

condition: service_healthy

redis:

condition: service_started

Важливим аспектом конфігурації є управління секретними даними. Паролі, ключі шифрування JWT та облікові дані сторонніх сервісів ніколи не зберігаються у вигляді простого тексту в коді. Вони винесені у файли змінних оточення .env, які додаються до списку ігнорування системи контролю версій Git. Відповідно, конфіденційна інформація не потрапить у публічний репозиторій. Під час розгортання на продакшн серверах ці змінні безпечно інжектуються через інтерфейс хостинг-провайдера або спеціалізовані менеджери секретів.

Фізична структура системи та логічні зв'язки між компонентами візуалізовані на діаграмі розгортання (рис. 3.1). Вона демонструє, як

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		87

клієнтський браузер взаємодіє з CDN та сервером додатків, а сервер, у свою чергу, звертається до ізольованих контейнерів бази даних та кешу.

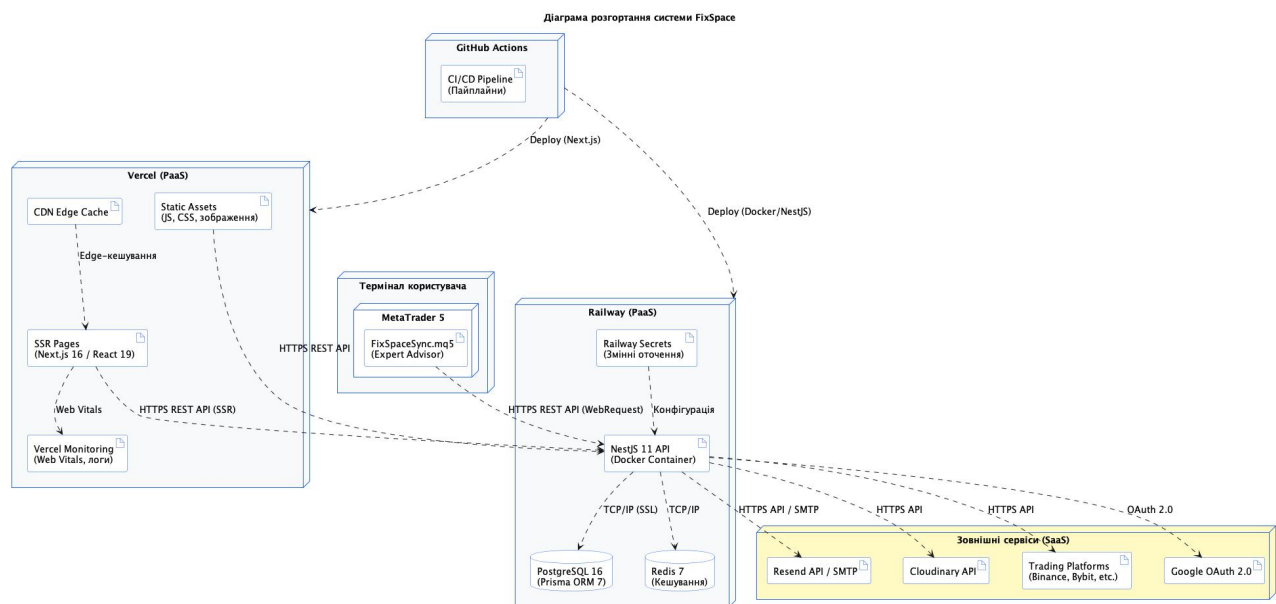


Рис. 3.1. Діаграма розгортання системи

Для забезпечення безперервної інтеграції та доставки (CI/CD) налаштовано автоматизовані пайплайни за допомогою GitHub Actions [28]. Цей підхід дозволяє автоматизувати рутинні процеси перевірки коду та його деплою. При кожному push-запиті або створенні Pull Request у гілки develop або main запускається послідовність дій. Спочатку виконується статичний аналіз коду для перевірки стилю та виявлення синтаксичних помилок. Далі запускається набір модульних тестів, які перевіряють коректність роботи окремих сервісів. Якщо ці етапи проходять успішно, система розгортає тимчасове середовище з тестовою базою даних для проведення інтеграційних тестів, які перевіряють взаємодію всіх компонентів разом.

Лише після успішного проходження всіх перевірок безпеки та функціональності пайплайн ініціює збирання нового Docker образу та його публікацію в реєстрі. Для гілки main налаштовано автоматичний деплой на продакшн.

Вибір платформ для хостингу зумовлений специфікою кожного компонента системи та потребою в оптимізації витрат та продуктивності.

Клієнтська частина, реалізована на Next.js, розгортається на платформі Vercel. Цей вибір є оптимальним, оскільки Vercel надає нативну підтримку фреймворку Next.js, автоматично налаштовує глобальну мережу доставки контенту для швидкого завантаження статичних ресурсів та забезпечує безшовну інтеграцію з GitHub.

Серверна частина та база даних PostgreSQL розміщуються на Railway. Ця платформа ідеально підходить для контейнеризованих додатків, оскільки вона автоматично керує масштабуванням ресурсів, моніторингом стану контейнерів та резервним копіюванням бази даних. Інтеграція між Vercel та Railway здійснюється через захищені API-запити, де клієнтська частина звертається до ендпоінтів сервера через HTTPS.

Загальний потік автоматизації, від написання коду до його появи у продуктивному середовищі, відображений на схемі пайплайну CI/CD. Вона наочно демонструє паралельність виконання тестів та умовні переходи між етапами збирання та деплою.

3.2 Порядок роботи з платформою

Знайомство з платформою розпочинається з лендінг сторінки. Саме тут користувач отримує первинне розуміння того, яким чином централізоване середовище здатне вирішити проблему хаотичного розподілу торгових даних (рис. 3.2).

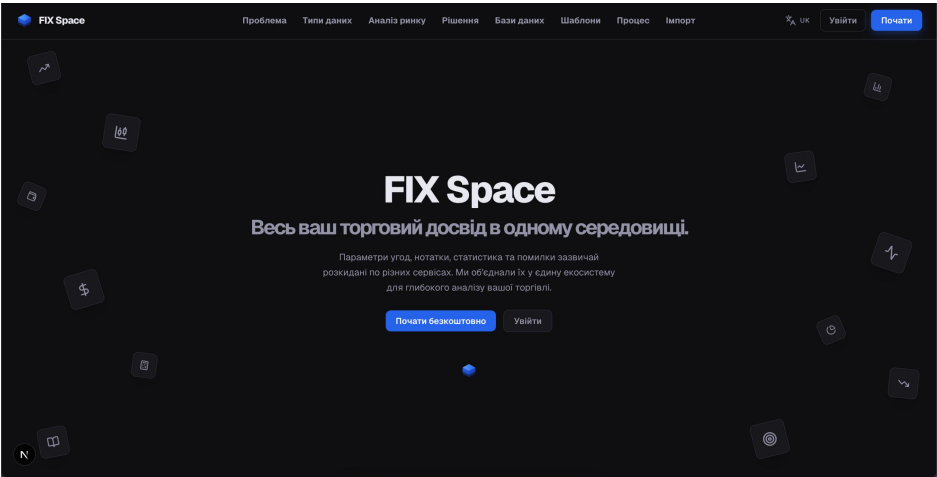


Рис. 3.2. Лендінг сторінка платформи

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				89
Змн.	Арк.	№ докум.	Підпис	Дата		

Процес реєстрації максимально спрощено для швидкого початку роботи. Для створення акаунта достатньо вказати лише ім'я, електронну пошту та пароль (рис. 3.3). Після цього автоматично формується особистий робочий простір із уже готовими базами даних. Завдяки цьому відпадає потреба у ручному налаштуванні, і забезпечується миттєвий перехід безпосередньо до справи. Передбачено також вхід через обліковий запис Google.

Рис. 3.3. Сторінка реєстрації

Для входу в систему передбачена стандартна сторінка авторизації, яка надійно захищає особисті дані та торговельну статистику (рис. 3.4). Доступ гарантується завдяки безпечним токенам, що забезпечує повну конфіденційність. Це вкрай важливо, оскільки в системі зберігаються чутливі фінансові показники та особисті нотатки.

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				90
Змн.	Арк.	№ докум.	Підпис	Дата		

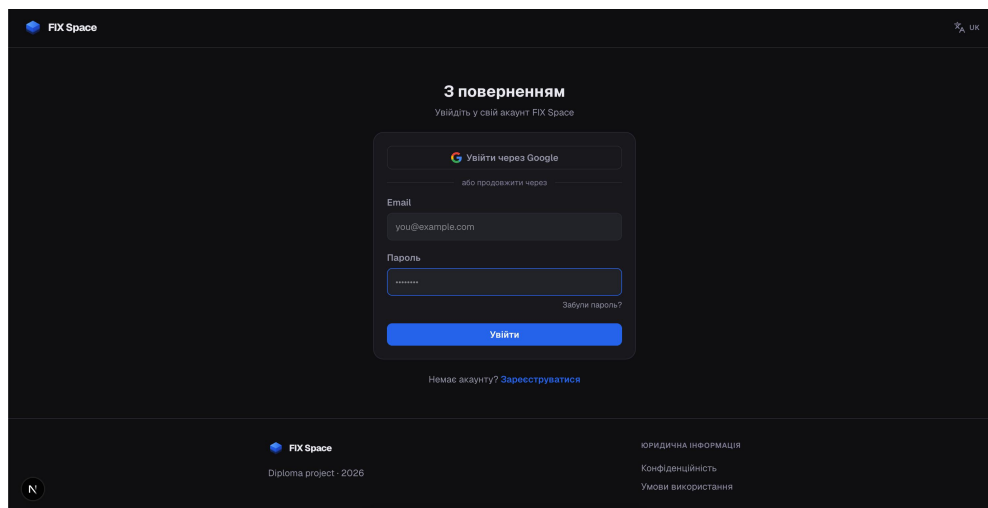


Рис. 3.4. Сторінка входу

Після успішної авторизації відкривається головна сторінка. На ній зібрана основна інформація про активність та забезпечений швидкий доступ до нещодавніх баз даних (рис. 3.5). Зліва розташоване меню навігації, яке дозволяє перемикатися між робочими просторами, створювати нові розділи або переходити до загальної аналітики.

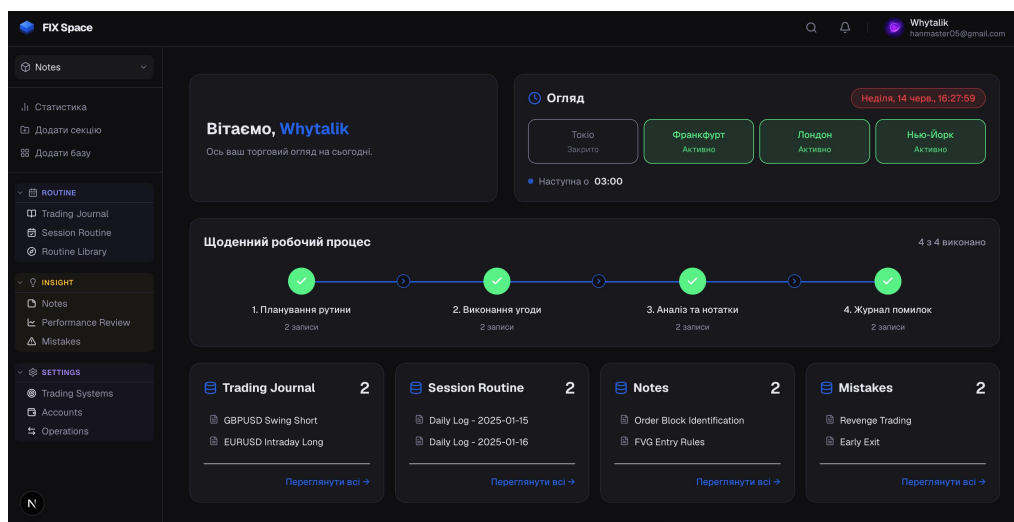


Рис. 3.5. Сторінка дашборду

Основою роботи є відкриття конкретної бази даних, наприклад, щоденних рутин (Routine Library). Вона відображається у вигляді звичайної таблиці, де кожен рядок це окремий запис, а стовпці відображають його властивості (рис.

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				91
Змн.	Арк.	№ докум.	Підпис	Дата		

3.6). Вбудовані фільтри та сортування допомагають швидко знайти потрібну інформацію, а новий запис створюється простим додаванням порожнього рядка.

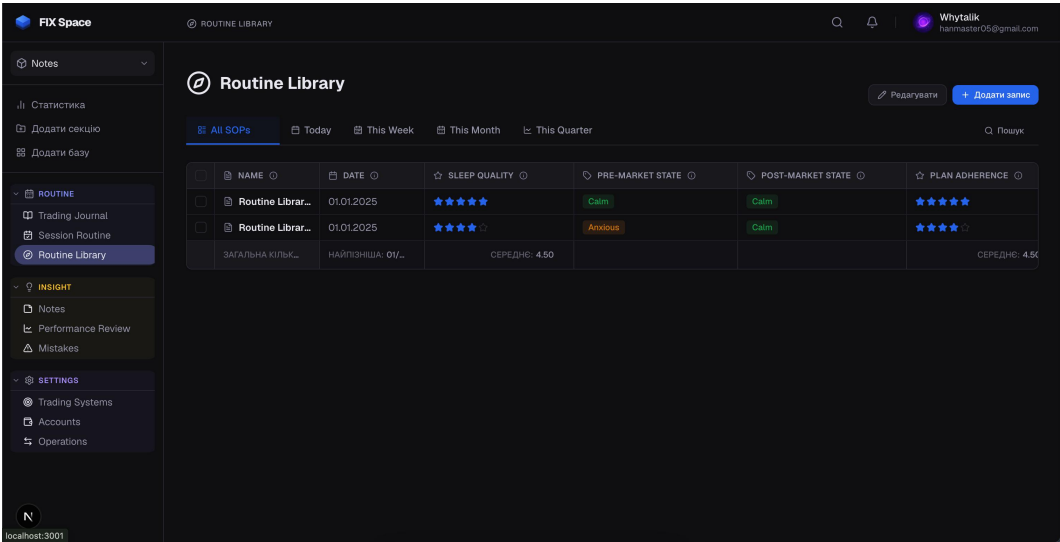


Рис. 3.6. Сторінка бази даних Routine Library

Детальний перегляд запису умовно поділений на дві частини. Верхня або бічна панель призначена для заповнення структурованих даних: ціни входу, статусу, дати або вибору стратегії (рис. 3.7).

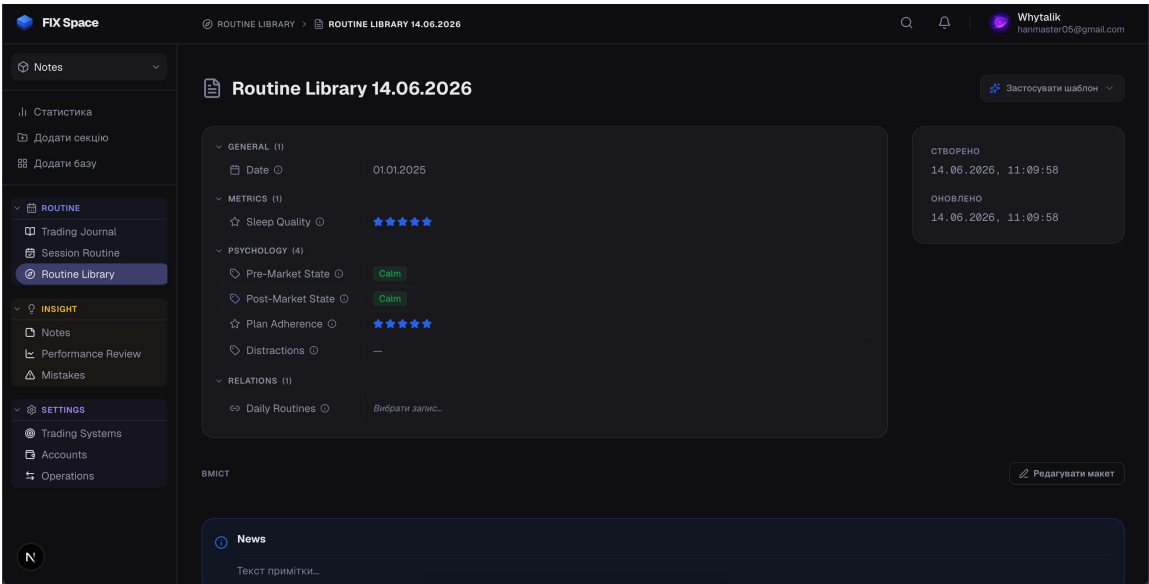


Рис. 3.7. Сторінка запису (властивості)

Нижче розташований зручний текстовий редактор. У цьому блоці передбачена можливість писати розгорнуті висновки, додавати скріншоти графіків або описувати що потрібно під час угоди (рис. 3.8).

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				92
Змн.	Арк.	№ докум.	Підпис	Дата		

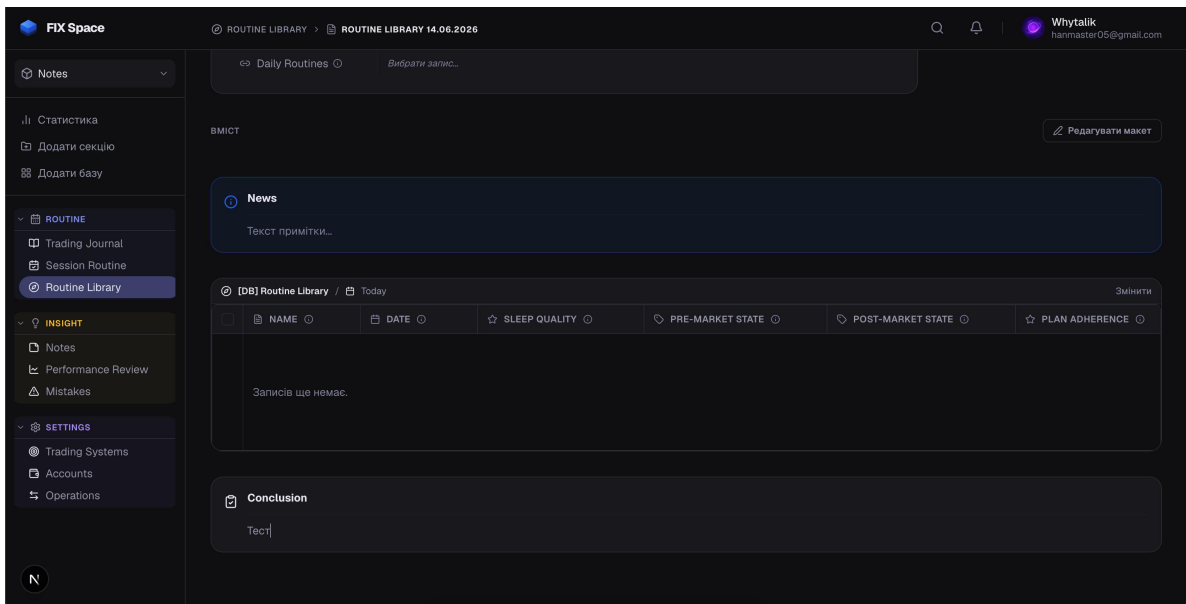


Рис. 3.8. Сторінка запису (вміст)

Перехід на сторінку глобальної статистики трансформує сухі цифри з розрізнених баз даних на цілісну візуальну картину: крива доходності, розподіл прибуткових та збиткових сесій, а також ключові метрики, такі як математичне очікування чи фактор прибутку (рис. 3.9).

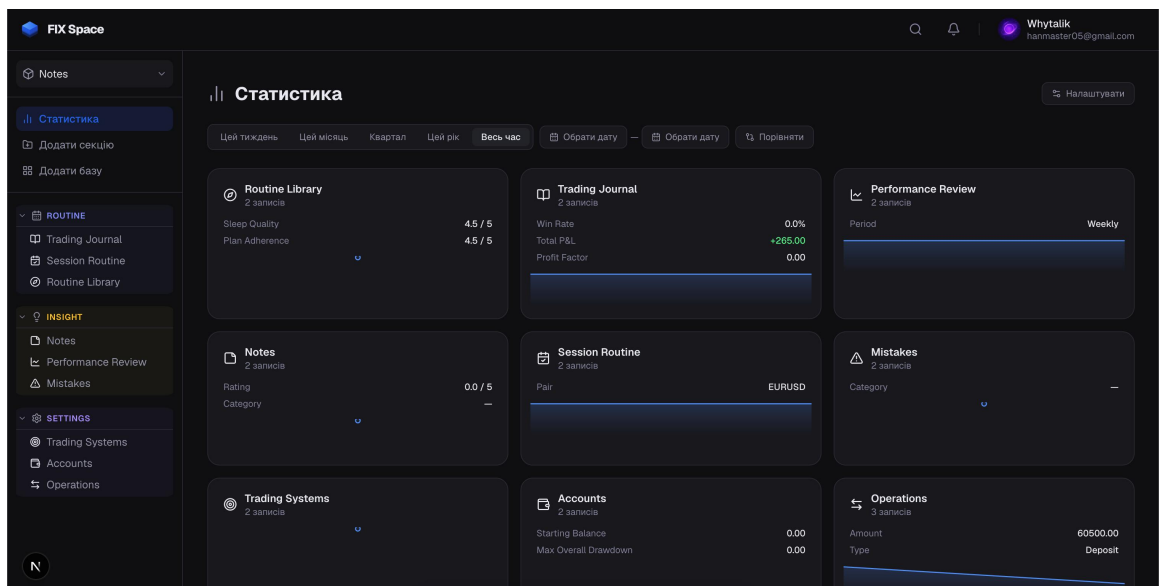


Рис. 3.9. Сторінка статистики

Для глибшого занурення система дозволяє ізолювати аналітику в межах окремої бази даних або специфічної категорії, що дає змогу об'єктивно оцінити

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				93
Змн.	Арк.	№ докум.	Підпис	Дата		

ефективність конкретної торговельної системи чи стратегії, нейтралізуючи тим самим упередження самовпевненості або надмірного песимізму (рис. 3.10).

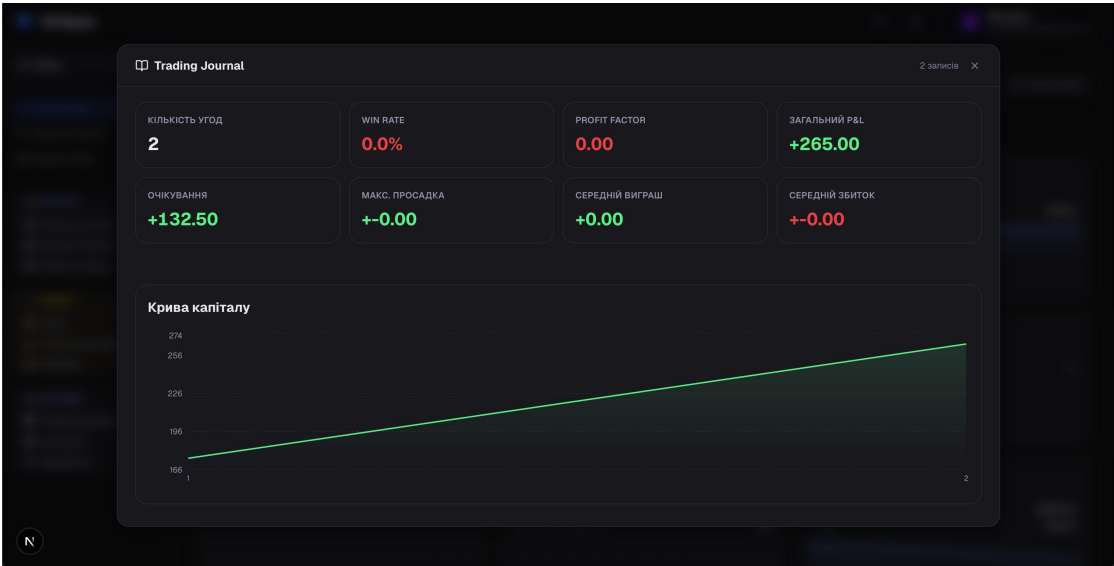


Рис. 3.10. Статистика по конкретній базі даних

Платформа дозволяє легко змінювати структуру даних під власні потреби. У налаштуваннях бази даних здійснюється зміна її назви або зовнішнього вигляду, а в розділі властивостей додаються нові поля для даних, включаючи зв'язки з іншими модулями (рис. 3.11).

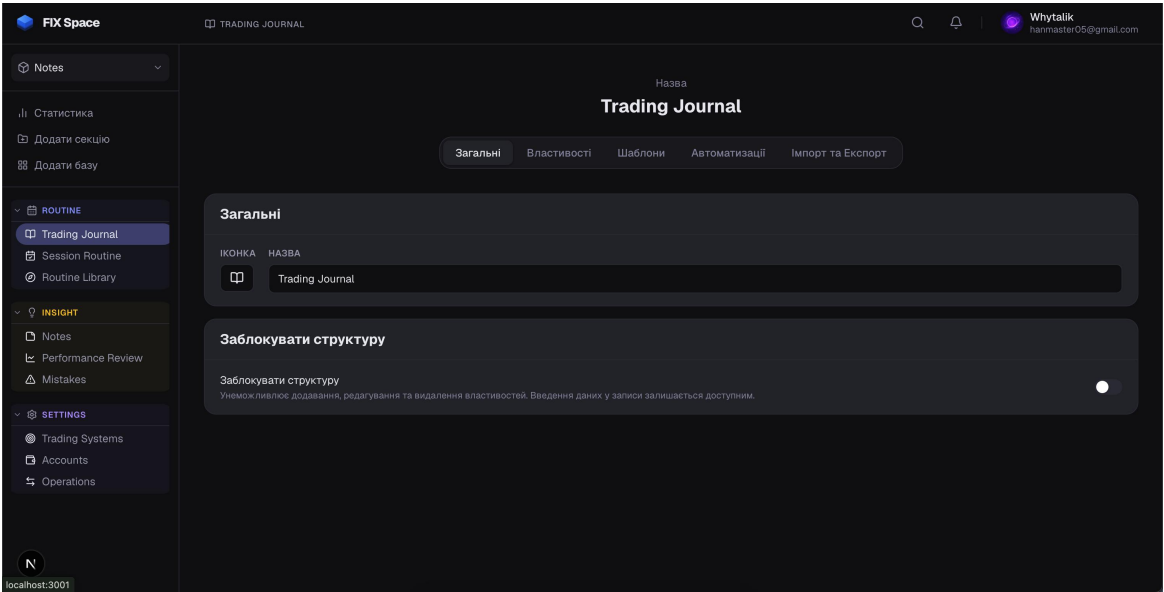


Рис. 3.11. Сторінка редагування бази даних

Процес створення нової властивості винесено в окреме модальне вікно, що дозволяє максимально детально конфігурувати її поведінку, формат відображення та умови видимості (рис. 3.12).

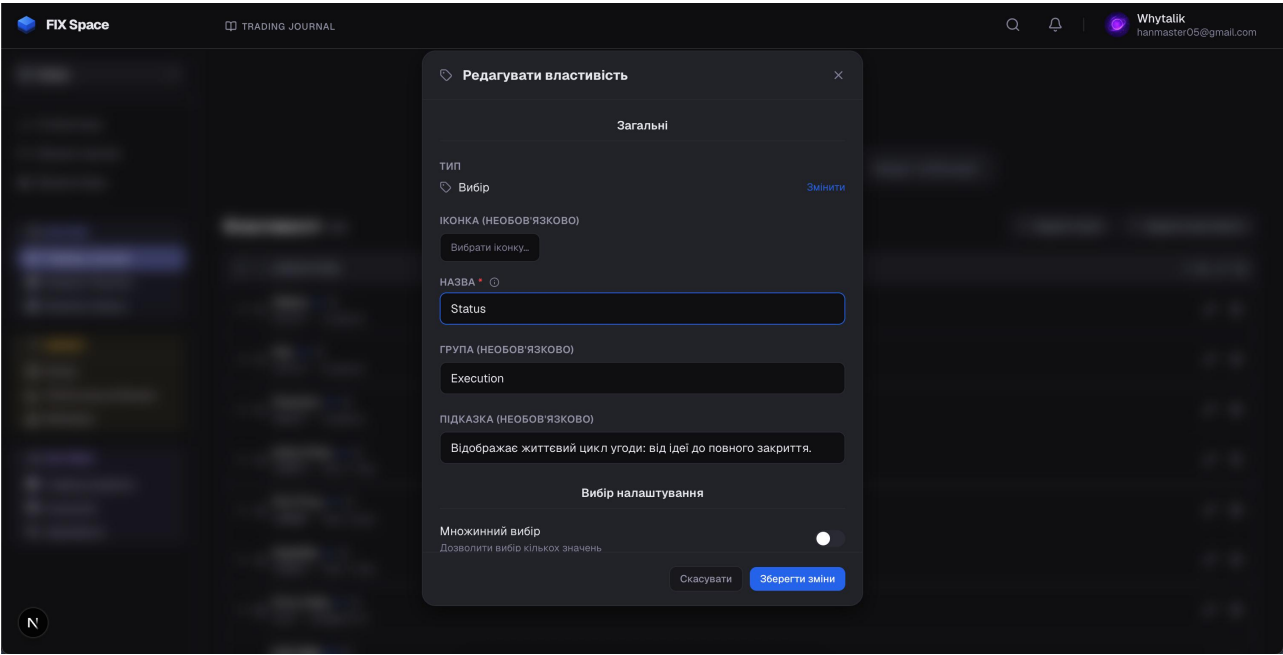


Рис. 3.12. Модальне вікно для створення нової властивості

Для оптимізації рутинних операцій передбачено механізм шаблонів. На відповідній вкладці налаштувань бази даних трейдер створює заготовки для майбутніх записів (рис. 3.13).

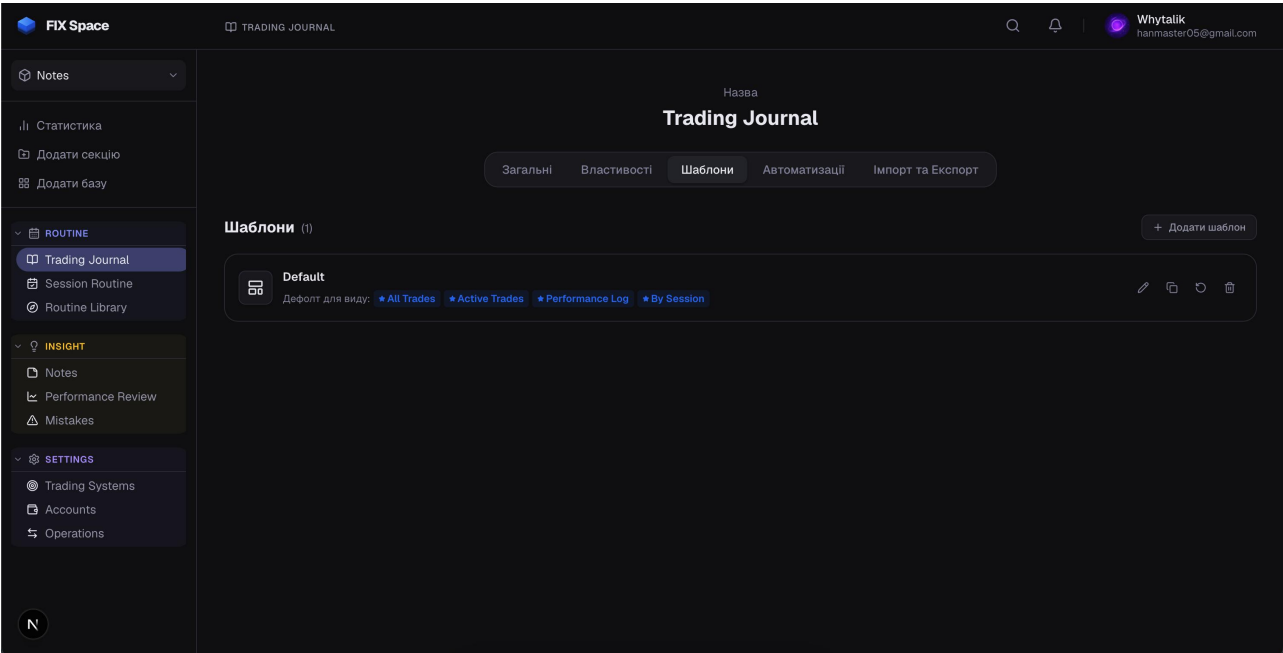


Рис. 3.13. Сторінка налаштувань бази даних, вкладка шаблонів

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				95
Змн.	Арк.	№ докум.	Підпис	Дата		

Один раз налаштувавши структуру контенту та стандартні значення властивостей на сторінці шаблону (рис. 3.14), користувач гарантує, що кожна нова угода чи нотатка автоматично отримає необхідний скелет для аналізу, що значно знижує тертя під час фіксації думок у стані стресу.

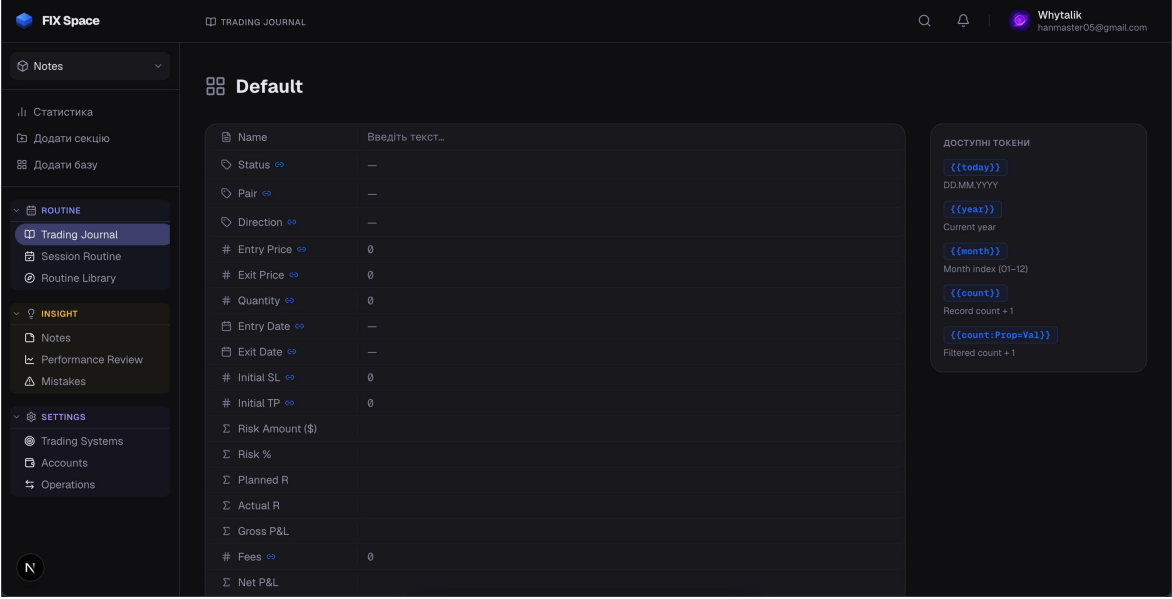


Рис. 3.14. Сторінка шаблону

Розуміючи, що трейдери часто приходять на платформу з вже накопиченою історією, розроблено інтуїтивний модуль імпорту та експорту даних (рис. 3.15).

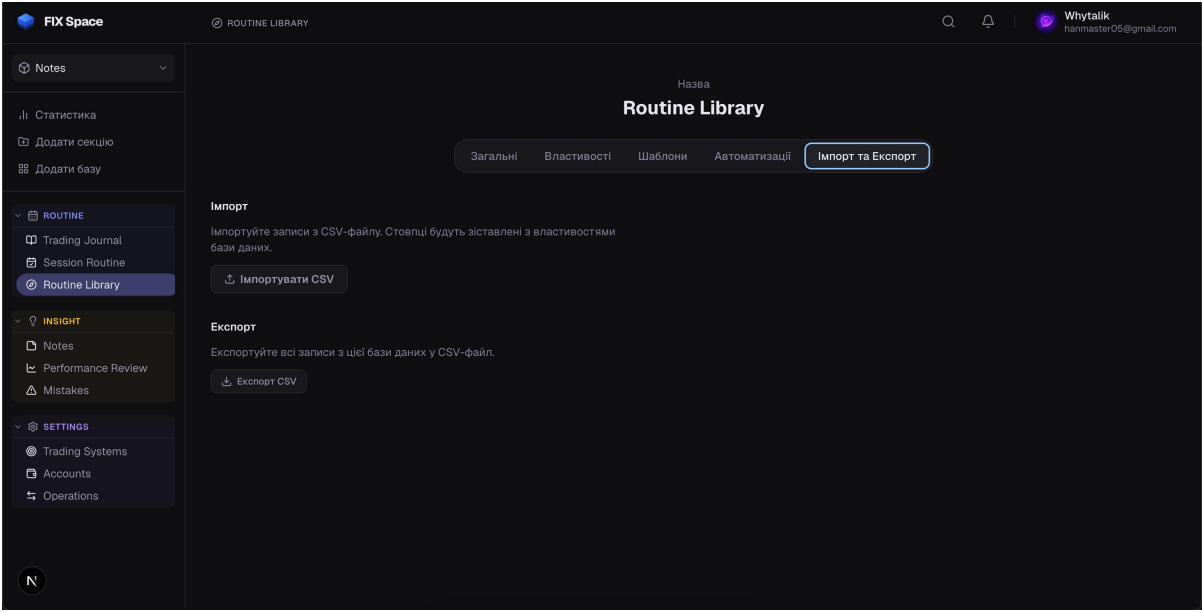


Рис. 3.15. Сторінка налаштувань бази даних, вкладка імпорту та експорту

Майстер імпорту CSV-файлів супроводжує користувача через процес мапінгу, де необхідно лише співставити колонки зовнішнього файлу з відповідними властивостями бази даних (рис. 3.16).

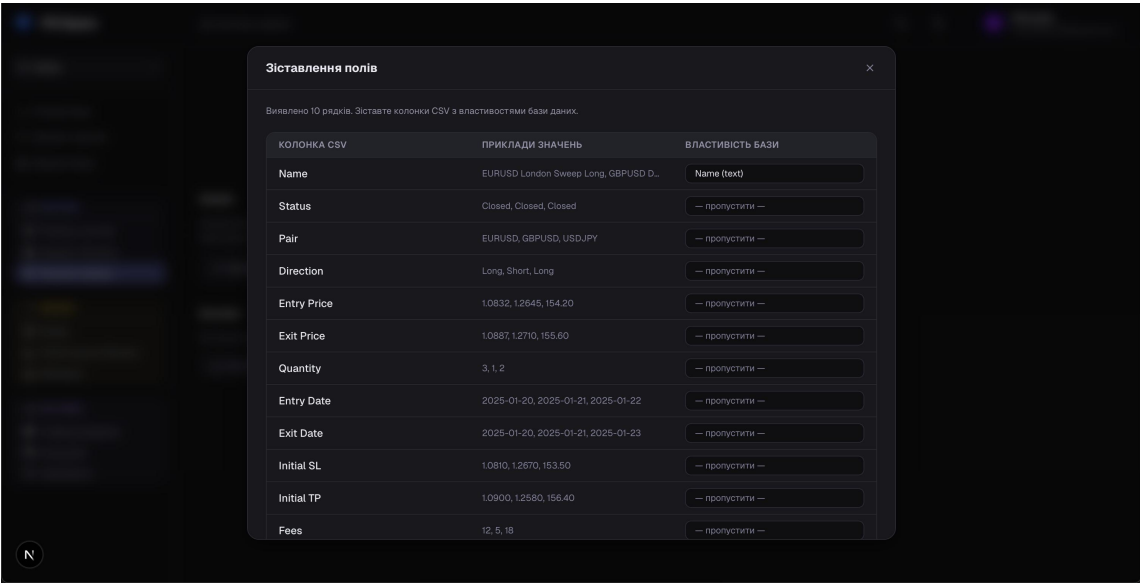


Рис. 3.16. Можливість співстановлення колонок при імпорті

Фінальним, найбільш професійним рівнем взаємодії з платформою стає подолання бар'єру між середовищем виконання ордерів та простором для їхнього глибокого аналізу. Цей процес розпочинається з глобального модального вікна налаштувань профілю (рис. 3.17), яке виступає єдиною точкою доступу до конфігурації всього робочого середовища.

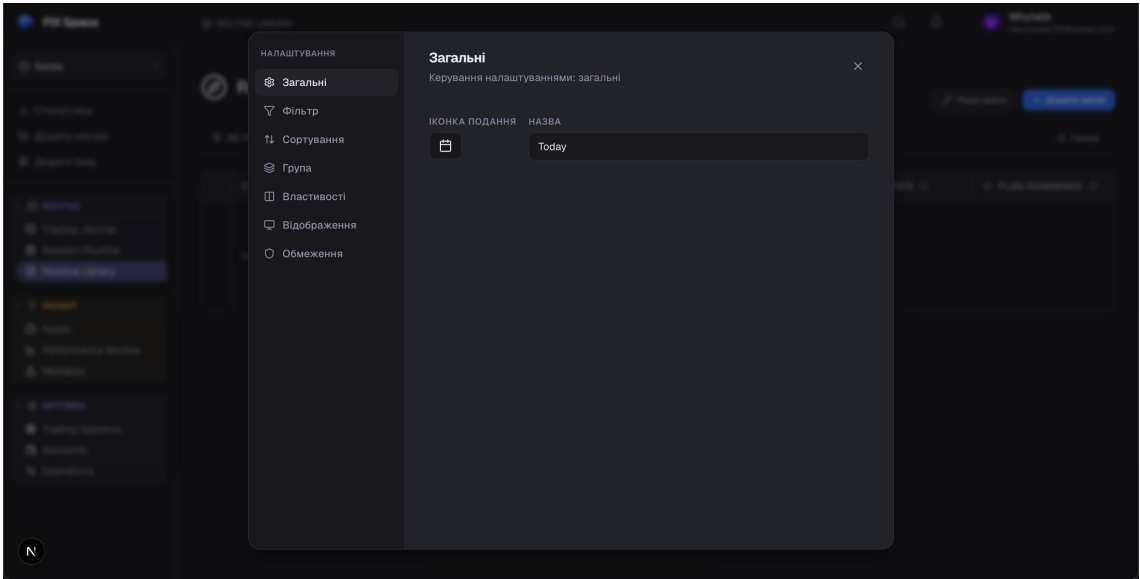


Рис. 3.17. Модальне вікно налаштувань

Саме тут, у структурованому меню, користувач знаходить розділ інтеграцій. Безпосередня конфігурація з'єднання відбувається на спеціалізованій сторінці налаштування інтеграцій (рис. 3.18), де трейдер авторизує підключення до таких торгових терміналів, як MetaTrader 5.

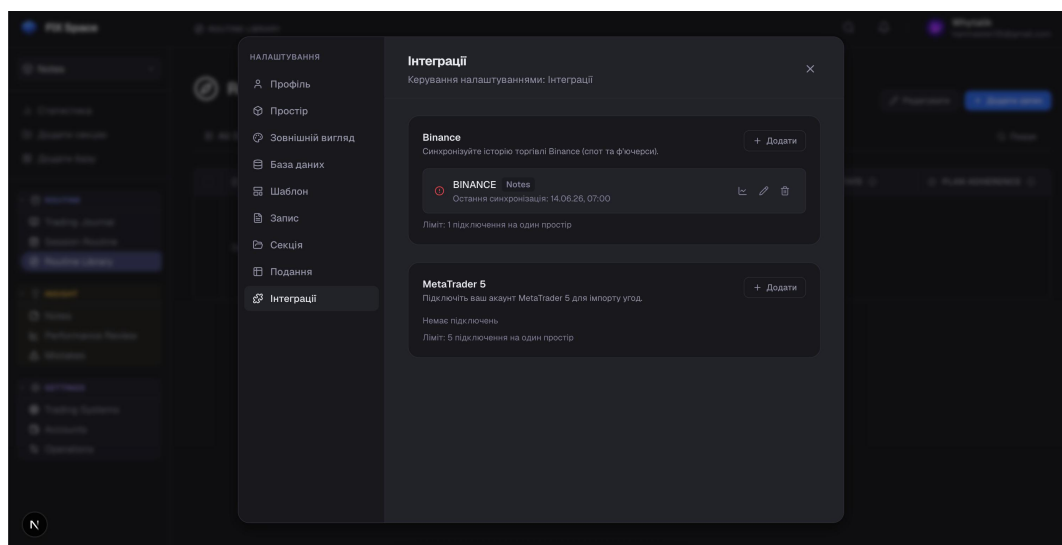


Рис. 3.18. Сторінка налаштування інтеграцій

Після встановлення з'єднання платформа відкриває доступ до історичних угод безпосередньо з торгового рахунку. Це дозволяє трейдеру буквально одним кліком перетворювати сирий масив ордерів на повноцінні, структуровані записи (рис. 3.19).

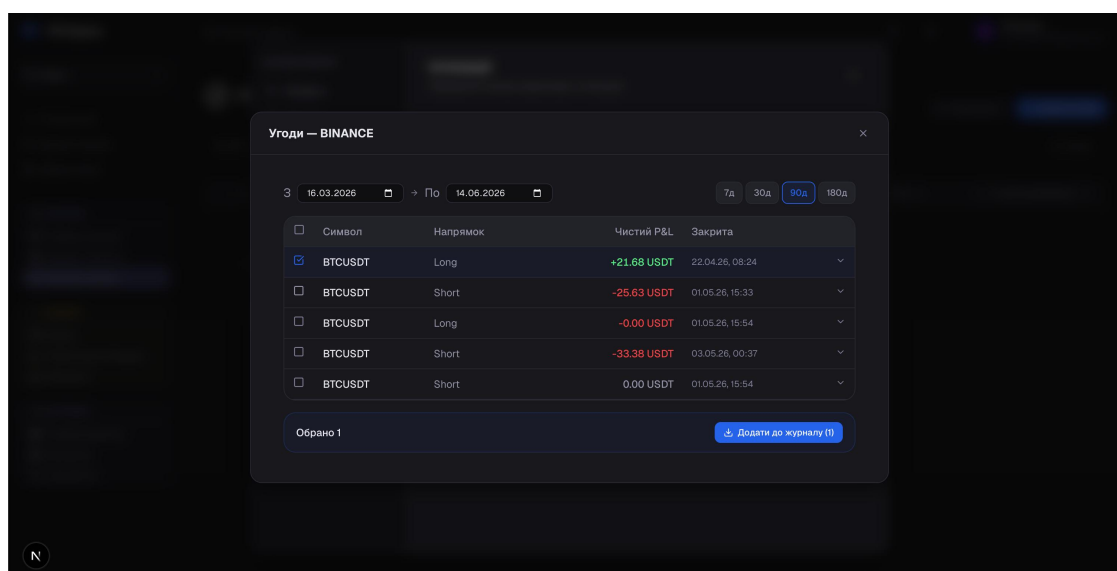


Рис. 3.19. Перегляд історичних угод з торгових платформ для імпорту

3.3 Забезпечення безпеки та надійності

Оскільки система оперує конфіденційними фінансовими даними, результатами торгівлі та особистою інформацією користувачів, побудова багаторівневої системи захисту є пріоритетним завданням архітектурного проєктування. Система безпеки розроблена з урахуванням принципу "Defense in Depth", що передбачає наявність кількох незалежних бар'єрів на шляху потенційного злоумисника.

Фундаментом безпеки платформи є надійна система автентифікації та управління сесіями, реалізована за допомогою механізму JSON Web Tokens. На відміну від класичних серверних сесій, які важко масштабуються, або зберігання токенів у localStorage, яке є вразливим до XSS-атак, у системі застосовано гібридний підхід. Після успішного входу користувач отримує пару токенів: з коротким терміном дії Access Token (термін дії 15 хвилин) для авторизації API-запитів та токен з тривалим терміном дії Refresh Token (термін дії 7 днів) для оновлення сесії без повторного введення пароля. Критично важливою особливістю реалізації є зберігання обох токенів виключно в HTTP-only cookies. Цей прапорець робить файли cookie недоступними для клієнтського JavaScript коду, що повністю нейтралізує ризик крадіжки сесії через скрипти третіх сторін. Додатково впроваджено сувору політику CORS, яка дозволяє приймати запити лише з довірених доменів клієнтської частини (Vercel), блокуючи будь-які спроби звернення з неавторизованих джерел.

Зберігання паролів користувачів у базі даних здійснюється з використанням алгоритму хешування bcrypt із автоматичним додаванням солі. Це забезпечує стійкість облікових даних навіть у гіпотетичному випадку повного витоку бази даних, оскільки відновлення оригінального пароля з хешу є обчислювально неможливим за сучасними стандартами. Повний потік автентифікації, включаючи перевірку креденшалів, генерацію токенів та їх безпечну передачу, візуалізований на діаграмі послідовності (рис. 3.20).

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		99

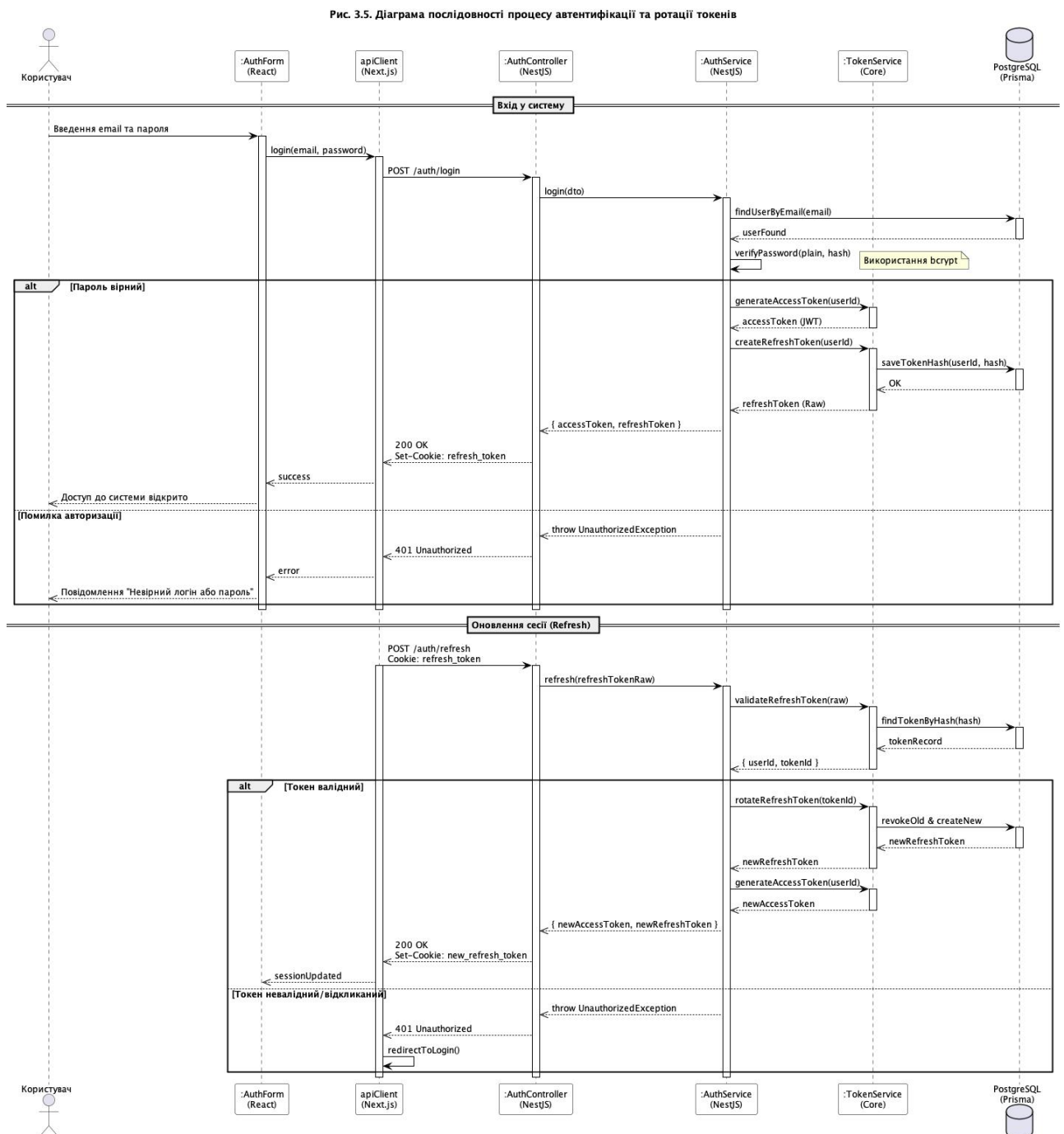


Рис. 3.20. Діаграма послідовності процесу автентифікації та ротації tokenів

Особливу увагу приділено захисту інтеграційних ключів від зовнішніх торгових платформ, таких як Binance або MetaTrader 5. Оскільки ці ключі надають доступ до фінансових рахунків користувачів, їх компрометація є неприпустимою. Для цього в системі реалізовано механізм шифрування з використанням сучасного стандарту AES-256-GCM [21]. Цей режим не лише забезпечує високу швидкість шифрування великих обсягів даних, але й

гарантує цілісність зашифрованого блоку: будь-яка спроба несанкціонованої модифікації даних у базі буде виявлена під час дешифрування через перевірку автентифікаційного тегу. Самі ключі ізольовані у захищених змінних оточення інфраструктури хостингу, доступ до яких мають лише процеси додатка під час виконання.

Після успішної автентифікації система забезпечує суворий контроль доступу до ресурсів через механізми авторизації NestJS Guards. Використання глобального JwtAuthGuard гарантує, що жоден ендпоінт API не залишиться відкритим для анонімних користувачів за замовчуванням. Однак простого факту входу недостатньо для доступу до чужих даних у мультикористувацькій системі. Тому додатково реалізовано кастомний ResourceOwnerGuard, який на рівні кожного запиту перевіряє приналежність цільового ресурсу (простору, бази даних або конкретного запису) поточному ідентифікатору користувача. Це забезпечує повну логічну ізоляцію даних між різними обліковими записами на рівні бізнес-логіки, унеможливлюючи ситуації, коли один користувач може переглянути або змінити торговий журнал іншого навіть знаючи його ID.

Надійність збереження даних та їхня внутрішня узгодженість забезпечуються завдяки використанню транзакційної моделі Prisma ORM. Для критичних операцій, таких як масовий імпорт угод, зміна структури бази даних або одночасне оновлення контенту та створення його знімка, застосовується метод \$transaction(). Це гарантує атомарність змін відповідно до властивостей ACID: операція або виконується повністю з переходом системи в новий стабільний стан, або, у разі виникнення будь-якої помилки (мережевий збій, порушення унікальності, помилка валідації), всі проміжні зміни автоматично відкочуються.

Для захисту від автоматизованих атак та зловживань ресурсами в системі впроваджено механізм лімітування частоти запитів. Використання модуля @nestjs/throttler дозволяє обмежити кількість запитів до API з однієї IP-адреси за певний проміжок часу. Це є ефективним методом протидії brute-force атакам на форми авторизації та запобігання DDoS-атакам низького рівня, які можуть

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				101
Змн.	Арк.	№ докум.	Підпис	Дата		

перевантажити сервер. Крім того, всі вхідні дані проходять сувору валідацію через об'єкти передачі даних з використанням декораторів бібліотеки class-validator. Цей шар валідації слугує додатковим бар'єром проти атак типу SQL-ін'єкцій та Mass Assignment, оскільки система відкидає будь-які поля, які не описані в схемі DTO, ще до того, як дані потраплять у логіку обробки.

Окремим рівнем забезпечення надійності є стратегія моніторингу та резервного копіювання. Система веде централізоване логування всіх критичних подій, помилок та дій користувачів за допомогою спеціалізованого LoggingInterceptor. Логи структуровані у форматі JSON, що дозволяє легко інтегрувати їх із системами аналізу для швидкого виявлення аномалій у поведінці додатку. База даних PostgreSQL, розміщена на платформі Railway, підтримує автоматичне щоденне створення фізичних знімків стану із можливістю відновлення до конкретної часової мітки. Це мінімізує ризики повної втрати даних у разі технічних збоїв інфраструктури, людського фактору або непередбачуваних помилок у роботі додатка, гарантуючи трейдеру спокій за збереженість свого торгового досвіду.

3.4 Тестування системи

Забезпечення якості програмного забезпечення є невід'ємною складовою процесу розробки. Для системи FIX Space реалізовано багаторівневу стратегію тестування, що охоплює юніт-тестування окремих компонентів, інтеграційне тестування міжмодульної взаємодії, навантажувальне тестування продуктивності під реальним навантаженням, а також наскрізне (E2E) тестування користувацького інтерфейсу. Кожен рівень верифікує окремий аспект коректності: ізольовану логіку сервісів, контракти HTTP API, цілісність користувацьких сценаріїв у реальних браузерях та поведінку системи при одночасному обслуговуванні кількох користувачів. Загальна стратегія відповідає характеристикам якості моделі ISO/IEC 25010 у частині придатності до тестування, надійності та продуктивності.

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				102
Змн.	Арк.	№ докум.	Підпис	Дата		

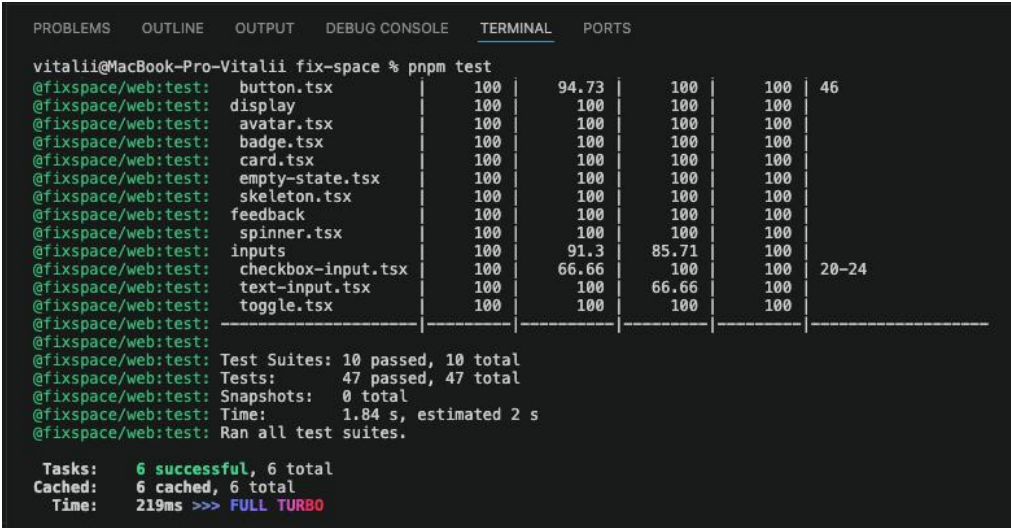
Для верифікації системи застосовано багаторівневий інтегрований підхід до забезпечення якості, що базується на концепції класичної піраміди тестування [30]. Вся кодова база та її логіка розподіляються за кількома взаємодоповнюючими шарами. Модульний рівень (Unit Testing) становить основу піраміди та охоплює приблизно сімдесят відсотків від загального обсягу автоматизованих перевірок. Його метою є швидка та ізольована верифікація алгоритмічної коректності бізнес-сервісів, сценаріїв використання, валідаційних функцій, обчислювальних обробників та формульного двигуна без звернення до зовнішніх ресурсів, таких як мережа чи бази даних. Середній шар піраміди, який складає близько двадцяти відсотків, займає інтеграційний рівень (Integration Testing). Він призначений для перевірки спільної роботи окремих модулів системи, взаємодії між компонентами.

Для ізоляції сховища даних застосовано окремий локальний екземпляр системи керування базами даних PostgreSQL, розгорнутий у Docker-контейнері відповідно до конфігурації профілю. Стан бази даних повністю очищується та переініціалізується шляхом виконання міграцій та початкового наповнення перед кожним повним тестовим запуском. Для проведення модульного та інтеграційного тестування використовується середовище Jest із пакетом транспіляції TypeScript ts-jest та допоміжним сервісним модулем NestJS для імітації залежностей через механізми впровадження залежностей. Верифікація інтеграційного HTTP-транспорту здійснюється за допомогою спеціалізованої бібліотеки supertest. Сценарії наскрізного тестування базуються на фреймворку Playwright, який забезпечує автоматичний запуск локального вебклієнта Next.js та сервера NestJS, одночасно керуючи тестуванням у браузерях на базі рушіїв Chromium, Firefox та WebKit. Навантажувальне тестування реалізовано за допомогою інструменту Artillery, який використовує HTTP-движок для імітації паралельної сесійної активності користувачів. Додатковим шаром верифікації виступає рівень навантажувальної стійкості (Load Testing), спрямований на дослідження поведінки системи під конкурентними паралельними потоками

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				103
Змн.	Арк.	№ докум.	Підпис	Дата		

запитів, зокрема для аналізу перцентилів часу відгуку, виявлення можливих таймаутів чи витоків пам'яті.

Юніт-тести перевіряють ізольовану бізнес-логіку кожного сервісу, сценарію використання та обробника типів властивостей. Усі зовнішні залежності замінюються імітаційними об'єктами (mock-об'єктами) та заглушками, що дозволяє перевіряти поведінку компонентів детерміновано та незалежно від інфраструктури. Тести структуровані за логічними групами та функціональними сценаріями: успішне виконання, граничні умови, обробка виняткових ситуацій та перевірка взаємодії із залежними компонентами [29]. Загалом реалізовано 481 юніт-тест у 44 специфікаційних файлах, що охоплюють усі 18 функціональних модулів системи. Результат виконання всього юніт-тестового набору наведено на рисунку 3.21.



```
vitalii@MacBook-Pro-Vitalii fix-space % npm test
@fixspace/web:test: button.tsx | 100 | 94.73 | 100 | 100 | 46
@fixspace/web:test: display | 100 | 100 | 100 | 100 |
@fixspace/web:test: avatar.tsx | 100 | 100 | 100 | 100 |
@fixspace/web:test: badge.tsx | 100 | 100 | 100 | 100 |
@fixspace/web:test: card.tsx | 100 | 100 | 100 | 100 |
@fixspace/web:test: empty-state.tsx | 100 | 100 | 100 | 100 |
@fixspace/web:test: skeleton.tsx | 100 | 100 | 100 | 100 |
@fixspace/web:test: feedback | 100 | 100 | 100 | 100 |
@fixspace/web:test: spinner.tsx | 100 | 100 | 100 | 100 |
@fixspace/web:test: inputs | 100 | 91.3 | 85.71 | 100 |
@fixspace/web:test: checkbox-input.tsx | 100 | 66.66 | 100 | 20-24
@fixspace/web:test: text-input.tsx | 100 | 66.66 | 100 |
@fixspace/web:test: toggle.tsx | 100 | 100 | 100 |
@fixspace/web:test: -----
@fixspace/web:test: Test Suites: 10 passed, 10 total
@fixspace/web:test: Tests: 47 passed, 47 total
@fixspace/web:test: Snapshots: 0 total
@fixspace/web:test: Time: 1.84 s, estimated 2 s
@fixspace/web:test: Ran all test suites.

Tasks: 6 successful, 6 total
Cached: 6 cached, 6 total
Time: 219ms >>> FULL TURBO
```

Рис. 3.21. Результат виконання юніт-тестів

Розподіл тестів по модулях та їхня відповідність задокументованим тест-кейсам наведено у таблиці 3.1.

Таблиця 3.1

Розподіл юніт-тестів по модулях

Модуль	Кількість спес-файлів	Тестів (Jest)	Задокументовани х ТС	Статус
Core	7	87	67	Виконано

Шаблони	4	102	69	Виконано
Перегляди	3	63	40	Виконано
Властивості	5	56	34	Виконано
Значення властивостей	1	16	16	Виконано
Автоматизації	3	42	9	Виконано

Кількість Jest-тестів перевищує кількість задокументованих ТС-ідентифікаторів, оскільки ряд спрес-файлів реалізовано поза процесом документування тест-кейсів і покривають допоміжні сценарії: перевірку HTTP-декораторів контролерів, поведінку репозиторіїв та рефлексивні виклики Prisma-транзакцій.

Окремо перевірялася коректність обробника формул. `FormulaEngine.evaluate()` тестується на п'яти пресетах (Запланований RR, Дотримання правил, Середній бал, Категорія за порогом, Rollup по пов'язаних записах) з повним набором вхідних значень та граничними ситуаціями: відсутні поля, нульові знаменники, циклічні залежності. Усі 13 тест-кейсів формульного двигуна проходять без відхилень.

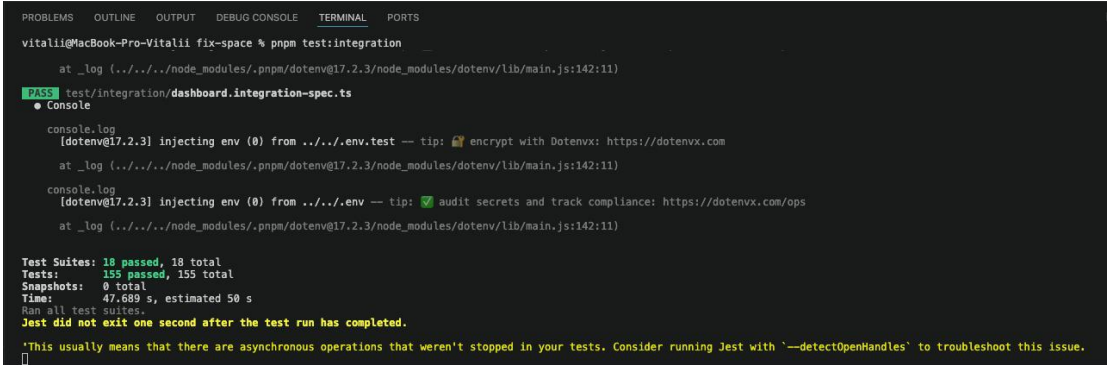
Інтеграційні тести перевіряють коректність HTTP API системи як єдиного цілого. На відміну від юніт-тестів, де кожна залежність замінюється стабом, інтеграційні тести запускають повноцінний NestJS-застосунок із реальним підключенням до тестової бази даних PostgreSQL. HTTP-запити надсилаються через supertest-агент безпосередньо до HTTP-сервера, а відповіді перевіряються за статусним кодом, структурою тіла та станом бази даних через Prisma-клієнт [31].

Для уникнення надмірного споживання з'єднань з базою даних (кожен NestJS-застосунок відкриває власний пул) застосовано патерн спільного

екземпляра сервера: `globalSetup` Jest запускає один екземпляр застосунку до старту будь-якого тестового файлу, а `globalTeardown` закриває його після завершення всіх 18 файлів. Тестові файли звертаються до спільного сервера через `supertest.agent(url)`, що еквівалентно HTTP-клієнту з підтримкою cookies між запитами. Такий підхід скоротив загальний час виконання інтеграційних тестів з ~420 с до ~87 с.

Поштовий сервіс у тестовому середовищі замінено стабом, що записує всі виклики до внутрішнього масиву; тести отримують доступ до цих викликів через спеціальний маршрут `/_test/mail`, зареєстрований виключно при запуску в тестовому режимі. Це дозволяє, зокрема, перевірити, що при реєстрації нового користувача токен верифікації email дійсно передається в аргументах `sendVerificationEmail` та відповідає токєну, прийнятому при верифікації.

Загалом реалізовано 155 інтеграційних тестів у 18 файлах-специфікаціях. Результат їхнього виконання наведено на рисунку 3.22.



```
PROBLEMS  OUTLINE  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
vitalii@MacBook-Pro-Vitalii fix-space % npm test:integration
at _log (/.../node_modules/.pnpm/dotenv@17.2.3/node_modules/dotenv/lib/main.js:142:11)
PASS test/integration/dashboard.integration-spec.ts
  ● Console
    console.log
      [dotenv@17.2.3] injecting env (0) from ../.env.test — tip: 🔐 encrypt with Dotenvx: https://dotenvx.com
    at _log (/.../node_modules/.pnpm/dotenv@17.2.3/node_modules/dotenv/lib/main.js:142:11)
    console.log
      [dotenv@17.2.3] injecting env (0) from ../.env — tip: ✅ audit secrets and track compliance: https://dotenvx.com/ops
    at _log (/.../node_modules/.pnpm/dotenv@17.2.3/node_modules/dotenv/lib/main.js:142:11)
Test Suites: 18 passed, 18 total
Tests:       155 passed, 155 total
Snapshots:   0 total
Time:        47.689 s, estimated 50 s
Ran all test suites.
Jest did not exit one second after the test run has completed.
*This usually means that there are asynchronous operations that weren't stopped in your tests. Consider running Jest with '--detectOpenHandles' to troubleshoot this issue.
```

Рис. 3.22. Результат виконання інтеграційних тестів

Наскрізне (end-to-end, E2E) тестування верифікує роботу всього програмного комплексу (інтерфейс користувача, HTTP-сервер, база даних, сторонні сервіси) в умовах, максимально наближених до реальної експлуатації. Головна мета цього рівня переконатися, що клієнтська частина коректно рендериться у веббраузері, успішно відправляє запити до бекенду, правильно обробляє відповіді та підтримує складний динамічний стан інтерфейсу.

Для реалізації E2E-тестів обрано фреймворк Playwright, який дозволяє запускати сценарії паралельно у реальних браузерах на трьох основних рушіях:

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				106
Змн.	Арк.	№ докум.	Підпис	Дата		

Chromium, Firefox та WebKit. Це гарантує кросбраузерну сумісність інтерфейсу та стабільність інтерактивних елементів на різних пристроях.

Тестова конфігурація описана у файлі `playwright.config.ts`. Перед запуском тестів Playwright автоматично ініціалізує та запускає локальні сервери Web App та API за допомогою інструменту Turboporo. Для повної ізоляції кожен тестовий сценарій виконується в окремому браузерному контексті.

Набір E2E-тестів FIX Space містить 75 тестів, розподілених по 17 специфікаційних файлах, які повністю покривають усі 18 функціональних модулів системи. Сценарії автентифікації перевіряють процеси реєстрації, верифікації електронної пошти за допомогою локального перехоплювача повідомлень, входу в систему, оновлення токенів та виходу. Сценарії ініціалізації робочого простору контролюють автоматичне розгортання простору при першому вході та перевіряють автоматичне створення всіх дев'яти встановлених баз даних із налагодженими зв'язками та шаблонами. Взаємодія з самими базами даних та секціями панелі навігації перевіряється через створення баз, редагування їхніх метаданих, групування баз та зміну порядку секцій за допомогою інтерфейсу перетягування.

Сценарії роботи з властивостями та записами покривають налаштування всіх 11 типів полів, додавання нових рядків таблиць, наповнення блокового контенту сторінок та перерахунок математичних формул на клієнті в реальному часі. Також тестуються механізми збереження та застосування шаблонів, робота з представленнями, налаштування фільтрів та сортувань, розрахунок KPI на аналітичних дашбордах статистики, процеси імпорту та експорту CSV-файлів, підключення API зовнішніх криптобірж для завантаження угод, обробка push-сповіщень при спрацюванні автоматизованих правил та зміна глобальних налаштувань профілю користувача.

Для стабільного виконання тестів у бічній панелі реалізовано автоматичне клієнтське розгортання секцій при першому вході в систему, якщо налаштування користувача ще не збережені. Це запобігає приховуванню баз

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				107
Змн.	Арк.	№ докум.	Підпис	Дата		

даних та гарантує детерміноване знаходження елементів інтерфейсу селекторами Playwright.

Результати виконання E2E-тестового набору підтверджують 100% успішність проходження (75 з 75 тестів успішно виконано) у браузерях Chromium, Firefox та WebKit.

Навантажувальне тестування проводилося з метою перевірки стійкості системи до одночасного обслуговування кількох користувачів, що є реалістичним сценарієм для SaaS-платформи. Використовувався інструмент Artillery у режимі HTTP-навантаження з визначеними фазами: поступове зростання кількості віртуальних користувачів (ramp-up), стабільне навантаження та пікова фаза.

Тестовий сценарій відтворює реальну послідовність дій трейдера, включаючи автентифікацію, зчитування просторів, зчитування баз даних простору та отримання записів першої бази даних, де кожен віртуальний користувач виконує цей сценарій циклічно. Конфігурація тестового запуску містила чотири основні фази: початкове лінійне зростання від нуля до десяти активних віртуальних користувачів протягом тридцяти секунд, фазу стабільного навантаження із десятима користувачами протягом шістдесяти секунд, подальше зростання навантаження від десяти до двадцяти користувачів за тридцять секунд, а також фінальне утримання пікового навантаження у двадцять користувачів протягом тридцяти секунд. Результати навантажувального тестування наведено на рисунку 3.23 та у таблиці 3.2.

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		108

FIX Space — Load Test Report

Artillery | 3161 requests (selected phases) | 0 errors

20-72

Median range
ms across phases

97-268

p95 range
ms across phases

49

Request rate
req/s

1815

VUs completed
of 1815

RESPONSE TIME BY PHASE

PHASE	DURATION	REQUESTS	RATE	MEDIAN (P50)	P95	STATUS
Ramp-up (1→10 VU)	30s	26	3/s	20 (20)	97 (97)	OK
Sustained (10 VU)	60s	1831	27/s	52 (33-59-56-59)	168 (123-159-162-194)	OK
Peak (10→20 VU)	60s	2184	37/s	61 (45-72-78-59-59-62)	199 (173-191-211-188-176-268)	OK

RESPONSE TIME DISTRIBUTION — DETAIL BY PHASE INTERVAL

#	PHASE	REQUESTS	RATE	P50	P95	P99	MEAN
1	Ramp-up	26	9/s	20	97	123	39
2	Sustained	120	19/s	33	123	125	44
3	Sustained	224	28/s	59	159	211	67
4	Sustained	335	37/s	56	162	247	69
5	Sustained	352	36/s	59	194	233	73
6	Peak	328	35/s	45	173	268	61
7	Peak	340	37/s	72	191	257	77
8	Peak	340	36/s	70	211	252	81
9	Peak	328	35/s	59	188	228	78
10	Peak	339	42/s	59	176	247	68
11	Peak	429	47/s	62	268	348	85

SUMMARY

Total requests (good intervals)

3 161

Total requests (full test)

6 183

HTTP 200

6 183 100%

Errors

0

Virtual users completed

1 815 / 1 815

Рис. 3.23. Результати навантажувального тестування API на читання (Artillery)

Усі 3 161 запит у вибраних фазах тесту завершилися зі статусом HTTP 200, жодної помилки не зафіксовано. У фазі 1 (ramp-up 1→10 VU) медіана відгуку склала 20 мс, р95 – 97 мс. У фазі 2 (sustained, 10 VU) медіана трималась у діапазоні 33–59 мс, р95 – 123–194 мс, що значно нижче цільового порогу 300 мс. У пікову фазу 3–4 (10→20 VU, 60 с) медіана зросла до 45–72 мс, р95 – до 173–268 мс. Таке зростання пояснюється тим, що bcrypt-хешування пароля при кожному логіні (cost factor 10) є CPU-bound операцією і стає вузьким місцем при паралельному виконанні. Для виробничого середовища це вирішується через кешування сесій або зменшення cost factor при незмінному рівні безпеки. При реалістичному навантаженні система стабільно утримує р95 нижче 200 мс.

Висновки до третього розділу

У третьому розділі здійснено практичну реалізацію спроектованої архітектури, що перетворило теоретичну модель на повноцінний, готовий до експлуатації програмний продукт.

Окрему увагу приділено побудові надійної інфраструктури та автоматизації процесів доставки коду. Контейнеризація застосунку за допомогою Docker із багатоступеневою збіркою та налаштування

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				109
Змн.	Арк.	№ докум.	Підпис	Дата		

безперервної інтеграції (CI/CD) через GitHub Actions дозволили усунути розбіжності між середовищами розробки, тестування та експлуатації. Розподіл хостингу між Vercel для клієнтської частини та Railway для серверного моноліту і бази даних є свідомим кроком для оптимізації витрат і забезпечення високої швидкості відгуку інтерфейсу завдяки глобальним CDN, зберігаючи при цьому повний контроль над реляційним сховищем даних та його резервним копіюванням.

Враховуючи конфіденційний характер фінансової інформації, фундаментальним аспектом реалізації стала багаторівнева система безпеки, що працює за принципом ешелонованої оборони. Застосування HTTP-only cookies для JWT-токенів унеможливило вектор атак через XSS, а шифрування інтеграційних ключів алгоритмом AES-256-GCM забезпечило захист доступу до брокерських рахунків. Логічна ізоляція даних на рівні бізнес-логіки через кастомні Guard-механізми та транзакційність операцій за стандартом ACID гарантують, що навіть у разі збоїв мережі чи помилок валідації торговий досвід користувача залишається цілісним, неушкодженим і категорично недоступним для сторонніх осіб.

Верифікація працездатності та стійкості платформи проведена через багаторівневу стратегію тестування, яка охопила понад сім сотень автоматизованих сценаріїв. Поєднання ізольованого юніт-тестування, інтеграційних перевірок із реальною базою даних та наскрізного (E2E) моделювання користувацької поведінки у різних браузерах дозволило виявити та усунути приховані дефекти взаємодії компонентів до моменту релізу. Результати навантажувального тестування підтвердили, що обрана архітектура здатна стабільно обслуговувати паралельні сесії, утримуючи час відгуку в межах, необхідних для комфортної аналітичної роботи без когнітивних затримок, що безпосередньо впливає на концентрацію трейдера.

Демонстрація порядку роботи з платформою від етапу реєстрації та автоматичної ініціалізації простору до глибокої аналітики, створення шаблонів і прямої інтеграції з MetaTrader 5 підтверджує, що інженерна задача

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				110
Змн.	Арк.	№ докум.	Підпис	Дата		

вирішена у повному обсязі. Розроблена вебплатформа FIX Space не лише технічно успішно реалізована, але й повністю відповідає початковій меті дослідження: вона надає трейдеру єдиний, безпечний та інтуїтивно зрозумілий інструментарій, який трансформує хаотичний масив ринкових даних і суб'єктивних вражень у структуровану базу знань, придатну для об'єктивного статистичного аналізу та подолання власних поведінкових помилок.

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				111
Змн.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

У кваліфікаційній роботі розроблено спеціалізовану вебплатформу FIX Spase для організації діяльності приватного трейдера, яка забезпечує централізоване зберігання, структурування та аналіз торгових даних у межах єдиного робочого середовища. Мету дослідження досягнуто в повному обсязі.

У першому розділі проведено аналіз предметної області та систематизовано типи даних, з якими щоденно працює трейдер: угоди, аналіз дня, нотатки, помилки та торгова статистика. Виявлено, що ключовою проблемою є інформаційна фрагментація розпорошення цих даних між різнорідними інструментами, що унеможлиблює побудову цілісної картини власної торгівлі та сприяє посиленню когнітивних викривлень у процесі прийняття рішень. Порівняльний аналіз наявних рішень показав їхню часткову відповідність потребам трейдера, що підтвердило доцільність розробки спеціалізованого інструменту.

У другому розділі сформовано систему вимог до платформи та спроектовано архітектуру системи. Визначено дві ролі акторів і детально описано варіанти використання, що охоплюють автентифікацію, управління робочим простором, структурування даних, аналітику, автоматизації та інтеграцію із зовнішніми торговими платформами. Розроблено гібридну модель даних, що поєднує реляційну цілісність PostgreSQL зі структурною гнучкістю формату JSONB для користувацьких властивостей. Обґрунтовано вибір модульної монолітної архітектури на стеку TypeScript як оптимального рішення для продукту, орієнтованого на індивідуального користувача. Спроектовано ключові алгоритми: безпечне обчислення формул через AST за допомогою бібліотеки jser, механізм версіонування контенту записів із ротацією знімків за принципом FIFO та багатоканальний алгоритм парсингу торгових CSV-звітів із зовнішніх брокерських платформ.

У третьому розділі здійснено повну реалізацію платформи та верифіковано її коректність. Реалізовано повний цикл автентифікації, тобто реєстрацію, вхід через Google OAuth, підтвердження email та відновлення

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				112
Змн.	Арк.	№ докум.	Підпис	Дата		

пароля. При першому вході система автоматично ініціалізує персональний робочий простір із дев'ятьма предметними базами даних, адаптованими під типові потреби трейдера. Система баз даних підтримує 11 типів властивостей, налаштовані подання та формульний двигун для автоматичного розрахунку показників. Контентна область кожного запису оснащена блоковим редактором, механізмом версіонування та системою шаблонів. Статистичний модуль обчислює ключові торгові метрики такі як win rate, математичне очікування, drawdown, розподіл прибутку за активами та інструментами. Автоматизації з подієвими тригерами та розкладом скорочують рутинне введення даних. Реалізовано імпорт та експорт даних у форматі CSV, глобальний пошук по всьому простору, а також інтеграцію з MetaTrader 5 через MQL5 Webhook для автоматичної синхронізації закритих угод. Безпеку платформи забезпечено на кількох рівнях: HTTP-only cookies унеможливлюють атаки через XSS, шифрування інтеграційних ключів алгоритмом AES-256-GCM захищає доступ до брокерських рахунків, а Guard-механізми ізолюють дані на рівні бізнес-логіки. Розгортання реалізовано засобами Docker із багатоступеневою збіркою та автоматизованим CI/CD-пайплайном через GitHub Actions, хостинг розподілено між Vercel для клієнтської частини та Railway для серверного моноліту і бази даних.

Якість платформи підтверджено багаторівневою стратегією тестування: 481 юніт-тест у 44 спец-файлах, 155 інтеграційних тестів із підключенням до реальної бази даних PostgreSQL, 75 наскрізних (E2E) тестів у трьох браузерних рушіях через Playwright. Результати навантажувального тестування за допомогою Artillery підтвердили стабільність відгуку системи в умовах паралельних сесій.

Практична цінність отриманих результатів полягає в тому, що розроблена вебплатформа FIX Space є готовим до експлуатації програмним продуктом, який трансформує хаотичний масив ринкових даних і суб'єктивних спостережень трейдера у структуровану базу знань, придатну для об'єктивного статистичного аналізу. Платформа надає трейдеру єдиний безпечний

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				113
Змн.	Арк.	№ докум.	Підпис	Дата		

інструментарій для подолання інформаційної фрагментації та поведінкових помилок, що безпосередньо впливає на якість торгових рішень.

Напрямами подальшого вдосконалення програмного комплексу є реалізація мобільного клієнта для оперативної фіксації угод, розширення списку підтримуваних брокерів та форматів імпорту, впровадження аналітичних можливостей на основі штучного інтелекту для виявлення поведінкових патернів, а також підтримка колаборативного режиму для роботи в командах чи з наставником.

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				114
Змн.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Тушак В.С., Чижмотря О.В. Інформаційна фрагментація у діяльності CFD-трейдера та засоби її подолання. Тези XVI Міжнародної науково-технічної конференції «Інформаційно-комп'ютерні технології», м. Житомир, 02-03 квітня 2026 р. – Житомир: Житомирська політехніка, 2026. (подано до друку)
2. Тушак В.С., Чижмотря О.В. Поведінкові помилки та роль статистичного аналізу у прийнятті торгових рішень. Тези XVI Міжнародної науково-технічної конференції «Інформаційно-комп'ютерні технології», м. Житомир, 02-03 квітня 2026 р. – Житомир: Житомирська політехніка, 2026. (подано до друку)
3. Канеман Д. Мислення швидко й повільно / пер. з англ. М. Яковлев. Київ : Наш Формат, 2017. 480 с.
4. Brown C., Dark J., Davis K. CFDs, forwards, futures and the cost-of-carry // Pacific-Basin Finance Journal. 2019. Vol. 54. P. 183–198. URL: ScienceDirect (дата звернення: 24.03.2026).
5. Tharp V. K. Trade Your Way to Financial Freedom. 2nd ed. New York : McGraw-Hill, 2007. 368 p.
6. Форте Т. Запасний мозок. Як організувати цифрове життя і розвантажити голову / пер. з англ. А. Дудченко. Київ : Наш Формат, 2024. 272 с.
7. Kahneman D., Tversky A. Prospect Theory: An Analysis of Decision under Risk // Econometrica. 1979. Vol. 47, No. 2. P. 263–291.
8. Google Sheets [Електронний ресурс]. – Режим доступу: <https://workspace.google.com/products/sheets/> (дата звернення: 24.03.2026).
9. Notion [Електронний ресурс]. – Режим доступу: <https://www.notion.com/> (дата звернення: 24.03.2026).
10. TraderSync Features [Електронний ресурс]. – Режим доступу: <https://tradersync.com/features/> (дата звернення: 24.03.2026).
11. ISO/IEC 25010:2023 Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Product quality model / International Organization for Standardization ; International Electrotechnical

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				115
Змн.	Арк.	№ докум.	Підпис	Дата		

Commission. — 2nd ed. — Geneva, 2023. — 22 p. — URL: <https://www.iso.org/standard/78176.html> (дата звернення: 24.03.2026).

12. Douglas M. Trading in the Zone: Master the Market with Confidence, Discipline and a Winning Attitude. New York : Prentice Hall Press, 2000. 240 p.

13. Sommerville I. Software Engineering. 10th ed. London : Pearson, 2016. 810 p.

14. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Boston : Prentice Hall, 2017. 432 p.

15. Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. Sebastopol : O'Reilly Media, 2021. 616 p.

16. Fowler M. Patterns of Enterprise Application Architecture. Boston : Addison-Wesley, 2002. 560 p.

17. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Boston : Addison-Wesley, 2003. 560 p.

18. Aho A. V., Lam M. S., Sethi R., Ullman J. D. Compilers: Principles, Techniques, and Tools. 2nd ed. Boston : Addison-Wesley, 2006. 1009 p.

19. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures [Електронний ресурс] : doctoral dissertation. Irvine : University of California, 2000. 162 p. URL: https://ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf (дата звернення: 24.03.2026).

20. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT) : RFC 7519. Internet Engineering Task Force, 2015. 30 p. URL: <https://www.rfc-editor.org/rfc/rfc7519> (дата звернення: 24.03.2026).

21. Dworkin M. J. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC : NIST Special Publication 800-38D. Gaithersburg : National Institute of Standards and Technology, 2007. 39 p. URL: <https://doi.org/10.6028/NIST.SP.800-38D> (дата звернення: 24.03.2026).

22. Bierman G., Abadi M., Torgersen M. Understanding TypeScript // Lecture Notes in Computer Science (ECOOP 2014 – Object-Oriented Programming). 2014. Vol.

		Тушак В.С.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Чижмоторя О.В.				116
Змн.	Арк.	№ докум.	Підпис	Дата		

8586. Р. 257–281. URL: https://doi.org/10.1007/978-3-662-44202-9_11 (дата звернення: 24.03.2026).
23. Stonebraker M., Rowe L. A. The design of POSTGRES // Proceedings of the 1986 ACM SIGMOD International Conference on Management of Data. 1986. P. 340–355. URL: <https://doi.org/10.1145/16894.16902> (дата звернення: 24.03.2026).
24. Date C. J. An Introduction to Database Systems. 8th ed. Boston : Addison-Wesley, 2003. 1024 p.
25. Banks A., Porcello E. Learning React: Modern Patterns for Developing React Apps. 2nd ed. Sebastopol : O'Reilly Media, 2020. 310 p.
26. Hunt A., Thomas D. The Pragmatic Programmer: Your Journey to Mastery. 20th Anniversary Edition. Boston : Addison-Wesley, 2019. 352 p.
27. Merkel D. Docker: Lightweight Linux Containers for Consistent Development and Deployment // Linux Journal. 2014. No. 239. P. 1–14.
28. Humble J., Farley D. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Boston : Addison-Wesley, 2010. 512 p.
29. Beck K. Test Driven Development: By Example. Boston : Addison-Wesley, 2002. 240 p.
30. Cohn M. Succeeding with Agile: Software Development Using Scrum. Boston : Addison-Wesley Professional, 2009. 504 p.
31. Meszaros G. xUnit Test Patterns: Refactoring Test Code. Boston : Addison-Wesley, 2007. 944 p.
32. jsep: A Tiny JavaScript Expression Parser [Електронний ресурс]. – Режим доступу: <https://github.com/EricSmekens/jsep> (дата звернення: 24.03.2026).
33. MQL5 Reference: WebRequest [Електронний ресурс]. – Режим доступу: <https://www.mql5.com/en/docs/network/webrequest> (дата звернення: 24.03.2026).

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				117
Змн.	Арк.	№ докум.	Підпис	Дата		

ДОДАТКИ

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				118
Змн.	Арк.	№ докум.	Підпис	Дата		

Лістинг servisy Record

```

import { BadRequestException, Injectable, NotFoundException } from "@nestjs/common";
import { EventEmitter2 } from "@nestjs/event-emitter";

import { Prisma } from "@fixspace/database";
import { CreateRecordDto, RecordResponseDto, UpdateRecordDto } from "@fixspace/domain";

import { AppLogger } from "@common/logger/app-logger.service";
import { filterUndefined } from "@common/utils/filter-undefined";
import { t } from "@common/utils/i18n.helper";
import { generateUniqueName } from "@common/utils/generate-unique-name";

import { SettingsCategory } from "@modules/settings/constants/settings.constants";
import { SettingsService } from "@modules/settings/settings.service";
import { FormulaRecalculator } from "@modules/property/types/formula/formula-recalculator.service";
import { RecordContentService } from "@modules/record-content/record-content.service";
import { DatabaseRepository } from "@modules/database/repositories/database.repository";
import { PropertyRepository } from "@modules/property/repositories/property.repository";
import { TemplateRepository } from "@modules/template/repositories/template.repository";
import { ViewRepository } from "@modules/view/repositories/view.repository";
import { RecordRepository } from "@modules/record/repositories/record.repository";
import { toRecordResponseDto } from "@modules/record-content/record-content.service";
import { parseNamePattern } from "@modules/record-content/record-content.service";
import { normalizeContentSchema, hasContentRows } from "@modules/record-content/record-content.service";

@Injectable()
export class RecordService {
  constructor(
    private readonly logger: AppLogger,
    private readonly eventEmitter: EventEmitter2,
    private readonly settingsService: SettingsService,
    private readonly formulaRecalculator: FormulaRecalculator,
    private readonly recordRepo: RecordRepository,
    private readonly recordContentService: RecordContentService,
    private readonly viewRepo: ViewRepository,
    private readonly databaseRepo: DatabaseRepository,
    private readonly propertyRepo: PropertyRepository,
    private readonly templateRepo: TemplateRepository,
  ) {
    this.logger.setContext(RecordService.name);
  }

  async create(
    databaseId: string,
    createRecordDto: CreateRecordDto,
    userId: string,
    options?: { skipAutomations?: boolean },
  ): Promise<RecordResponseDto> {
    this.logger.debug("Creating record", { databaseId, viewId: createRecordDto.viewId });
  }

```

		Тушак В.С.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Чижмоторя О.В.				119
Змн.	Арк.	№ докум.	Підпис	Дата		

```

const database = await this.databaseRepo.findDatabaseByOwner(databaseId, userId);

if (!database) {
  throw new NotFoundException(t("errors.DATABASE_NOT_FOUND_ID", { id: databaseId }));
}

const properties = await this.propertyRepo.findManyByDatabase(databaseId);

const view = createRecordDto.viewId ? await this.viewRepo.findById(createRecordDto.viewId) :
null;

let capturedTemplateContent: unknown = null;

const result = await this.recordRepo.transaction(async (transaction) => {
  let templateId = createRecordDto.templateId;
  let recordIcon = createRecordDto.icon;

  this.logger.debug("Starting transaction for record creation", { templateId, providedName:
createRecordDto.name });

  if (templateId === undefined) {
    if (view?.useDefaultTemplate && view.defaultTemplateId) {
      templateId = view.defaultTemplateId;
      this.logger.debug("Resolved template from view default", { templateId });
    } else if (view?.useDefaultTemplate !== false) {
      const defaultTemplate = await this.templateRepo.findDefaultInDatabase(databaseId,
transaction);
      if (defaultTemplate) {
        templateId = defaultTemplate.id;
        this.logger.debug("Resolved template from database default", { templateId });
      }
    }
  }

  const templateValues: Record<string, unknown> = {};
  let recordName = createRecordDto.name;

  if (templateId) {
    const template = await transaction.template.findFirst({
      where: {
        id: templateId,
        databaseId,
      },
      include: {
        values: true,
      },
    });

    if (!template) {
      throw new NotFoundException(t("errors.TEMPLATE_NOT_FOUND"));
    }
  }
}

```

		Тушак В.С.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Чижемотря О.В.				120
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    if (!recordIcon && template.icon) {
      recordIcon = template.icon;
    }

    let patternToUse = template.namePattern;
    if (!patternToUse && template.name?.includes("{ { }")) {
      patternToUse = template.name;
    }

    if (patternToUse && (!recordName || recordName === "Untitled" || recordName === "Без
назви")) {
      recordName = await parseNamePattern(patternToUse, databaseId, transaction);
    }

    for (const templateValue of template.values) {
      if (templateValue.value !== undefined && templateValue.value !== null) {
        templateValues[templateValue.propertyId] = templateValue.value;
      }
    }

    capturedTemplateContent = template.content;
  }

  const { icon: effectiveIcon } = await this.settingsService.resolveDefaults(userId,
SettingsCategory.RECORD, {
    icon: recordIcon,
  });

  const record = await this.recordRepo.create(
    {
      databaseId,
      name: recordName,
      icon: effectiveIcon,
      templateId: templateId,
    },
    transaction,
  );

  for (const property of properties) {
    const templateValue = templateValues[property.id];
    const value = templateValue !== undefined ? templateValue : Prisma.DbNull;

    await transaction.propertyValue.create({
      data: {
        recordId: record.id,
        propertyId: property.id,
        value: value as Prisma.InputJsonValue,
        computed: false,
      },
    });
  }
}

```

		Тушак В.С.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Чижевотря О.В.				121
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    await this.formulaRecalculator.recalculate(record.id, databaseId, transaction);

    this.logger.log("Record created", {
      recordId: record.id,
      databaseId,
      templateId: templateId,
    });

    const createdRecord = await this.recordRepo.findUniqueOrThrowWithValues(record.id,
transaction);
    return toRecordResponseDto(createdRecord);
  });

  if (hasContentRows(capturedTemplateContent)) {
    await this.recordContentService.update(result.id, { content:
normalizeContentSchema(capturedTemplateContent) });
  }

  if (!options?.skipAutomations) {
    await this.eventEmitter.emitAsync("automation.recordCreated", { record: result, userId });
  }

  return result;
}

async findAll(databaseId: string, userId: string): Promise<RecordResponseDto[]> {
  this.logger.debug("Finding all records", { databaseId });
  const records = await this.recordRepo.findAllByDatabase(databaseId, userId);
  return records.map(toRecordResponseDto);
}

async findAllPaged(
  databaseId: string,
  userId: string,
  page: number,
  pageSize: number,
): Promise<{ data: RecordResponseDto[]; total: number; page: number; pageSize: number }> {
  if (page < 1 || pageSize < 1) {
    throw new BadRequestException(t("errors.PAGE_PAGE_SIZE_MUST_BE_POSITIVE"));
  }

  this.logger.debug("Finding paged records", { databaseId, page, pageSize });

  const [records, total] = await this.recordRepo.findPagedByDatabase(databaseId, userId, (page - 1)
* pageSize, pageSize);

  return {
    data: records.map(toRecordResponseDto),
    total,
    page,
    pageSize,
  }
}

```

		Тушак В.С.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Чижевотря О.В.				122
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    };
}

async findOne(id: string): Promise<RecordResponseDto> {
    this.logger.debug("Finding record", { id });

    const record = await this.recordRepo.findById(id);

    if (!record) {
        throw new NotFoundException(t("errors.RECORD_NOT_FOUND_ID", { id }));
    }

    return toRecordResponseDto(record);
}

async update(id: string, updateRecordDto: UpdateRecordDto): Promise<RecordResponseDto> {
    this.logger.debug("Updating record", { id });

    const existingRecord = await this.recordRepo.findById(id);

    if (!existingRecord) {
        throw new NotFoundException(t("errors.RECORD_NOT_FOUND_ID", { id }));
    }

    const updateData = filterUndefined({
        fields: {
            name: updateRecordDto.name,
            icon: updateRecordDto.icon,
        },
    });

    const record = await this.recordRepo.update(id, updateData);

    this.logger.log("Record updated", { id });
    return toRecordResponseDto(record);
}

async remove(id: string): Promise<RecordResponseDto> {
    this.logger.debug("Removing record", { id });

    const existingRecord = await this.recordRepo.findById(id);

    if (!existingRecord) {
        throw new NotFoundException(t("errors.RECORD_NOT_FOUND_ID", { id }));
    }

    const record = await this.recordRepo.delete(id);

    this.logger.log("Record removed", { id });
    return toRecordResponseDto(record);
}

```

		Тушак В.С.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Чижмотря О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		123

```

async applyTemplate(id: string, templateId: string): Promise<RecordResponseDto> {
  this.logger.debug("Applying template to record", { recordId: id, templateId });

  const record = await this.recordRepo.findById(id);
  if (!record) {
    throw new NotFoundException(t("errors.RECORD_NOT_FOUND_ID", { id }));
  }

  let capturedTemplateContent: unknown = null;

  const result = await this.recordRepo.transaction(async (transaction) => {
    const template = await transaction.template.findFirst({
      where: { id: templateId, databaseId: record.databaseId },
      include: { values: true },
    });

    if (!template) {
      throw new NotFoundException(t("errors.TEMPLATE_NOT_FOUND"));
    }

    capturedTemplateContent = template.content;

    let recordName = record.name;
    if (template.namePattern && (!recordName || recordName === "Untitled" || recordName ===
"Без назви")) {
      recordName = await parseNamePattern(template.namePattern, record.databaseId, transaction);
    }

    for (const templateValue of template.values) {
      if (templateValue.value !== undefined && templateValue.value !== null) {
        await transaction.propertyValue.upsert({
          where: {
            recordId_propertyId: {
              recordId: id,
              propertyId: templateValue.propertyId,
            },
          },
          update: {
            value: templateValue.value as Prisma.InputJsonValue,
          },
          create: {
            recordId: id,
            propertyId: templateValue.propertyId,
            value: templateValue.value as Prisma.InputJsonValue,
          },
        });
      }
    }

    await transaction.record.update({
      where: { id },
      data: {

```

		Тушак В.С.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Чижемотря О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		124

```

        templateId,
        name: recordName,
    },
    });

    await this.formulaRecalculator.recalculate(id, record.databaseId, transaction);

    if (capturedTemplateContent) {
        await this.recordContentService.update(id, {
            content: normalizeContentSchema(capturedTemplateContent),
        });
    }

    this.logger.log("Template applied", { recordId: id, templateId });
    const updated = await this.recordRepo.findUniqueOrThrowWithValues(id, transaction);
    return toRecordResponseDto(updated);
});

return result;
}

async duplicate(id: string): Promise<RecordResponseDto> {
    this.logger.debug("Duplicating record", { id });

    const source = await this.recordRepo.findById(id);

    if (!source) {
        throw new NotFoundException(t("errors.RECORD_NOT_FOUND_ID", { id }));
    }

    return this.recordRepo.transaction(async (transaction) => {
        const newRecord = await this.recordRepo.create(
            {
                databaseId: source.databaseId,
                name: generateUniqueName(source.name),
                icon: source.icon ?? undefined,
            },
            transaction,
        );

        for (const propertyValue of source.values) {
            await transaction.propertyValue.create({
                data: {
                    recordId: newRecord.id,
                    propertyId: propertyValue.propertyId,
                    value: propertyValue.value as Prisma.InputJsonValue,
                    computed: propertyValue.computed,
                },
            });
        }
    });

    await this.formulaRecalculator.recalculate(newRecord.id, source.databaseId, transaction);
}

```

		Тушак В.С.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Чижмотря О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		125

```

    this.logger.log("Record duplicated", { sourceId: id, newId: newRecord.id });

    const created = await this.recordRepo.findUniqueOrThrowWithValues(newRecord.id,
transaction);
    return toRecordResponseDto(created);
  });
}
}

```

		Тушак В.С.			ПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижевотря О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		126

Лістинг CI/CD пайплайну

```

name: CI

on:
  push:
    branches: [develop, main]
  pull_request:
    branches: [develop, main]

env:
  NODE_VERSION: '24'
  PNPM_VERSION: '9.15.1'
  DATABASE_URL: 'postgresql://postgres:postgres@localhost:5432/unused'

jobs:
  lint:
    name: Lint
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - uses: pnpm/action-setup@v4
        with:
          version: ${{ env.PNPM_VERSION }}

      - uses: actions/setup-node@v4
        with:
          node-version: ${{ env.NODE_VERSION }}
          cache: pnpm

      - name: Install dependencies
        run: pnpm install --frozen-lockfile

      - name: Cache turbo build setup
        uses: actions/cache@v4
        with:
          path: .turbo
          key: ${{ runner.os }}-turbo-${{ hashFiles('**/turbo.json', '**/package.json', 'pnpm-lock.yaml') }}
          restore-keys: |
            ${{ runner.os }}-turbo-

      - name: Generate Prisma Client
        run: pnpm turbo db:generate

      - name: Lint
        run: pnpm turbo lint

  test:

```

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		127

```

name: Test
runs-on: ubuntu-latest
needs: [lint]
steps:
  - uses: actions/checkout@v4
    with:
      fetch-depth: 0

  - uses: pnpm/action-setup@v4
    with:
      version: ${{ env.PNPM_VERSION }}

  - uses: actions/setup-node@v4
    with:
      node-version: ${{ env.NODE_VERSION }}
      cache: pnpm

  - name: Install dependencies
    run: pnpm install --frozen-lockfile

  - name: Cache turbo build setup
    uses: actions/cache@v4
    with:
      path: .turbo
      key: ${{ runner.os }}-turbo-${{ hashFiles('**/turbo.json', '**/package.json', 'pnpm-lock.yaml') }}
      restore-keys: |
        ${{ runner.os }}-turbo-

  - name: Generate Prisma Client
    run: pnpm turbo db:generate

  - name: Test
    run: pnpm turbo test -- --coverage

  - name: Upload coverage to Codecov
    uses: codecov/codecov-action@75cd11691c0faa626561e295848008c8a7dddffe # v5
    with:
      token: ${{ secrets.CODECOV_TOKEN }}
      fail_ci_if_error: false

  - uses: actions/upload-artifact@v4
    if: always()
    with:
      name: coverage-report
      path: '**/coverage/'
      retention-days: 7

security:
  name: Security Audit
  runs-on: ubuntu-latest
  needs: [lint]

```

		Тушак В.С.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		128

steps:

- uses: actions/checkout@v4
- uses: pnpm/action-setup@v4
with:
version: \${{ env.PNPM_VERSION }}
- uses: actions/setup-node@v4
with:
node-version: \${{ env.NODE_VERSION }}
cache: pnpm
- name: Install dependencies
run: pnpm install --frozen-lockfile
- name: Audit dependencies
run: pnpm audit --prod --audit-level=high
continue-on-error: true

test-integration:

name: Test (Integration)
runs-on: ubuntu-latest
needs: [test]
environment:
name: staging
services:
postgres:
image: postgres:16
env:
POSTGRES_USER: postgres
POSTGRES_PASSWORD: postgres
POSTGRES_DB: fixspace_test
options: >-
--health-cmd pg_isready
--health-interval 10s
--health-timeout 5s
--health-retries 5
ports:
- 5432:5432
env:
DATABASE_URL:
postgres://postgres:postgres@localhost:5432/fixspace_test?schema=public
NODE_ENV: test
JWT_SECRET: super-secret-key-for-diploma-project
JWT_REFRESH_SECRET: super-refresh-secret-key-for-diploma-project
GOOGLE_CLIENT_ID: test-google-client-id
GOOGLE_CLIENT_SECRET: test-google-client-secret
GOOGLE_CALLBACK_URL: http://localhost:3000/auth/google/callback
steps:
- uses: actions/checkout@v4

- uses: pnpm/action-setup@v4

		Тушак В.С.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Чижемотря О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		129

```

with:
  version: ${{ env.PNPM_VERSION }}

- uses: actions/setup-node@v4
  with:
    node-version: ${{ env.NODE_VERSION }}
    cache: pnpm

- name: Install dependencies
  run: pnpm install --frozen-lockfile

- name: Generate Prisma Client
  run: pnpm turbo db:generate

- name: Build domain package
  run: pnpm --filter @fixspace/domain build

- name: Run migrations
  run: pnpm --filter @fixspace/database exec prisma migrate deploy

- name: Integration tests
  run: pnpm --filter @fixspace/api test:integration

build:
  name: Build
  runs-on: ubuntu-latest
  needs: [test]
  steps:
    - uses: actions/checkout@v4

    - uses: pnpm/action-setup@v4
      with:
        version: ${{ env.PNPM_VERSION }}

    - uses: actions/setup-node@v4
      with:
        node-version: ${{ env.NODE_VERSION }}
        cache: pnpm

    - name: Install dependencies
      run: pnpm install --frozen-lockfile

    - name: Cache turbo build setup
      uses: actions/cache@v4
      with:
        path: .turbo
        key: ${{ runner.os }}-turbo-${{ hashFiles('**/turbo.json', '**/package.json', 'pnpm-lock.yaml') }}
        restore-keys: |
          ${{ runner.os }}-turbo-

    - name: Generate Prisma Client

```

		Тушак В.С.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		130

run: pnpm turbo db:generate

- name: Build

run: pnpm turbo build

- uses: actions/upload-artifact@v4

with:

name: build-output

path: |

apps/api/dist/

apps/web/.next/

retention-days: 1

docker:

name: Docker Build

runs-on: ubuntu-latest

needs: [build]

steps:

- uses: actions/checkout@v4

- name: Build API image

run: docker build -t fixspace-api:\${{ github.sha }} -f apps/api/Dockerfile .

- name: Show image info

run: docker images fixspace-api

deploy-staging:

name: Deploy (Staging)

runs-on: ubuntu-latest

needs: [docker]

if: github.ref == 'refs/heads/develop'

environment:

name: staging

steps:

- uses: actions/checkout@v4

- name: Create deployment manifest

run: |

mkdir -p deployment

echo "Environment: staging"

> deployment/manifest.txt

echo "Branch: \${{ github.ref_name }}"

>> deployment/manifest.txt

echo "Commit: \${{ github.sha }}"

>> deployment/manifest.txt

echo "Actor: \${{ github.actor }}"

>> deployment/manifest.txt

echo "Run ID: \${{ github.run_id }}"

>> deployment/manifest.txt

echo "Timestamp: \$(date -u +%Y-%m-%dT%H:%M:%SZ)" >> deployment/manifest.txt

cat deployment/manifest.txt

echo "Deployment to staging: IN PROGRESS"

- name: Install Railway CLI

run: npm install -g @railway/cli

- name: Deploy to Railway

		Тушак В.С.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Чижмотря О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		131

```

env:
  RAILWAY_TOKEN: ${ secrets.RAILWAY_TOKEN_STAGING ||
secrets.RAILWAY_TOKEN }}
  if: env.RAILWAY_TOKEN != "
  run: railway up --ci --service ${ secrets.RAILWAY_SERVICE_STAGING || 'fix-space-api-
staging' }}

- uses: actions/upload-artifact@v4
  with:
    name: deployment-manifest
    path: deployment/
    retention-days: 30

deploy-production:
  name: Deploy (Production)
  runs-on: ubuntu-latest
  needs: [docker]
  if: github.ref == 'refs/heads/main'
  environment:
    name: production
  steps:
    - uses: actions/checkout@v4

    - name: Create deployment manifest
      run: |
        mkdir -p deployment
        echo "Environment: production" > deployment/manifest.txt
        echo "Branch: ${ github.ref_name }}" >> deployment/manifest.txt
        echo "Commit: ${ github.sha }}" >> deployment/manifest.txt
        echo "Actor: ${ github.actor }}" >> deployment/manifest.txt
        echo "Run ID: ${ github.run_id }}" >> deployment/manifest.txt
        echo "Timestamp: $(date -u +%Y-%m-%dT%H:%M:%SZ)" >> deployment/manifest.txt
        cat deployment/manifest.txt
        echo "Deployment to production: IN PROGRESS"

    - name: Install Railway CLI
      run: npm install -g @railway/cli

    - name: Deploy to Railway
      env:
        RAILWAY_TOKEN: ${ secrets.RAILWAY_TOKEN }}
        if: env.RAILWAY_TOKEN != "
        run: railway up --ci --service ${ secrets.RAILWAY_SERVICE_PRODUCTION || 'fix-space-
api' }}

    - uses: actions/upload-artifact@v4
      with:
        name: deployment-manifest-prod
        path: deployment/
        retention-days: 30

```

Схема процесів безперервної інтеграції та доставки (CI/CD)

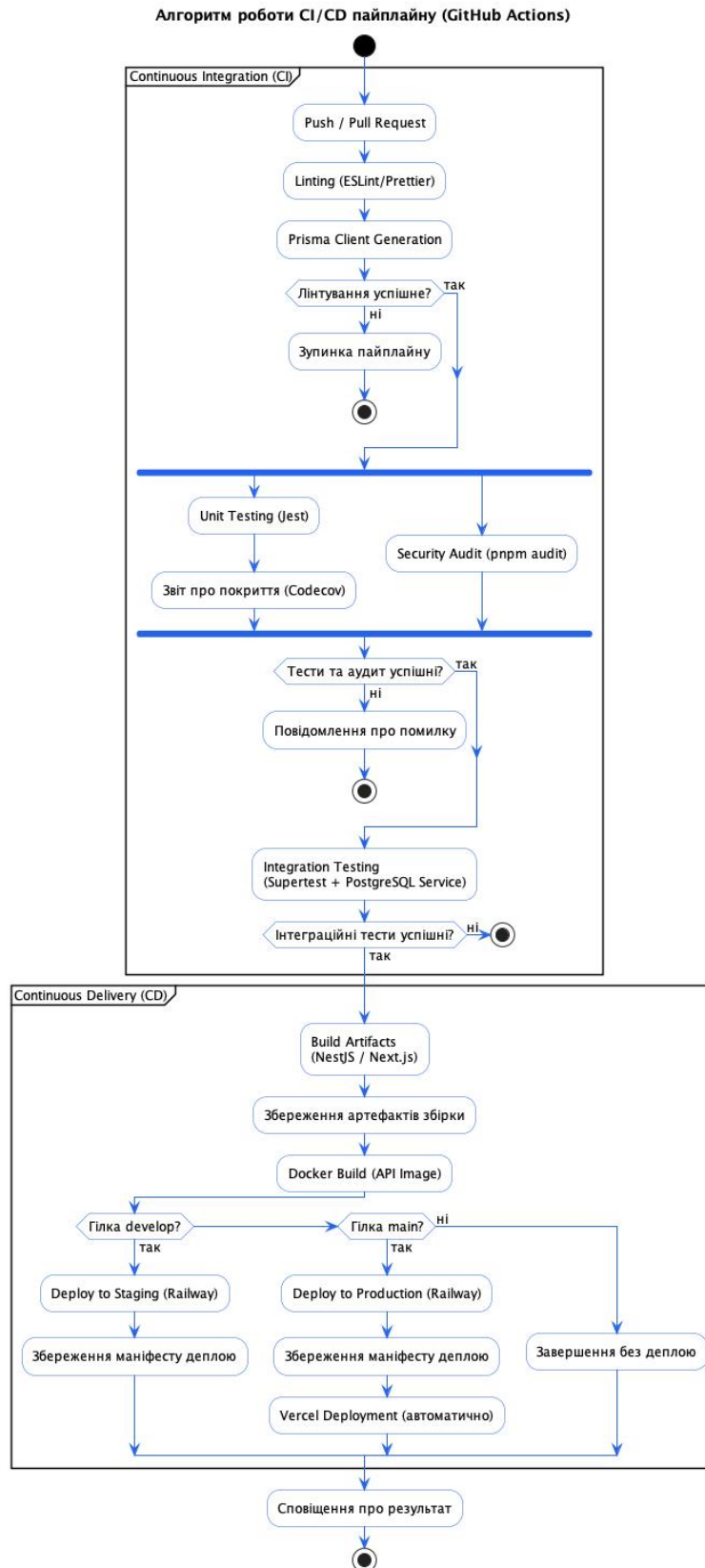


Рис. В.1. Схема процесів безперервної інтеграції та доставки (CI/CD)