ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка платформи керування криптотрейдинговими ботами»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Олександр ПОМАЗАН
(підпис)

Виконав: здобувач вищої освіти групи ПД-41

Олександр ПОМАЗАН

Керівник: Наталія ТРІНТІНА
канд.техн.наук, доц.

Рецензент:

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Помазану Олександру Олександровичу

1. Тема кваліфікаційної роботи: «Розробка платформи керування криптотрейдинговими ботами»
керівник кваліфікаційної роботи канд.техн.наук, доц. Наталія ТРІНТІНА,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026 року № 51.

2. Строк подання кваліфікаційної роботи «29» травня 2026 року.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про автоматизовану криптовалютну торгівлю, принципи роботи торгових ботів, методи обробки webhook-сигналів, підходи до керування біржовими акаунтами, контролю ризикових обмежень і розрахунку торгової аналітики, а також технічна документація щодо використання Java Spring Boot, React, PostgreSQL, Django, WebSocket, Docker Compose та API криптовалютних бірж.

4. Зміст розрахунково-пояснювальної записки:

1. Огляд та аналіз предметної галузі автоматизованої криптовалютної торгівлі, керування торговими ботами, біржовими акаунтами, сигналами та ризиками.
2. Аналіз існуючих програмних засобів для автоматизованої криптовалютної торгівлі, керування ботами, обробки webhook-сигналів і торгової аналітики.
3. Огляд засобів реалізації web-платформи CryptoKeps для керування криптотрейдинговими ботами.
4. Проєктування архітектури платформи, структури бази даних, основних модулів системи та взаємодії компонентів.
5. Програмна реалізація та опис функціонування web-платформи CryptoKeps.
6. Тестування основних функцій платформи та оцінка відповідності поставленим вимогам.

5. Перелік графічного матеріалу:

1. Аналіз аналогів програмного забезпечення для керування криптотрейдинговими ботами.
2. Вимоги до web-платформи CryptoKeps.
3. Архітектура програмної реалізації платформи.
4. Взаємодія React, Java Spring Boot, Django, PostgreSQL, Gateway та WebSocket.
5. Алгоритм обробки webhook-сигналу та оновлення стану торгової стратегії.
6. Діаграма класів або структура основних сутностей системи.
7. Екранні форми web-платформи: вхід, аналітика, боти, акаунти, Risk Manager.
8. Апробація результатів дослідження.

6. Дата видачі завдання«23» лютого 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури за темою автоматизованої криптовалютної торгівлі	23.02–28.02.2026	

2	Аналіз та дослідження існуючих аналогів програмного забезпечення для керування криптотрейдинговими ботами	01.03–07.03.2026	
3	Аналіз предметної галузі автоматизованої криптовалютної торгівлі та визначення вимог до web-платформи CryptoKeps	09.03–15.03.2026	
4	Огляд засобів реалізації web-платформи CryptoKeps	16.03–22.03.2026	
5	Проєктування web-платформи CryptoKeps, логічної структури системи, бази даних та UML-діаграм	23.03–05.04.2026	
6	Програмна реалізація серверної та клієнтської частин web-платформи CryptoKeps	06.04–26.04.2026	
7	Реалізація модуля Risk Manager, webhook-сигналів, торгової аналітики та live-моніторингу	27.04–03.05.2026	
8	Тестування web-платформи CryptoKeps та оформлення результатів тестування	04.05–10.05.2026	
9	Оформлення роботи: вступ, висновки, реферат, список джерел та додатки	11.05–17.05.2026	
10	Розробка демонстраційних матеріалів та підготовка до попереднього захисту роботи	18.05–22.05.2026	

Здобувач вищої освіти. _____ Олександр ПОМАЗАН
(підпис)

Керівник
кваліфікаційної роботи _____ Наталія ТРІНТІНА
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 52 стор., 4 табл., 20 рис., 24 джерела.

Мета роботи – спрощення процесу керування автоматизованою криптовалютною торгівлею за рахунок розробки web-платформи CryptoKeps для централізованого моніторингу торгових ботів, біржових акаунтів, webhook-сигналів, історії угод, торгової аналітики та risk-профілів.

Об'єкт дослідження – процеси автоматизованого керування криптовалютною торгівлею, торговими ботами, біржовими акаунтами, торговими сигналами та ризиковими обмеженнями.

Предмет дослідження – засоби, методи та технології реалізації web-платформи CryptoKeps для керування криптовалютними торговими ботами, обробки webhook-сигналів, моніторингу біржових профілів, аналізу результатів торгівлі та контролю ризиків.

Короткий зміст роботи: у кваліфікаційній роботі проведено аналіз предметної галузі автоматизованої криптовалютної торгівлі та визначено основні проблеми, пов'язані з розрізненим зберіганням даних про торгові акаунти, ботів, сигнали, історію угод і ризикові параметри. Встановлено, що використання кількох бірж, стратегій і сервісів ускладнює контроль торгових процесів, збільшує час аналізу стану акаунтів і підвищує ризик несвоєчасного реагування на збитки.

Проаналізовано існуючі програмні засоби для автоматизованої криптовалютної торгівлі, зокрема 3Commas, Cryptohopper, Coinrule, WunderTrading і TradingView. Визначено їхні переваги та недоліки, серед яких залежність від тарифних планів, обмежена гнучкість власної серверної логіки, часткова реалізація risk-management і прив'язка до зовнішніх сервісів.

У роботі сформовано функціональні та нефункціональні вимоги до платформи CryptoKeps, обґрунтовано вибір засобів реалізації клієнтської частини, серверної частини, бази даних, модуля risk-management і засобів розгортання. Для

реалізації web-платформи використано Java, Spring Boot, Spring Data JPA, PostgreSQL, Liquibase, React, TypeScript, Vite, Django, Spring Cloud Gateway, WebSocket, Docker Compose, REST API, Git та API криптовалютних бірж.

Спроєктовано загальну архітектуру платформи, логічну структуру системи, структуру бази даних, діаграму класів серверної частини, діаграму розгортання, схему обробки webhook-сигналу та схему роботи Risk Manager з watcher-сервісами OKX і Binance. Реалізовано основні модулі системи: авторизацію користувача, профіль користувача, сторінки моніторингу, ботів, акаунтів, аналітики, Risk Manager, обробку тестових сигналів, перегляд історії угод і live-оновлення даних.

Особливістю розробленої системи є модуль Risk Manager, який дозволяє створювати risk-профілі для бірж OKX і Binance, задавати ризикові обмеження, переглядати баланс, доступний баланс, максимальний баланс, PnL, серію збиткових угод, статус активної торгівлі та блокування профілю. Також передбачено Telegram-сповіщення про критичні події та можливість live-моніторингу через WebSocket.

Проведено тестування платформи CryptoKeps, перевірено коректність роботи авторизації, навігації, сторінок ботів, акаунтів, аналітики, Risk Manager, обробки webhook-сигналів, збереження історії угод і відображення live-даних.

КЛЮЧОВІ СЛОВА: CRYPTOKEPS, КРИПТОВАЛЮТНА ТОРГІВЛЯ, ТОРГОВИЙ БОТ, WEB-ПЛАТФОРМА, WEBHOOK-СИГНАЛ, RISK MANAGER, TRADINGVIEW, SPRING BOOT, REACT, POSTGRESQL, DJANGO, WEBSOCKET, DOCKER COMPOSE.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – інтерфейс прикладного програмування, що використовується для взаємодії між програмними компонентами або зовнішніми сервісами.

REST – архітектурний стиль організації обміну даними між клієнтською та серверною частинами web-застосунку.

HTTP – протокол передавання даних у мережі Інтернет.

HTTPS – захищена версія протоколу HTTP, що забезпечує безпечний обмін даними між клієнтом і сервером.

JSON – текстовий формат обміну даними, який використовується для передавання структурованої інформації між frontend- і backend-частинами системи.

WebSocket – технологія постійного двостороннього з'єднання між клієнтом і сервером для передавання live-даних.

Webhook – механізм автоматичного надсилання повідомлення або сигналу з одного сервісу до іншого після настання певної події.

Frontend – клієнтська частина web-платформи, з якою безпосередньо взаємодіє користувач.

Backend – серверна частина web- платформи, яка виконує бізнес-логіку, обробляє запити та працює з базою даних.

Java – об'єктно-орієнтована мова програмування, що використовується для реалізації серверної частини платформи.

Spring – фреймворк для створення серверних Java-застосунків.

Spring Boot – інструмент екосистеми Spring, який спрощує створення, налаштування та запуск backend-сервісів.

Spring Data JPA – модуль Spring для роботи з реляційними базами даних через об'єктну модель.

Spring Cloud Gateway – компонент Spring Cloud, який використовується для маршрутизації запитів між сервісами системи.

React – бібліотека JavaScript для створення компонентного користувацького інтерфейсу.

TypeScript – мова програмування, що розширює JavaScript за рахунок статичної типізації.

Vite – інструмент для швидкого створення, запуску та збирання frontend-застосунків.

Django – web-фреймворк мови Python, який використовується для реалізації backend-модуля авторизації та risk-профілів.

PostgreSQL – об’єктно-реляційна система керування базами даних.

Liquibase – інструмент для керування змінами структури бази даних за допомогою міграцій.

Docker – платформа контейнеризації, що використовується для запуску компонентів системи в ізольованому середовищі.

Docker Compose – інструмент для опису та запуску багатоконтейнерних застосунків.

Git – система контролю версій, що використовується для відстеження змін у програмному коді.

GitHub – web-сервіс для зберігання репозиторіїв, спільної роботи з кодом та контролю версій.

CryptoKeps – web-платформа для керування криптотрейдинговими ботами, торговими акаунтами, сигналами, аналітикою та risk-профілями.

TradingView – сервіс для аналізу фінансових графіків, створення індикаторів, стратегій та webhook-сигналів.

Bybit – криптовалютна біржа, API якої може використовуватися для роботи з торговими акаунтами, ордерами та історією угод.

OKX – криптовалютна біржа, дані якої використовуються watcher-сервісами для live-моніторингу risk-профілів.

Binance – криптовалютна біржа, дані якої використовуються watcher-сервісами для контролю балансів, позицій і ризикових параметрів.

Risk Manager – модуль платформи CryptoKeps для створення, запуску, зупинки та моніторингу risk-профілів.

Watcher-сервіс – окремий сервіс, який підключається до біржового API, отримує live-дані та перевіряє ризикові обмеження.

PnL – показник прибутку або збитку за торговою операцією чи періодом.

Win rate – частка прибуткових угод від загальної кількості угод.

Drawdown – просадка, тобто зниження торгового балансу або equity відносно попереднього максимуму.

KPI – ключові показники ефективності, які використовуються для оцінювання результатів роботи торгових ботів.

Telegram Bot API – інтерфейс для створення Telegram-ботів і надсилання повідомлень користувачу.

Dashboard – головна аналітична сторінка системи, на якій відображаються ключові показники роботи платформи.

SPA – односторінковий web-застосунок, у якому взаємодія з користувачем відбувається без повного перезавантаження сторінки.

IDE – інтегроване середовище розробки програмного забезпечення.

ВСТУП

У сучасних умовах розвитку фінансових технологій криптовалютна торгівля є одним із напрямів, який активно використовує програмні засоби автоматизації. Криптовалютний ринок працює цілодобово, характеризується високою волатильністю та потребує швидкого реагування на зміну торгової ситуації. Через це ручне керування торговими акаунтами, ботами, сигналами та ризиковими параметрами стає складним, особливо у випадку використання кількох бірж, стратегій і зовнішніх аналітичних сервісів [1; 2; 5].

Актуальність теми кваліфікаційної роботи зумовлена необхідністю створення зручної web-платформи для централізованого керування автоматизованою криптовалютною торгівлею. У реальних умовах дані про торгові акаунти, сигнали, історію угод, активність ботів, PnL, win rate, drawdown і risk-профілі можуть бути розподілені між різними біржами, ботами, таблицями, Telegram-сповіщеннями та аналітичними сервісами. Така розрізненість ускладнює контроль торгових процесів і підвищує ризик несвоєчасного реагування на збитки [8; 9; 10].

У зв’язку з цим актуальною є розробка платформи CryptoKeps, яка поєднує в одному середовищі керування торговими ботами, обробку webhook-сигналів, перегляд торгових акаунтів, збереження історії угод, розрахунок торгової аналітики, live-моніторинг біржових профілів і контроль ризикових обмежень. На відміну від окремих сервісів автоматизованої торгівлі, така платформа орієнтована на модульну архітектуру та можливість подальшого розширення під конкретні торгові сценарії [14; 15; 16; 17; 18].

Особливо важливим елементом системи є модуль Risk Manager. Він дозволяє створювати профілі для бірж OKX і Binance, задавати ліміти втрат, переглядати баланс, доступний баланс, максимальний баланс, PnL, серію збиткових угод і статус блокування профілю. Завдяки використанню watcher-сервісів і WebSocket користувач може отримувати актуальні дані без ручного оновлення сторінки, а

Telegram-сповіщення дозволяють повідомляти про критичні події навіть поза web-інтерфейсом [22; 24].

Мета роботи – спрощення процесу керування автоматизованою криптовалютною торгівлею за рахунок розробки web-платформи CryptoKeps для централізованого моніторингу торгових ботів, біржових акаунтів, webhook-сигналів, історії угод, торгової аналітики та risk-профілів.

Об’єкт дослідження – процеси автоматизованого керування криптовалютною торгівлею, торговими ботами, біржовими акаунтами, торговими сигналами та ризиковими обмеженнями.

Предмет дослідження – засоби, методи та технології реалізації web-платформи CryptoKeps для керування криптовалютними торговими ботами, обробки webhook-сигналів, моніторингу біржових профілів, аналізу результатів торгівлі та контролю ризиків.

Для досягнення поставленої мети необхідно вирішити такі завдання:

1. Проаналізувати предметну галузь автоматизованої криптовалютної торгівлі та визначити основні проблеми керування торговими ботами, сигналами, біржовими акаунтами й ризиками [1; 5; 8].
2. Дослідити існуючі програмні рішення для автоматизованої торгівлі, порівняти їх функціональні можливості та визначити основні недоліки [14; 15; 16; 17; 18].
3. Сформувати функціональні та нефункціональні вимоги до платформи CryptoKeps.
4. Обґрунтувати вибір технологій для реалізації web-платформи, зокрема Java Spring Boot, React, PostgreSQL, Django, WebSocket та Docker Compose [19; 20; 21; 22; 23; 24].
5. Спроєктувати архітектуру платформи, структуру бази даних та основні UML-діаграми системи.
6. Реалізувати серверну частину для керування торговими ботами, обробки webhook-сигналів, збереження історії угод і розрахунку торгової аналітики.

7. Реалізувати клієнтський web-інтерфейс для перегляду ботів, акаунтів, аналітики та Risk Manager.
8. Розробити модуль Risk Manager для live-моніторингу біржових профілів OKX і Binance, контролю ризикових обмежень та Telegram-сповіщень.
9. Провести тестування основних функцій платформи та оцінити відповідність розробленого програмного забезпечення поставленим вимогам.

Методи дослідження. У бакалаврській роботі використовуються такі методи: аналіз предметної галузі, порівняльний аналіз існуючих програмних засобів автоматизованої торгівлі, об'єктно-орієнтоване проєктування, UML-моделювання, проєктування структури реляційної бази даних, розробка клієнт-серверного web-платформи, реалізація REST API, робота з WebSocket-з'єднаннями, тестування програмного забезпечення та аналіз результатів роботи системи.

Наукова новизна одержаних результатів полягає у розробці структури web-платформи CryptoKeps, яка поєднує керування криптотрейдинговими ботами, обробку webhook-сигналів, торгову аналітику та окремий модуль Risk Manager. На відміну від типових сервісів автоматизованої торгівлі, запропоноване рішення орієнтоване не лише на запуск ботів, а й на централізований контроль торгових акаунтів, live-моніторинг ризикових параметрів і сповіщення користувача про критичні події.

Практична значущість одержаних результатів полягає в тому, що розроблена web-платформа може бути використана як основа для подальшого розвитку системи автоматизованої криптовалютної торгівлі. CryptoKeps дозволяє об'єднати в одному інтерфейсі перегляд ботів, акаунтів, сигналів, історії угод, торгових KPI та risk-профілів. У майбутньому систему можна розширити додаванням нових бірж, нових торгових стратегій, розширеної аналітики, рольової моделі доступу, журналу дій користувача, генерації PDF або Excel-звітів і повноцінного виконання торгових операцій через біржові API.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ АВТОМАТИЗОВАНОЇ КРИПТОВАЛЮТНОЇ ТОРГІВЛІ

1.1 Аналіз процесу автоматизованої криптовалютної торгівлі

Автоматизована криптовалютна торгівля є одним із напрямів розвитку сучасних фінансово-технологічних систем. Її сутність полягає у використанні програмних засобів, які здатні виконувати частину торгових дій без постійної участі користувача. До таких дій належать отримання торгових сигналів, аналіз стану ринку, відкриття та закриття позицій, збереження історії угод, розрахунок торгових показників і контроль ризикових обмежень [1; 2; 5; 6].

На відміну від ручної торгівлі, де трейдер самостійно аналізує графіки, приймає рішення та виконує операції на біржі, автоматизована система дозволяє частково перенести ці процеси на програмні модулі. Це особливо актуально для криптовалютного ринку, оскільки він працює безперервно, не має класичних торгових сесій і характеризується високою волатильністю. Через це користувач не завжди може вручну відстежувати зміну ціни, появу сигналів, стан відкритих позицій і ризикові ситуації [1; 2; 3; 4; 7].

У межах даної роботи розглядається платформа CryptoKeps, призначена для керування криптотрейдинговими ботами, торговими акаунтами, webhook-сигналами, історією угод, аналітикою та risk-профілями. Платформа має об’єднати в одному середовищі ті процеси, які в реальних умовах часто виконуються через різні сервіси: біржовий кабінет, окремий бот, таблиці для аналізу, Telegram-сповіщення та сторонні інструменти моніторингу [8; 9; 10; 11].

Основна ідея такої системи полягає в тому, щоб користувач міг через єдиний web-інтерфейс отримувати інформацію про стан торгової системи, переглядати ботів, аналізувати результати угод, контролювати біржові профілі та швидко реагувати на ризикові події. Це дозволяє зменшити кількість ручних дій, скоротити

час аналізу торгової ситуації та підвищити загальну керованість автоматизованої торгівлі [8; 11; 13].

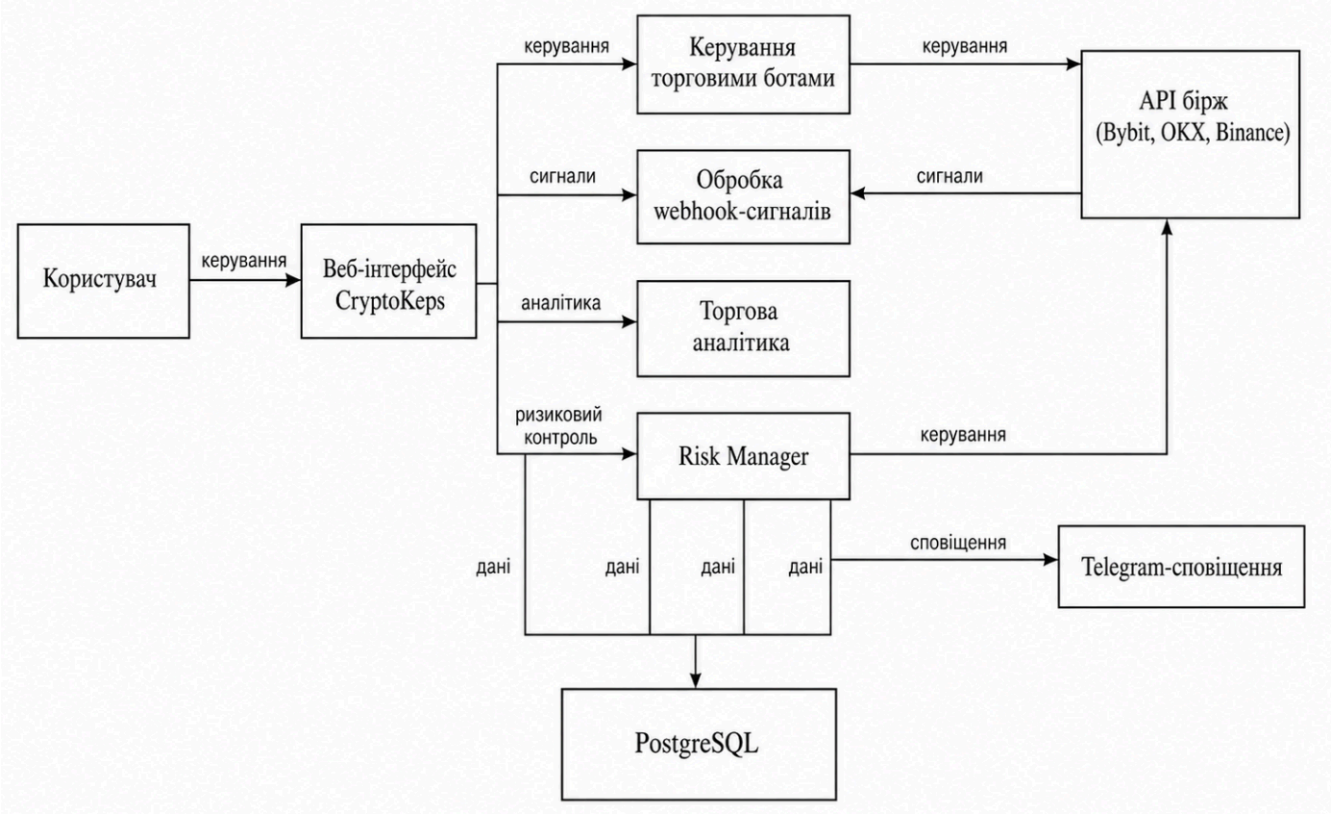


Рис. 1.1 Загальна схема процесу автоматизованої криптовалютної торгівлі

1.2 Особливості керування криптовалютними торговими ботами

Торговий бот у системах автоматизованої торгівлі є програмним компонентом, який працює за заданою логікою або стратегією. Він може реагувати на сигнали, аналізувати стан індикаторів, виконувати торгові дії або передавати інформацію для подальшої обробки. У платформі CryptoKeps бот розглядається не як ізольований скрипт, а як частина загальної системи керування торговими процесами [8; 9; 10].

Кожен торговий бот може мати власну назву, тип стратегії, торгову пару, таймфрейм, статус роботи та прив’язку до конкретного біржового акаунта. Для зручного керування такими ботами система повинна забезпечувати їх перегляд, пошук, фільтрацію, відображення основних показників і перехід до детальної

інформації. Якщо кількість ботів збільшується, ручний контроль кожної стратегії стає незручним, тому важливо мати централізований інтерфейс.

У процесі керування ботами користувачеві важливо бачити не лише факт існування стратегії, а й її реальний стан. Наприклад, бот може бути активним, призупиненим або перебувати в помилковому стані. Крім того, для оцінки ефективності стратегії потрібно бачити PnL, кількість угод, win rate, просадку, середнє співвідношення ризику до прибутку та історію останніх операцій.

У платформі CryptoKeps передбачено сторінку ботів, яка дозволяє переглядати список торгових стратегій, аналізувати їхні показники та взаємодіяти з окремими ботами. Такий підхід дає змогу швидше оцінити, які стратегії працюють ефективно, які потребують зупинки або додаткової перевірки, а які можуть бути використані для подальшого розвитку.

1.3 Обробка торгових сигналів та webhook-повідомлень

Одним із важливих елементів автоматизованої криптовалютної торгівлі є обробка торгових сигналів. Сигнал може надходити від зовнішнього сервісу, наприклад TradingView, або від іншого аналітичного інструменту. Зазвичай він містить інформацію про торгову пару, напрям угоди, таймфрейм, індикатор, ціну, значення ATR або інші параметри, які використовуються торговою логікою [18].

Webhook-повідомлення дозволяють передавати торгові сигнали між різними системами автоматично. Це означає, що після спрацювання певної умови у зовнішньому сервісі сигнал може бути надісланий на сервер платформи CryptoKeps, де він проходить перевірку, нормалізацію та збереження. Такий підхід дозволяє автоматизувати передачу торгових рішень і зменшити потребу в ручному введенні даних [18].

Важливою проблемою при роботі з webhook-сигналами є захист від повторної обробки однакових повідомлень. Якщо один і той самий сигнал буде оброблено декілька разів, це може призвести до помилкового оновлення стану стратегії або неправильного відкриття угоди. Для вирішення цієї проблеми

використовується механізм idempotency key, який дозволяє визначити, чи був сигнал уже оброблений раніше [18].

У платформі CryptoKeps сигнал після отримання передається до серверної частини, де проходить кілька етапів: приймання запиту, перевірка даних, нормалізація, формування унікального ключа, збереження в базі даних, оновлення стану індикаторів і перерахунок готовності бота до торгової дії. Така послідовність забезпечує контрольовану обробку сигналів і дозволяє підтримувати актуальний стан торгової стратегії.

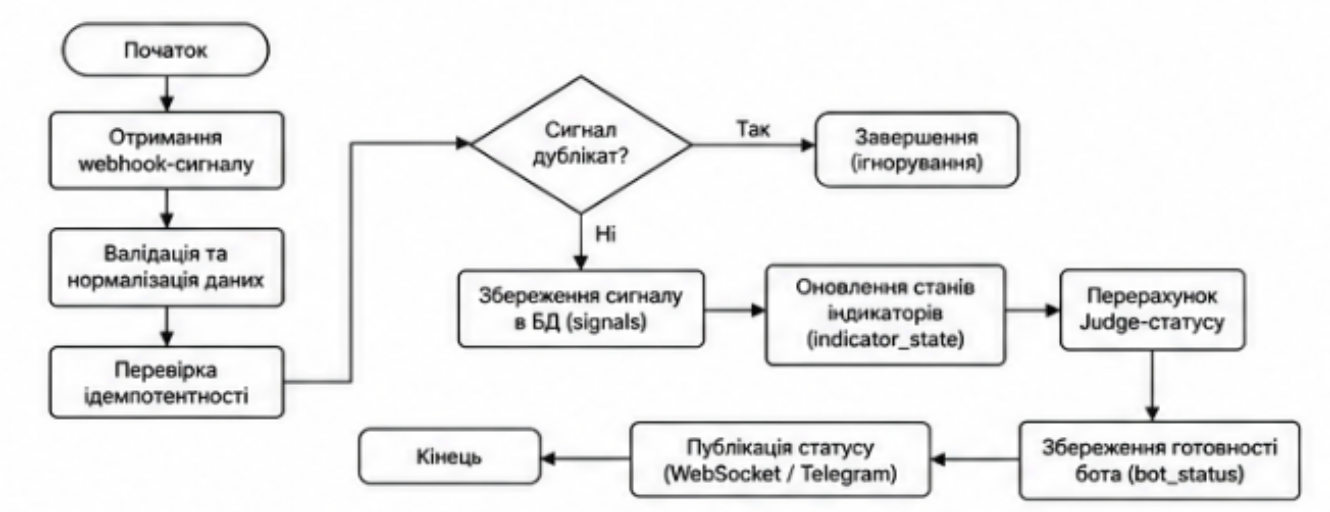


Рис. 1.2 Блок-схема серверної обробки торгового сигналу в платформі CryptoKeps

1.4 Роль risk-management у системах автоматизованої торгівлі

Risk-management є одним із ключових елементів автоматизованої криптовалютної торгівлі. Навіть якщо торгова стратегія має позитивні результати на певному проміжку часу, відсутність контролю ризиків може призвести до значних втрат. Для криптовалютного ринку ця проблема є особливо важливою через високу волатильність, різкі рухи ціни та можливість швидкої зміни ліквідності [5; 6; 7].

У системах автоматизованої торгівлі risk-management повинен контролювати не лише одну окрему угоду, а й загальний стан біржового профілю. До таких параметрів можна віднести поточний баланс, доступний баланс, максимальний

баланс, PnL, кількість збиткових угод поспіль, активність торгівлі та статус блокування профілю. Якщо один із ризикових показників перевищує допустиме значення, система повинна повідомити користувача або автоматично обмежити подальшу роботу профілю [8; 11; 24].

У платформі CryptoKeps для цього передбачено модуль Risk Manager. Він дозволяє створювати risk-профілі для бірж OKX і Binance, задавати ліміти збитку, запускати або зупиняти профілі, переглядати live-дані та контролювати поточний стан торгової активності. Окремі watcher-сервіси отримують дані з біржових API, перевіряють ризикові обмеження та передають оновлення у web-інтерфейс [24].

Особливістю такого підходу є те, що risk-management винесено в окремий функціональний модуль. Це дозволяє не змішувати логіку торгових ботів із контролем біржових профілів. У результаті система стає більш структурованою: Java Backend відповідає за торгову логіку, сигнали та аналітику, а Django Backend разом із watcher-сервісами забезпечує роботу з профілями ризику.

Рис. 1.3 Діаграма роботи Risk Manager та watcher-сервісів OKX і Binance

1.5 Аналіз існуючих програмних рішень для керування криптотрейдинговими ботами

Для обґрунтування доцільності розробки платформи CryptoKeps було розглянуто існуючі програмні рішення, які використовуються для автоматизованої криптовалютної торгівлі. До таких рішень належать 3Commas, Cryptohopper, Coinrule, WunderTrading і TradingView. Кожна з цих платформ має власне призначення та набір функцій, однак не всі вони повністю закривають потреби централізованого керування ботами, сигналами, аналітикою та ризиками [14; 15; 16; 17; 18].

Платформа 3Commas дозволяє створювати торгових ботів, підключати біржові акаунти, використовувати DCA- та grid-стратегії, а також працювати з інструментами напівавтоматичної торгівлі. Її перевагою є готовий функціонал і підтримка багатьох бірж. Водночас користувач залежить від тарифних планів і логіки, яку пропонує сам сервіс [14].

Cryptohopper також орієнтований на автоматизовану торгівлю та підтримує створення ботів, використання стратегій, backtesting і marketplace. Це зручно для користувачів, які хочуть швидко використати готові шаблони, але така модель не завжди підходить для реалізації власної серверної логіки або специфічних механізмів risk-management [15].

Coinrule пропонує створення торгових правил за принципом “якщо умова виконується — виконати дію”. Такий підхід зручний для простих сценаріїв, але може бути недостатнім для складних стратегій, де потрібно враховувати стан декількох індикаторів, історію сигналів, власні обмеження ризику та інтеграцію з окремими backend-сервісами [16].

WunderTrading добре підходить для роботи з TradingView-сигналами та автоматичного виконання торгових дій. Однак його можливості risk-management також залишаються обмеженими порівняно з окремим модулем контролю біржових профілів. TradingView, у свою чергу, є потужним інструментом для

аналізу графіків і створення сигналів, але не є повноцінною платформою для керування ботами, акаунтами та ризиками [17; 18].

Порівняно з цими рішеннями, CryptoKeps орієнтована на поєднання декількох напрямів: керування торговими ботами, приймання webhook-сигналів, збереження історії угод, розрахунок торгових KPI, live-моніторинг профілів, контроль ризикових обмежень і Telegram-сповіщення. Це робить систему більш гнучкою для подальшого розвитку та адаптації під конкретні торгові сценарії [14; 15; 16; 17; 18].

Таблиця 1.1

Порівняльний аналіз програмних засобів

Показник	3Commas	Cryptohopper	WunderTrading	TradingView	CryptoKeps
Підключення бірж через API	+	+	+	частково	+
Webhook-сигнали	+	+	+	+	+
Інтеграція з TradingView	+	+	+	власна система alert	+
Торгова аналітика	+	+	+	+	+
Risk-management	частково	частково	частково	-	+
Live-моніторинг балансів	частково	частково	частково	-	+
Telegram-сповіщення	частково	-	частково	через зовнішні інтеграції	+

Продовження таблиці 1.1

Порівняльний аналіз програмних засобів

Показник	3Commas	Cryptohopper	WunderTrading	TradingView	CryptoKeps
Гнучкість власної серверної логіки	-	-	-	частково через Pine Script	+
Підтримка декількох бірж	+	+	+	+	+

1.6 Постановка проблеми та обґрунтування необхідності розробки

Проведений аналіз предметної галузі показує, що основною проблемою автоматизованої криптовалютної торгівлі є розрізненість інструментів. Часто один сервіс використовується для створення сигналів, інший — для запуску бота, третій — для перегляду біржового акаунта, а результати торгівлі можуть додатково фіксуватися в таблицях або окремих аналітичних сервісах. Такий підхід ускладнює контроль торгової системи та збільшує ризик помилок [14; 15; 16; 17; 18].

Окремою проблемою є контроль ризиків. Багато готових платформ мають лише базові інструменти risk-management, наприклад stop loss або take profit у межах конкретної угоди [8; 11; 13].

Крім цього, у межах розробки необхідно врахувати, що користувачеві важливо не просто отримувати окремі торгові дані, а бачити цілісну картину стану системи. Для цього платформа має об’єднувати інформацію про активність ботів, підключені акаунти, отримані сигнали, історію угод, поточні ризикові показники та сповіщення про критичні події. Такий підхід дозволяє зменшити кількість ручних перевірок і забезпечити більш оперативне прийняття рішень у процесі автоматизованої торгівлі.

Саме тому розробка платформи CryptoKeps є доцільною. Вона дозволяє об’єднати в одному програмному середовищі web-інтерфейс, Java Backend для торгової логіки й аналітики, Django Backend для авторизації та risk-профілів, PostgreSQL для централізованого збереження даних, Gateway для маршрутизації запитів і watcher-сервіси для live-моніторингу біржових профілів.

Майбутня платформа повинна забезпечити такі ключові можливості: авторизацію користувача, перегляд загального стану торгової системи, керування ботами, перегляд акаунтів, обробку webhook-сигналів, розрахунок торгових показників, збереження історії угод, керування risk-профілями, live-оновлення балансів і Telegram-сповіщення про критичні події.

Додатково важливо врахувати, що ефективність автоматизованої торгівлі залежить не лише від якості торгової стратегії, а й від організації взаємодії між окремими компонентами системи. Якщо сигнали, торгові акаунти, боти, історія угод і risk-профілі не пов’язані між собою в єдиній структурі, користувач змушений витрачати більше часу на перевірку стану системи та може пропустити критичні зміни. Тому для CryptoKeps важливою є не тільки наявність окремих функцій, а й їх узгоджена робота в межах єдиної архітектури.

Для наочного відображення цієї ідеї доцільно представити узагальнену схему функціональних напрямів платформи CryptoKeps. На схемі необхідно показати зв’язок між frontend-інтерфейсом, gateway, Java Backend, Django Backend, PostgreSQL, біржовими API, Telegram Bot, webhook-джерелами та модулями моніторингу. Така схема дозволяє зрозуміти, як окремі частини платформи взаємодіють між собою та як дані проходять від зовнішнього сигналу або біржі до користувацького інтерфейсу.

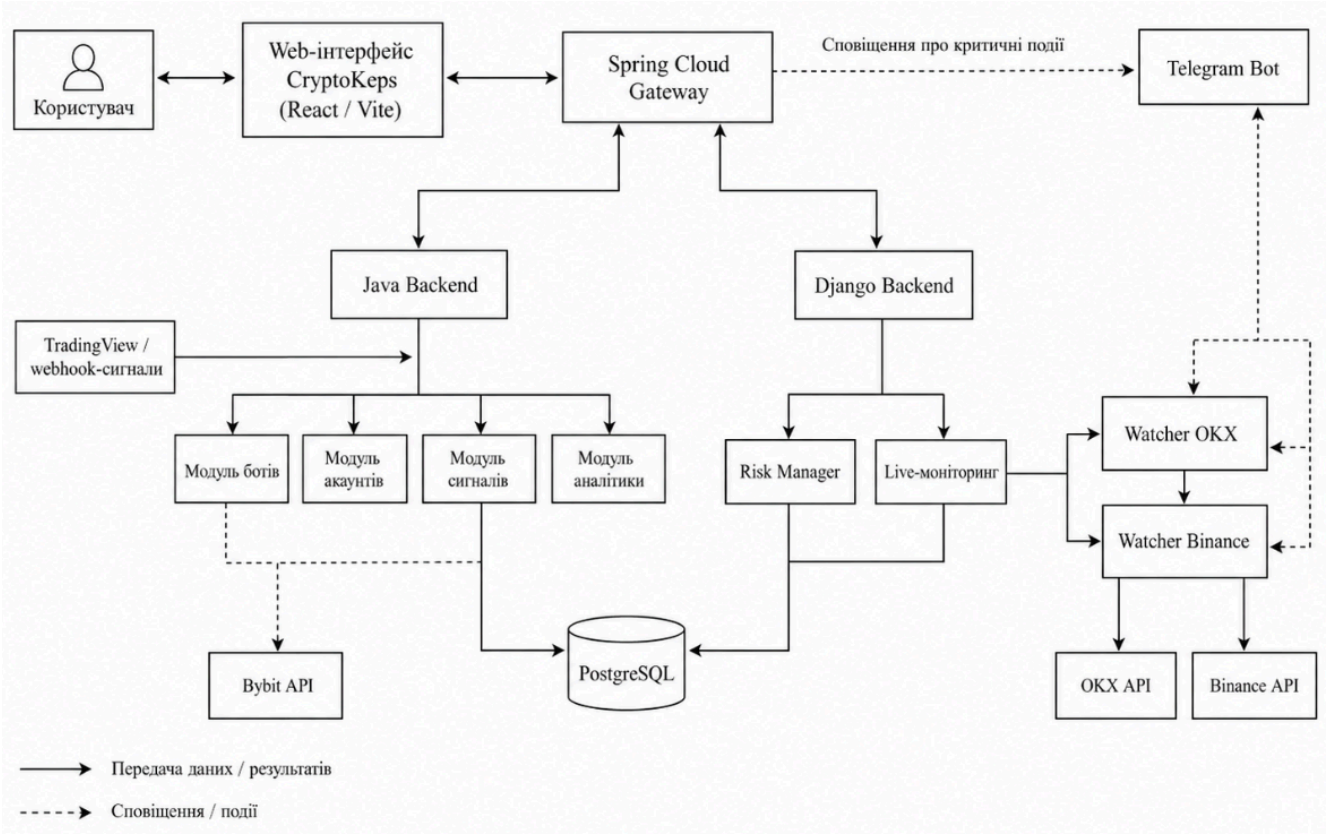


Рис. 1.4 Узагальнена схема функціональних напрямів платформи CryptoKeps

Окрему роль у цій структурі відіграють зовнішні інтеграції: TradingView або інші джерела webhook-сигналів, біржові API Bybit, OKX і Binance, а також Telegram Bot для сповіщень. Завдяки такій організації CryptoKeps може не лише відображати стан торгової системи, а й забезпечувати повний ланцюг роботи з даними: від отримання сигналу або біржового оновлення до збереження інформації, розрахунку показників, перевірки ризикових обмежень і повідомлення користувача про критичні події.

Отже, предметна галузь автоматизованої криптовалютної торгівлі потребує комплексного програмного рішення, яке поєднує керування торговими ботами, обробку сигналів, аналітику, моніторинг і risk-management. Платформа CryptoKeps спрямована на вирішення цієї задачі за рахунок модульної архітектури, централізованого збереження даних [8; 9; 10; 11; 13].

2 ВИБІР ТЕХНІЧНИХ ЗАСОБІВ ДЛЯ РЕАЛІЗАЦІЇ ПЛАТФОРМИ CRYPTOKEPS

2.1 Визначення функціональних та нефункціональних вимог

Платформа CryptoKeps розробляється як web-система для керування криптотрейдинговими ботами, торговими акаунтами, webhook-сигналами, історією угод, аналітикою та risk-профілями. Оскільки система має працювати з декількома напрямами одночасно, на етапі вибору технічних засобів важливо визначити основні вимоги до майбутнього програмного забезпечення [8; 11; 13].

Функціональні вимоги описують дії, які повинна виконувати платформа. Для CryptoKeps ключовими функціями є авторизація користувача, перегляд загального стану системи, робота з торговими ботами, перегляд торгових акаунтів, обробка зовнішніх сигналів, відображення історії угод, розрахунок торгових показників, керування risk-профілями та отримання live-даних.

Платформа повинна забезпечувати можливість перегляду списку ботів із відображенням їхнього статусу, торгової пари, таймфрейму, біржового акаунта та основних показників ефективності. Для зручності користувача необхідно передбачити пошук і фільтрацію ботів за назвою, біржею, акаунтом або статусом. Це дозволяє швидко знаходити потрібну стратегію навіть у випадку, якщо в системі використовується багато торгових ботів.

Окремою функцією є приймання webhook-сигналів від зовнішніх сервісів, зокрема TradingView або інших аналітичних інструментів. Після отримання сигналу система повинна перевірити його, нормалізувати дані, сформувати унікальний ключ для захисту від дублювання, зберегти сигнал у базі даних і оновити стан відповідної торгової логіки [18].

Також платформа повинна зберігати історію угод. Для кожної угоди доцільно фіксувати торговий акаунт, символ, сторону операції, кількість, ціну входу, ціну виходу, stop loss, take profit, PnL і час виконання. На основі цих даних система має

формувати торгову аналітику: сумарний PnL, win rate, кількість угод, максимальну просадку та інші показники.

Важливою частиною системи є модуль Risk Manager. Він повинен забезпечувати створення, редагування, запуск і зупинку risk-профілів для бірж OKX і Binance. Для кожного профілю користувач має можливість задати ризикові обмеження: максимальний відсоток втрати, граничну суму збитку, кількість збиткових угод поспіль та інші параметри. При перевищенні лімітів система повинна блокувати профіль або повідомляти користувача про критичну ситуацію.

До нефункціональних вимог належать вимоги до швидкодії, надійності, безпеки, масштабованості та зручності використання. Основні сторінки платформи повинні завантажуватися не довше ніж за 3 секунди, типові API-запити мають оброблятися до 1 секунди, а live-дані через WebSocket повинні оновлюватися кожні 1–5 секунд. Система має підтримувати не менше 50 торгових ботів, 100 risk-профілів і зберігати не менше 10 000 торгових операцій. API-ключі біржових акаунтів повинні зберігатися у захищеному вигляді, а помилка одного зовнішнього сервісу не повинна зупиняти роботу всієї платформи.

2.2 Вибір мови програмування та фреймворку

Для реалізації основної серверної частини платформи CryptoKeps обрано мову програмування Java та фреймворк Spring Boot. Такий вибір є доцільним, оскільки серверна частина системи повинна виконувати значний обсяг бізнес-логіки: обробляти торгових ботів, сигнали, акаунти, історію угод, торгову аналітику та інтеграцію з біржовими API [19].

Java є однією з найпоширеніших мов програмування для створення корпоративних і серверних застосунків. Вона підтримує об’єктно-орієнтований підхід, має велику екосистему бібліотек, стабільну роботу з багатопотоковістю та добре підходить для довготривалих backend-сервісів. Для платформи CryptoKeps це важливо, оскільки серверна частина повинна працювати стабільно, обробляти запити від frontend-частини та взаємодіяти з базою даних [19].

Spring Boot використовується для спрощення розробки backend-застосунку. Він дозволяє швидко створювати REST API, підключати базу даних, організовувати структуру проєкту за шарами та налаштовувати взаємодію між компонентами. У платформі CryptoKeps Spring Boot доцільно застосувати для реалізації контролерів, сервісів, репозиторіїв, DTO-моделей і сутностей бази даних [19].

Серверна частина Java Backend повинна містити окремі контролери для роботи з ботами, акаунтами, торговими сигналами, історією угод і статусами стратегій. Наприклад, BotController може відповідати за отримання списку ботів і перегляд їхніх показників, WebhookController — за приймання торгових сигналів, AccountInfoController — за роботу з біржовими акаунтами, а TradeController — за історію торгових операцій.

Бізнес-логіка виноситься в сервісний рівень. Саме сервіси відповідають за обробку webhook-сигналів, формування idempotency key, оновлення стану індикаторів, розрахунок готовності Judge-логіки, обчислення KPI та підготовку даних для frontend-частини. Такий підхід дозволяє відокремити приймання HTTP-запитів від основної логіки системи.

Для доступу до бази даних доцільно використовувати Spring Data JPA. Він дозволяє працювати з таблицями PostgreSQL через Java-сутності та репозиторії, що спрощує розробку й підтримку коду. Наприклад, для таблиць bots, trade_history, signals, indicator_state і bot_status можуть бути створені відповідні entity-класи та repository-інтерфейси [19; 22].

2.3 Вибір засобів реалізації клієнтського web-інтерфейсу

Для реалізації клієнтської частини платформи CryptoKeps обрано React, TypeScript та Vite. Ці засоби дозволяють створити сучасний, швидкий і зручний web-інтерфейс для роботи з торговими ботами, акаунтами, аналітикою, risk-профілями та профілем користувача [20; 21].

React є популярною бібліотекою для побудови інтерфейсів користувача. Його основною перевагою є компонентний підхід. Кожна частина інтерфейсу може бути оформлена як окремий компонент: таблиця ботів, картка акаунта, графік аналітики, форма входу, блок risk-профілю або навігаційна панель. Це спрощує підтримку проєкту, оскільки окремі компоненти можна змінювати або розширювати без повного переписування всього інтерфейсу [20].

TypeScript використовується для підвищення надійності frontend-коду. Оскільки платформа працює з великою кількістю структурованих даних, важливо чітко описати типи об’єктів: ботів, акаунтів, сигналів, угод, risk-профілів, користувачів і відповідей API. Це зменшує кількість помилок під час розробки та полегшує взаємодію frontend-частини з backend-сервісами [20].

Vite застосовується як інструмент для запуску й збирання frontend-проєкту. Він забезпечує швидкий старт локального середовища розробки, зручне оновлення сторінки під час зміни коду та оптимізоване збирання фінальної версії застосунку. Для CryptoKeps це зручно, оскільки frontend містить декілька сторінок і багато динамічних елементів [21].

Клієнтський інтерфейс CryptoKeps повинен включати декілька основних сторінок. Сторінка Login відповідає за авторизацію користувача. Сторінка Monitor показує загальний стан торгової системи. Сторінка Bots призначена для перегляду ботів, їхніх KPI та історії угод. Сторінка Accounts дозволяє переглядати торгові акаунти. Сторінка Risk Manager використовується для роботи з risk-профілями, а сторінка Profile — для перегляду й редагування даних користувача.

Окрему роль відіграє навігаційна панель. Вона повинна забезпечувати швидкий перехід між основними розділами платформи, відображати активну сторінку та надавати доступ до профілю користувача. Також у платформі передбачено перемикання світлої та темної теми інтерфейсу, що підвищує зручність використання системи.

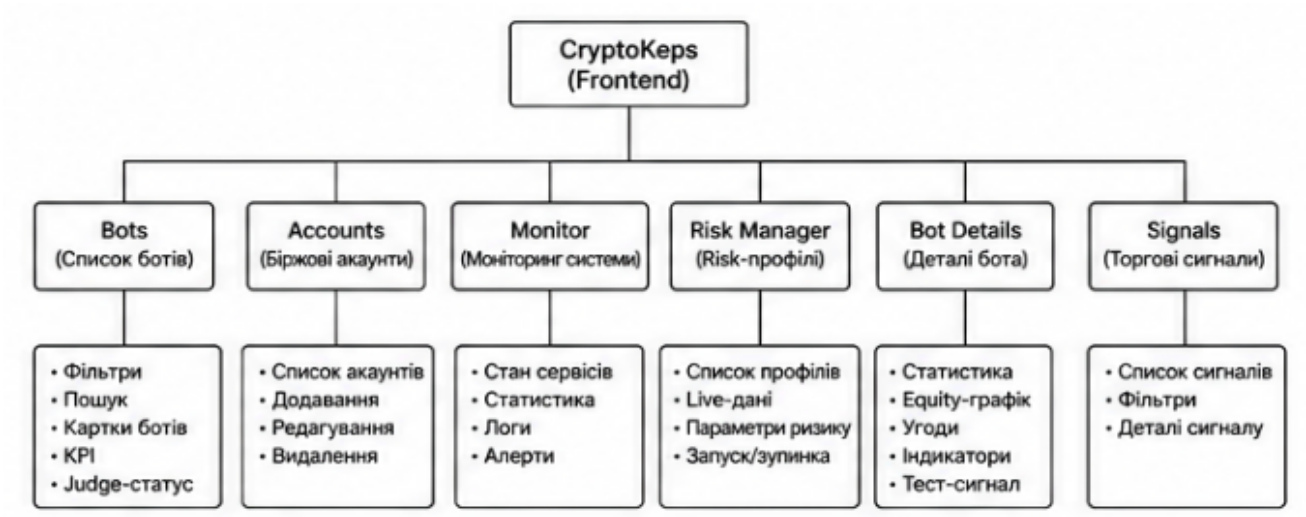


Рис. 2.1 Структура клієнтського web-інтерфейсу платформи CryptoKeps

2.4 Вибір бази даних та засобів керування міграціями

Для збереження даних платформи CryptoKeps обрано PostgreSQL. Це реляційна система керування базами даних, яка добре підходить для зберігання структурованої інформації та підтримує зв’язки між сутностями. Для даного проєкту це важливо, оскільки система містить багато пов’язаних об’єктів: користувачів, акаунти, ботів, сигнали, історію угод, стани індикаторів, статуси ботів, watcher-сервери та risk-профілі [22].

PostgreSQL дозволяє організувати централізоване збереження даних для різних частин платформи. Java Backend використовує базу для торгових акаунтів, ботів, сигналів, історії угод і торгової аналітики. Django Backend може використовувати її для зберігання користувачів, профілів Risk Manager, watcher-серверів і періодів доступу. Завдяки єдиному сховищу дані системи залишаються узгодженими.

Для торгової частини системи можуть бути використані таблиці account_info, bots, trade_history, signals, indicator_state і bot_status. Таблиця account_info зберігає біржові акаунти, bots — торгових ботів, trade_history — історію угод, signals — отримані сигнали, indicator_state — стан індикаторів, a bot_status — готовність бота до торгової дії.

Для модуля Risk Manager використовуються таблиці, пов’язані з користувачами та risk-профілями. До них можна віднести auth_user, tw_profile, tw_server і tw_access_period. Таблиця tw_profile містить параметри risk-профілю, зокрема біржу, назву, API-ключі, ліміти збитку, refresh time і прив’язку до watcher-сервера.

Для керування змінами структури бази даних у Java Backend доцільно використовувати Liquibase. Він дозволяє описувати зміни бази даних у вигляді міграцій. Це означає, що створення таблиць, додавання нових полів або зміна зв’язків між сутностями може виконуватися контрольовано й послідовно [23].

Використання Liquibase особливо корисне під час розвитку проєкту. Якщо в майбутньому потрібно буде додати нову таблицю для сповіщень, розширити структуру торгової історії або змінити параметри ботів, ці зміни можна оформити як окремі changeset-и. Це полегшує підтримку системи та дозволяє відстежувати історію змін структури бази даних.

2.5 Вибір засобів маршрутизації та інтеграції з біржовими API

Оскільки платформа CryptoKeps складається з декількох сервісів, необхідно забезпечити зручну маршрутизацію запитів між frontend-частиною, Java Backend, Django Backend і watcher-сервісами. Для цього використовується Spring Cloud Gateway, який виконує роль єдиної точки входу до системи [19].

Gateway дозволяє приховати внутрішню структуру платформи від клієнтської частини. Frontend може звертатися до одного базового адреса, а Gateway вже перенаправляє запити до потрібного сервісу. Наприклад, запити до Java Backend можуть проходити через маршрут /api/java/..., запити до Django Backend — через /api/py/dj/..., а WebSocket-з’єднання watcher-сервісів — через відповідні маршрути для OKX і Binance.

Такий підхід спрощує масштабування системи. Якщо в майбутньому буде додано новий backend-сервіс або нову біржову інтеграцію, її можна підключити через окремий маршрут Gateway без зміни логіки frontend-частини. Крім того,

Gateway може використовуватися для централізованої обробки заголовків, ідентифікації запитів, логування та базових механізмів безпеки [19; 24].

Для передавання live-даних використовується WebSocket. На відміну від звичайного HTTP-запиту, WebSocket дозволяє підтримувати постійне з'єднання між клієнтом і сервером. Це необхідно для сторінки Risk Manager, де користувач повинен бачити актуальний баланс, PnL, доступний баланс, максимальний баланс, статус блокування, серію збитків і активність торгівлі [24].

Інтеграція з біржовими API є важливою частиною платформи. У структурі системи передбачено взаємодію з Bybit API, OKX API і Binance API. Bybit може використовуватися для торгової логіки, акаунтів, ордерів та історії угод. OKX і Binance використовуються watcher-сервісами для отримання live-даних по профілях Risk Manager [24].

Також у системі передбачено використання Telegram Bot API для надсилання повідомлень користувачу. Telegram-сповіщення потрібні для інформування про критичні події: перевищення ризикового ліміту, блокування профілю, закриття позиції або інші ситуації, які потребують уваги користувача [24].

2.6 Вибір інструментів контейнеризації, тестування та контролю версій

Для запуску платформи CryptoKeps як набору окремих сервісів доцільно використовувати Docker Compose. Система складається з кількох компонентів: frontend, gateway, Java Backend, Django Backend, PostgreSQL, OKX watcher, Binance watcher і Telegram Bot service. Запускати всі ці компоненти вручну окремо незручно, тому контейнеризація дозволяє спростити процес розгортання [19; 22].

Docker Compose дає можливість описати всі сервіси в одному конфігураційному файлі. Для кожного сервісу можна визначити образ, порти, змінні середовища, залежності та параметри запуску. Завдяки цьому платформа може бути швидко запущена в тестовому або демонстраційному середовищі [19; 22].

Контейнеризація також полегшує подальший розвиток системи. Якщо потрібно оновити тільки frontend або перезапустити watcher-сервіс, це можна зробити без повного перезапуску всієї платформи. Крім того, Docker Compose дозволяє ізолювати середовище виконання, що зменшує залежність від налаштувань конкретного комп’ютера розробника.

Для тестування серверної частини можуть використовуватися стандартні засоби Java та Spring Boot. Вони дозволяють перевіряти роботу контролерів, сервісів, репозиторіїв і логіки обробки сигналів. Окремо доцільно тестувати обробку webhook-сигналів, формування idempotency key, збереження сигналів у базі даних, розрахунок KPI та роботу з історією угод.

Frontend-частина також потребує тестування. Необхідно перевірити коректність роботи сторінок Login, Monitor, Bots, Accounts, Risk Manager і Profile. Важливо переконатися, що система правильно відображає дані, обробляє помилки API, перемикає теми інтерфейсу та не порушує навігацію між сторінками.

Для контролю версій використовується Git. Він дозволяє зберігати історію змін коду, працювати з гілками, повертатися до попередніх версій і організовувати командну розробку. Завантаження проєкту на GitHub або іншу платформу контролю версій є важливим етапом, оскільки це забезпечує зручне зберігання коду та можливість подальшого супроводу системи.

Отже, для реалізації платформи CryptoKeps обрано набір технічних засобів, які відповідають її архітектурі та функціональним вимогам. Java Spring Boot забезпечує серверну логіку, React і TypeScript — клієнтський інтерфейс, PostgreSQL — збереження даних, Liquibase — керування змінами бази, Django — модуль користувачів і risk-профілів, Spring Cloud Gateway — маршрутизацію запитів, WebSocket — live-оновлення, Docker Compose — контейнеризоване розгортання, а Git — контроль версій проєкту [19].

3 ПРОЄКТУВАННЯ ПЛАТФОРМИ КЕРУВАННЯ КРИПТОТРЕЙДИНГОВИМИ БОТАМИ

3.1 Проєктування загальної архітектури платформи

Платформа CryptoKeps проєктується як багатокомпонентна web-платформа, призначена для централізованого керування криптотрейдинговими ботами, торговими акаунтами, сигналами, аналітичними показниками та risk-профілями. Основна ідея архітектури полягає в розділенні функцій між окремими модулями, кожен з яких відповідає за власну частину системи.

На верхньому рівні користувач взаємодіє з платформою через web-інтерфейс, реалізований на React та Vite. Через цей інтерфейс користувач може авторизуватися, переглядати сторінку моніторингу, працювати зі списком торгових ботів, аналізувати торгові акаунти, переглядати risk-профілі та налаштовувати власний профіль.

Запити з клієнтської частини надходять до Gateway, який виконує роль єдиної точки входу. Його задача полягає в маршрутизації запитів між різними backend-сервісами. Наприклад, запити, пов’язані з ботами, акаунтами, торговими сигналами та історією угод, передаються до Java Backend. Запити, пов’язані з авторизацією, профілем користувача та risk-профілями, передаються до Django Backend [19; 20; 21; 22; 23; 24].

Java Backend відповідає за торгову частину платформи. Він обробляє дані про торгові акаунти, ботів, webhook-сигнали, історію угод, статуси ботів і торгову аналітику. У цьому модулі реалізується логіка приймання сигналів, перевірки їх унікальності, оновлення станів індикаторів, розрахунку готовності бота та формування KPI.

Django Backend використовується для роботи з користувачами та risk-профілями. Він забезпечує авторизацію, збереження профілю користувача, керування TW/risk-профілями та зв’язок із watcher-сервісами. Такий поділ дозволяє

відокремити торгову логіку від логіки керування користувачами й ризиковими параметрами.

Окремими компонентами системи є watcher-сервіси для Binance та OKX. Вони призначені для отримання live-даних із біржових API, перевірки стану профілів, контролю балансу, PnL, серії збитків і статусу активної торгівлі. У разі виникнення критичної ситуації система може сформувати повідомлення для користувача через Telegram Bot.

Усі основні дані платформи зберігаються в PostgreSQL. База даних містить інформацію про користувачів, торгові акаунти, ботів, сигнали, історію угод, статуси ботів, стани індикаторів, watcher-сервери та risk-профілі. Така централізація дозволяє забезпечити цілісність даних і використовувати їх для аналітики, моніторингу та подальшого розширення платформи.

3.2 Проєктування логічної структури системи

Логічна структура CryptoKeps відображає поділ системи на основні рівні та функціональні модулі. Для зручності проєктування платформу можна поділити на три основні рівні: рівень представлення, рівень бізнес-логіки та рівень даних [19].

Рівень представлення відповідає за взаємодію користувача з платформою. До нього належить web-інтерфейс трейдера, оператора або адміністратора. Через цей рівень користувач переглядає інформацію, виконує дії з ботами, акаунтами, risk-профілями та отримує результати аналітики. Основними сторінками інтерфейсу є Login, Monitor, Bots, Accounts, Risk Manager і Profile.

Рівень бізнес-логіки містить основні функціональні модулі системи. До них належать модуль ботів, модуль акаунтів, модуль сигналів, модуль аналітики, модуль Risk Manager, модуль моніторингу та модуль Telegram-сповіщень. Кожен із цих модулів виконує окремий набір задач і взаємодіє з іншими частинами платформи.

Модуль ботів відповідає за перегляд торгових стратегій, їхніх статусів, торгових пар, таймфреймів і показників ефективності. Модуль акаунтів забезпечує

роботу з біржовими підключеннями. Модуль сигналів приймає та обробляє webhook-повідомлення. Модуль аналітики формує торгові показники на основі історії угод. Модуль Risk Manager контролює risk-профілі, а модуль моніторингу забезпечує відображення поточного стану системи [22].

Рівень даних відповідає за збереження та отримання інформації з бази даних, а також за інтеграцію з зовнішніми сервісами. До нього належать репозиторії, засоби доступу до PostgreSQL, інтеграційні сервіси для біржових API та допоміжні компоненти для роботи з Telegram Bot API.

Такий поділ системи є зручним для подальшої підтримки й масштабування. Якщо потрібно змінити логіку обробки сигналів, це можна зробити в межах відповідного сервісу, не змінюючи frontend. Якщо необхідно додати нову сторінку інтерфейсу, це не потребує зміни структури бази даних. Якщо потрібно підключити нову біржу, можна розширити інтеграційний рівень без повного перепроєктування системи.

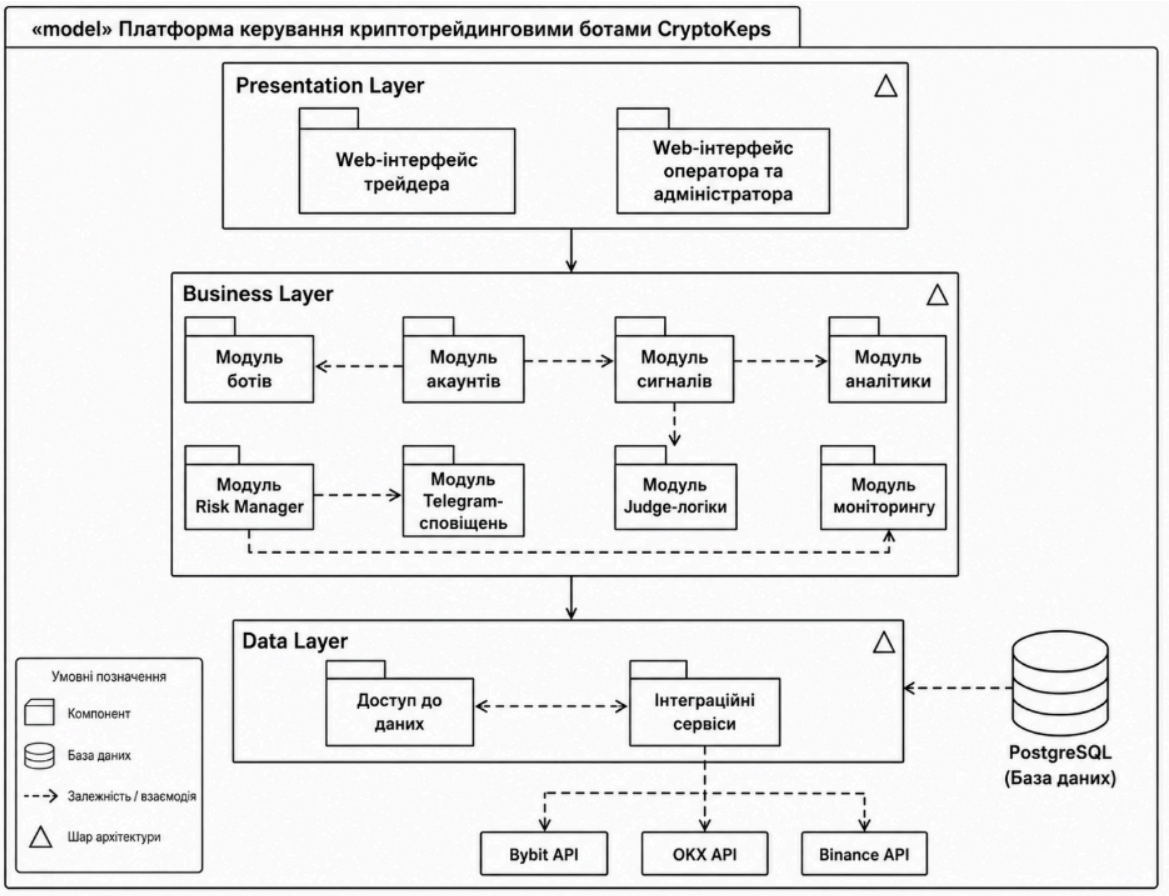


Рис. 3.1 Логічна структура проєкту CryptoKeps

3.3 Проєктування структури бази даних

База даних є центральним елементом платформи CryptoKeps, оскільки саме в ній зберігаються основні дані про користувачів, торгові акаунти, ботів, сигнали, угоди, аналітичні стани та risk-профілі. Для реалізації бази даних використовується PostgreSQL, а структура даних поділяється на дві логічні частини: торгову частину та частину risk-management.

Торгова частина бази даних пов’язана з Java Backend. Вона містить таблиці, які описують торгові акаунти, ботів, історію угод, сигнали, стани індикаторів і статуси ботів. Основними таблицями цієї частини є account_info, bots, trade_history, signals, indicator_state і bot_status.

Таблиця account_info зберігає дані про торгові акаунти. До таких даних належать ідентифікатор акаунта, назва ключа, біржа, API key, API secret і статус підключення. Ця таблиця є базовою для зв’язку акаунтів із торговими ботами та історією угод.

Таблиця bots містить інформацію про торгових ботів. Кожен бот має назву, тип стратегії, торговий символ, таймфрейм, статус активності та прив’язку до акаунта. Це дозволяє системі розуміти, який бот працює з якою біржею та якою торговою парою.

Таблиця trade_history призначена для збереження історії угод. Вона містить символ, сторону угоди, кількість, ціну входу, ціну виходу, stop loss, take profit, PnL і часову мітку. На основі цієї таблиці система може розраховувати прибутковість, кількість угод, win rate, drawdown та інші торгові показники.

Таблиця signals використовується для збереження webhook-сигналів. Для захисту від повторної обробки сигналів передбачено поле idempotency_key. Це дозволяє системі перевіряти, чи був конкретний сигнал уже прийнятий раніше.

Таблиця indicator_state зберігає стан індикаторів за торговим символом, таймфреймом та ключем індикатора. Вона використовується для визначення активності сигналів у логіці Judge. Таблиця bot_status містить агрегований статус

готовності бота, зокрема кількість активних індикаторів, поріг готовності та логічний показник ready.

Друга частина бази даних пов’язана з Django Backend і Risk Manager. До неї належать таблиці auth_user, tw_profile, tw_server і tw_access_period. Таблиця auth_user зберігає користувачів платформи, їхні email, пароль, ім’я, прізвище та Telegram chat ID. Таблиця tw_profile містить risk-профілі, біржу, назву профілю, ключі доступу, ліміти збитку, refresh time і зв’язок із watcher-сервером. Таблиця tw_server описує watcher-сервери, а tw_access_period зберігає інформацію про період доступу користувача й обмеження за кількістю профілів [22].

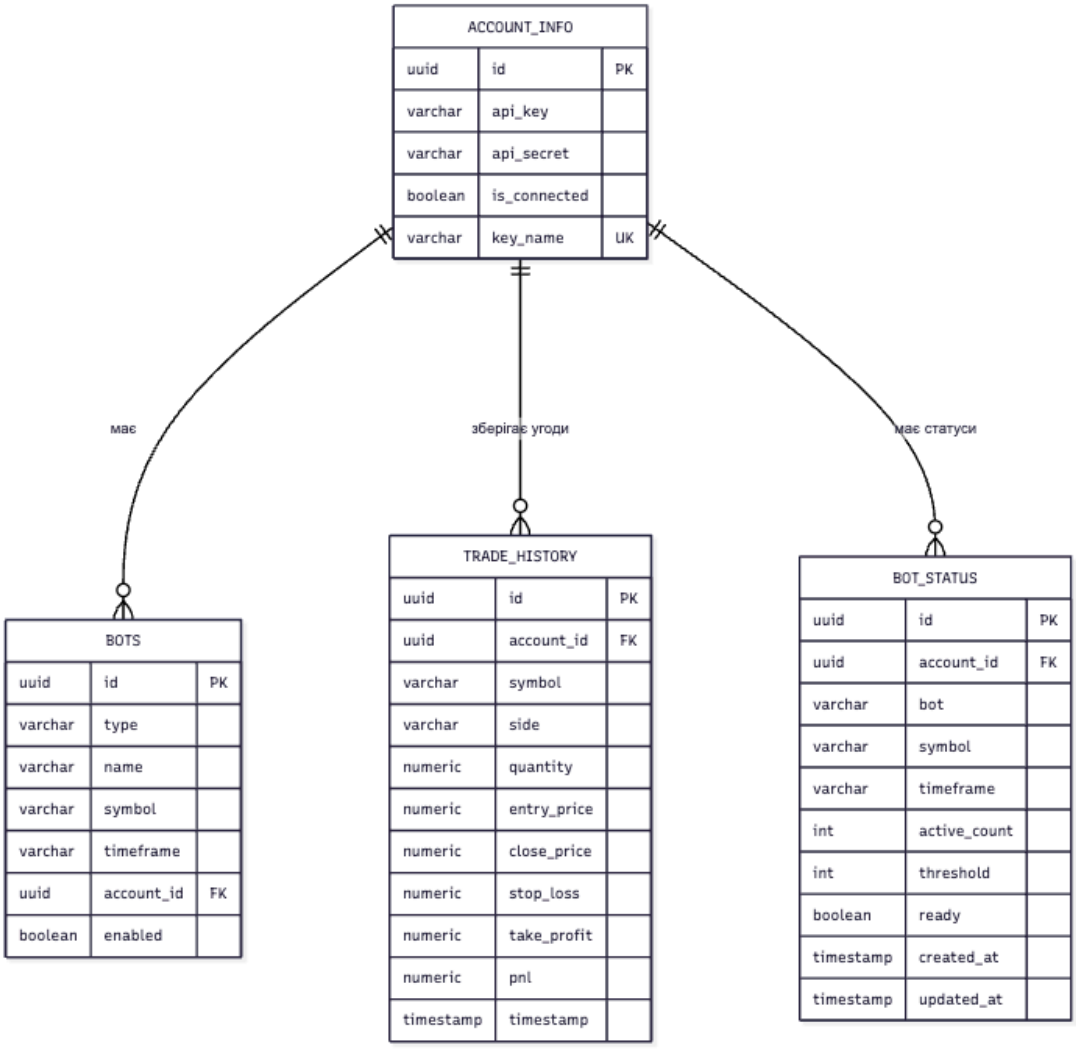


Рис. 3.2 Структура бази даних платформи CryptoKeps, частина 1

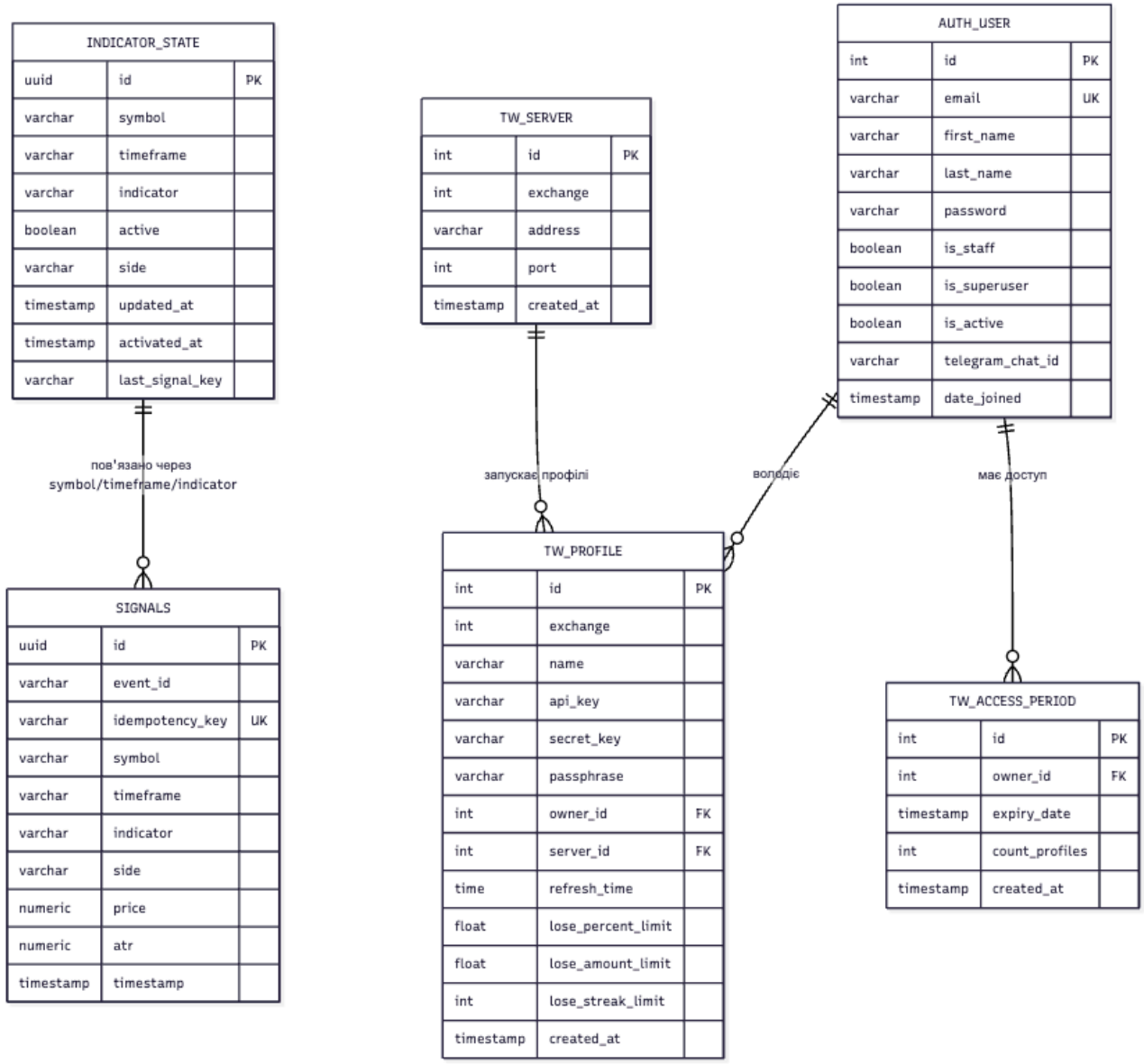


Рис. 3.3 Структура бази даних платформи CryptoKeps, частина 2

3.4 Проєктування серверної частини Java Backend

Java Backend є основним серверним модулем торгової частини платформи CryptoKeps. Він відповідає за обробку торгових ботів, акаунтів, сигналів, історії угод, статусів стратегій і торгової аналітики. Структура Java Backend побудована за класичною багаторівневою схемою Spring Boot: controllers, services, repositories, entities і DTO.

Рівень контролерів відповідає за приймання HTTP-запитів від frontend-частини. До основних контролерів можна віднести BotController,

WebhookController, AccountInfoController, TradeController, BotStatusController і TestSignalController. Кожен контролер відповідає за окрему групу endpoint-ів і передає запити на сервісний рівень [19].

BotController використовується для отримання списку ботів, перегляду угод конкретного бота, отримання статусу Judge-логіки та відправлення тестового сигналу. WebhookController приймає зовнішні торгові сигнали. AccountInfoController відповідає за роботу з торговими акаунтами. TradeController забезпечує отримання історії угод, а BotStatusController використовується для перегляду статусів ботів.

Сервісний рівень містить основну бізнес-логіку. Наприклад, WebhookServiceImpl відповідає за обробку отриманого webhook-сигналу, SignalIngestServiceImpl — за нормалізацію сигналу та формування idempotency key, StrategyEngineImpl — за оновлення логіки стратегії, IndicatorStateServiceImpl — за стан індикаторів, а TradeHistoryServiceImpl — за роботу з історією угод.

Окремим сервісом є BotQueryService, який готує дані для відображення ботів у frontend. Він може розраховувати кількість угод, PnL, win rate, drawdown і equity series. Завдяки цьому frontend отримує вже підготовлені дані, а не виконує складні обчислення самостійно.

Рівень репозиторіїв відповідає за доступ до PostgreSQL. Для кожної основної сутності створюється repository-інтерфейс: AccountInfoRepository, BotRepository, TradeHistoryRepository, SignalRepository, IndicatorStateRepository і BotStatusRepository. Це дозволяє ізолювати логіку роботи з базою даних від бізнес-логіки.

Entity-класи описують структуру таблиць бази даних у Java-коді. До них належать AccountInfoEntity, BotEntity, TradeHistoryEntity, SignalEntity, IndicatorStateEntity і BotStatusEntity. DTO-моделі використовуються для передачі даних між backend і frontend, щоб не передавати напряму внутрішні entity-об’єкти.

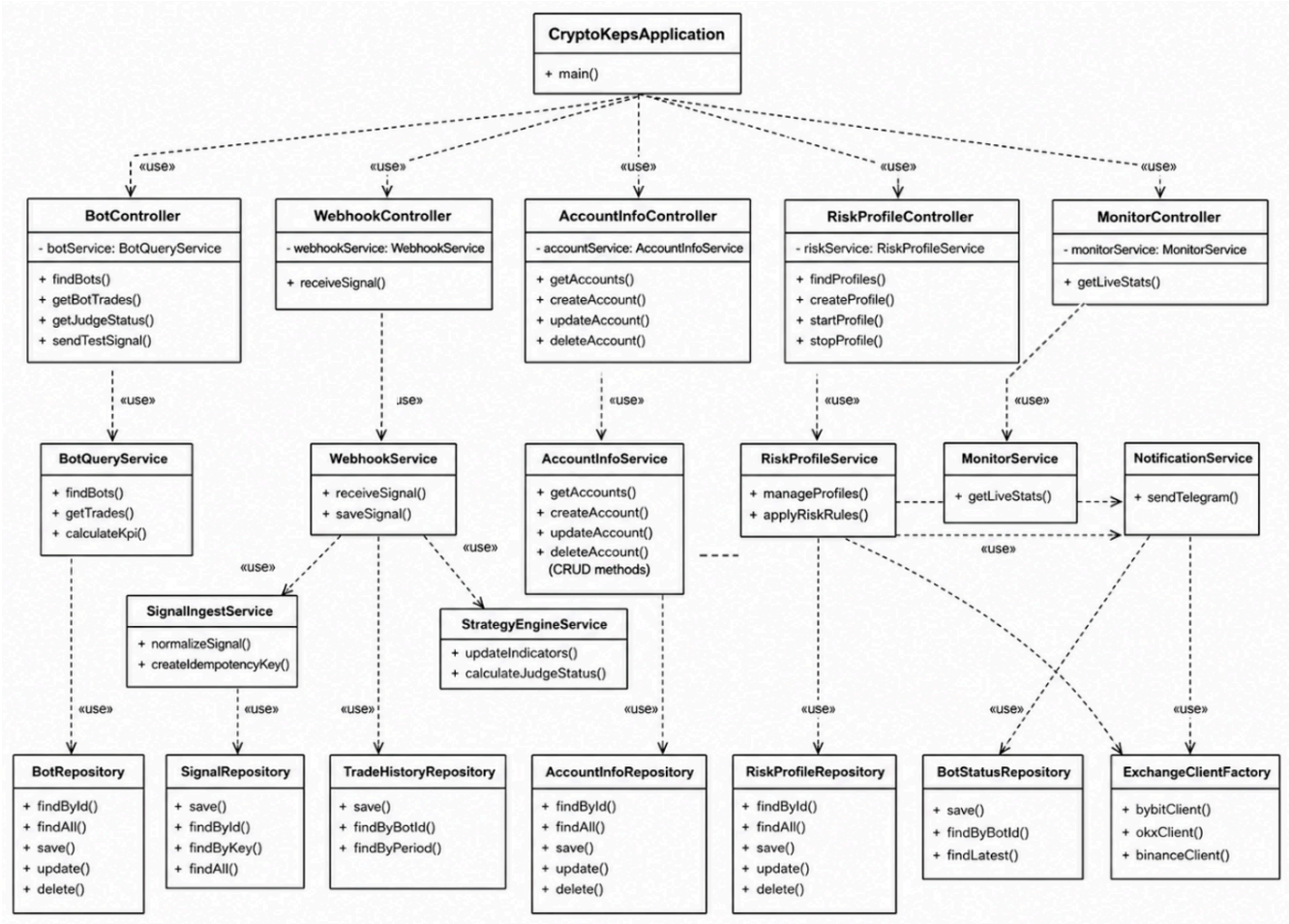


Рис. 3.4 Діаграма класів серверної частини платформи CryptoKeps

3.5 Проєктування модуля Risk Manager та watcher-сервісів

Модуль Risk Manager є важливою частиною платформи CryptoKeps, оскільки він відповідає за контроль ризикових параметрів біржових профілів. Його основне призначення полягає в тому, щоб користувач міг створювати профілі для бірж OKX і Binance, задавати ризикові ліміти та отримувати live-дані про стан профілю.

Risk Manager складається з декількох частин: frontend-сторінки Risk Manager, Django Backend, таблиць risk-профілів у базі даних, watcher-сервісів OKX і Binance, а також Telegram-сповіщень. Така структура дозволяє відокремити контроль ризиків від основної торгової логіки Java Backend [24].

Користувач створює risk-профіль у web-інтерфейсі. У профілі вказується біржа, назва профілю, API-ключі, passphrase, refresh time, максимально допустимий

відсоток втрати, максимально допустима сума збитку та допустима серія збиткових угод. Після створення профіль зберігається в таблиці tw_profile.

Django Backend відповідає за операції з risk-профілями: створення, редагування, видалення, запуск і зупинку. Також він може надавати watcher-сервісам список профілів, які потрібно контролювати на конкретному сервері.

Watcher-сервіси є окремими процесами, які підключаються до біржових API. Binance Watcher отримує інформацію про баланс, відкриті позиції та активність торгівлі через Binance API. OKX Watcher виконує аналогічну функцію для OKX API. Дані watcher-сервісів передаються у frontend через WebSocket, що дозволяє користувачу бачити актуальні значення без ручного оновлення сторінки.
Якщо watcher-сервіс виявляє перевищення ризикового ліміту, він може змінити статус профілю, позначити його як заблокований, ініціювати закриття позицій або сформувати Telegram-сповіщення. Це дозволяє швидше реагувати на критичні ситуації та зменшити ризик неконтрольованих збитків [24].

Крім основних операцій створення та запуску risk-профілів, модуль Risk Manager повинен забезпечувати постійний контроль змін балансу й торгової активності. Це важливо, оскільки ризикова ситуація може виникнути не лише в момент відкриття угоди, а й у процесі подальшого руху ціни. Тому система має регулярно отримувати дані з біржових API, порівнювати їх із заданими обмеженнями та оновлювати стан профілю у web-інтерфейсі.

Окрему роль у роботі модуля відіграє поділ відповідальності між Django Backend і watcher-сервісами. Django Backend зберігає налаштування профілів, інформацію про користувача та параметри ризику, а watcher-сервіси виконують безпосередній моніторинг біржових даних. Такий підхід дозволяє не перевантажувати основний backend і водночас забезпечити більш гнучку роботу з різними біржами.

Для користувача робота Risk Manager відображається у вигляді зрозумілого сценарію: створення профілю, запуск моніторингу, отримання live-даних, перевірка ризикових лімітів і реакція системи на результат перевірки. Якщо значення залишаються в межах норми, дані просто оновлюються в інтерфейсі.

Якщо ліміти перевищені, система фіксує критичний стан, може заблокувати профіль і надіслати повідомлення через Telegram.

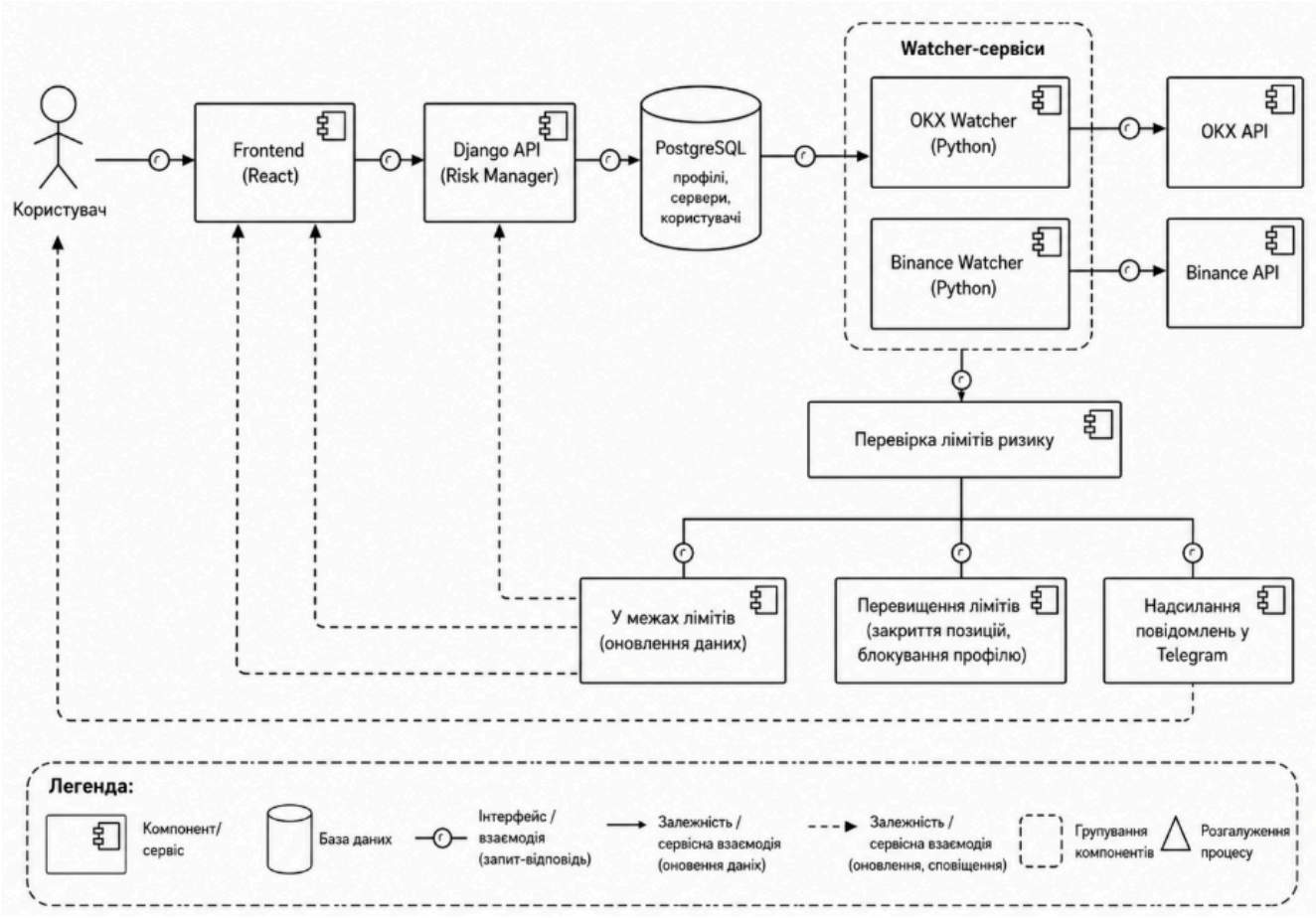


Рис. 3.5 Діаграма компонентів модулю RiskManager

3.6 Проєктування клієнтського web-інтерфейсу

Клієнтський web-інтерфейс CryptoKeps проєктується як набір сторінок, кожна з яких відповідає за окремий сценарій роботи користувача. Основними сторінками є Login, Monitor, Bots, Accounts, Risk Manager і Profile. Навігація між сторінками здійснюється через верхню панель, яка також містить логотип платформи, посилання на розділи та перемикач теми інтерфейсу.

Сторінка Login призначена для авторизації користувача. Вона містить поля для email і пароля, кнопку входу та обробку можливих помилок авторизації. Для демонстрації роботи платформи можуть бути підставлені тестові дані користувача, що спрощує запуск системи під час презентації.

Сторінка Monitor є загальною сторінкою моніторингу. Вона показує агреговані показники платформи: кількість ботів, стан акаунтів, торгову активність, основні метрики та загальне уявлення про роботу системи. Ця сторінка потрібна для швидкої оцінки поточного стану платформи.

Сторінка Bots призначена для роботи з торговими ботами. На ній відображаються картки ботів або таблиця зі списком стратегій. Для кожного бота можуть показуватися назва, біржа, торговий символ, таймфрейм, статус, PnL, win rate, кількість угод і drawdown. Також передбачено можливість перегляду останніх угод і відправлення тестового сигналу.

Сторінка Accounts використовується для перегляду торгових акаунтів. Вона показує назву акаунта, біржу, статус підключення, кількість пов’язаних ботів, кількість угод, PnL і останню активність. Це дозволяє користувачу контролювати біржові підключення в одному інтерфейсі.

Сторінка Risk Manager відповідає за керування risk-профілями. На ній користувач може переглядати профілі OKX і Binance, запускати або зупиняти профіль, бачити balance, available balance, max balance, PnL, lose streak, trade now і статус блокування. Для демонстрації також може бути реалізоване оновлення балансових значень у реальному часі після натискання кнопки Start.

Сторінка Profile призначена для роботи з даними користувача. Вона дозволяє переглядати email, редагувати ім’я, прізвище, Telegram chat ID і виконувати вихід із системи. Ця сторінка є допоміжною, але важливою для персоналізації роботи платформи та налаштування сповіщень.

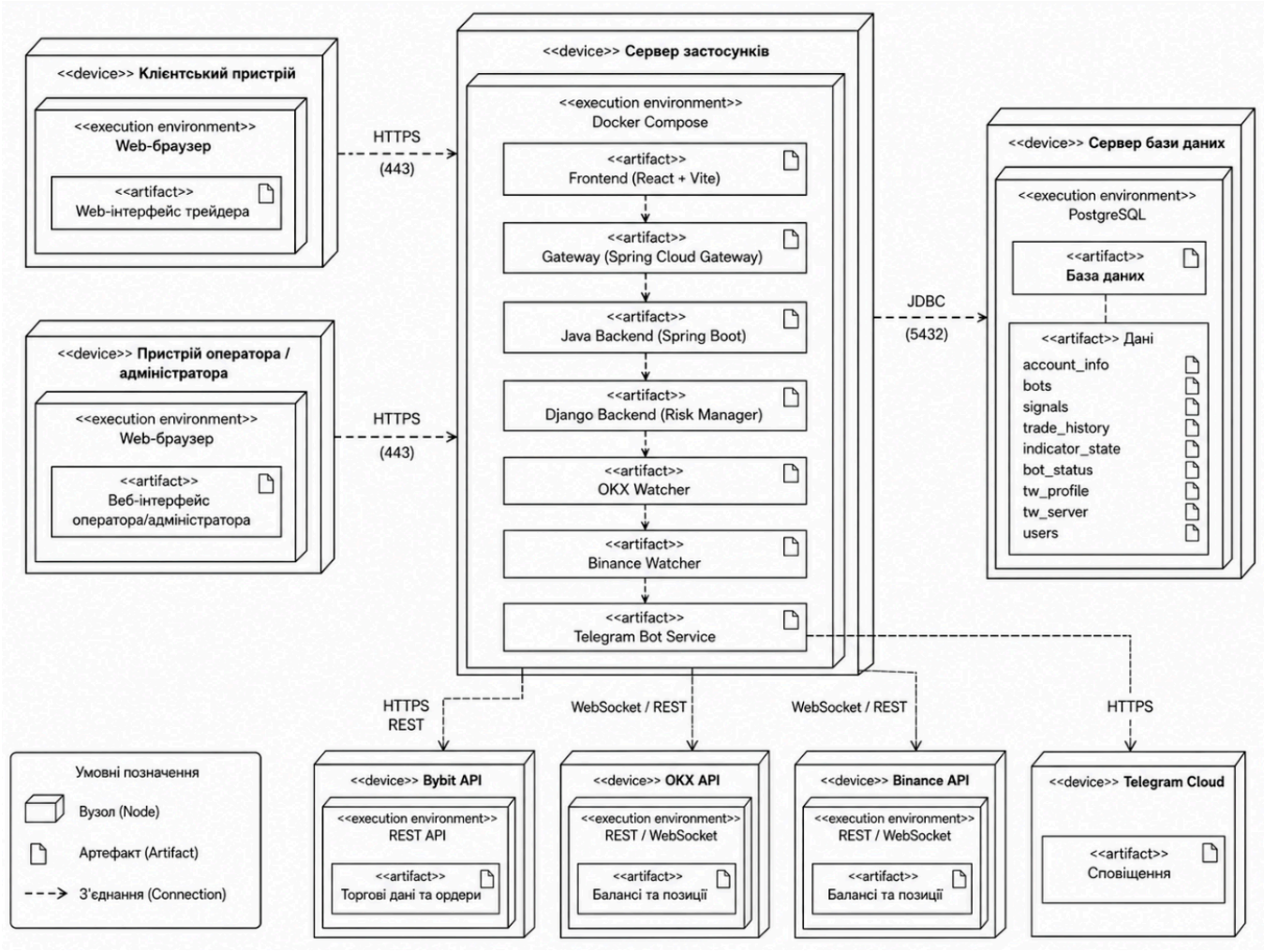


Рис. 3.6 Діаграма розгортання платформи CryptoKeps

Отже, проєктування платформи CryptoKeps передбачає побудову модульної системи, у якій кожен компонент має чітко визначену роль. Frontend забезпечує взаємодію користувача з платформою, Java Backend відповідає за торгову логіку й аналітику, Django Backend забезпечує авторизацію та risk-профілі, watcher-сервіси виконують live-моніторинг біржових акаунтів, Gateway маршрутизує запити, а PostgreSQL зберігає дані системи.

4 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПЛАТФОРМИ CRYPTOKEPS

4.1 Реалізація авторизації користувача та профілю

Реалізація платформи CryptoKeps передбачає створення повноцінного web-інтерфейсу, через який користувач може отримати доступ до основних функцій системи. Початковою точкою роботи з платформою є сторінка авторизації. Вона призначена для входу користувача до системи за email та паролем [19; 22; 24].

Авторизація є важливою частиною платформи, оскільки CryptoKeps працює з торговими акаунтами, risk-профілями, історією угод, аналітичними даними та API-ключами біржових сервісів. Тому доступ до основних сторінок системи повинен бути обмежений лише для авторизованих користувачів.

У frontend-частині сторінка входу реалізована як окремий компонент. Вона містить поля для введення email і пароля, кнопку входу та механізм обробки помилок авторизації. Для демонстраційного режиму в системі можуть бути попередньо підставлені тестові дані користувача, що дозволяє швидко перейти до перегляду функціоналу платформи без додаткового налаштування.

Після успішного входу користувач переходить до основної частини системи. Дані про поточного користувача зберігаються в контексті авторизації frontend-додатку. Для цього використовується AuthProvider, який відповідає за отримання інформації про користувача, виконання входу, виходу та оновлення профілю.

Backend-частина авторизації реалізується на стороні Django Backend. Він відповідає за обробку запитів входу, виходу, отримання поточного користувача та редагування профілю. Основними API-напрямами є login, logout, me та profile. Такий підхід дозволяє відокремити користувацьку частину від торгової логіки Java Backend.

Сторінка профілю користувача дозволяє переглядати email, ім’я, прізвище та Telegram chat ID. Telegram chat ID використовується для подальшого надсилання

повідомлень користувачу про критичні події, пов’язані з risk-профілями, перевищенням лімітів або іншими ситуаціями, які потребують уваги.

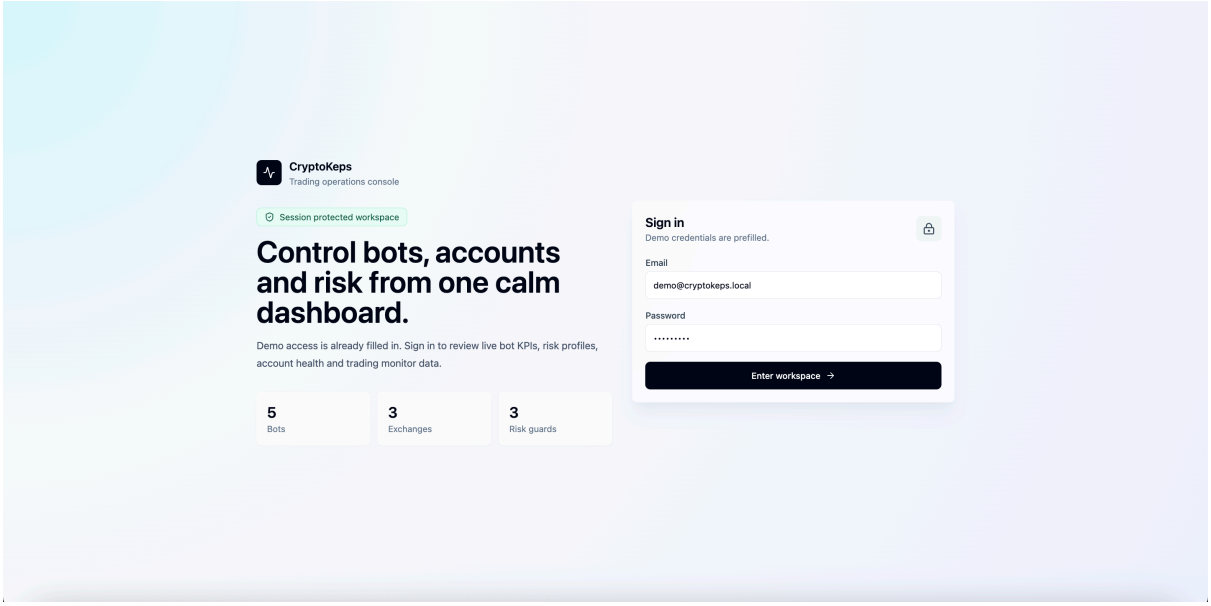


Рис. 4.1 Екранна форма входу до платформи CryptoKeps

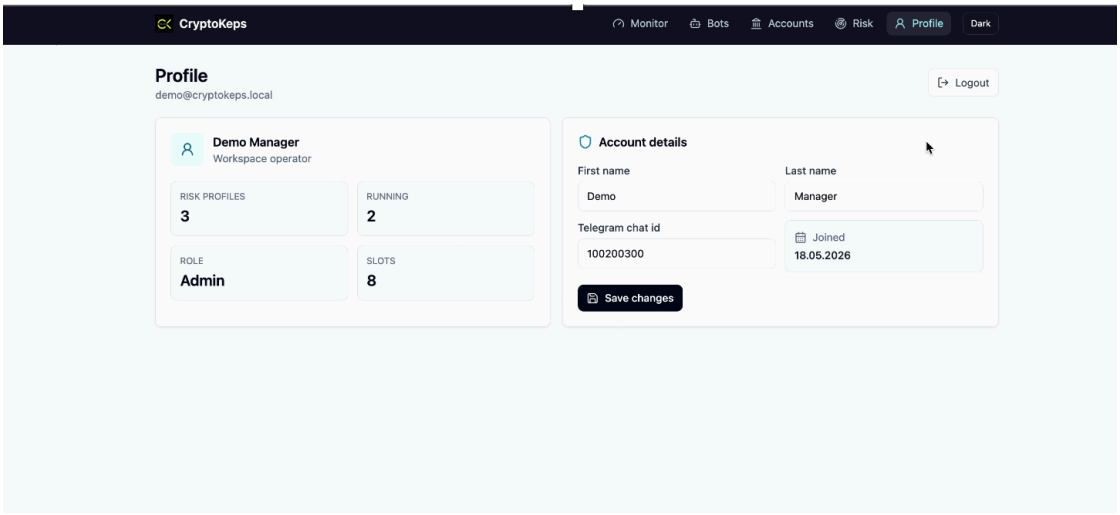


Рис. 4.2 Екранна форма профілю користувача платформи CryptoKeps

4.2 Реалізація клієнтського web-інтерфейсу платформи

Клієнтський web-інтерфейс платформи CryptoKeps реалізовано з використанням React, TypeScript та Vite. Основною задачею frontend-частини є

забезпечення зручної взаємодії користувача з торговими ботами, акаунтами, аналітикою, risk-профілями та власним профілем.

Інтерфейс побудований за компонентним принципом. Це означає, що окремі частини сторінок оформлені як самостійні компоненти: навігаційна панель, картки ботів, таблиці акаунтів, блоки аналітики, форми авторизації, картки risk-профілів і кнопки керування. Такий підхід спрощує підтримку системи та дозволяє повторно використовувати компоненти в різних частинах застосунку [20; 21].

Основними сторінками клієнтської частини є Login, Monitor, Bots, Accounts, Risk Manager і Profile. Сторінка Login використовується для входу до системи. Сторінка Monitor є загальним екраном моніторингу, де користувач може швидко оцінити стан торгової системи. Сторінка Bots призначена для перегляду й аналізу торгових ботів. Сторінка Accounts відображає торгові акаунти. Сторінка Risk Manager використовується для контролю risk-профілів, а Profile — для редагування персональних даних користувача.

Для зручної навігації реалізована верхня панель. Вона містить назву або логотип платформи, посилання на основні сторінки, активний стан поточного розділу та елементи керування профілем. Також у системі передбачено перемикання світлої та темної теми інтерфейсу, що покращує комфорт використання платформи в різних умовах.

Сторінка Monitor призначена для загального огляду системи. На ній можуть відображатися агреговані показники: кількість торгових ботів, кількість акаунтів, загальний PnL, активність стратегій, останні сигнали або статуси ризику. Цей екран потрібний для швидкого розуміння поточного стану платформи без переходу до кожного окремого модуля [20].

Сторінка Bots реалізує візуальне представлення торгових ботів. Для кожного бота можуть бути показані назва, біржа, торговий символ, таймфрейм, статус, PnL, win rate, drawdown, кількість угод і остання активність. Також користувач може переглядати останні угоди, фільтрувати ботів за періодом і виконувати тестові дії, наприклад надсилати тестовий сигнал [20].

Сторінка Accounts відображає торгові акаунти, які підключені до платформи. Для кожного акаунта можна показати назву, біржу, статус підключення, кількість пов’язаних ботів, кількість угод, PnL, win rate та час останньої активності. Це дозволяє користувачеві швидко зрозуміти, які акаунти активні та як вони використовуються в системі.

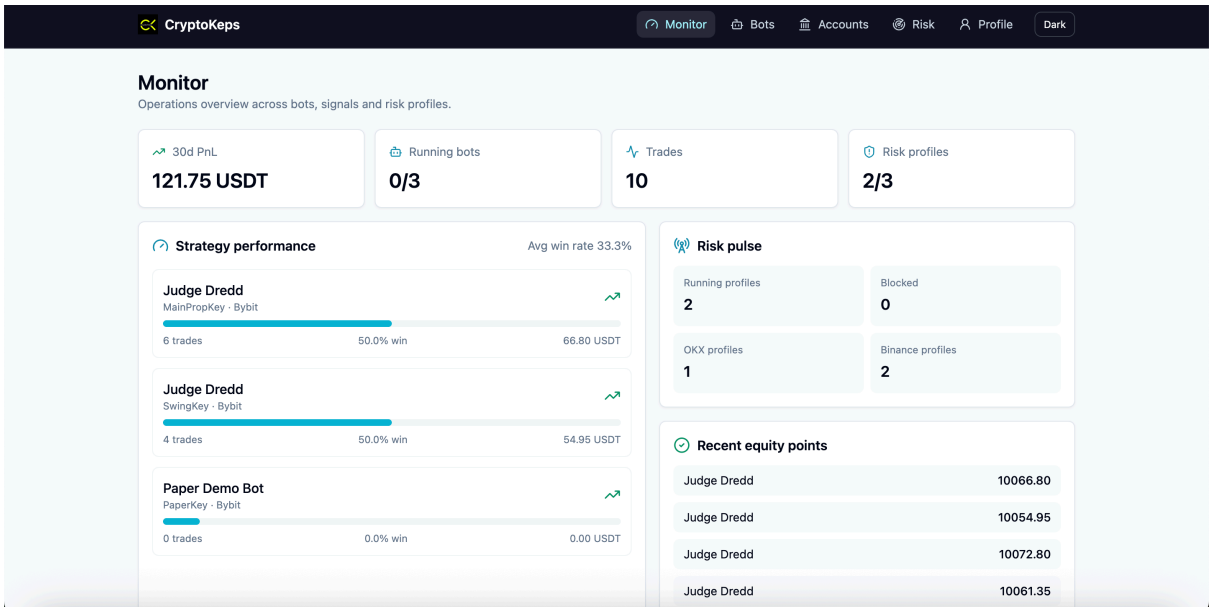


Рис. 4.3 Екранна форма сторінки аналітики платформи CryptoKeps

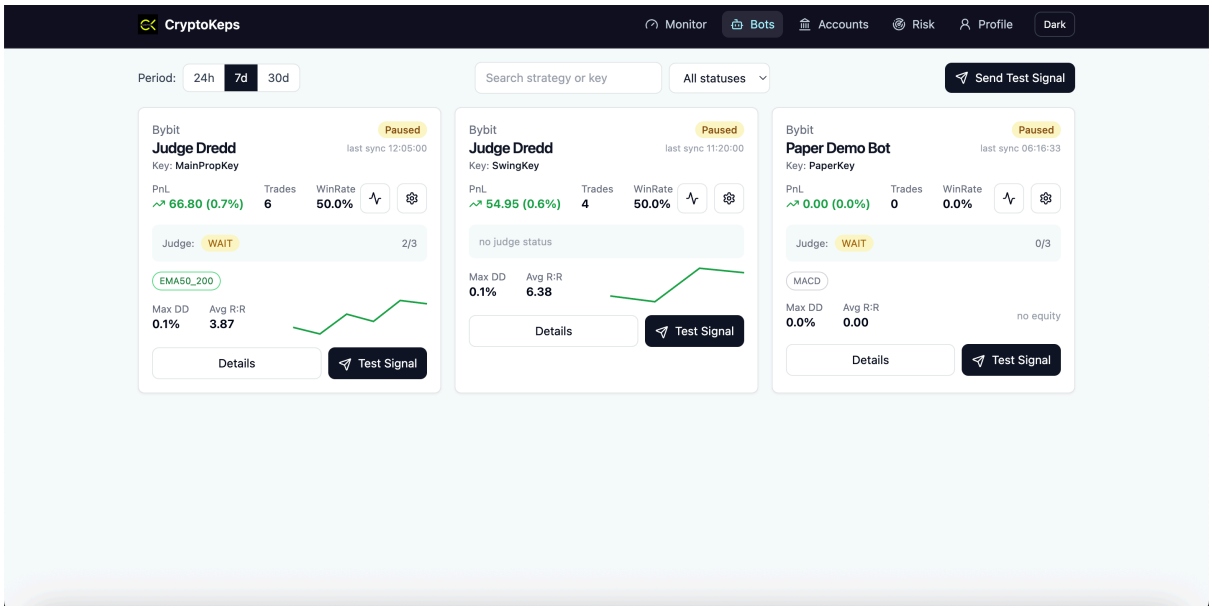


Рис. 4.4 Екранна форма сторінки торгових ботів платформи CryptoKeps

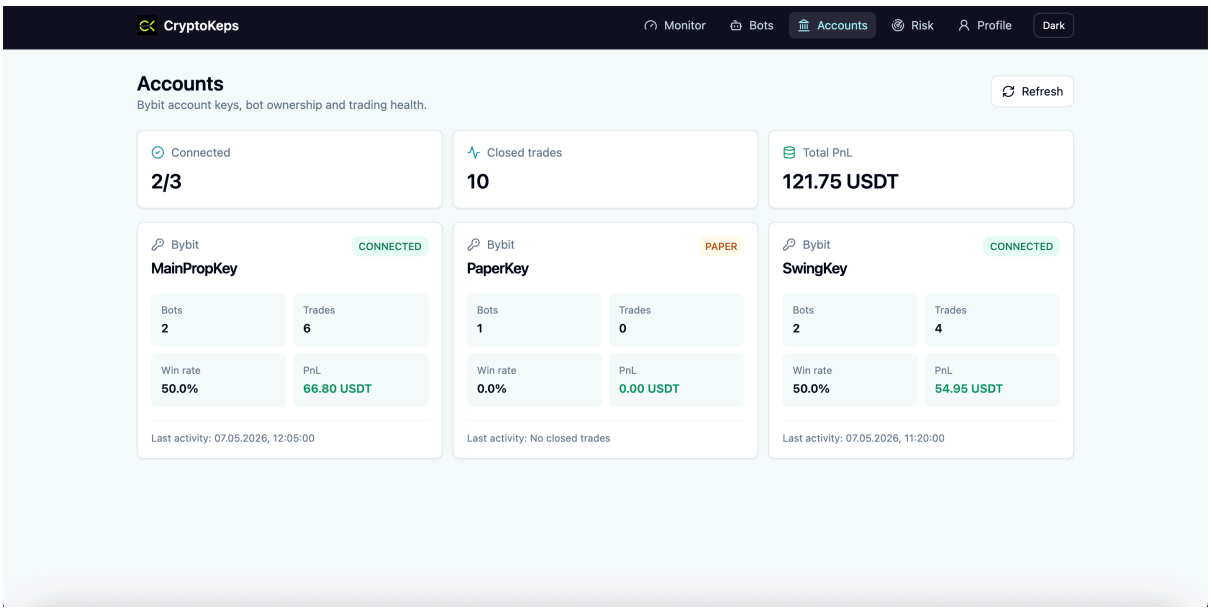


Рис. 4.5 Екранна форма сторінки торгових акаунтів платформи CryptoKeps

4.3 Реалізація серверної частини для роботи з ботами, акаунтами та сигналами

Серверна частина CryptoKeps реалізується у вигляді Java Backend на основі Spring Boot. Вона відповідає за основну торгову логіку платформи: роботу з ботами, торговими акаунтами, сигналами, історією угод, статусами стратегій і аналітичними показниками [19].

Архітектура Java Backend поділена на декілька рівнів. Рівень контролерів приймає HTTP-запити від frontend-частини. Рівень сервісів виконує бізнес-логіку. Рівень репозиторіїв відповідає за доступ до бази даних PostgreSQL. Entity-класи описують структуру таблиць, а DTO використовуються для передавання даних між backend і frontend [19; 22].

Для роботи з торговими ботами використовується BotController. Він надає можливість отримати список ботів, переглянути інформацію про конкретного бота, отримати його угоди та статус Judge-логіки. Дані для відображення ботів готуються на сервісному рівні, де також можуть розраховуватися основні KPI.

Для роботи з торговими акаунтами використовується AccountInfoController. Він відповідає за отримання списку акаунтів, створення нового акаунта, оновлення

даних і видалення акаунта. Акаунт містить інформацію про біржу, назву ключа, API key, API secret і статус підключення.

Для обробки webhook-сигналів використовується WebhookController. Він приймає зовнішній сигнал у форматі JSON і передає його до WebhookServiceImpl. Далі сигнал проходить нормалізацію через SignalIngestServiceImpl, де формується idempotency key. Якщо такий ключ уже існує в базі, сигнал не обробляється повторно. Якщо сигнал новий, він зберігається в таблиці signals [18; 19].

Після збереження сигналу система оновлює стан індикатора в таблиці indicator_state. Це необхідно для роботи Judge-логіки, яка визначає готовність бота до торгової дії на основі кількості активних індикаторів. Далі оновлений результат зберігається в таблиці bot_status.

Окремо реалізується робота з історією угод. TradeHistoryServiceImpl відповідає за збереження та отримання торгових операцій. Історія угод використовується не лише для перегляду минулих операцій, а й для розрахунку аналітичних показників: сумарного PnL, кількості угод, win rate і drawdown.

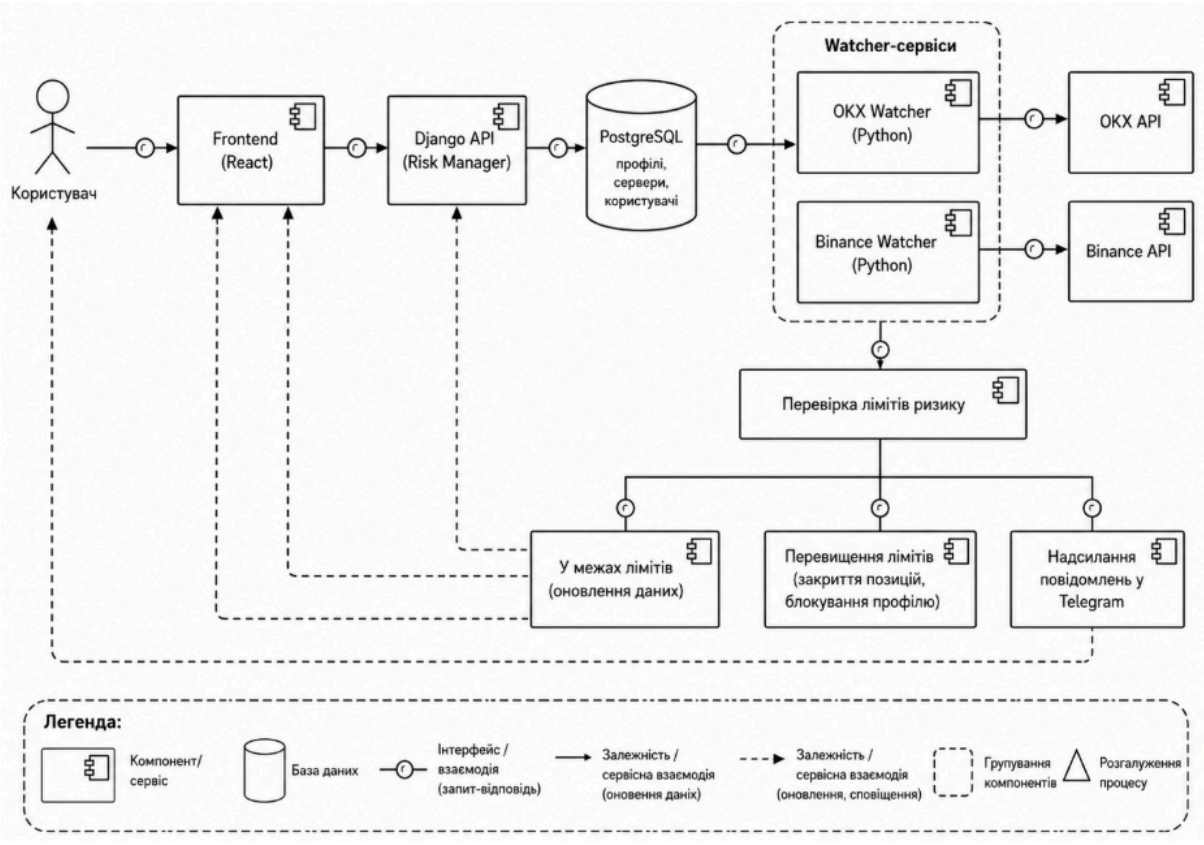


Рис. 4.6 Схема роботи серверної частини з торговими ботами та акаунтами

4.4 Реалізація модуля торгової аналітики та історії угод

Модуль торгової аналітики є важливою частиною платформи CryptoKeps, оскільки дозволяє користувачу оцінювати ефективність роботи торгових ботів і акаунтів. Без такого модуля система могла б лише зберігати угоди, але не давала б узагальненої інформації для прийняття рішень.

Історія угод зберігається в таблиці trade_history. Для кожної операції фіксуються торговий акаунт, символ, сторона угоди, кількість, ціна входу, ціна закриття, stop loss, take profit, PnL і час виконання. Ці дані формують основу для подальшого аналізу [22].

На основі історії угод серверна частина розраховує основні торгові KPI. До них належать сумарний прибуток або збиток, кількість угод, кількість прибуткових угод, win rate, максимальна просадка та equity series. Такі показники допомагають користувачеві оцінити не лише результат однієї угоди, а й стабільність стратегії загалом [12].

PnL показує фінансовий результат за вибраний період. Win rate дозволяє оцінити частку прибуткових угод. Drawdown показує максимальне зниження відносно попереднього максимуму equity. Кількість угод дозволяє оцінити активність стратегії. Разом ці показники дають більш повну картину роботи торгового бота.

У frontend-частині аналітичні дані можуть відображатися у вигляді карток, таблиць і графіків. Наприклад, на сторінці Bots користувач бачить KPI кожного бота, а на сторінці аналітики може переглянути загальні показники платформи. Для акаунтів також можуть відображатися PnL, win rate, кількість угод і остання активність.

Важливо, що розрахунок показників виконується на backend-стороні. Це дозволяє зменшити навантаження на frontend і забезпечити однакову логіку розрахунків для різних сторінок системи. Frontend отримує вже підготовлені дані через API та відповідає переважно за їхнє відображення.

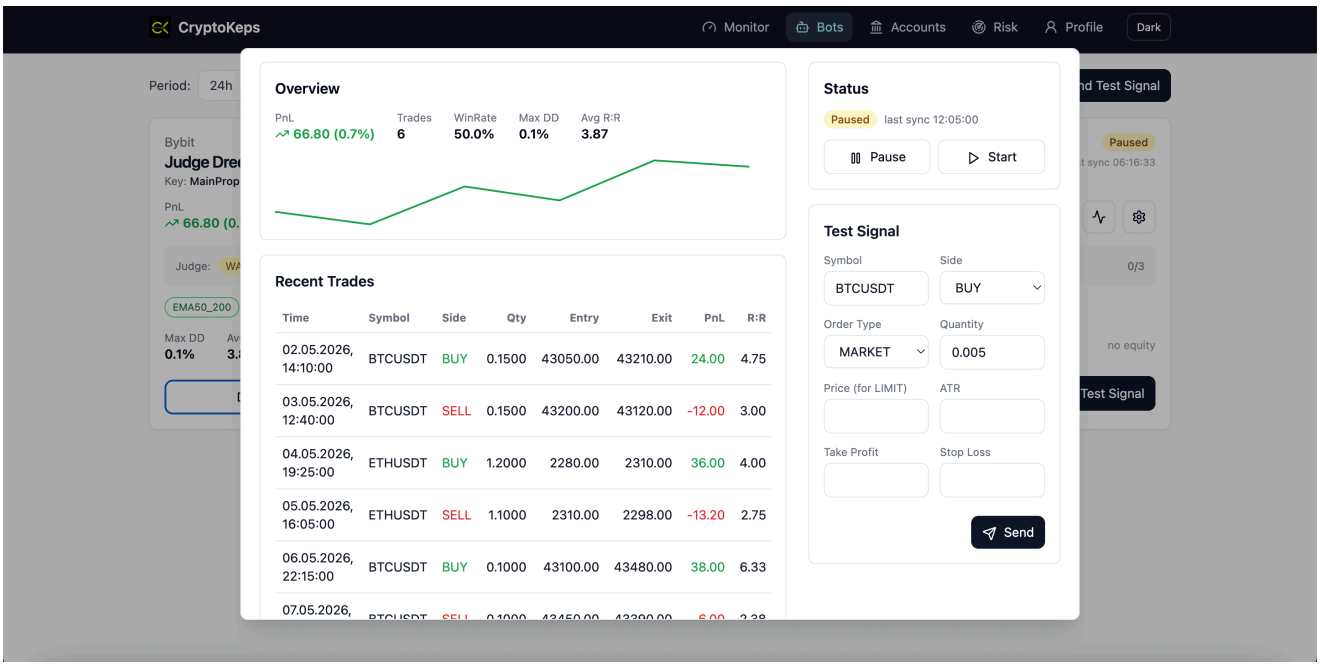


Рис. 4.7 Екранна форма відображення торгової аналітики платформи

4.5 Реалізація модуля Risk Manager і live-моніторингу

Модуль Risk Manager реалізований для контролю біржових профілів і ризикових обмежень. Він поєднує frontend-сторінку Risk Manager, Django Backend, таблиці risk-профілів, watcher-сервіси OKX і Binance та механізм Telegram-сповіщень [24].

У web-інтерфейсі користувач може створювати risk-профілі, переглядати їхній стан, запускати або зупиняти профіль, редагувати параметри та видаляти непотрібні записи. Для кожного профілю відображаються біржа, назва, баланс, доступний баланс, максимальний баланс, PnL, серія збиткових угод, статус активної торгівлі та ознака блокування.

Django Backend відповідає за збереження й обробку risk-профілів. Він реалізує API для створення профілю, запуску, зупинки, оновлення та видалення. Також він забезпечує отримання списку профілів для watcher-сервісів. У таблиці tw_profile зберігаються ключові параметри профілю: біржа, назва, API key, secret key, passphrase, refresh time, lose_percent_limit, lose_amount_limit і lose_streak_limit [22; 24].

Watcher-сервіси працюють як окремі процеси. Їхня задача — отримувати дані з біржових API та регулярно перевіряти стан risk-профілів. Binance Watcher і OKX Watcher отримують баланс, доступний баланс, активні позиції, PnL та інші показники. Отримані live-дані передаються у frontend через WebSocket [24].

WebSocket використовується для оновлення даних без ручного перезавантаження сторінки. Після запуску профілю користувач може бачити зміну балансових показників у режимі реального часу. У демонстраційному режимі також може бути реалізовано симуляцію зміни балансу після натискання кнопки Start, щоб показати принцип роботи live-моніторингу [24].

Якщо watcher-сервіс виявляє перевищення ризикового ліміту, профіль може бути заблокований. Також система може надіслати Telegram-сповіщення користувачу. Це дозволяє отримувати повідомлення про критичні події навіть тоді, коли користувач не перебуває безпосередньо у web-інтерфейсі.

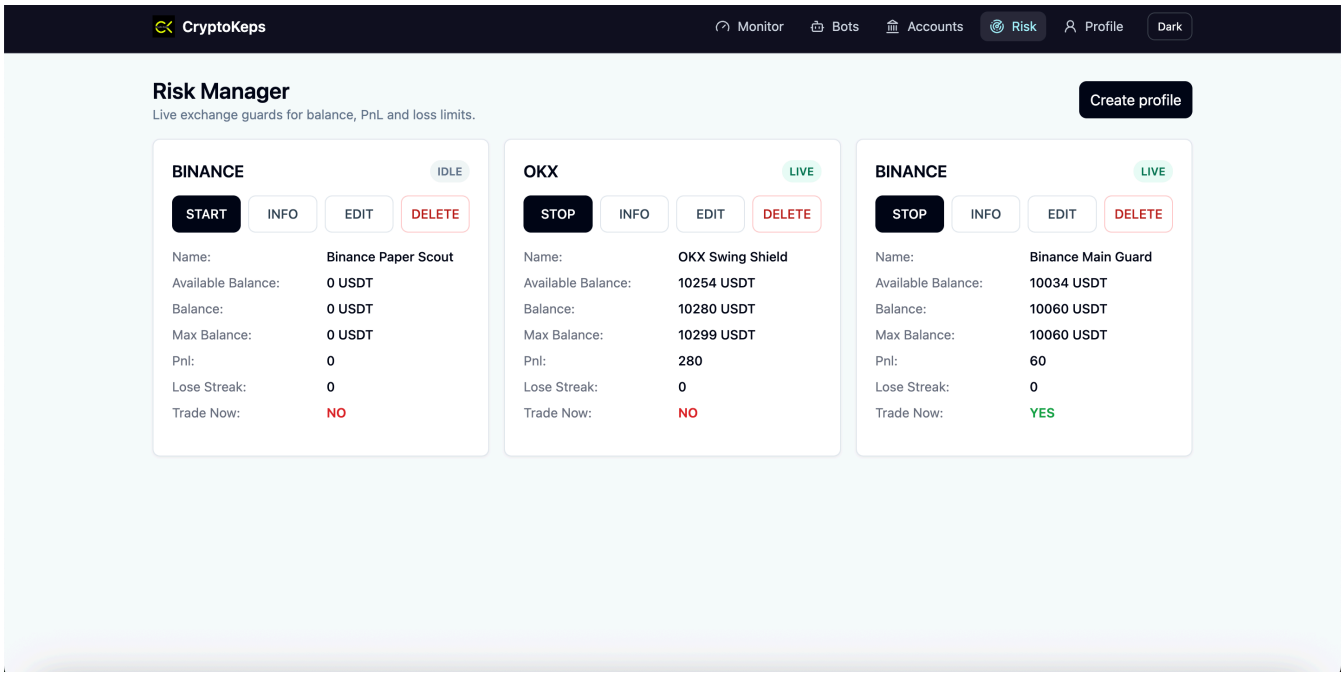


Рис. 4.8 Екранна форма сторінки Risk Manager платформи CryptoKeps

4.6 Реалізація обробки webhook-сигналів та Judge-логіки

Окремим важливим напрямом реалізації платформи CryptoKeps є обробка webhook-сигналів, оскільки саме через них система може отримувати торгові події від зовнішніх сервісів. У межах проєкту webhook-сигнал розглядається як структуроване повідомлення, яке містить дані про торговий символ, таймфрейм, індикатор, напрям сигналу, ціну, значення ATR та інші параметри, необхідні для подальшої обробки торговою логікою. Основним джерелом таких сигналів може бути TradingView або інший аналітичний сервіс, який підтримує надсилання webhook-повідомлень [18].

У серверній частині платформи приймання webhook-сигналу виконується через окремий контролер WebhookController. Він приймає HTTP-запит у форматі JSON і передає отримані дані до сервісного рівня. Такий підхід дозволяє відокремити зовнішню точку входу від основної бізнес-логіки. Контролер відповідає лише за приймання запиту, а перевірка, нормалізація, збереження та оновлення стану стратегії виконуються окремими сервісами Java Backend [19].

Після отримання сигналу дані передаються до WebhookService, який координує подальшу обробку. На цьому етапі перевіряється наявність основних полів сигналу, зокрема символу, таймфрейму, індикатора та напряму сигналу. Якщо структура повідомлення є некоректною або частина обов’язкових даних відсутня, сигнал не повинен впливати на стан торгової стратегії. Це дозволяє уникнути помилкового оновлення даних у системі.

Наступним етапом є нормалізація сигналу. Вона виконується для того, щоб привести отримані дані до єдиного формату, який використовується всередині платформи. Наприклад, торговий символ, назва індикатора, таймфрейм або напрям сигналу можуть надходити в різних форматах залежно від зовнішнього сервісу. Нормалізація дозволяє уніфікувати ці значення та забезпечити коректну подальшу обробку.

Особливе значення має механізм захисту від повторної обробки однакових сигналів. Для цього в CryptoKeps використовується idempotency key. Він

формується на основі ключових параметрів сигналу, які дозволяють ідентифікувати конкретну торгову подію. Якщо сигнал із таким ключем уже існує в базі даних, система не створює новий запис і не змінює повторно стан індикатора. Це важливо, оскільки повторне надсилання одного й того самого webhook-повідомлення може призвести до некоректного стану торгової логіки [18; 19].

Якщо сигнал є новим, він зберігається в таблиці signals. У записі можуть фіксуватися event_id, idempotency_key, symbol, timeframe, indicator, side, price, atr, value та часові мітки. Збереження сигналів дозволяє не лише обробляти поточні події, а й аналізувати історію надходження торгових сигналів у майбутньому. Це також спрощує налагодження системи, оскільки адміністратор або розробник може перевірити, які саме сигнали надходили до платформи.

Після збереження сигналу оновлюється стан відповідного індикатора в таблиці indicator_state. Якщо сигнал має напрям LONG, індикатор може переводитися в активний стан. Якщо сигнал має напрям SHORT, індикатор може деактивуватися. Така логіка дозволяє зберігати поточний стан набору індикаторів для конкретної торгової пари та таймфрейму.

На основі станів індикаторів працює Judge-логіка. Її завдання полягає в тому, щоб визначити, чи готовий бот до торгової дії. Для цього система підраховує кількість активних індикаторів для певного symbol і timeframe та порівнює її із заданим порогом. Якщо кількість активних індикаторів дорівнює або перевищує поріг, бот отримує статус готовності. Якщо активних індикаторів недостатньо, бот залишається в режимі очікування.

Результат роботи Judge-логіки зберігається в таблиці bot_status. У ній фіксуються бот, торговий символ, таймфрейм, кількість активних індикаторів, поріг готовності, ознака ready та часові мітки оновлення. Завдяки цьому frontend може отримати вже підготовлений статус і відобразити його користувачу у зрозумілому вигляді, наприклад як READY або WAIT.

Таким чином, обробка webhook-сигналу в CryptoKeps не обмежується простим збереженням повідомлення. Сигнал проходить повний цикл: приймання, перевірку, нормалізацію, захист від дублювання, збереження в базі даних,

оновлення стану індикатора та перерахунок готовності торгової стратегії. Це забезпечує контрольовану й передбачувану роботу торгової логіки.

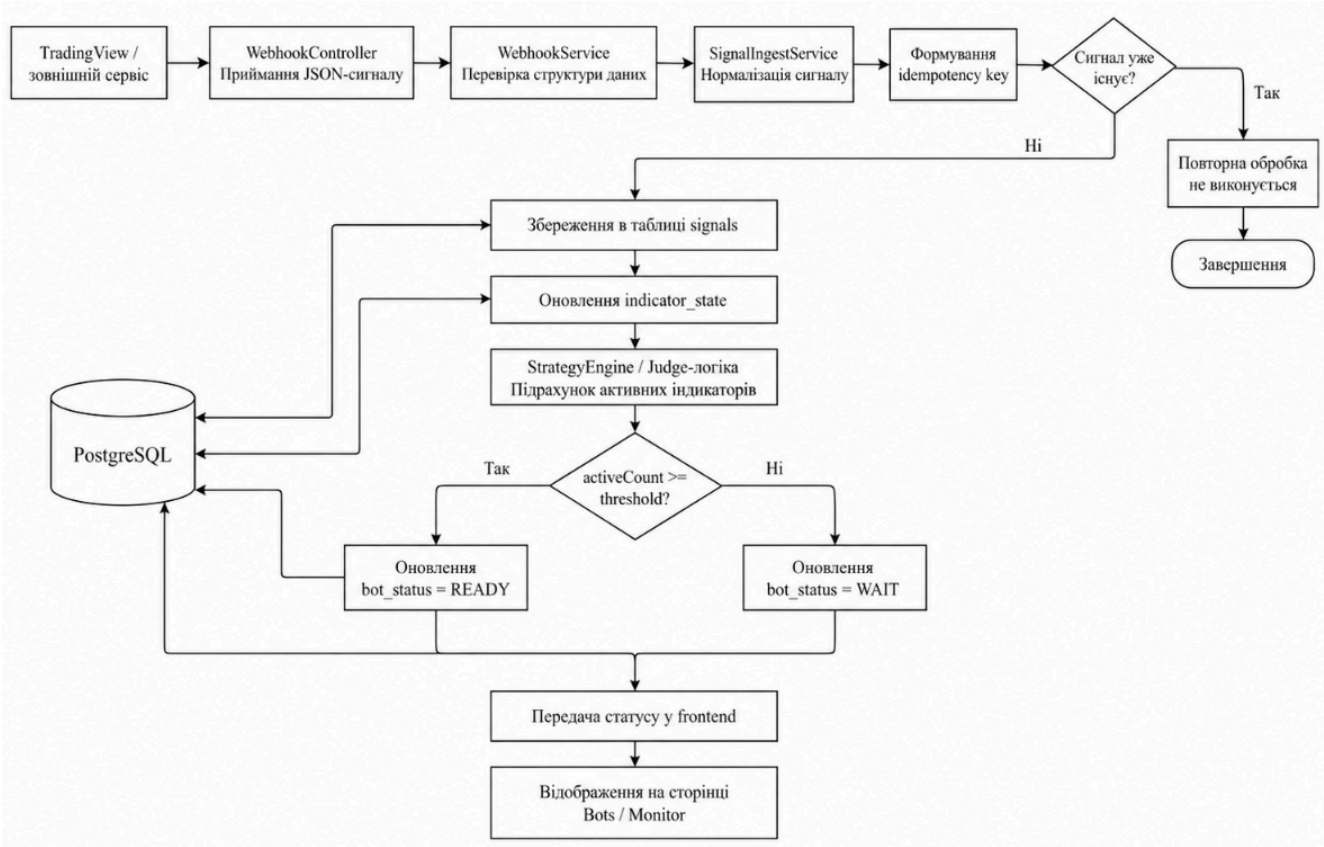


Рис. 4.9 Блок-схема обробки webhook-сигналу та оновлення Judge-статусу бота

4.7 Реалізація інфраструктури розгортання та взаємодії сервісів

Платформа CryptoKeps має багатокомпонентну структуру, тому для її коректної роботи важливо організувати взаємодію між окремими сервісами. До складу системи входять frontend-частина, Spring Cloud Gateway, Java Backend, Django Backend, PostgreSQL, watcher-сервіси OKX і Binance, а також зовнішні інтеграції з біржовими API та Telegram Bot. Кожен із цих компонентів виконує окрему функцію, але в межах платформи вони працюють як єдина система.

Frontend-частина реалізована як окремий web-застосунок на React, TypeScript і Vite. Вона відповідає за відображення сторінок Login, Monitor, Bots, Accounts, Risk Manager і Profile. Користувач взаємодіє саме з frontend-інтерфейсом, а всі дані

отримує через API-запити до серверної частини. Такий підхід дозволяє відокремити візуальну частину системи від бізнес-логіки [20; 21].

Spring Cloud Gateway виконує роль єдиної точки входу для запитів frontend-частини. Це дозволяє не звертатися до кожного backend-сервісу напряму, а використовувати централізовану маршрутизацію. Запити, пов’язані з торговими ботами, акаунтами, сигналами та аналітикою, можуть спрямовуватися до Java Backend. Запити, пов’язані з авторизацією, профілем користувача та risk-профілями, передаються до Django Backend [19].

Java Backend відповідає за торгову частину платформи. Він обробляє ботів, торгові акаунти, webhook-сигнали, історію угод, стани індикаторів, статуси ботів і торгову аналітику. Саме цей сервіс реалізує основну доменну логіку, пов’язану з автоматизованою криптовалютною торгівлею. Взаємодія з базою даних здійснюється через репозиторії та сутності Spring Data JPA [19; 22].

Django Backend відповідає за користувацьку частину та модуль Risk Manager. Він реалізує авторизацію, отримання поточного користувача, оновлення профілю, створення й керування risk-профілями. Також цей backend надає watcher-сервісам дані про профілі, які потрібно контролювати. Такий поділ дозволяє розмежувати торгову логіку та логіку контролю ризикових профілів.

PostgreSQL використовується як централізоване сховище даних. У ньому зберігаються торгові акаунти, боти, сигнали, історія угод, стани індикаторів, статуси ботів, користувачі, watcher-сервери та risk-профілі. Завдяки централізованій базі даних різні частини платформи можуть працювати з узгодженими даними, а результати роботи одного модуля можуть використовуватися іншими модулями [22].

Окремим елементом інфраструктури є watcher-сервіси. Вони працюють як незалежні процеси та підключаються до API бірж OKX і Binance. Їхнє завдання полягає в отриманні live-даних про баланс, активні позиції, доступні кошти, PnL, серію збиткових угод і стан торгівлі. Після отримання таких даних watcher-сервіси можуть передавати їх у frontend через WebSocket або оновлювати відповідні дані через backend-модулі [24].

Telegram Bot використовується як канал сповіщень про критичні події. Якщо risk-профіль перевищує встановлений ліміт, система може сформувати повідомлення для користувача. Це дозволяє повідомляти про ризикові ситуації навіть у випадку, якщо користувач не перебуває безпосередньо у web-інтерфейсі. Такий механізм підвищує практичну цінність платформи як інструменту контролю автоматизованої торгівлі.

Для розгортання системи доцільно використовувати Docker Compose. Оскільки CryptoKeps складається з кількох сервісів, контейнеризований підхід дозволяє описати всі компоненти в одному конфігураційному файлі та запускати їх разом. У складі такого розгортання можуть бути визначені контейнери для frontend, gateway, Java Backend, Django Backend, PostgreSQL, OKX watcher і Binance watcher.

Перевагою Docker Compose є те, що кожен компонент запускається в ізольованому середовищі, але водночас може взаємодіяти з іншими сервісами через внутрішню мережу. Це спрощує налаштування платформи, зменшує залежність від локального середовища розробника та дозволяє швидко відновити роботу системи після перезапуску.

Інфраструктура CryptoKeps також є зручною для подальшого масштабування. У майбутньому до системи можна додати нові watcher-сервіси для інших бірж, окремий сервіс для розширеної аналітики, модуль журналу дій користувача, сервіс генерації звітів або додаткові канали сповіщень. При цьому базова архітектура не потребуватиме повної перебудови, оскільки система вже побудована за модульним принципом.

4.8 Тестування основних функцій платформи

Тестування платформи CryptoKeps необхідне для перевірки коректності роботи основних модулів системи. Оскільки платформа складається з frontend-частини, Java Backend, Django Backend, PostgreSQL, Gateway і watcher-сервісів, тестування повинно охоплювати як окремі компоненти, так і взаємодію між ними [19; 20].

Першим етапом є тестування авторизації. Перевіряється можливість входу користувача за email і паролем, отримання даних поточного користувача, вихід із системи та оновлення профілю. Також необхідно перевірити, що неавторизований користувач не може отримати доступ до захищених сторінок платформи.

Другим етапом є тестування сторінок frontend-частини. Перевіряється коректність відображення сторінок Monitor, Bots, Accounts, Risk Manager і Profile. Особливу увагу слід приділити тому, чи правильно відображаються дані з API, чи працюють кнопки, фільтри, перемикання теми та переходи між сторінками.

Третім етапом є тестування роботи з торговими ботами. Необхідно перевірити отримання списку ботів, відображення KPI, перегляд історії угод, отримання статусу Judge-логіки та відправлення тестового сигналу. Також потрібно перевірити поведінку системи у випадку відсутності даних або помилки API.

Четвертим етапом є тестування webhook-сигналів. Перевіряється приймання сигналу, нормалізація даних, формування idempotency key, збереження сигналу в базі даних, захист від повторної обробки та оновлення стану індикаторів. Це важливо, оскільки помилки в обробці сигналів можуть впливати на логіку торгового бота [18; 19].

П’ятим етапом є тестування Risk Manager. Перевіряється створення, редагування, запуск, зупинка та видалення risk-профілів. Також тестується відображення live-даних, зміна балансу, статус блокування, PnL, lose streak і trade now. Окремо перевіряється робота Telegram-сповіщень у разі критичної події [24].

Для оформлення результатів тестування доцільно використати таблицю тест-кейсів. У ній можна зазначити назву тесту, вхідні дані, очікуваний результат і фактичний результат. Це дозволяє структуровано показати, що основні функції платформи перевірені та працюють відповідно до вимог.

Таблиця 4.1

Результати тестування основних функцій платформи CryptoKeps

№	Функція, що тестується	Вхідні дані / дія користувача	Очікуваний результат	Фактичний результат	Статус
1	Авторизація користувача	Введення коректного email та пароля	Користувач входить до системи та переходить на головну сторінку	Вхід виконано успішно	Успішно
2	Обробка помилки авторизації	Введення неправильного пароля	Система показує повідомлення про помилку входу	Повідомлення відображається	Успішно
3	Перегляд сторінки Monitor	Перехід до розділу Monitor	Відображаються загальні показники торгової системи	Дані відображаються коректно	Успішно
4	Перегляд списку ботів	Перехід до сторінки Bots	Система показує список торгових ботів із їхніми статусами та KPI	Список ботів завантажується	Успішно
5	Фільтрація ботів за періодом	Вибір періоду 24h, 7d або 30d	Дані ботів оновлюються відповідно до вибраного періоду	Фільтрація працює коректно	Успішно
6	Перегляд історії угод бота	Відкриття деталей торгового бота	Відображаються останні угоди, PnL, ціни входу/виходу та час операцій	Історія угод відображається	Успішно
7	Надсилання тестового сигналу	Натискання кнопки відправлення тестового сигналу	Система приймає сигнал і передає на серверну обробку	Сигнал обробляється	Успішно

Продовження таблиці 4.1

Результати тестування основних функцій платформи CryptoKeps

№	Функція, що тестується	Вхідні дані / дія користувача	Очікуваний результат	Фактичний результат	Статус
8	Захист від дублювання сигналів	Повторне надсилання однакового webhook-сигналу	Повторний сигнал не повинен оброблятися як новий	Дублювання блокується через idempotency key	Успішно
9	Перегляд торгових акаунтів	Перехід до сторінки Accounts	Відображаються підключені акаунти, статус, біржа, PnL і кількість угод	Дані акаунтів відображаються	Успішно
10	Розрахунок торгової аналітики	Завантаження сторінки аналітики або картки бота	Система розраховує PnL, win rate, drawdown і кількість угод	Показники формуються коректно	Успішно
11	Створення risk-профілю	Заповнення форми профілю OKX або Binance	Новий risk-профіль зберігається в системі	Профіль створюється	Успішно
12	Запуск risk-профілю	Натискання кнопки Start у Risk Manager	Профіль переходить у активний стан, починається оновлення даних	Профіль запускається	Успішно
13	Live-оновлення балансу	Запущений risk-профіль у Risk Manager	Баланс, PnL і статус торгівлі оновлюються без перезавантаження сторінки	Live-дані оновлюються	Успішно
14	Перевірка ризикових лімітів	Перевищення заданого ліміту збитку	Система блокує профіль або показує критичний статус	Перевищення ліміту фіксується	Успішно

ВИСНОВКИ

У ході виконання кваліфікаційної роботи проведено аналіз предметної галузі автоматизованої криптовалютної торгівлі, включаючи процеси керування торговими ботами, біржовими акаунтами, webhook-сигналами, історією угод, аналітичними показниками та ризиковими обмеженнями. Визначено, що основними проблемами цієї галузі є розрізнене зберігання даних у різних біржах і сервісах, складність одночасного контролю кількох торгових стратегій, ризик повторної обробки сигналів, обмеженість стандартних засобів risk-management та потреба у швидкому реагуванні на збиткові ситуації.

Досліджено існуючі програмні рішення для автоматизованої криптовалютної торгівлі, зокрема 3Commas, Cryptohopper, Coinrule, WunderTrading і TradingView. Встановлено, що такі системи мають корисні можливості для створення торгових ботів, роботи з біржовими API, використання сигналів і перегляду торгової аналітики. Водночас вони мають низку обмежень: залежність від тарифних планів, недостатню гнучкість власної серверної логіки, часткову реалізацію контролю ризиків і прив’язку до функціоналу конкретного зовнішнього сервісу.

Визначено функціональні та нефункціональні вимоги до платформи CryptoKeps. До основних функціональних вимог віднесено авторизацію користувача, перегляд торгових ботів і акаунтів, обробку webhook-сигналів, захист від дублювання сигналів через idempotency key, збереження історії угод, розрахунок PnL, win rate, drawdown, керування risk-профілями, live-моніторинг біржових профілів і надсилання Telegram-сповіщень. Серед нефункціональних вимог визначено швидкодію, стабільність роботи, захист API-ключів, масштабованість, зручність інтерфейсу та коректну обробку помилок зовнішніх біржових API.

Обґрунтовано вибір засобів реалізації платформи. Для серверної частини використано Java та Spring Boot, що забезпечують стабільну роботу backend-логіки, створення REST API, обробку торгових сигналів і взаємодію з базою даних. Для

клієнтської частини обрано React, TypeScript і Vite, які дозволяють реалізувати зручний web-інтерфейс. Для зберігання даних використано PostgreSQL, для керування міграціями — Liquibase, для модуля Risk Manager — Django, для live-оновлення — WebSocket, а для контейнеризованого запуску компонентів — Docker Compose.

Розроблено архітектуру платформи CryptoKeps, яка включає frontend-інтерфейс, Java Backend, Django Backend, Spring Cloud Gateway, PostgreSQL, watcher-сервіси для OKX і Binance, зовнішні біржові API та Telegram Bot. Спроєктовано логічну структуру системи, структуру бази даних, діаграму класів, діаграму розгортання, схему обробки webhook-сигналу та схему роботи Risk Manager. Така архітектура дозволяє розділити торгову логіку, користувацькі профілі, risk-management, live-моніторинг і збереження даних між окремими модулями.

Реалізовано основні функціональні модулі платформи: авторизацію користувача, профіль користувача, сторінки моніторингу, ботів, акаунтів, аналітики та Risk Manager. Серверна частина забезпечує роботу з торговими ботами, акаунтами, webhook-сигналами, історією угод і торговими показниками. Клієнтська частина дозволяє користувачу переглядати стан торгової системи, аналізувати результати роботи ботів, контролювати підключені акаунти та працювати з risk-профілями через єдиний web-інтерфейс.

Особливою функціональною частиною платформи є модуль Risk Manager. Він забезпечує створення, запуск, зупинку, редагування та видалення risk-профілів для бірж OKX і Binance. У межах модуля реалізовано перегляд балансу, доступного балансу, максимального балансу, PnL, статусу активної торгівлі, серії збиткових угод і стану блокування профілю. При перевищенні ризикових обмежень система може фіксувати критичний стан, блокувати профіль і надсилати повідомлення користувачу через Telegram.

Проведено тестування основних функцій платформи CryptoKeps. Перевірено авторизацію, навігацію між сторінками, відображення ботів і акаунтів, обробку тестових сигналів, захист від дублювання webhook-повідомлень, перегляд історії

угод, розрахунок торгових KPI, роботу Risk Manager, live-оновлення балансових показників і надсилання Telegram-сповіщень. Результати тестування підтвердили відповідність реалізованого програмного забезпечення поставленим вимогам.

Розроблена платформа CryptoKeps відповідає поставленій меті та може бути використана як основа для подальшого розвитку системи автоматизованої криптовалютної торгівлі. У перспективі систему можна розширити додаванням нових бірж, розширеної аналітики, журналу дій користувача, рольової моделі доступу, генерації PDF або Excel-звітів, глибшої інтеграції з TradingView та повноцінного виконання торгових операцій через біржові API.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Помазан О. О., Трінтіна Н. А. Розробка платформи керування криптотрейдинговими ботами. Матеріали II Всеукраїнської науково-технічної конференції «Виклики та рішення в програмній інженерії», 26 листопада 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. Київ: ДУІКТ, 2025. С. 508–511.
2. Помазан О. О., Трінтіна Н. А. Розробка платформи керування криптотрейдинговими ботами. Матеріали Всеукраїнської науково-технічної конференції «Технології цифрового розвитку: програмні системи та децентралізація», 1–3 травня 2026 року, Київ, ДУІКТ. Збірник тез. Подано до друку.

ПЕРЕЛІК ПОСИЛАНЬ

1. Закон України «Про віртуальні активи». Верховна Рада України. URL: https://zakon.rada.gov.ua/go/2074-20
2. Віртуальні активи. Міністерство цифрової трансформації України. URL: https://thedigital.gov.ua/projects/business/virtual_assets
3. Мінцифра – регулятор ринку віртуальних активів. Міністерство цифрової трансформації України. URL: https://thedigital.gov.ua/news/technologies/mintsifra-regulyator-rinku-virtualnikh-aktiviv
4. Розвінчуємо міфи про законопроєкт «Про віртуальні активи». Міністерство цифрової трансформації України. URL: https://thedigital.gov.ua/news/technologies/rozvinchuemo-mifi-pro-zakonoproekt-pro-virtualni-aktivi
5. Кірмач А. В. Законодавче забезпечення обігу віртуальних активів в Україні. Інформація і право. 2021. № 4(39). С. 49–56. URL: https://ippi.org.ua/sites/default/files/21_5.pdf
6. Лук'янчук Р. В. Сучасні виклики, пов'язані із розвитком криптоіндустрії. Інформація і право. 2022. URL: https://il.ippi.org.ua/article/view/254344
7. Лук'янчук Р. В. Проблемні питання правового регулювання обігу криптовалют в Україні. Інформація і право. 2023. URL: https://il.ippi.org.ua/article/view/282329/276526
8. Гунавардана Ш. С. Д. Автоматизована система торгових стратегій крипто-бота. Київ: КПІ ім. Ігоря Сікорського, 2023. URL: https://ela.kpi.ua/items/9ccbe2ef-9e05-475a-97f4-8f7a81d4c9bf
9. Песоцький О. В. Автоматизований бот для торгівлі на крипто-валютній біржі. Київ: КПІ ім. Ігоря Сікорського, 2020. URL: https://ela.kpi.ua/items/5e29825c-7044-4335-8a8d-6f3a2d0a1f34
10. Білий Д. Ю. Торговий бот для криптовалютних бірж. Київ: КПІ ім. Ігоря Сікорського, 2025. URL: https://ela.kpi.ua/items/f7608933-d9b4-4663-bc59-920c1fe697c6

11. Приходченко К. О. Програмна система автоматизованої торгівлі криптовалютами. Харків: ХНУРЕ, 2025. URL: https://openarchive.nure.ua/bitstreams/f3bb17c0-b5e3-4bc4-ae67-451e8b8ecdcd/download
12. Джура Є. С. Веб-орієнтована система прогнозування цін криптовалют та акцій. Харків: ХНУРЕ, 2025. URL: https://openarchive.nure.ua/bitstreams/c1384716-811c-4f1c-948a-b5332b0ce5c4/download
13. Бузова К. В. Розробка торгового бота з використанням методів машинного навчання. Харків: ХНУРЕ, 2024. URL: https://openarchive.nure.ua/bitstreams/ea45c707-e162-4f67-907b-d863b63c4253/download
14. 3Commas. Crypto Trading Bots & Automation Platform. 3Commas. URL: https://3commas.io/
15. Cryptohopper. Crypto Trading Bot for Automated Trading. Cryptohopper. URL: https://www.cryptohopper.com/
16. Coinrule. Crypto Trading Bot. Coinrule. URL: https://coinrule.com/
17. WunderTrading. Crypto Trading Bot for Smart Automated Trading. WunderTrading. URL: https://wundertrading.com/
18. TradingView. How to configure webhook alerts. TradingView. URL: https://www.tradingview.com/support/solutions/43000529348-how-to-configure-webhook-alerts/
19. Spring Boot Reference Documentation. Spring. URL: https://docs.spring.io/spring-boot/reference/
20. React Documentation. React. URL: https://react.dev/
21. Vite Guide. Vite. URL: https://vite.dev/guide/
22. PostgreSQL Documentation. PostgreSQL. URL: https://www.postgresql.org/docs/
23. Liquibase Documentation. Liquibase. URL: https://docs.liquibase.com/
24. Binance Developers. WebSocket API General Info. Binance. URL: https://developers.binance.com/docs/derivatives/usds-margined-futures/websocket-api-general-info

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка платформи керування криптотрейдинговими ботами

Виконав студент 4 курсу
групи ПД-41
Помазан Олександр Олександрович
Керівник роботи
доц., канд.техн.наук Трінтіна Наталія Альбертівна

Київ - 2026

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Провести аналіз предметної галузі автоматизованої криптовалютної торгівлі та визначити основні проблеми керування торговими ботами, сигналами, біржовими акаунтами й ризиками.
2. Дослідити існуючі програмні рішення для автоматизованої торгівлі, порівняти їх функціональні можливості та визначити основні недоліки.
3. Сформувати функціональні та нефункціональні вимоги до платформи CryptoKeps.
4. Обґрунтувати вибір технологій для реалізації web-платформи, зокрема Java Spring Boot, React, PostgreSQL, Django, WebSocket та Docker Compose.
5. Спроєктувати архітектуру платформи, структуру бази даних та основні UML-діаграми системи.
6. Реалізувати серверну частину для керування торговими ботами, обробки webhook-сигналів, збереження історії угод і розрахунку торгової аналітики.
7. Реалізувати клієнтський web-інтерфейс для перегляду ботів, акаунтів, аналітики та Risk Manager.
8. Розробити модуль Risk Manager для live-моніторингу біржових профілів OKX і Binance, контролю ризикових обмежень та Telegram-сповіщень.
9. Провести тестування основних функцій платформи та оцінити відповідність розробленого програмного забезпечення поставленим вимогам.

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи – спрощення процесу керування автоматизованою криптовалютною торгівлею.

Об'єкт дослідження – процеси автоматизованого керування криптовалютною торгівлею, торговими ботами, біржовими акаунтами, торговими сигналами та ризиковими обмеженнями.

Предмет дослідження – програмна платформа для керування криптовалютними торговими ботами, обробки webhook-сигналів, моніторингу біржових профілів, аналізу результатів торгівлі та контролю ризиків.

АНАЛІЗ АНАЛОГІВ

Показник	3Commas	Cryptohopper	Coinrule	WunderTrading	TradingView	CryptoKeps
Керування торговими ботами	+	+	+	+	-	+
Підключення бірж через API	+	+	+	+	частково	+
Webhook-сигнали	+	+	частково	+	+	+
Інтеграція з TradingView	+	+	частково	+	власна система alert	+
Торгова аналітика	+	+	частково	+	+	+
Risk-management	частково	частково	частково	частково	-	+
Live-моніторинг балансів	частково	частково	-	частково	-	+
Telegram-сповіщення	частково	-	-	частково	через зовнішні інтеграції	+
Гнучкість власної серверної логіки	-	-	-	-	частково через Pine Script	+
Підтримка декількох бірж	+	+	+	+	+	+

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги

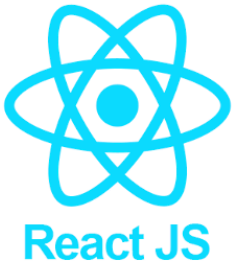
1. Керування ботами — створення, перегляд, запуск, зупинка та видалення торгових ботів.
2. Пошук і фільтрація — відбір ботів за назвою, біржею, акаунтом і статусом.
3. Обробка webhook-сигналів — приймання сигналів від TradingView або інших сервісів.
4. Захист від дублікатів — перевірка повторних сигналів через idempotency key.
5. Історія угод — збереження даних про угоди, ціни входу/виходу, SL, TP, PnL і час.
6. Торгова аналітика — розрахунок PnL, win rate, drawdown і кількості угод
7. Risk Manager — створення та налаштування профілів ризику для OKX і Binance.
8. Live-моніторинг — перегляд балансу, PnL, активності торгівлі та статусу блокування.
9. Автоблокування — блокування профілю при перевищенні ліміту збитку.
10. Telegram-сповіщення — повідомлення про критичні події та закриття позицій.

Нефункціональні вимоги

1. Завантаження сторінок — до 3 с.
2. Відповідь API — до 1 с.
3. WebSocket-оновлення — кожні 1–5 с.
4. Доступність системи — не менше 99% у тестовому середовищі.
5. Захист ключів — API-ключі зберігаються у зашифрованому вигляді.
6. Масштабованість — не менше 50 ботів і 100 risk-профілів.
7. Обсяг історії — не менше 10 000 угод.
8. Зручність — основні дії за 2–4 кліки.
9. Сумісність — Chrome, Edge, Firefox.
10. Обробка помилок — помилка біржі не зупиняє всю платформу.

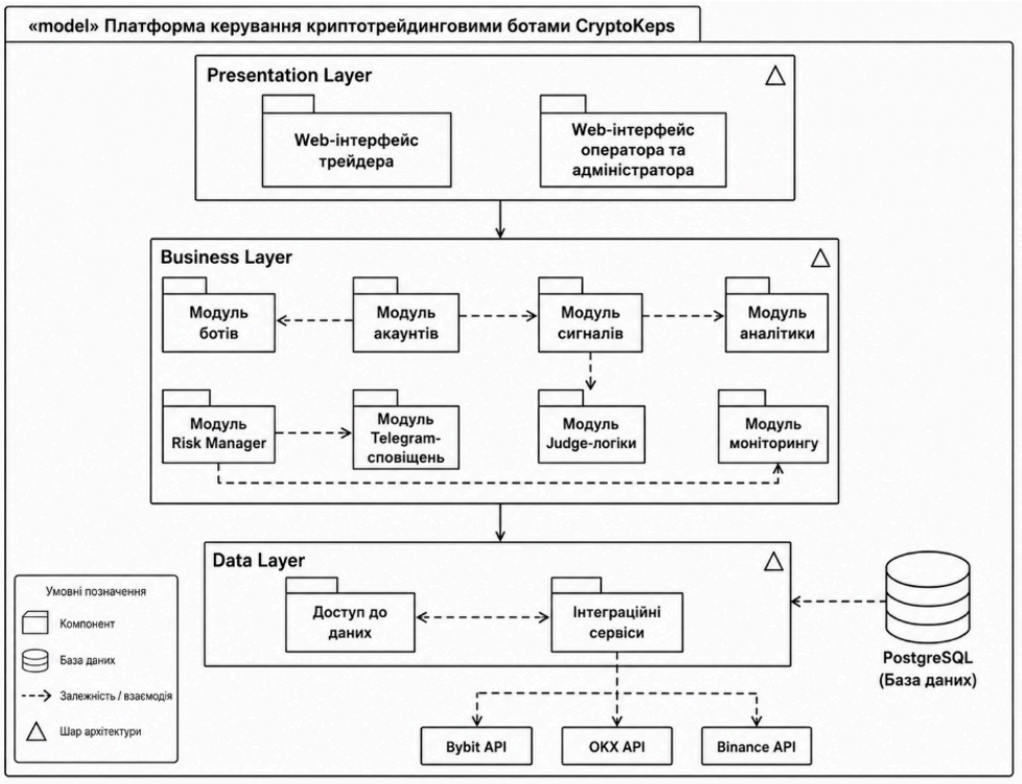
5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ



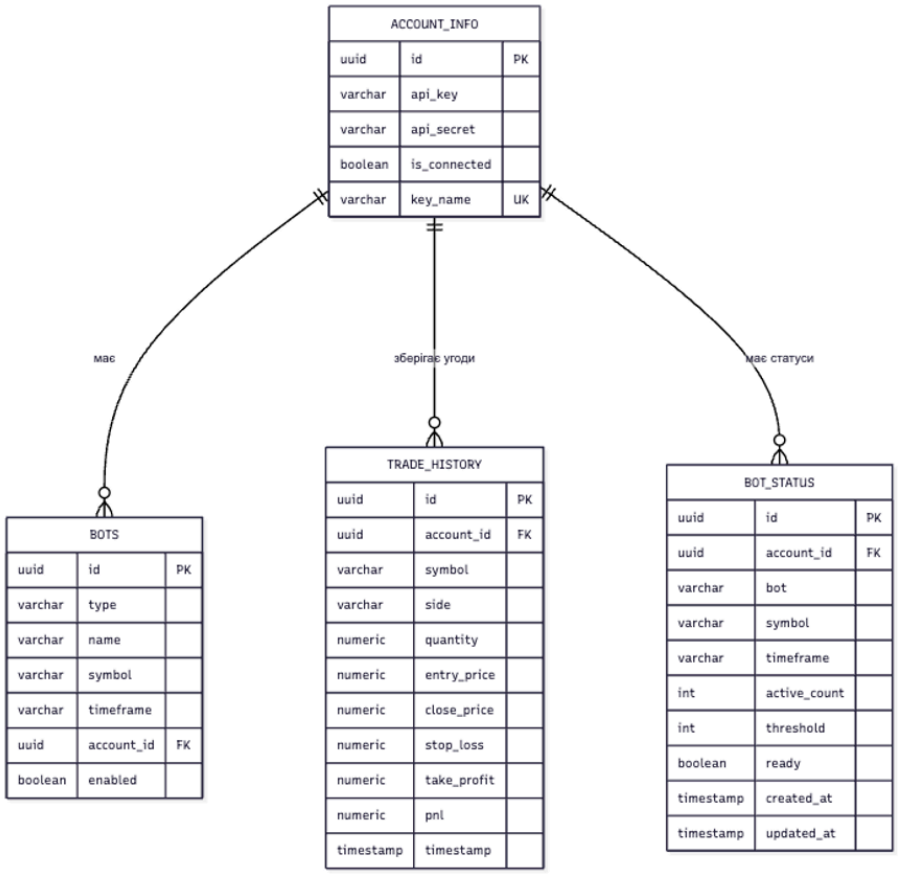
6

ЛОГІЧНА СТРУКТУРА ПРОЕКТУ



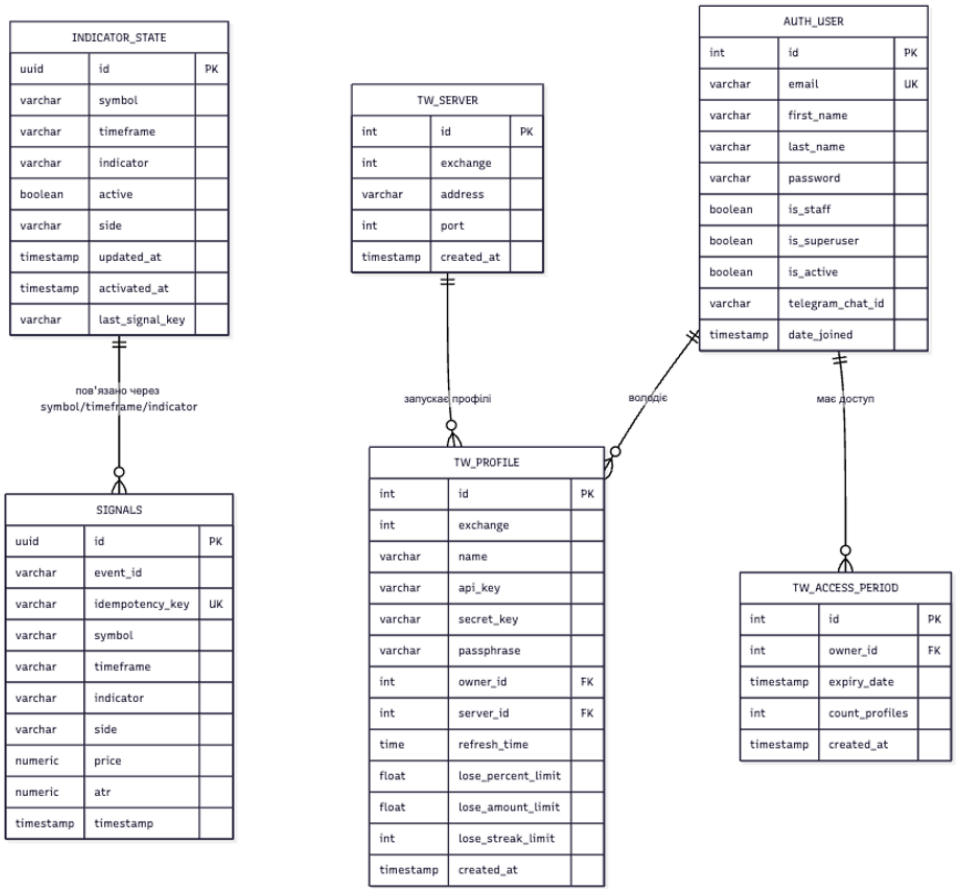
7

СТРУКТУРА БАЗИ ДАНИХ (Ч.1)



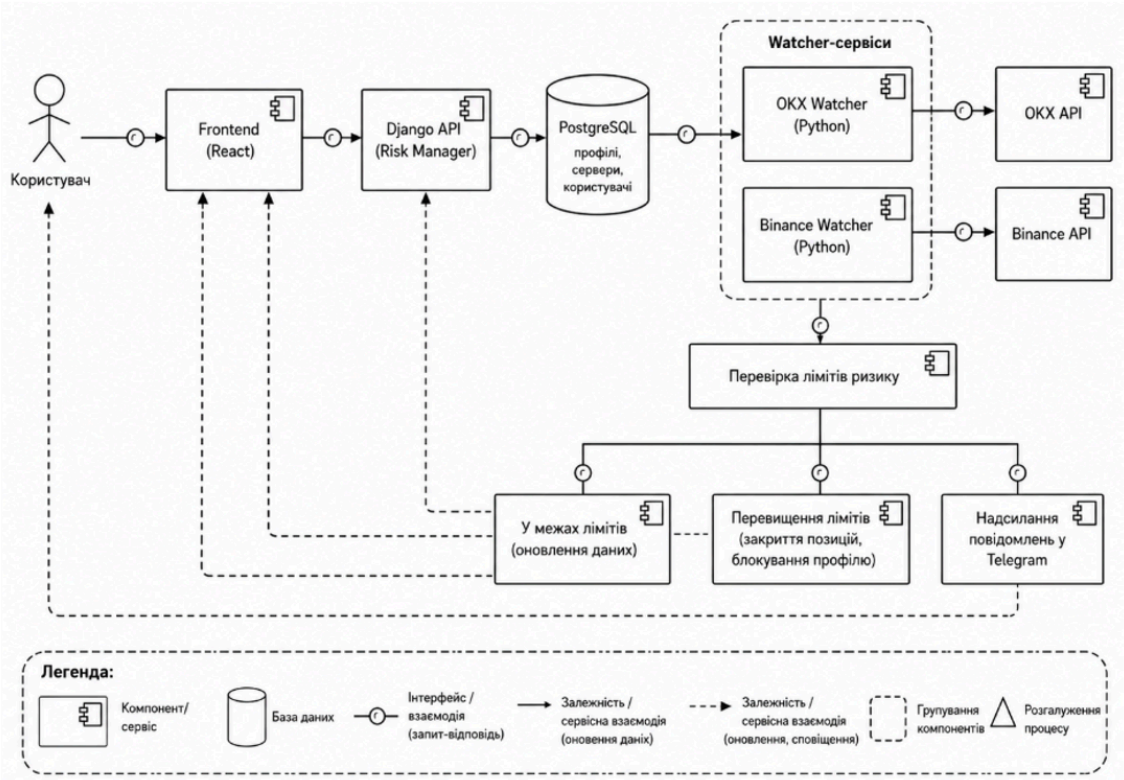
8

СТРУКТУРА БАЗИ ДАНИХ(Ч.2)



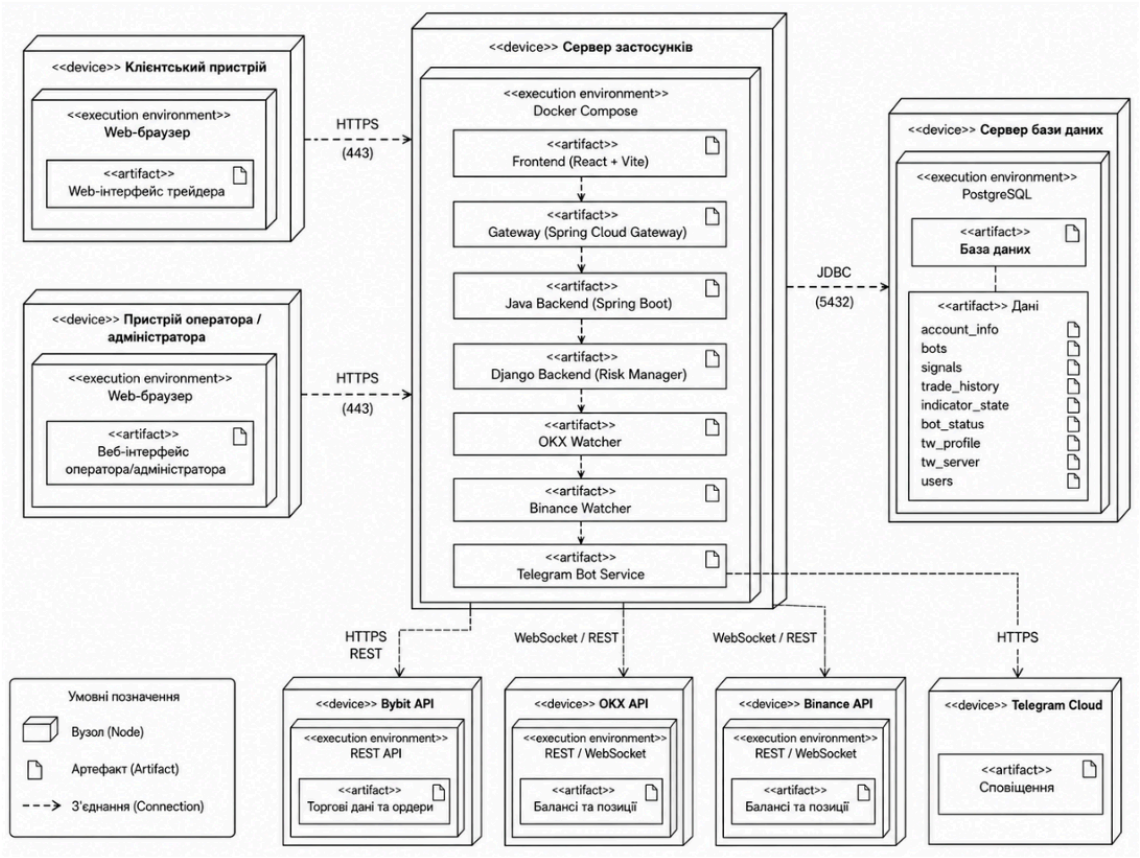
9

ДІАГРАМА КОМПОНЕНТІВ МОДУЛЮ RISK MANAGER TA WATCHER-СЕРВІСІВ

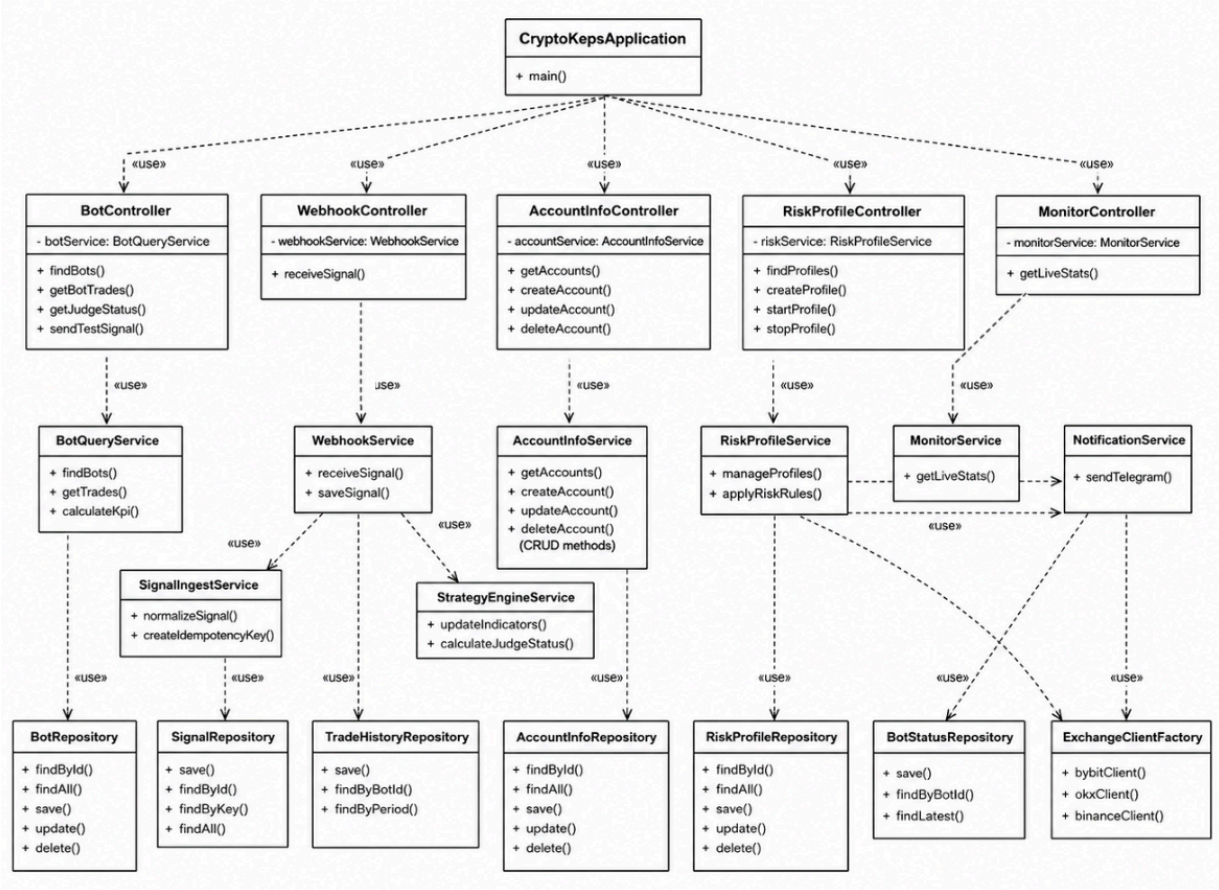


10

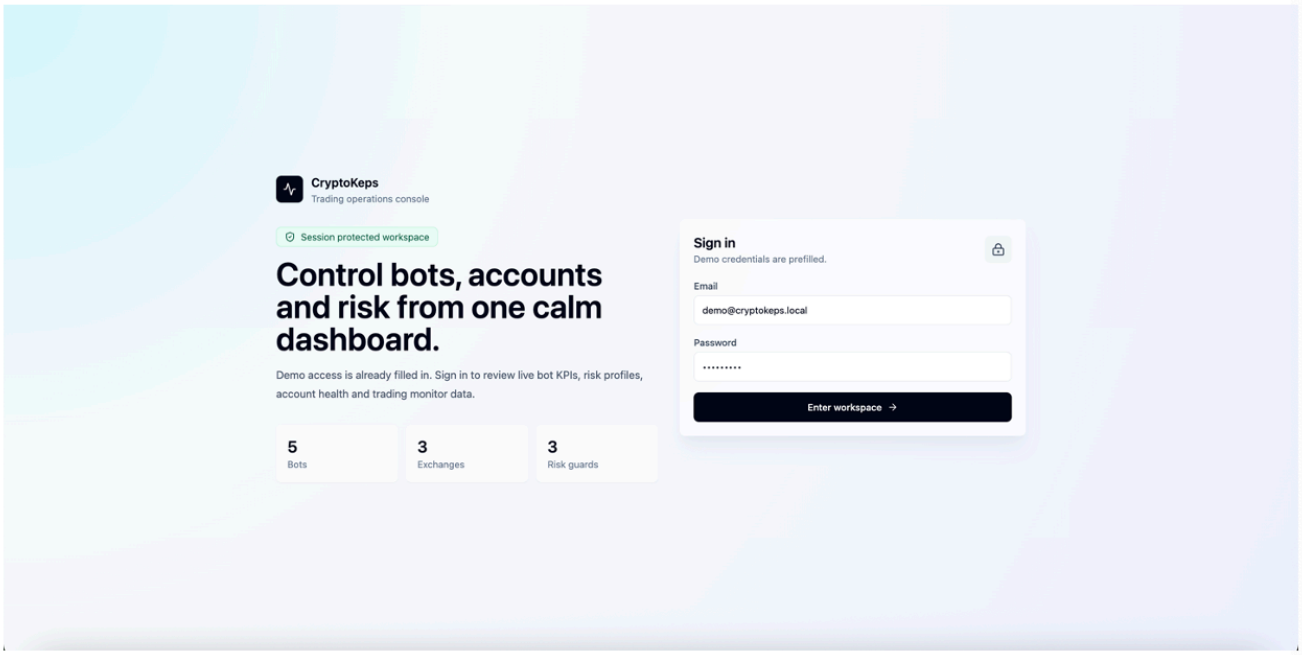
ДІАГРАМА РОЗГОРТАННЯ



ДІАГРАМА КЛАСІВ



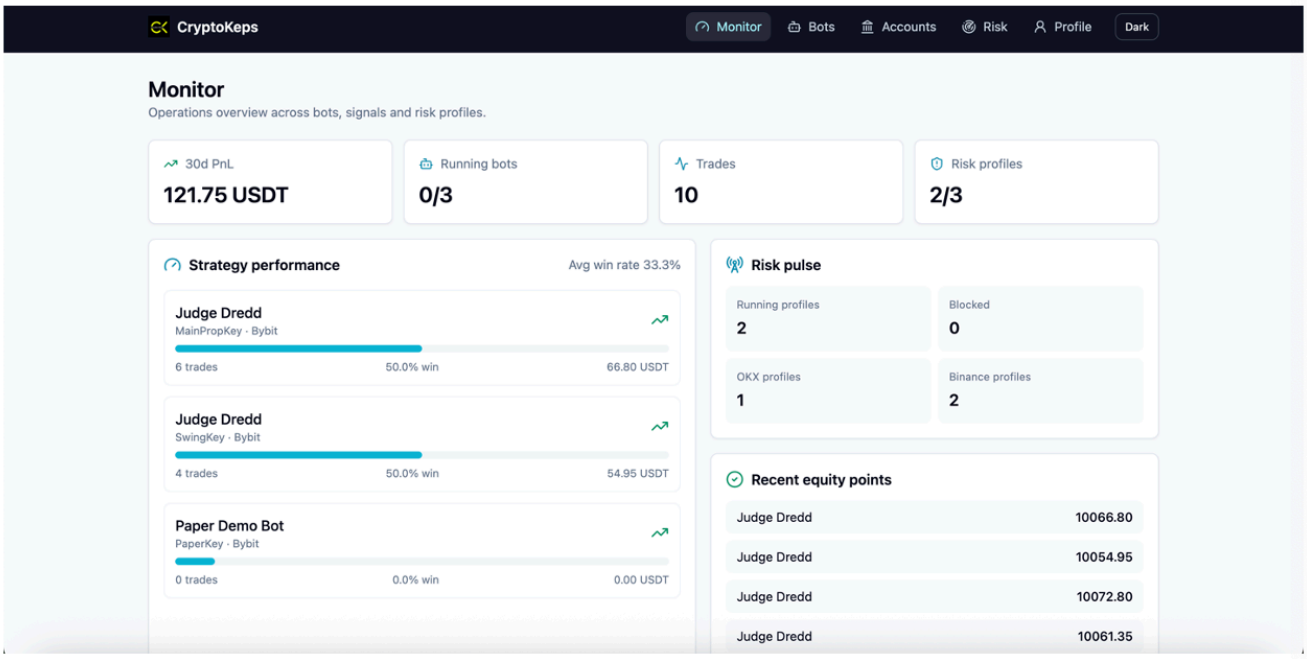
ЕКРАННА ФОРМА



ВХІД

13

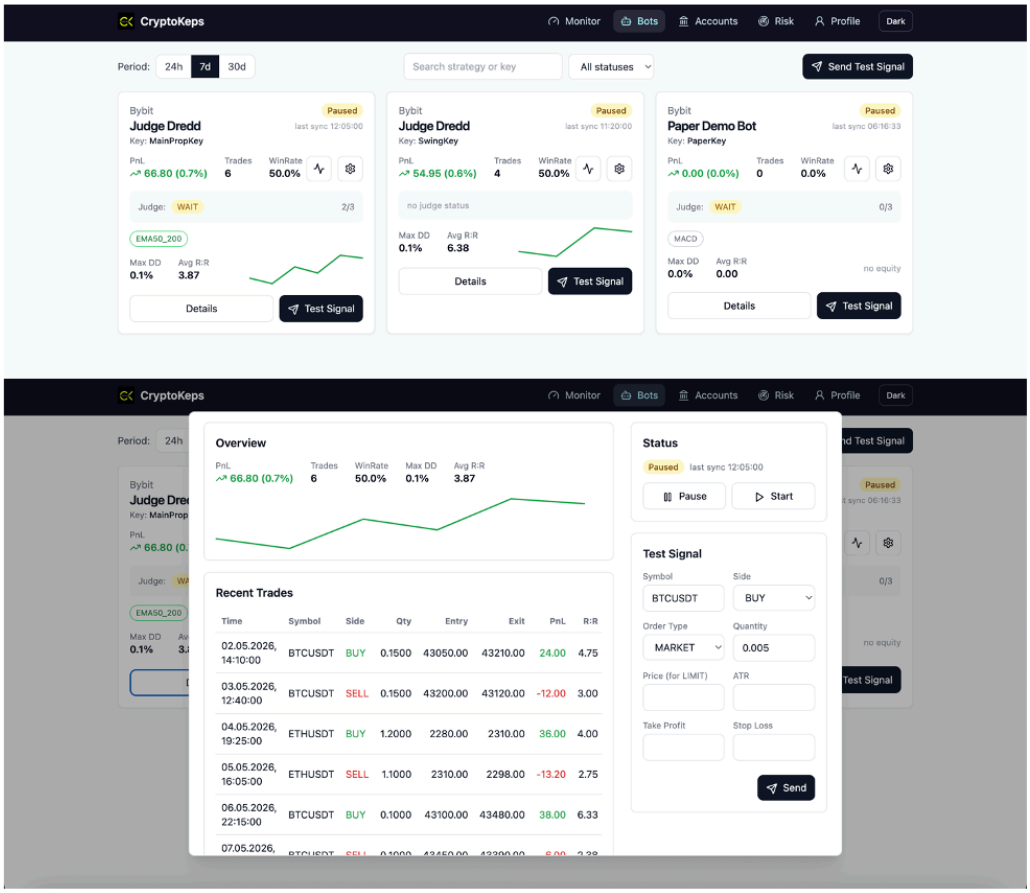
ЕКРАННА ФОРМА



АНАЛІТИКА

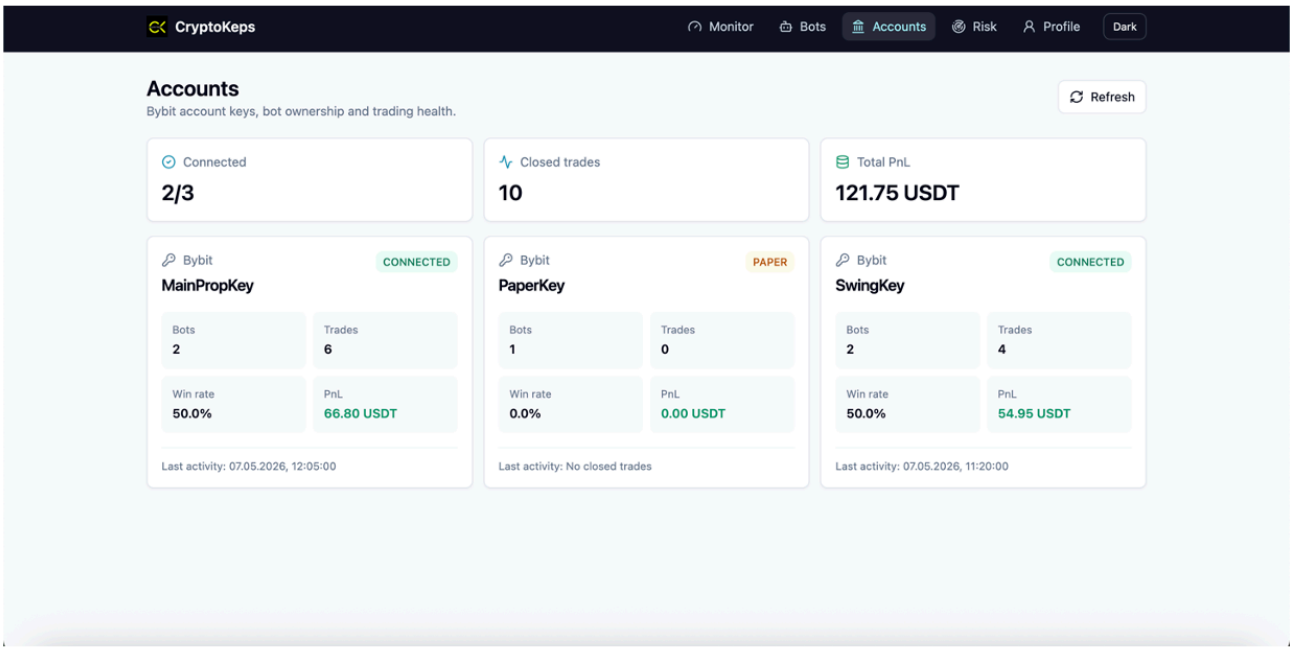
14

ЕКРАННА ФОРМА – БОТИ



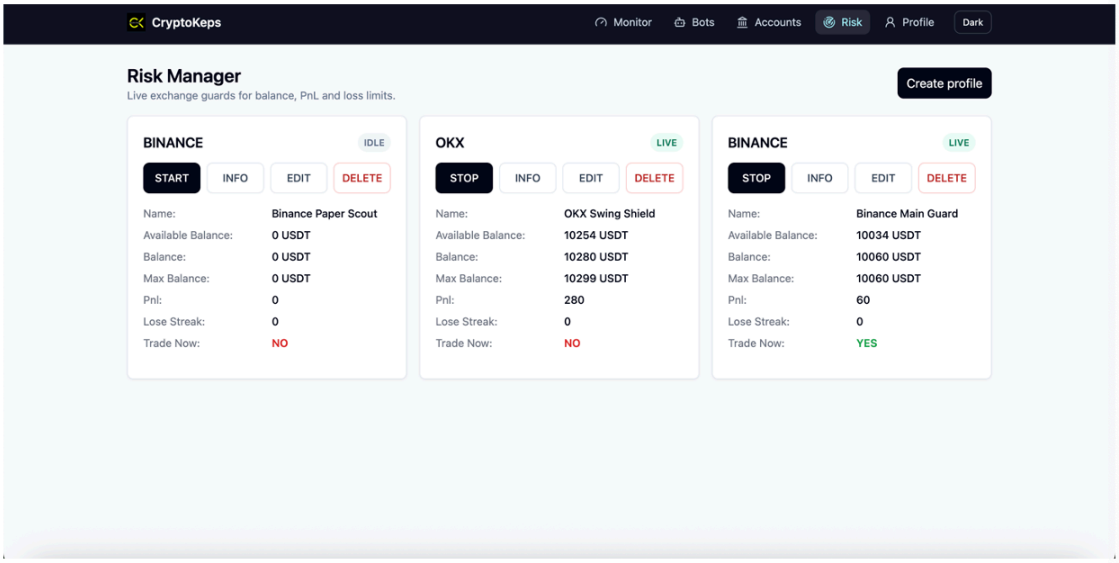
15

ЕКРАННА ФОРМА



АКАУНТИ

ЕКРАННА ФОРМА



RISK MANAGER

17

ВИСНОВКИ

1. Проведено аналіз предметної галузі автоматизованої криптовалютної торгівлі та визначено основні проблеми керування торговими ботами, біржовими акаунтами, webhook-сигналами й ризиковими обмеженнями.
2. Досліджено існуючі програмні рішення для автоматизованої торгівлі, зокрема 3Commas, Cryptohopper, Coinrule, WunderTrading і TradingView.
3. Визначено їхні переваги та недоліки, серед яких обмежена гнучкість, залежність від тарифів і недостатній risk-management.
4. Сформовано функціональні та нефункціональні вимоги до платформи CryptoKeps, зокрема керування ботами, обробка сигналів, збереження історії угод, торгова аналітика, live-моніторинг і Telegram-сповіщення.
5. Обґрунтовано вибір технологій для реалізації платформи: Java Spring Boot, React, PostgreSQL, Django, WebSocket, Spring Cloud Gateway та Docker Compose.
6. Спроєктовано архітектуру платформи, структуру бази даних і основні UML-діаграми, що відображають логіку взаємодії користувача, frontend, backend, біржових API та Risk Manager.
7. Визначено подальший напрям розробки платформи CryptoKeps: реалізація серверної частини, web-інтерфейсу, модуля Risk Manager, live-моніторингу біржових профілів і тестування основних функцій системи.

20

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

```
package com.trading.bybit_api.entity;

import
com.trading.bybit_api.domain.strategy.BotTyp
e;
import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.time.LocalDateTime;
import java.util.UUID;

@Data
@Entity
@Builder(toBuilder = true)
@AllArgsConstructor
@NoArgsConstructor
@Table(name = "bot_status",
      uniqueConstraints =
@UniqueConstraint(columnNames =
{"symbol", "timeframe", "bot"}))
public class BotStatusEntity {

   @Id
   @GeneratedValue(strategy =
GenerationType.UUID)
   private UUID id;

   @Column(name = "account_id", nullable =
false)
   private UUID accountId;

   @Column(name = "symbol", nullable =
false)
   private String symbol;

   @Column(name = "timeframe", nullable =
false)
   private String timeframe;

   @Enumerated(EnumType.STRING)
   @Column(name = "bot", nullable = false)
   private BotType bot;

   @Column(name = "active_count", nullable =
false)
   private int activeCount;

   @Column(name = "threshold", nullable =
false)
   private int threshold;

   @Column(name = "ready", nullable = false)
   private boolean ready;

   @Column(name = "created_at", nullable =
false)
   private LocalDateTime createdAt;

   @Column(name = "updated_at", nullable =
false)
   private LocalDateTime updatedAt;
}
package com.trading.api.controller;
```

```
import com.trading.api.dto.BotCardDto;
import
com.trading.api.service.BotQueryService;
import lombok.RequiredArgsConstructor;
import
org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/bots")
@RequiredArgsConstructor
public class BotController {

   private final BotQueryService service;

   @GetMapping
   public List<BotCardDto> list(
         @RequestParam(value = "q", required
= false) String q,
         @RequestParam(value = "status",
required = false) String status,
         @RequestParam(value = "period",
defaultValue = "24h") String period
   ) {
      return service.listBots(q, status, period);
   }
}
```

```
package com.trading.bybit_api.controller;

import
com.trading.bybit_api.dto.WebhookSignalDto;
import
com.trading.bybit_api.service.interfaces.IWebh
ookService;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import
org.springframework.http.ResponseEntity;
import
org.springframework.web.bind.annotation.*;

@Slf4j
@RestController
@RequestMapping("/api/v1/webhook")
@RequiredArgsConstructor
public class WebhookController {

   private final IWebhookService
webhookService;

   @PostMapping("/signal")
   public ResponseEntity<String>
processTradingSignal(@RequestBody
WebhookSignalDto signal) {
      String responseMessage =
webhookService.processSignal(signal);
      return
ResponseEntity.ok(responseMessage);
   }
}
```

```
package com.trading.bybit_api.service.impl;

import
com.trading.bybit_api.dto.WebhookSignalDto;
import
com.trading.bybit_api.service.interfaces.*;
```

```
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import
org.springframework.stereotype.Service;

@Slf4j
@Service
@RequiredArgsConstructor
public class WebhookServiceImpl implements
IWebhookService {
   private final ISignalIngestService
signalIngestService;
   private final IndicatorStateServiceImpl
indicatorStateService;
   private final StrategyEngineImpl
strategyEngine;

   @Override
   public String
processSignal(WebhookSignalDto signal) {
      log.info("Processing signal: {}", signal);
      try {
         var s =
signalIngestService.normalize(signal);
         if (!s.isValid()) return "invalid";
         if (!signalIngestService.recordIfNew(s))
return "duplicate";

         indicatorStateService.apply(s);
         var st =
strategyEngine.onStateChanged(s.getSymbol(),
s.getTimeframe());
         strategyEngine.publish(st);
         return "ok";
      } catch (Exception e) {
         log.error("Error processing trading
signal", e);
         return "Failed";
      }
   }
}
package com.trading.bybit_api.service.impl;

import
com.trading.bybit_api.domain.strategy.Normali
zedSignal;
import
com.trading.bybit_api.entity.SignalEntity;
import
com.trading.bybit_api.repository.SignalReposit
ory;
import
com.trading.bybit_api.service.interfaces.ISigna
lIngestService;
import
com.trading.bybit_api.dto.WebhookSignalDto;
import
com.trading.bybit_api.service.util.WebhookMa
pper;
import lombok.RequiredArgsConstructor;
import
org.springframework.stereotype.Service;

import java.time.Instant;
import java.time.LocalDateTime;
import java.time.ZoneOffset;

@Service
@RequiredArgsConstructor
```

```
public class SignalIngestServiceImpl
implements ISignalIngestService {

   private final SignalRepository signalRepo;

   @Override
   public NormalizedSignal
normalize(WebhookSignalDto dto) {
      return WebhookMapper.toDomain(dto);
   }

   @Override
   public boolean
recordIfNew(NormalizedSignal s) {
      String key = s.idempotencyKey();
      if
(signalRepo.findByIdempotencyKey(key).isPre
sent()) return false;

      var e = new SignalEntity();
      e.setEventId(key);
      e.setIdempotencyKey(key);
      e.setSymbol(s.getSymbol());
      e.setTimeframe(s.getTimeframe());
      e.setIndicator(s.getIndicator());
      e.setSide(s.getSide());
      e.setPrice(s.getPrice());
      e.setAtr(s.getAtr());

e.setTs(LocalDateTime.ofInstant(Instant.ofEpo
chMilli(s.getTs()), ZoneOffset.UTC));

      signalRepo.save(e);
      return true;
   }
}
package com.trading.bybit_api.service.impl;

import
com.trading.bybit_api.domain.strategy.Normali
zedSignal;
import
com.trading.bybit_api.domain.strategy.Side;
import
com.trading.bybit_api.entity.IndicatorStateEntit
y;
import
com.trading.bybit_api.repository.IndicatorState
Repository;
import
com.trading.bybit_api.service.interfaces.IIndica
torStateService;
import lombok.RequiredArgsConstructor;
import
org.springframework.stereotype.Service;

import java.time.Instant;
import java.time.LocalDateTime;
import java.time.ZoneOffset;

@Service
@RequiredArgsConstructor
public class IndicatorStateServiceImpl
implements IIndicatorStateService {

   private final IndicatorStateRepository repo;

   @Override
   public void apply(NormalizedSignal s) {
```

```
        var state =
repo.findBySymbolAndTimeframeAndIndicato
r(
                s.getSymbol(),
                s.getTimeframe(),
                s.getIndicator()
        ).orElseGet(IndicatorStateEntity::new);

        state.setSymbol(s.getSymbol());
        state.setTimeframe(s.getTimeframe());
        state.setIndicator(s.getIndicator());

        boolean active = (s.getSide() ==
Side.LONG);
        var signalTime =
LocalDateTime.ofInstant(
                Instant.ofEpochMilli(s.getTs()),
                ZoneOffset.UTC
        );

        state.setActive(active);
        state.setSide(s.getSide().name());
        state.setUpdatedAt(signalTime);

state.setLastSignalKey(s.idempotencyKey());
        state.setActivatedAt(active ? signalTime :
state.getActivatedAt());

        repo.save(state);
    }
}
package com.trading.bybit_api.service.impl;

import
com.trading.bybit_api.domain.strategy.BotStat
us;
import
com.trading.bybit_api.domain.strategy.BotTyp
e;
import
com.trading.bybit_api.repository.BotRepositor
y;
import
com.trading.bybit_api.repository.BotStatusRep
ository;
import
com.trading.bybit_api.repository.IndicatorState
Repository;
import
com.trading.bybit_api.service.interfaces.IStrate
gyEngine;
import lombok.RequiredArgsConstructor;
import
org.springframework.stereotype.Service;

import java.time.Instant;
import java.time.LocalDateTime;
import java.time.ZoneOffset;

@Service
@RequiredArgsConstructor
public class StrategyEngineImpl implements
IStrategyEngine {

    private final IndicatorStateRepository
stateRepo;
    private final BotStatusRepository
statusRepo;
```

```
    private final BotRepository botRepo;

    private static final int
DEFAULT_THRESHOLD = 3;

    @Override
    public BotStatus onStateChanged(String
symbol, String timeframe) {
        int active = (int)
stateRepo.countBySymbolAndTimeframeAnd
ActiveTrue(symbol, timeframe);
        int threshold =
DEFAULT_THRESHOLD;
        boolean ready = active >= threshold;
        var now =
LocalDateTime.ofInstant(Instant.now(),
ZoneOffset.UTC);

        var bot =
botRepo.findFirstBySymbolAndTimeframeAn
dTypeAndEnabledTrue(
                symbol,
                timeframe,
                BotType.JUDGE
        ).orElse(null);

        var e =
statusRepo.findBySymbolAndTimeframeAndB
ot(symbol, timeframe, BotType.JUDGE)
                .orElseGet(() -> {
                    var ne = new
com.trading.bybit_api.entity.BotStatusEntity();
                    ne.setSymbol(symbol);
                    ne.setTimeframe(timeframe);
                    ne.setBot(BotType.JUDGE);
                    ne.setCreatedAt(now);
                    return ne;
                });

        if (bot != null) {
            e.setAccountId(bot.getAccountId());
        }

        if (e.getAccountId() == null) {
            return new BotStatus(symbol,
timeframe, BotType.JUDGE, active, threshold,
ready, Instant.now());
        }

        e.setActiveCount(active);
        e.setThreshold(threshold);
        e.setReady(ready);
        if (e.getCreatedAt() == null)
e.setCreatedAt(now);
        e.setUpdatedAt(now);
        statusRepo.save(e);

        return new BotStatus(symbol, timeframe,
BotType.JUDGE, active, threshold, ready,
Instant.now());
    }

    @Override
    public void publish(BotStatus status) {
        // TODO: WS/SSE/Telegram
    }
}
package
com.trading.bybit_api.domain.strategy;
```

```
import lombok.Builder;
import lombok.Value;
import lombok.extern.jackson.Jacksonized;

import java.math.BigDecimal;

@Value
@Builder(toBuilder = true)
@Jacksonized
public class NormalizedSignal {
   String eventId;
   String symbol;
   String timeframe;
   String indicator;
   Side side;
   BigDecimal price;
   BigDecimal atr;
   long ts;

   public String idempotencyKey() {
      if (eventId != null && !eventId.isBlank())
return eventId;
      return symbol + ":" + timeframe + ":" +
indicator + ":" + side + ":" + ts;
   }

   public boolean isValid() {
      return symbol != null &&
!symbol.isBlank()
            && indicator != null && side != null
            && price != null &&
price.compareTo(BigDecimal.ZERO) > 0;
   }
}
```

```
package
com.trading.bybit_api.domain.strategy;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Value;
import lombok.extern.jackson.Jacksonized;

import java.time.Instant;

@Value
@AllArgsConstructor
@Builder(toBuilder = true)
@Jacksonized
public class BotStatus {
   String symbol;
   String timeframe;
   BotType bot;
   int activeCount;
   int threshold;
   boolean ready;
   Instant updatedAt;
}
```

```
from django.contrib.auth.models import
AbstractUser
from django.db import models

from auth.managers import UserManager


class User(AbstractUser):
   username = None
   email = models.EmailField(unique=True)
```

```
    telegram_chat_id =
models.CharField(max_length=20,
blank=True, null=True)

    USERNAME_FIELD = 'email'
    REQUIRED_FIELDS = []

    objects = UserManager()

    def __str__(self):
        return self.email
from django.db import models
from django.conf import settings
from cryptography.fernet import Fernet
from auth.models import User
from tw.constants import EXCHANGES

fernet =
Fernet(settings.CRYPTOGRAPHY_KEY)


def _decrypt_or_empty(token: str) -> str:
    if not token:
        return ""
    try:
        return
fernet.decrypt(token.encode()).decode()
    except Exception:
        return token


def _encrypt_or_empty(value: str) -> str:
    if not value:
        return ""
    return
fernet.encrypt(value.encode()).decode()


class TWServer(models.Model):
    exchange =
models.PositiveIntegerField(choices=EXCHA
NGES)
    address = models.CharField(max_length=20)
    port = models.PositiveSmallIntegerField()
    created_at =
models.DateTimeField(auto_now_add=True)


class TWProfile(models.Model):
    exchange =
models.PositiveIntegerField(choices=EXCHA
NGES)
    name = models.CharField(max_length=50)

    _api_key =
models.CharField(max_length=255,
blank=True, default="", db_column='api_key')
    _secret_key =
models.CharField(max_length=255,
blank=True, default="",
db_column='secret_key')
    _passphrase =
models.CharField(max_length=255,
blank=True, default="",
db_column='passphrase')

    owner = models.ForeignKey(User,
related_name="tw_profiles",
```

```
on_delete=models.DO_NOTHING,
db_index=True)
    server = models.ForeignKey(TWServer,
related_name="tw_profiles",
on_delete=models.DO_NOTHING, null=True,
blank=True)

    refresh_time = models.TimeField(null=True,
blank=True)
    lose_percent_limit =
models.FloatField(null=True, blank=True)
    lose_amount_limit =
models.FloatField(null=True, blank=True)
    lose_streak_limit =
models.PositiveSmallIntegerField(null=True,
blank=True)
    created_at =
models.DateTimeField(auto_now_add=True)

    @property
    def is_started(self) -> bool:
        return self.server_id is not None

    @property
    def api_key(self) -> str:
        return _decrypt_or_empty(self._api_key)

    @api_key.setter
    def api_key(self, value: str):
        self._api_key = _encrypt_or_empty(value)

    @property
    def secret_key(self) -> str:
        return
_decrypt_or_empty(self._secret_key)

    @secret_key.setter
    def secret_key(self, value: str):
        self._secret_key =
_encrypt_or_empty(value)

    @property
    def passphrase(self) -> str:
        return
_decrypt_or_empty(self._passphrase)

    @passphrase.setter
    def passphrase(self, value: str):
        self._passphrase =
_encrypt_or_empty(value)


class TWAccessPeriod(models.Model):
    owner = models.ForeignKey(
        User,
        related_name="tw_access_periods",
        on_delete=models.DO_NOTHING,
        db_index=True
    )
    expiry_date = models.DateTimeField()
    count_profiles =
models.PositiveSmallIntegerField()
    created_at =
models.DateTimeField(auto_now_add=True)
```