ДОДАТОК А

Структура бази даних системи

База даних arbitrage.db (SQLite) містить три незалежні таблиці: arbitrage_signals (історія виявлених сигналів з дедуплікацією), backtest_runs (журнал запусків бектестингу) та portfolio (single-row таблиця депозиту). Між таблицями навмисно не введено зовнішніх ключів - сутності концептуально автономні, а як «м'які посилання» використовуються string-поля symbol, buy_exchange, sell_exchange (обґрунтування – у підрозділі 2.3.2). DDL-визначення схеми наведено у лістингу А.1.

Лістинг А.1 – SQL-схема таблиць бази даних системи

```
 CREATE TABLE IF NOT EXISTS arbitrage_signals (
    id              INTEGER PRIMARY KEY AUTOINCREMENT,
    detected_at     TEXT    NOT NULL,
    symbol          TEXT    NOT NULL,
    buy_exchange    TEXT    NOT NULL,
    sell_exchange   TEXT    NOT NULL,
    buy_price       REAL    NOT NULL,
    sell_price      REAL    NOT NULL,
    spread_pct      REAL    NOT NULL,
    fee_pct         REAL    NOT NULL,
    net_profit_pct REAL    NOT NULL,
    investment      REAL    NOT NULL,
    profit_usd      REAL    NOT NULL,
    risk_level      TEXT    NOT NULL,
    risk_score      INTEGER NOT NULL,
    tier            INTEGER DEFAULT 0,
    rank            INTEGER DEFAULT 0,
    quote_volume_24h REAL  DEFAULT 0
);

CREATE TABLE IF NOT EXISTS backtest_runs (
    id             INTEGER PRIMARY KEY AUTOINCREMENT,
    run_at         TEXT NOT NULL, symbol TEXT NOT NULL,
timeframe TEXT NOT NULL,
    start_date     TEXT NOT NULL, end_date TEXT NOT NULL,
    exchange_buy   TEXT NOT NULL, exchange_sell TEXT NOT NULL,
    total_signals INTEGER NOT NULL, win_rate_pct REAL NOT
NULL,
    total_profit_pct REAL NOT NULL, max_drawdown_pct REAL NOT
NULL,
    sharpe_ratio   REAL
);
```

```
CREATE TABLE IF NOT EXISTS portfolio (
    id          INTEGER PRIMARY KEY AUTOINCREMENT,
    created_at  TEXT NOT NULL,
    deposit_usd REAL NOT NULL
);
```

Розширення схеми (колонки tier, rank, quote_volume_24h) виконано програмною міграцією _migrate_signals_table через PRAGMA table_info + ALTER TABLE ADD COLUMN, що забезпечує сумісність зі старими версіями БД без втрати даних.

Таблиця А.1 – Призначення основних полів таблиці arbitrage_signals

Поле	Тип	Призначення
id	INTEGER	первинний ключ, автоінкремент
detected_at	TEXT	час виявлення сигналу (ISO-8601)
symbol	TEXT	торгова пара (наприклад, BTC/USDT)
buy_exchange / sell_exchange	TEXT	біржі купівлі та продажу
buy_price / sell_price	REAL	ціни купівлі та продажу
spread_pct	REAL	валовий спред, %
fee_pct	REAL	сумарна комісія бірж, %
net_profit_pct	REAL	чистий прибуток після комісій, %
profit_usd	REAL	очікуваний прибуток у доларах
risk_level / risk_score	TEXT / INTEGER	рівень та числова оцінка ризику
tier / rank	INTEGER	рівень ліквідності та ранг пари
quote_volume_24h	REAL	добовий обсяг торгів у котирувальній валюті

Таблиця backtest_runs зберігає журнал запусків бектестингу (пара, таймфрейм, період, біржі, кількість сигналів, win-rate, сумарний прибуток, максимальну просадку.

ДОДАТОК Б

Лістинги ключових модулів

Нижче наведено репрезентативні фрагменти ключових модулів системи (повний код – у репозиторії проєкту) (лістинги Б.1–Б.6).

Лістинг Б.1 – Модуль core.arbitrage (ядро двобіржевого арбітражу)

```
     def calculate_arbitrage(df, investment,
min_profit_pct=0.1):
    valid = df[(df['Bid'] > 0) & (df['Ask'] > 0)]
    if len(valid) < 2:
        return _empty_result(exchange_count=len(valid))
    i_min = valid['Ask'].idxmin(); j_max =
valid['Bid'].idxmax()
    buy_price, sell_price = valid.loc[i_min,'Ask'],
valid.loc[j_max,'Bid']
    buy_id, sell_id = valid.loc[i_min,'Exchange_ID'],
valid.loc[j_max,'Exchange_ID']
    gross = (sell_price - buy_price) / buy_price * 100
    is_valid = (buy_id != sell_id) and (sell_price >
buy_price)
    net_pct, fee_pct = calculate_net_profit(buy_id, sell_id,
gross)
    return {'buy_exchange_id': buy_id, 'sell_exchange_id':
sell_id,
            'spread_pct': gross, 'fee_pct': fee_pct,
'net_profit_pct': net_pct,
            'profit_usd': investment * net_pct / 100,
            'is_valid': is_valid, 'is_profitable': is_valid
and net_pct > min_profit_pct}
```

Лістинг Б.2 – Модуль core.orderbook (оцінювання executable USD)

```
     def estimate_executable_usd(levels, side, investment_usd):
    remaining, executed = investment_usd, 0.0
    ordered = sorted(levels, key=lambda x: x[0], reverse=(side
== 'sell'))
    for price, amount in ordered:
        if remaining <= 0: break
        take = min(remaining, price * amount)
        executed += take; remaining -= take
    return executed
```

Лістинг Б.3 – Модуль core.exchanges.pool (batch-збір тикерів з відмовостійкістю)

```
     async def fetch_all_exchanges_tickers(symbols):
    tasks = [_fetch_one(eid, symbols) for eid in EXCHANGES]
    results = await asyncio.gather(*tasks,
return_exceptions=True)
    return {eid: (res if isinstance(res, dict) else {})
            for eid, res in zip(EXCHANGES, results)}
```

Лістинг Б.4 – Модуль core.risk (оцінювання рівня ризику сигналу)

```
     def assess_risk(spread_pct, volatility, liquidity_depth,
exchange_count):
    score = 0
    score += 30 if spread_pct > 1.0 else 15 if spread_pct >
0.4 else 0
    score += 25 if volatility > 2.0 else 12 if volatility >
1.0 else 0
    score += 20 if liquidity_depth < 5000 else 8 if
liquidity_depth < 20000 else 0
    score += 15 if exchange_count < 3 else 0
    level = 'high' if score >= 60 else 'medium' if score >= 30
else 'low'
    return {'risk_score': min(score, 100), 'risk_level':
level}
```

Лістинг Б.5 – Модуль database.queries (дедуплікація сигналів за часовим вікном)

```
     def should_insert_signal(conn, signal, window_sec=300):
    row = conn.execute(
        "SELECT detected_at FROM arbitrage_signals "
        "WHERE symbol=? AND buy_exchange=? AND sell_exchange=?
"
        "ORDER BY id DESC LIMIT 1",
        (signal['symbol'], signal['buy_exchange'],
signal['sell_exchange'])
    ).fetchone()
    if row is None:
        return True
    last = datetime.fromisoformat(row[0])
    return (datetime.utcnow() - last).total_seconds() >
window_sec
```

Лістинг Б.6 – Модуль core.market.fair_price (розрахунок VWAP за рівнями стакану)

```
 def fair_price_vwap(levels):
    total_qty = sum(amount for _, amount in levels)
    if total_qty == 0:
        return 0.0
    return sum(price * amount for price, amount in levels) /
total_qty
```

ДОДАТОК В

Схематичні представлення інтерфейсу системи

Інтерфейс системи побудовано за однорівневою плоскою навігацією: постійна бічна панель (sidebar) із блоками керування та центральна робоча область із сімома вкладками. Схематичні представлення (wireframe) кожної вкладки наведено на рисунках В.1–В.7. Кольорове кодування інтерфейсу послідовне: зелений – позитив/купівля, червоний – негатив/продаж, бірюзовий – нейтральна інформація.

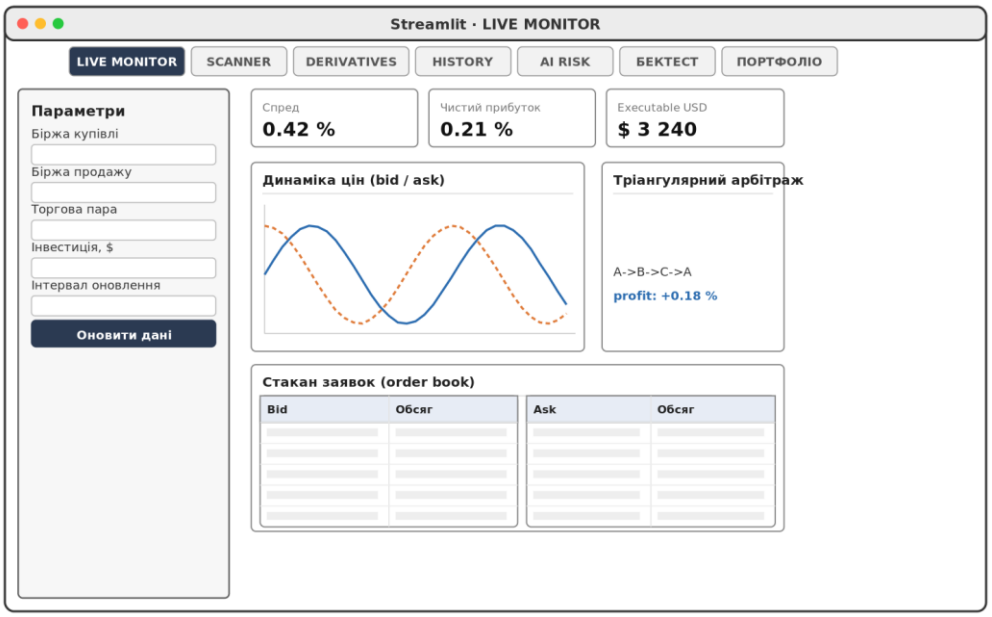


Рисунок В.1 – Вкладка LIVE MONITOR: метрики спреду, динаміка цін, тріангулярний арбітраж та стакан заявок

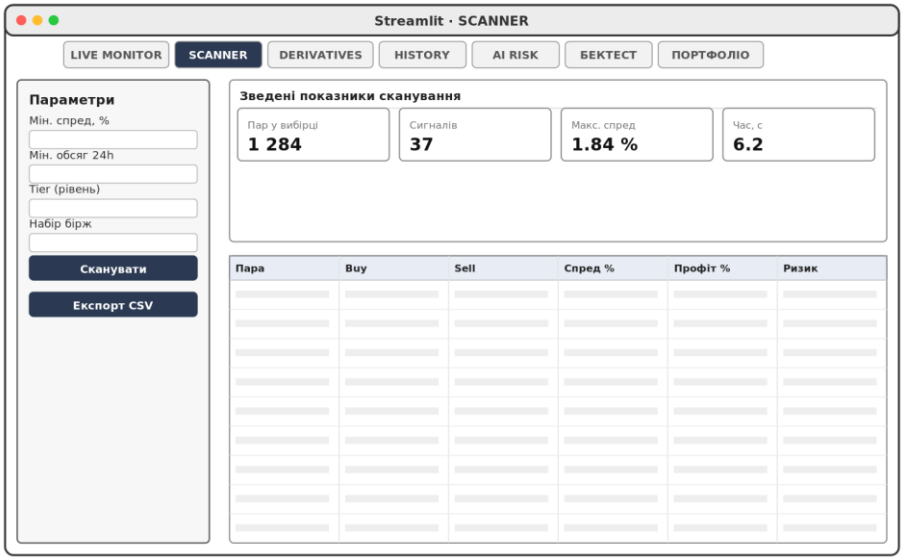


Рисунок В.2 – Вкладка SCANNER: масове сканування universe торгових пар із фільтрами та експортом у CSV

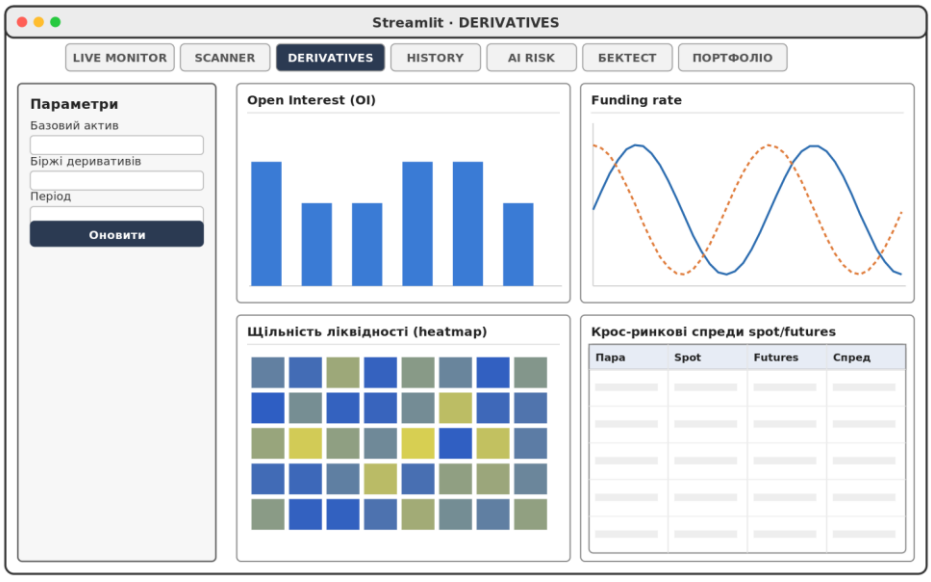


Рисунок В.3 – Вкладка DERIVATIVES: open interest, funding rate, щільність ліквідності та крос-ринкові спреди

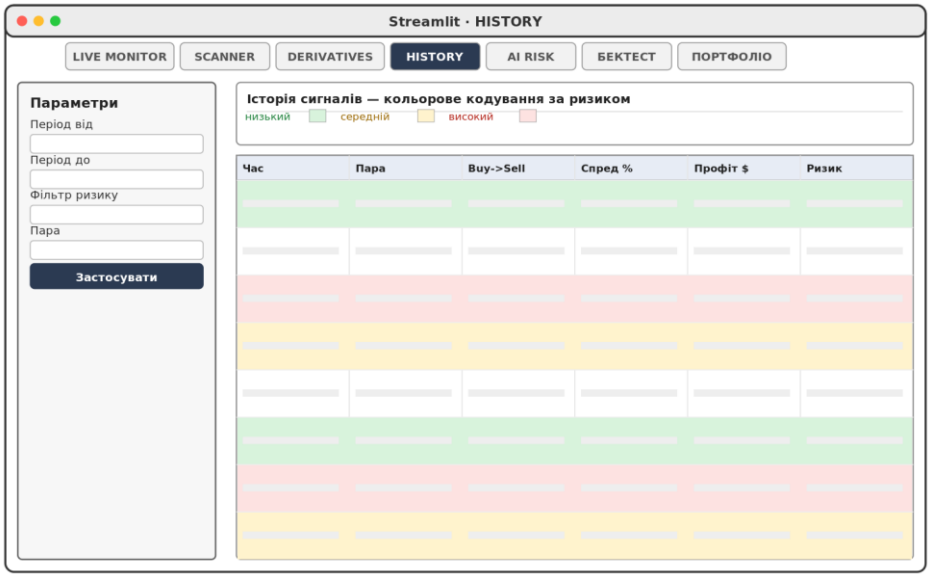


Рисунок В.4 – Вкладка HISTORY: історія виявлених сигналів із кольоровим кодуванням за рівнем ризику

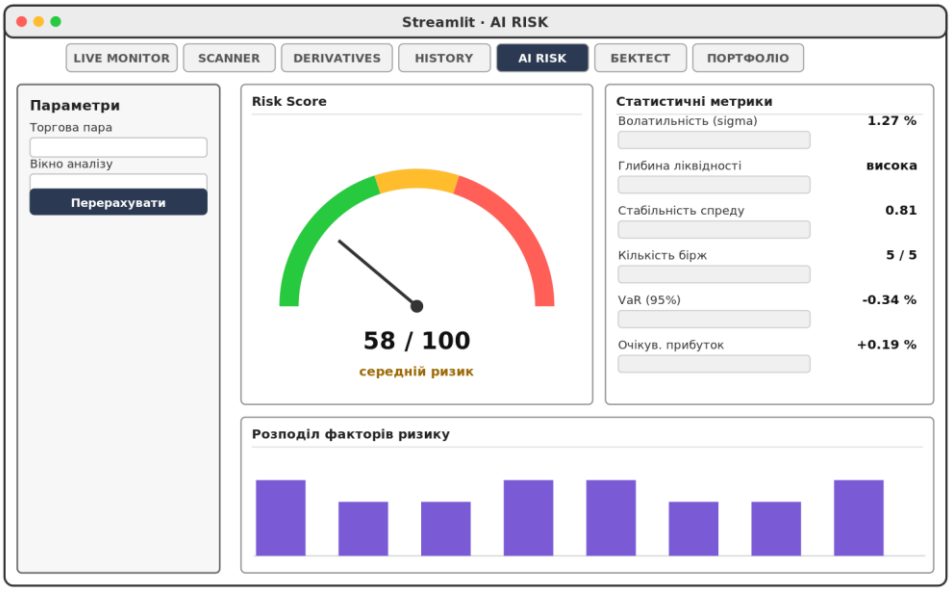


Рисунок В.5 – Вкладка AI RISK: гейдж Risk Score та панель статистичних метрик ризику



Рисунок В.6 – Вкладка БЕКТЕСТИНГ: параметри симуляції, крива капіталу та журнал угод



Рисунок В.7 – Вкладка ПОРТФОЛІО: депозит, баланс, крива капіталу та розподіл прибутку

ДОДАТОК Г

Звіт виконання тестів і фрагмент журналу подій

Для перевірки коректності ключових модулів реалізовано набір автоматизованих тестів на базі pytest. Нижче наведено результат прогону модульних тестів (лістинг Г.1), фрагмент журналу подій під час інтеграційного тестування (лістинг Г.2) та зведені підсумки тестування (таблиця Г.1).

Лістинг Г.1 – Результат прогону модульних тестів (pytest -v)

```
============================= test session starts
==============================
platform linux -- Python 3.10.12, pytest-9.0.3, pluggy-1.6.0
rootdir: /Users/.../dyploma   configfile: pytest.ini
testpaths: tests
collected 9 items

tests/test_core.py::test_level_price_amount_two_fields
PASSED   [ 11%]
tests/test_core.py::test_level_price_amount_three_fields
PASSED   [ 22%]
tests/test_core.py::test_normalize_levels
PASSED   [ 33%]
tests/test_core.py::test_fair_price_vwap
PASSED   [ 44%]
tests/test_core.py::test_arbitrage_valid_spread
PASSED   [ 55%]
tests/test_core.py::test_arbitrage_same_exchange_invalid
PASSED  [ 66%]
tests/test_core.py::test_spread_row
PASSED   [ 77%]
tests/test_core.py::test_demo_spreads_types             PASSED
[ 88%]
tests/test_queries.py::test_should_insert_signal_dedup  PASSED
[100%]

============================== 9 passed in 1.01s
==============================
```

Лістинг Г.2 – Фрагмент журналу подій arbitrage.log (інтеграційне тестування)

```
2026-05-19 14:32:18 [INFO] core.market.ticker_service:
snapshot symbols=1 duration=1.42s
2026-05-19 14:32:18 [INFO] core.market.exchange_health:
online=5/5 (100%)
2026-05-19 14:35:21 [WARNING] core.exchanges.pool:
fetch_tickers_batch coinbase: ConnectionError(...)
2026-05-19 14:35:21 [INFO] core.market.exchange_health:
online=3/5 (60%)
2026-05-19 14:42:09 [INFO] backtesting.engine: backtest run
completed: 716 candles, 247 signals
```

Таблиця Г.1 – Зведені підсумки модульного тестування

Група тестів	К-ть	Результат
Нормалізація рівнів стакану	3	пройдено
Розрахунок справедливої ціни (VWAP)	1	пройдено
Арбітражна логіка	2	пройдено
Аналіз спредів	1	пройдено
Демо-генератори тикерів	1	пройдено
Дедуплікація сигналів у БД	1	пройдено
Усього	9	пройдено

ДОДАТОК Д

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	74 с.
	Інші документи	
2.	Презентаційні матеріали	__ плакатів

.	.	Прізвище та ініціали.	Підп.	Дата			
Розробив		Котенко К.О.			Інтелектуальна система пошуку арбітражних можливостей між криптобіржами	Шифр групи	Код напр./спец.
Перевірив		Малєєв Д.В.				ІТШІ-22-1	122
Н.контр.		Малєєва І.А.				ХНУРЕ кафедра ШІ	
Затв.		Чала Л.Е.					