UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

AI-POWERED PREDICTIVE TRADING ANALYTICS PLATFORM

David Alexander Colorado Rodríguez 20211020031
Andres Felipe Martín Rodriguez 20201020137

Data Bases II
Carlos Andres Sierra Virgüez

Universidad Distrital Francisco José de Caldas
Systems Engineering Curriculum Project
Colombia, Bogotá D.C.
May 13, 2025

Contents

1 Business Model Canvas

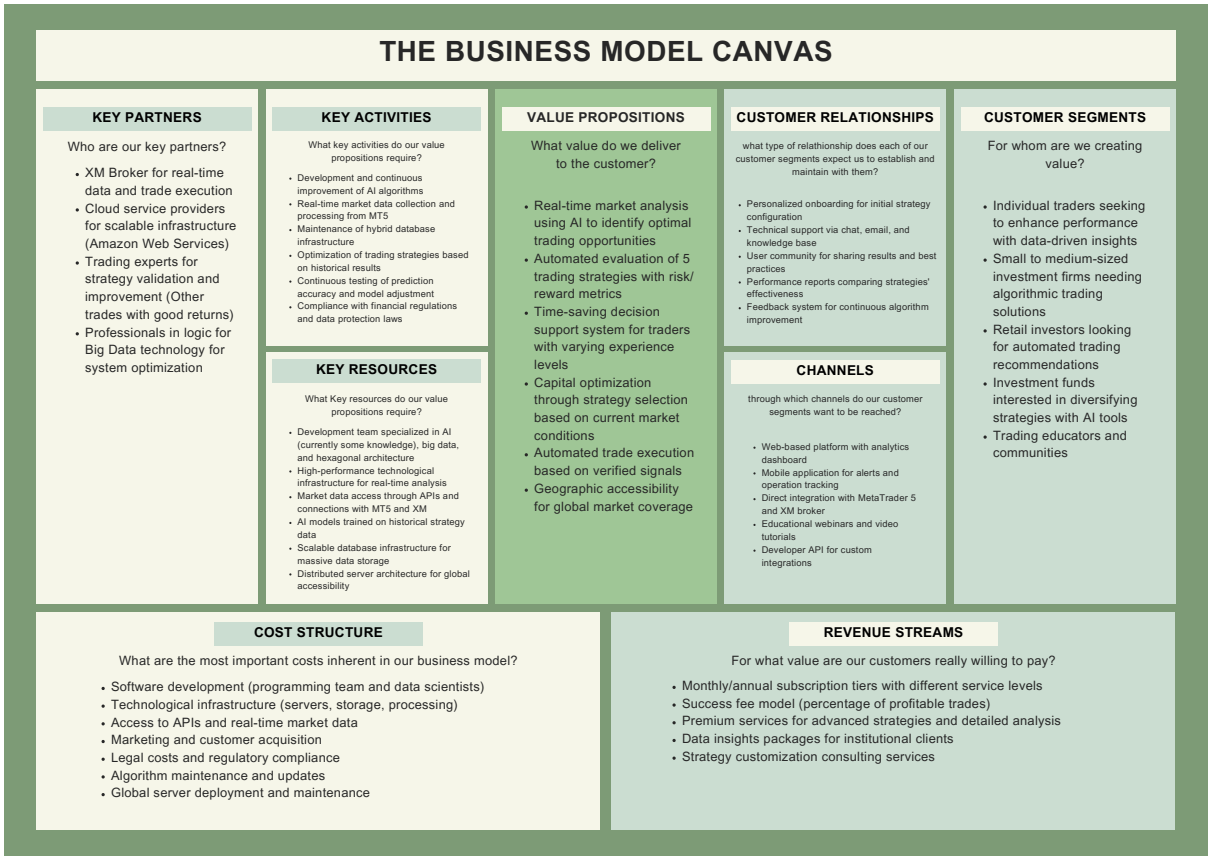

Figure 1: Business Model

2 User Stories

2.1 Use Stories – Individual Trader

Title: Real-Time Trading Alerts	**Priority:** High	**Estimate:** –
User Story: As an individual trader, I want to receive real-time alerts about buying/selling opportunities based on the analysis of the 5 strategies so that I can act quickly on market movements.		
Acceptance Criteria: Given the strategy is enabled When the system detects a market signal Then an alert is instantly delivered to the user		

Title: Mobile Platform Access	**Priority:** High	**Estimate:** –
User Story: As an individual trader, I want to access the trading platform from my mobile device so that I can stay updated and execute trades on the go.		
Acceptance Criteria: Given I have internet access When I log in from mobile Then I can use all trading features		

Title: Trade History Review	**Priority:** Medium	**Estimate:** –
User Story: As an individual trader, I want to review my past trades and strategy performance so that I can improve my future decisions.		
Acceptance Criteria: Given I'm logged in When I navigate to my trade history Then I can filter and view past operations		

2.2 Use Stories – Portfolio Manager

Title: Daily Strategy Performance Reports	**Priority:** High	**Estimate:** –
User Story: As a portfolio manager, I want to receive daily reports on each strategy's performance under various market conditions so that I can optimize capital distribution.		
Acceptance Criteria: Given I'm subscribed to reports When the day ends Then I receive a detailed summary of strategy results		

Title: Capital Allocation Simulation	**Priority:** High	**Estimate:** –
User Story: As a portfolio manager, I want to simulate how allocating capital across different strategies would have performed in the past so that I can make data-driven decisions.		
Acceptance Criteria: Given I input allocation percentages When I run the simulation Then the system displays historical performance results		

Title: Risk/Reward Monitoring	**Priority:** Medium-High	**Estimate:** –
User Story: As a portfolio manager, I want to monitor the risk/reward ratio of my strategy mix in real time so that I can proactively rebalance the portfolio.		
Acceptance Criteria: Given my portfolio is active When market data changes Then the system recalculates and alerts on deviations		

2.3 Use Stories – Advanced Trader

Title: AI Model Configuration	**Priority:** High	**Estimate:** –
User Story: As an advanced trader, I want to fine-tune AI model parameters so that I can adapt strategy behavior to my trading approach.		
Acceptance Criteria: Given I access the configuration panel When I change a model parameter Then recommendations are adjusted accordingly		

Title: Strategy Backtesting	**Priority:** High	**Estimate:** –
User Story: As an advanced trader, I want to run backtests with custom parameters so that I can validate my theories before trading live.		
Acceptance Criteria: Given I select a strategy and timeframe When I run a backtest Then the system shows detailed historical performance		

3 Data Sources and Structures

To obtain the raw financial data, given their ease of integration, usability in the market and wide recognition, the Yahoo Finance and Alpha Vantage sources were selected.

Yahoo Finance (yfinance)

Overview: yfinance is a popular library that provides access to Yahoo Finance data, including stock prices, financial statements, and historical data. Some features are:

- Retrieve historical market data
- Fetch fundamental data (income statements, balance sheets, cash flows)
- Get real-time stock prices
- Download options data

Alpha Vantage

Overview: Alpha Vantage provides free access to real-time and historical financial market data via APIs. Some features are:

- Stock, forex, and cryptocurrency data
- Fundamental data
- Technical indicators
- Economic indicators

Structures

The main structures that potentially will be needed are:

Customer data

User ID, name, email, strategy preferences, subscription level, transaction history.
Format: Relational database (e.g., PostgreSQL).

Transaction data

Trade ID, timestamp, strategy used, financial instrument, volume, result (profit/loss).
Format: Relational database with indexes for quick queries.

Real-time market data

Stock prices, forex, cryptocurrencies; technical indicators; volume.
Format: non-relational database (e.g., MongoDB) to handle massive flows.

Analytical data

Historical performance of strategies, risk/reward metrics, benchmark comparisons.
Format: data warehouse (e.g., Snowflake) for aggregated analysis.

Configuration data

User-customized parameters for AI models.
Format: Relational database.

4 Expected Data Volume and Transactions

Assumptions were used based on the customer segments and their needs as described in the user stories.
Users: Assume 10,000 individual traders, 500 investment firms and 1,000 retail investors in the first year (11,500 total users).
Interactions per user: Individual traders: 10 daily alerts, 5 trades executed.
Firms: 50 daily alerts, 20 trades.
Retail investors: 5 daily alerts, 2 trades.
Data generated per trade: 1 KB (includes timestamp, instrument, volume, result).
Daily volume:
Alerts: (10,000 × 10) + (500 × 50) + (1,000 × 5) = 130,000 alerts.
Trades: (10,000 × 5) + (500 × 20) + (1,000 × 2) = 62,000 trades.
Total data: 62,000 × 1 KB = 62 MB/day (trades only) + market data (1 GB/day per tracked exchange).
Monthly volume: 1.86 GB (transactions) + 30 GB (market) = 32 GB/month (minimum).
Daily transactions: 62,000 trades executed.

5 Potential Problems and Considerations

Real-time data processing

Problem: Constant data ingestion from multiple exchanges (NYSE, NASDAQ, etc.) can overload infrastructure if not optimized.
Consideration: Use message queuing (e.g., Kafka) and distributed processing (e.g., Spark).

Scalability

Problem: User and data growth can degrade performance.
Consideration: Implement geographic replication and auto-scaling in AWS.

Security

Problem: Sensitive data (transactions, configurations) are targets for attacks.
Consideration: AES-256 encryption, multi-factor authentication and regular audits.

Regulatory compliance

Problem: Financial regulations vary by country (e.g., GDPR, SEC).
Consideration: Store data by region and generate automated compliance reports.

6 Database Design

6.1 Define Components

The components in a hexagonal architecture separate the core domain from external concerns. For your trading platform, these components are:

6.1.1 Domain Layer (Core)

Trading Domain

- Strategy evaluation logic
- Signal generation algorithms
- Risk calculation engines
- Portfolio optimization models

User Domain

- Authentication business rules
- Permission management
- Subscription handling logic

6.1.2 Ports (Interfaces)

Primary Ports (API Interfaces)

- StrategyService: Interface for strategy operations
- TradeService: Interface for trade execution
- AlertService: Interface for notification generation
- PortfolioService: Interface for portfolio management
- UserService: Interface for user operations

Secondary Ports (Repository Interfaces)

- StrategyRepository: Interface for strategy persistence
- TradeRepository: Interface for trade persistence
- MarketDataRepository: Interface for market data retrieval
- UserRepository: Interface for user persistence
- AlertRepository: Interface for alert persistence

6.1.3 Adapters

Primary Adapters (Driving)

- RESTController: REST API endpoints
- WebSocketController: Real-time communication
- ScheduledTaskRunner: Time-based operations

Secondary Adapters (Driven)

- PostgreSQLRepository: Implementation for relational data
- MongoDBRepository: Implementation for high-volume data
- YahooFinanceAdapter: Market data integration
- AlphaVantageAdapter: Additional market data
- XMBrokerAdapter: Trade execution with XM

6.2 Define Entities

6.2.1 User Domain Entities

1. **User**: System users with different roles (individual traders, portfolio managers, advanced traders)
2. **Subscription**: User subscription plans and payment status
3. **Role**: User permission roles (could be embedded in User or separate)
4. **UserPreference**: User settings and UI preferences

6.2.2 Trading Domain Entities

1. **Strategy**: Trading strategies with their logic and parameters
2. **StrategyConfig**: User-specific strategy configurations
3. **Trade**: Trading decisions and outcomes
4. **TradeExecution**: Broker interactions for trade execution
5. **Alert**: Trading opportunity notifications

6.2.3 Market Data Entities

1. **FinancialInstrument**: Tradable assets across markets
2. **MarketData**: Price and volume information
3. **TechnicalIndicator**: Calculated indicators based on market data

6.2.4 Portfolio Domain Entities

1. **Portfolio**: Collection of strategies with allocated capital
2. **PortfolioAllocation**: Distribution of capital across strategies
3. **PortfolioPerformance**: Performance metrics for portfolios

6.2.5 Analysis Domain Entities

1. **StrategyPerformance**: Performance metrics for strategies
2. **BacktestResult**: Results of historical strategy testing
3. **MarketCondition**: Classification of market environments

6.3 Define Relationship

Aca va la matriz pegala

6.4 Define Relationship Types

Here are the specific relationship types with cardinality constraints:

6.4.1 One-to-One (1:1) Relationships

[label=0.]**User - UserPreference**

1. • Each user has exactly one set of preferences.
   • Implementation: Foreign key from `UserPreference` to `User` with unique constraint.

6.4.2 One-to-Many (1:N) Relationships

[label=0.]**User - Subscription**

1. • A user can have multiple subscriptions (historical record).
   • Implementation: Foreign key from `Subscription` to `User`.

2. **User - StrategyConfig**
   - A user can create multiple strategy configurations.
   - Implementation: Foreign key from `StrategyConfig` to `User`.

3. **User - Trade**
   - A user can execute multiple trades.
   - Implementation: Foreign key from `Trade` to `User`.

4. **User - Alert**
   - A user can receive multiple alerts.
   - Implementation: Foreign key from `Alert` to `User`.

5. **User - Portfolio**
   - A user can manage multiple portfolios.
   - Implementation: Foreign key from `Portfolio` to `User`.

6. **Strategy - StrategyConfig**
   - A strategy can have multiple configurations.
   - Implementation: Foreign key from `StrategyConfig` to `Strategy`.

7. **Strategy - Alert**
   - A strategy can generate multiple alerts.
   - Implementation: Foreign key from `Alert` to `Strategy`.

8. **Strategy - StrategyPerformance**
   - A strategy has multiple performance records.
   - Implementation: Foreign key from `StrategyPerformance` to `Strategy`.

9. **Strategy - BacktestResult**
   - A strategy can have multiple backtest results.
   - Implementation: Foreign key from `BacktestResult` to `Strategy`.

10. **FinancialInstrument - MarketData**
    - An instrument generates multiple market data points.
    - Implementation: Foreign key from `MarketData` to `FinancialInstrument`.

11. **FinancialInstrument - Trade**
    - An instrument can be involved in multiple trades.
    - Implementation: Foreign key from `Trade` to `FinancialInstrument`.

12. **MarketData - TechnicalIndicator**
    - A market data point can generate multiple technical indicators.
    - Implementation: Foreign key from `TechnicalIndicator` to `MarketData`.
13. **Trade - TradeExecution**
    - A trade can have multiple execution records.
    - Implementation: Foreign key from `TradeExecution` to `Trade`.
14. **Portfolio - PortfolioAllocation**
    - A portfolio contains multiple strategy allocations.
    - Implementation: Foreign key from `PortfolioAllocation` to `Portfolio`.
15. **Portfolio - PortfolioPerformance**
    - A portfolio has multiple performance records.
    - Implementation: Foreign key from `PortfolioPerformance` to `Portfolio`.
16. **StrategyConfig - BacktestResult**
    - A configuration can have multiple backtest results.
    - Implementation: Foreign key from `BacktestResult` to `StrategyConfig`.

6.4.3 Many-to-Many (M:N) Relationships

[label=0.]**Strategy - Portfolio**

1. • A strategy can be used in multiple portfolios.
   • A portfolio can use multiple strategies.
   • Implementation: Resolved through `PortfolioAllocation` junction table.
2. **Strategy - MarketCondition**
   - A strategy can perform differently under multiple market conditions.
   - A market condition affects multiple strategies.
   - Implementation: Relationship is captured through `StrategyPerformance` table.
3. **FinancialInstrument - Strategy**
   - A strategy can trade multiple instruments.
   - An instrument can be traded by multiple strategies.
   - Implementation: Implicit through `Trade` records.

6.4.4 Reflexive Relationships

[label=0.]**Strategy - Strategy (Base/Derived)**

1. • A strategy can be derived from another strategy.
   • Implementation: Add `parent_strategy_id` (nullable FK) to `Strategy` table.
2. **Portfolio - Portfolio (Master/Sub)**
   - A portfolio can contain sub-portfolios.
   - Implementation: Add `parent_portfolio_id` (nullable FK) to `Portfolio` table.

7 Entity-Relationship Diagram

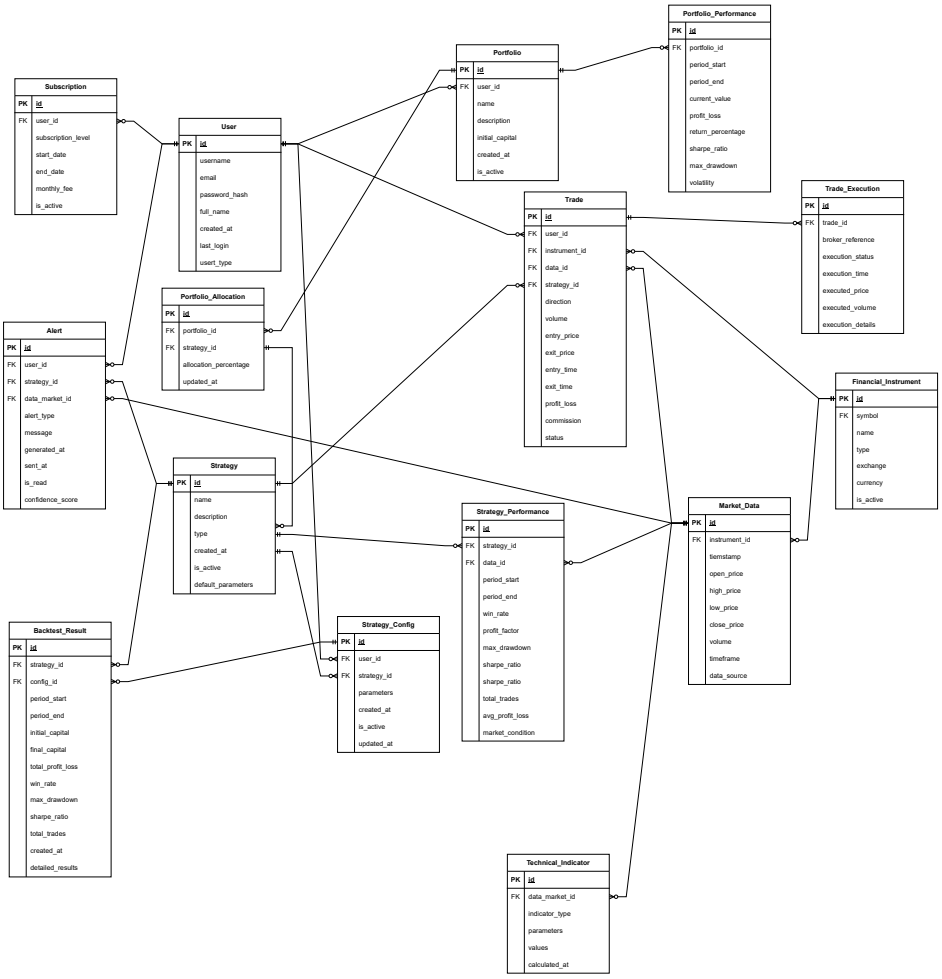

Figure 2: Model ER

Link al repositorio: `https://github.com/felimarod/project-databases-ii`

8 References