Ezorro Technical Document

Baha Abunojaim
Nov 2024

eZorro Technical Overview

Introduction

eZorro is an AI-driven platform that enables users to design, validate, and automate trading strategies. By transforming plain language inputs into executable trading algorithms, eZorro provides a seamless pipeline from ideation to deployment. This document outlines the platform's key features and technical workflow, ensuring clarity for developers and advanced users.

Key Features

1. **Natural Language Input**
    - Users describe their trading ideas in plain English.
    - eZorro's AI parses the input, identifies key components (e.g., asset types, order types, conditions), and engages in iterative dialogue to refine the strategy.
2. **Collaborative Refinement**
    - The AI assistant ensures all parameters are accurately defined.
    - Supports complex conditions, multi-asset scenarios, and nested logic.
    - Real-time validation of syntax and structure ensures compatibility with backend systems.
3. **Algorithm Generation**
    - Finalized strategies are converted into robust trading algorithms.
    - The system generates a detailed explanation of the algorithm for user review, ensuring transparency.
4. **Backtesting**
    - Strategies are backtested against historical data to assess performance.
    - Outputs include:
        - Profitability metrics (e.g., ROI, Sharpe ratio)
        - Risk assessments (e.g., drawdown, volatility)
        - Performance under varied market conditions
    - Users can iteratively optimize strategies based on results.
5. **Automation and Execution**
    - Validated algorithms are deployed to live markets.
    - Automated execution of trades and alerts in response to real-time data.
    - Integration with brokers (e.g., Alpaca, Interactive Brokers) and market data providers (e.g., Polygon, QuiverQuant).
6. **Dashboards**
    - Provides comprehensive dashboards and charts for both stocks and crypto.
    - Interactive heatmap
    - Screening, news and events

    - Portfolio tailored news
    - Portfolio snapshot across all integrated platforms

All Features

AI-Assisted Strategy Building

eZorro's AI streamlines the strategy creation process by guiding users through building and refining their trading strategies. The AI interprets natural language inputs, identifies missing parameters, and ensures logical consistency. This ensures even users with minimal technical expertise can create sophisticated trading strategies.

AI Assistant for Market Insights

The built-in AI assistant provides real-time answers to market-related questions, leveraging:

- **Fundamental Data**: Earnings reports, financial ratios, and company insights.
- **Market Data**: Current prices, historical trends, and volume analysis.
- **Technical Data**: Indicators like RSI, moving averages, and MACD.
- **Alternative Data**: Social media trends, sentiment analysis, and unconventional data points.

Additionally, the assistant offers:

- **Tips**: Suggestions on trading practices and market opportunities.
- **News and Updates**: Tailored information on market events impacting user portfolios.
- **Recommendations**: Data-driven insights for potential trades or adjustments.

Chat via eZorro or WhatsApp

Users can interact with eZorro's AI via:

- **eZorro Platform**: Directly through the application interface.
- **WhatsApp Integration**: Access the same powerful AI tools through a WhatsApp conversation, enabling convenient, on-the-go interactions.

Link Existing Broker Accounts Through SnapTrade

By integrating with SnapTrade, users can:

- Connect multiple broker accounts in one place.
- Execute trades directly through eZorro.
- Synchronize portfolio data across platforms for unified management.

Portfolio-Tailored News Dashboard

eZorro curates a personalized news feed based on the user's portfolio holdings. This dashboard includes:

- Breaking news and updates relevant to the user's investments.
- Insights on events likely to impact specific sectors or assets.
- Alternative data-driven analyses for deeper market understanding.

Portfolio Snapshot Across Linked Accounts

Provides a consolidated view of the user's holdings across all linked broker accounts, showing:

- Asset allocation.
- Current value and unrealized P/L (profit and loss).
- Key metrics, such as sectoral exposure and diversification levels.

Market Dashboards for Stocks and Crypto

Comprehensive dashboards offering real-time market insights:

- **Stock Dashboard**: Gainers/Losers, sector movements, and trending tickers.
- **Crypto Dashboard**: Prices, market caps, volume data, and sentiment analysis for cryptocurrencies.

Refine Strategies Using AI Feedback

eZorro's AI offers iterative feedback on trading strategies, suggesting improvements based on:

- Historical data patterns.
- Risk-reward analysis.
- Alternative conditions or triggers to enhance strategy effectiveness.

Suggest Strategies

AI-generated strategy ideas based on user preferences, market conditions, and portfolio goals. Users can explore new opportunities with pre-designed strategies tailored to their trading style.

Portfolio Building and Rebalancing

- **Portfolio Building**: Use AI recommendations to construct diversified portfolios aligned with user goals (e.g., growth, income, or risk aversion).
- **Rebalancing**: Maintain optimal allocation by adjusting weights periodically based on market movements and user-defined criteria.

Assign Schedule to Strategy

Users can set evaluation intervals for strategies, choosing from:

- 1 min, 5 mins, 10 mins, 15 mins, 30 mins, 60 mins.
- Daily (specific time).
  This feature provides flexibility to align strategy execution with market volatility and user availability.

Start/Stop Strategy Evaluations

Control strategy activation with a single click:

- **Start**: Begin real-time evaluation and execution based on live market data.
- **Stop**: Halt evaluations to pause trading or refine strategies without impacting live portfolios.

Backtesting Strategies

eZorro allows users to simulate strategy performance using historical market data. Outputs include:

- Key metrics (e.g., ROI, Sharpe ratio, drawdowns).
- Scenario analysis for varying market conditions.
- A detailed breakdown of trades executed during backtesting.

Alerts (WhatsApp or eZorro Notifications)

Stay informed with real-time alerts delivered via:

- **WhatsApp Messages**: Instant updates directly to the user's device.
- **eZorro Notifications**: Notifications within the platform for trades, triggers, and critical updates.

These alerts ensure users never miss significant events or actions required for their strategies.

eZorro Specific Data Models

Aside from the data models associated with 3rd party services and data sources such as Polygon.io, QuiverQuant and SnapTrade. eZorro holds specific data models. The following are suggested data models:

User Model

```
{
  "id": "UUID",
  "name": "String",
  "email": "String",
  "phone_number": "String",
  "whatsapp_number": "String (optional)",
  “snapTrade”:”JSON”,
  “alpaca”:”JSON”,
  "created_at": "Timestamp",
  "updated_at": "Timestamp"
```

Thread Model

```
{
  "id": "UUID",
  "title": "String",
  "user_id": "UUID (Reference to User)",
  "created_at": "Timestamp",
  "messages": [
    {
      "id": "UUID",
      "role": "String (user, system, assistant)",
      "content": {},
      "visible": "Boolean",
      "created_at": "Timestamp"
    }
  ]
}
```

Strategy Model

```
{
  "id": "UUID",
  "user_id": "UUID (Reference to User)",
  "title": "String",
  "strategy_text": "String",
  "summary": "String",
  "explanation": "String (optional)",
  "examples": "String (optional)",
  "strategy_json": "JSON",
  "information_json": "JSON",
  "backtest_result_id": "UUID (Reference to BacktestResult) (optional)",
  "report_id": "UUID (Reference to Report) (optional)",
  "created_at": "Timestamp",
  "updated_at": "Timestamp"
}
```


Schedule Model


```
{
  "id": "UUID",
  "strategy_id": "UUID (Reference to Strategy)",
  "interval": "Enum (1m, 5m, 10m, 15m, 30m, 60m, Daily)",
  "start_time": "Timestamp",
  "end_time": "Timestamp (optional)",
  "start_date": "Timestamp (optional)",
  “end_date": "Timestamp (optional)",
  "created_at": "Timestamp"
}
```


Run Model


```
{
  "id": "UUID",
  "strategy_id": "UUID (Reference to Strategy)",
  "executed_at": "Timestamp",
  "result": "JSON (Run results data)",
  "created_at": "Timestamp"
}
```


Report Model


```
{
  "id": "UUID",
  "strategy_id": "UUID (Reference to Strategy)",
  "backtest_id": "UUID (Reference to BacktestResult)",
  "generated_at": "Timestamp",
  "report_url": "String (Cloud Storage URL)",
```

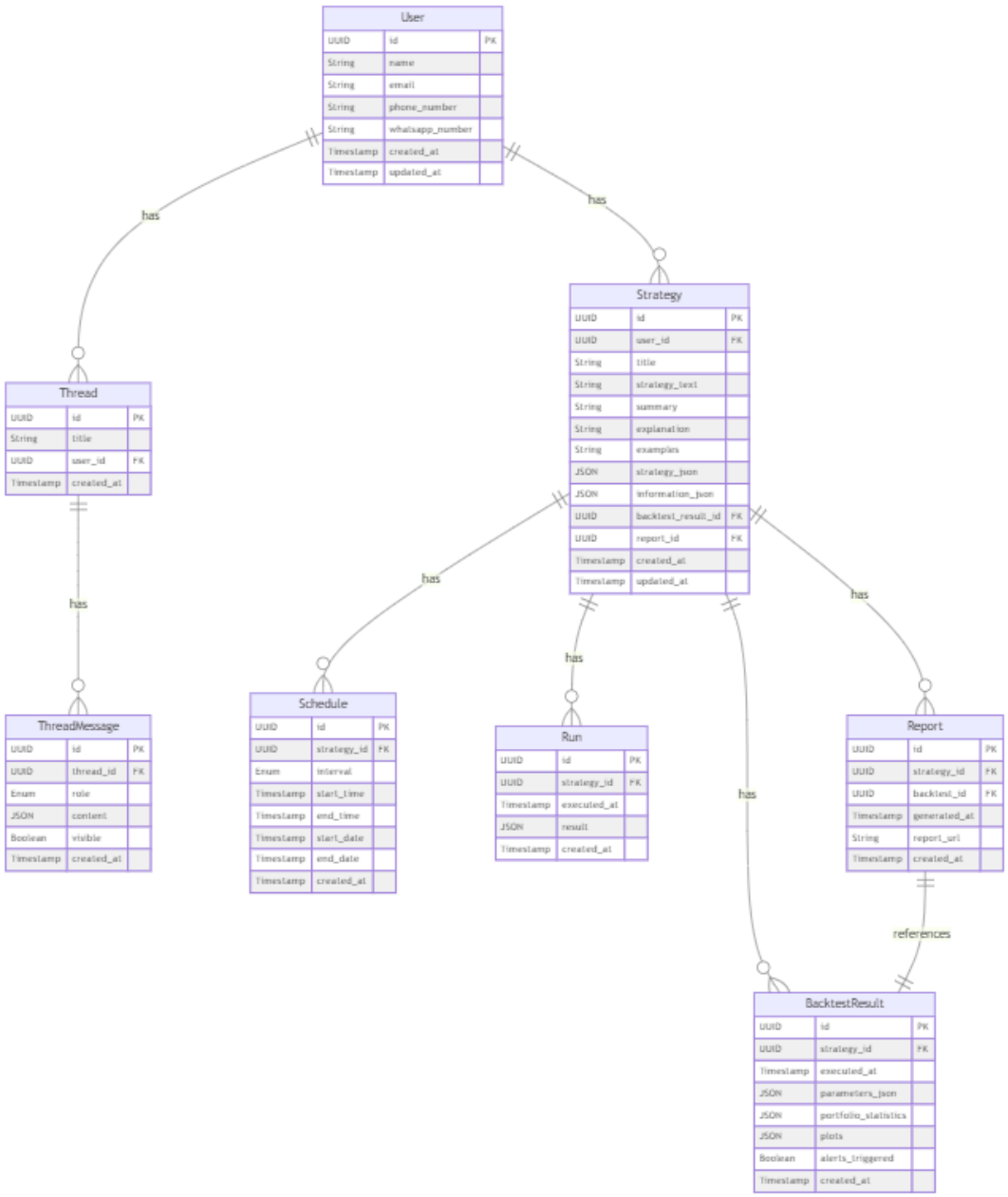
```
  "created_at": "Timestamp"
}
```

BacktestResult Model

```
{
  "id": "UUID",
  "strategy_id": "UUID (Reference to Strategy)",
  "executed_at": "Timestamp",
  "parameters_json": "JSON (Backtest parameters)",
  "portfolio_statistics": {
    "start_date": "Datetime",
    "end_date": "Datetime",
    "start_value": "Decimal",
    "end_value": "Decimal",
    "total_return": "Decimal",
    "benchmark_return": "Decimal",
    "max_gross_exposure": "Decimal",
    "total_fees_paid": "Decimal",
    "max_drawdown": "Decimal",
    "max_drawdown_duration": "Duration",
    "total_trades": "Integer",
    "total_closed_trades": "Integer",
    "total_open_trades": "Integer",
    "open_trade_pnl": "Decimal",
    "win_rate": "Decimal",
    "profit_factor": "Decimal",
    "sharpe_ratio": "Decimal",
    "calmar_ratio": "Decimal",
    "sortino_ratio": "Decimal",
    "omega_ratio": "Decimal"
  },
  "plots": {
    "portfolio_value_url": "String (Cloud Storage URL)",
    "normalized_prices_url": "String (Cloud Storage URL)"
  },
  "alerts_triggered": "Boolean",
  "created_at": "Timestamp"
}
```

Database Diagram

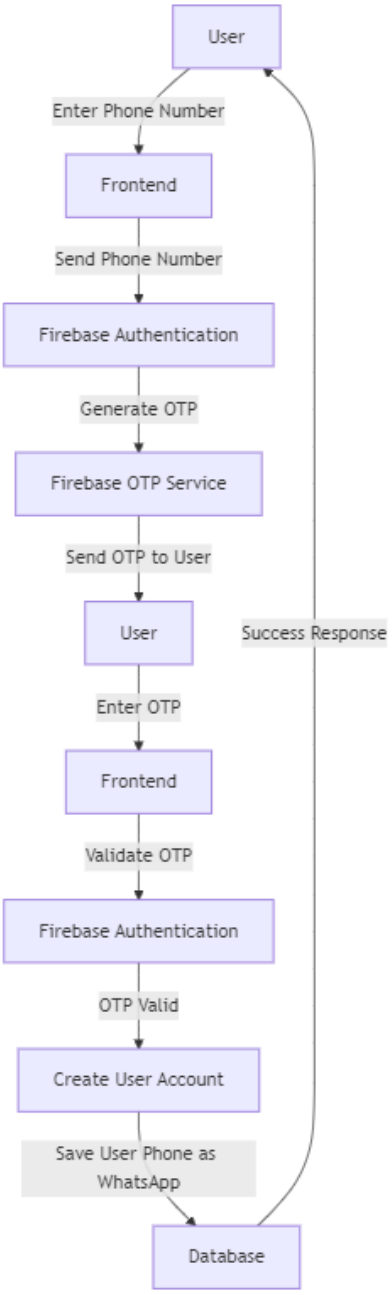
The following is a database diagram showing the above data models and their relationships:



User Actions flowcharts

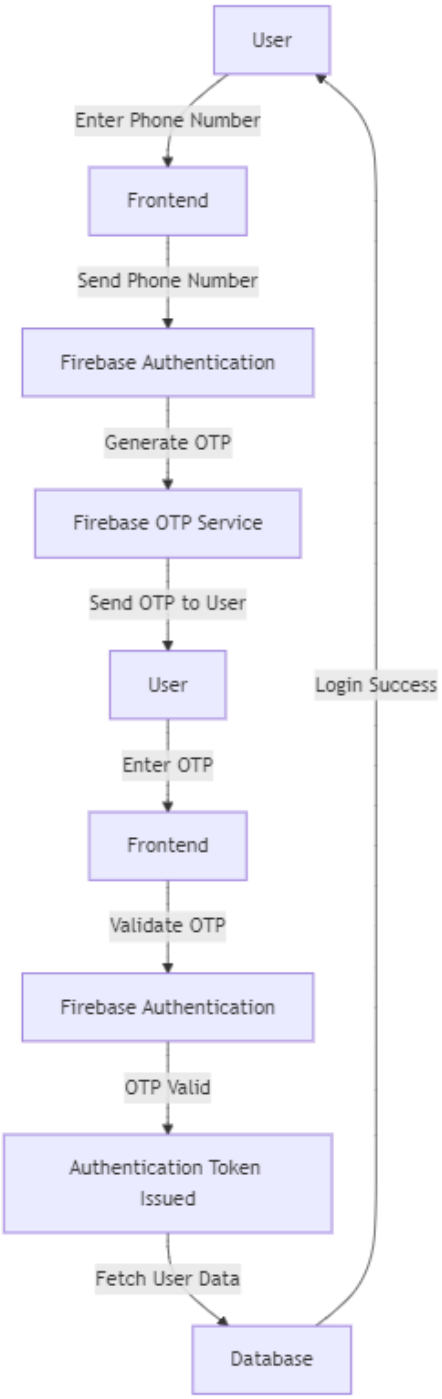
Signup Flow

Description:
New users sign up by providing their phone number and verifying it with an OTP. The phone number is saved as both the primary contact and the linked WhatsApp number.



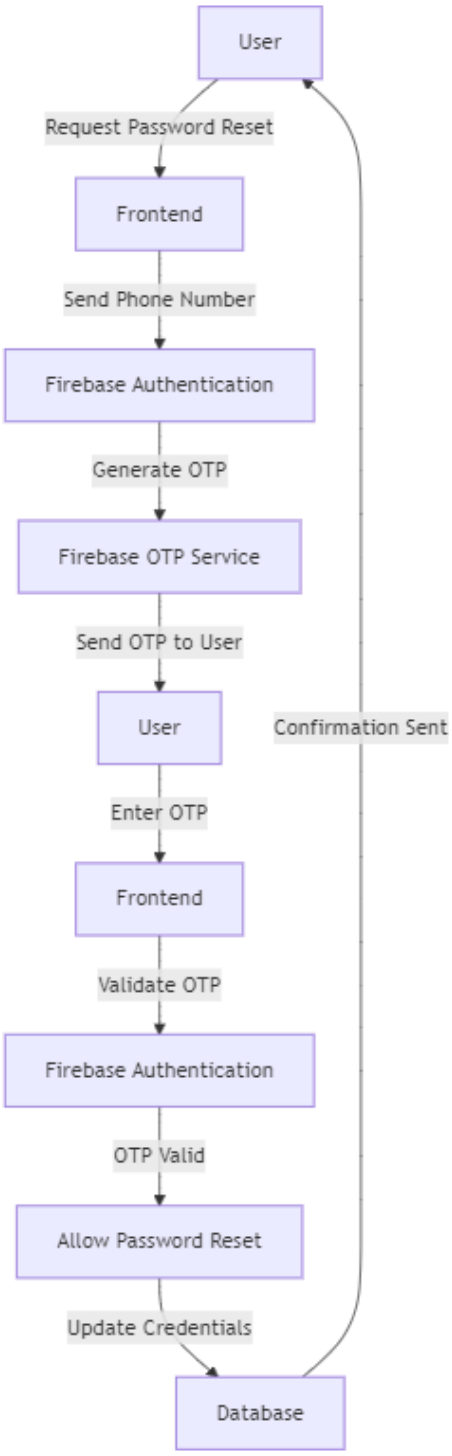
Login Flow

Description:
Existing users log in by providing their phone number, verifying it with an OTP, and receiving an authentication token.



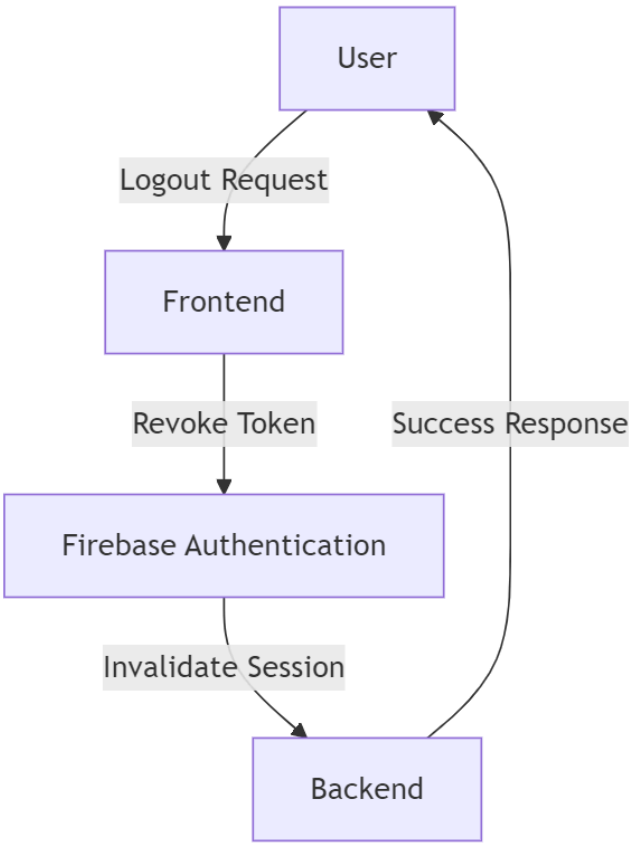
Reset Password Flow

Description:
Users reset their password by verifying their phone number and optionally updating other credentials if allowed.



Logout Flow

Description:
Users log out by revoking the authentication token.



Link/Unlink Account Using SnapTrade

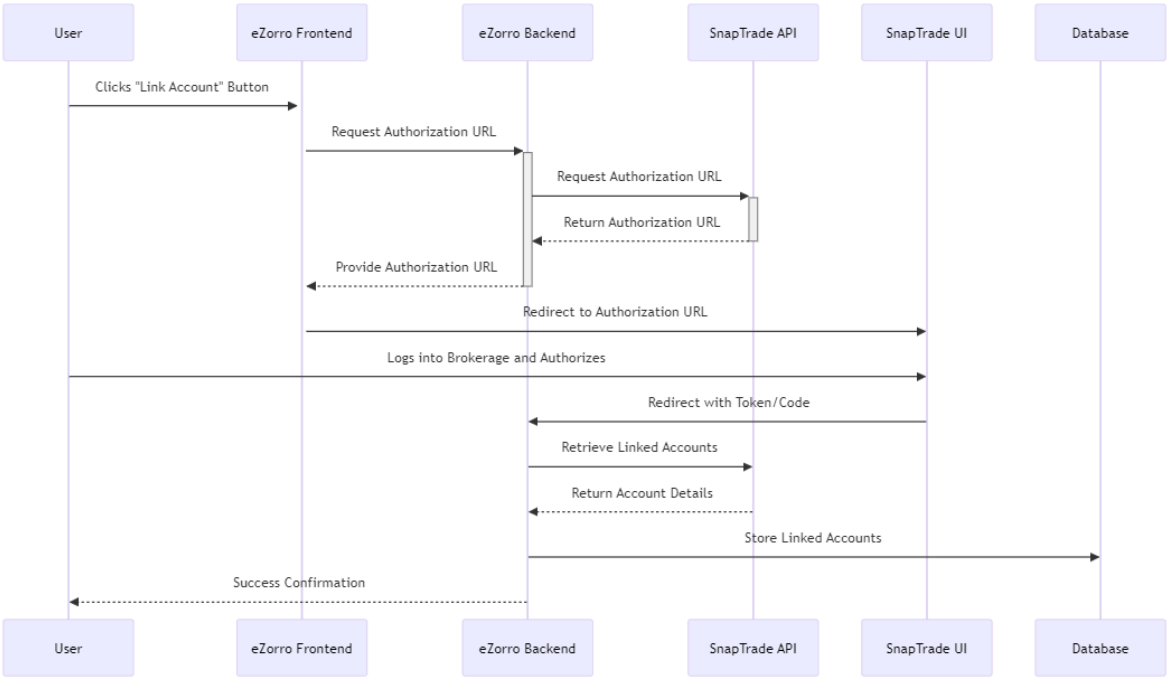
This process integrates SnapTrade's API into eZorro, allowing users to securely link or unlink their brokerage accounts. Here's the step-by-step breakdown:

1. Linking an Account

Steps:

1. **User Action**:
    - The user clicks the **"Link Account"** button in the eZorro interface.
2. **Redirect to SnapTrade**:
    - eZorro generates a request to SnapTrade's **Authorization URL** with the user's details.
    - The user is redirected to SnapTrade's interface to select and authorize their brokerage accounts.
3. **Brokerage Authorization**:
    - The user logs into their brokerage account(s) via SnapTrade and grants permission to link their account.

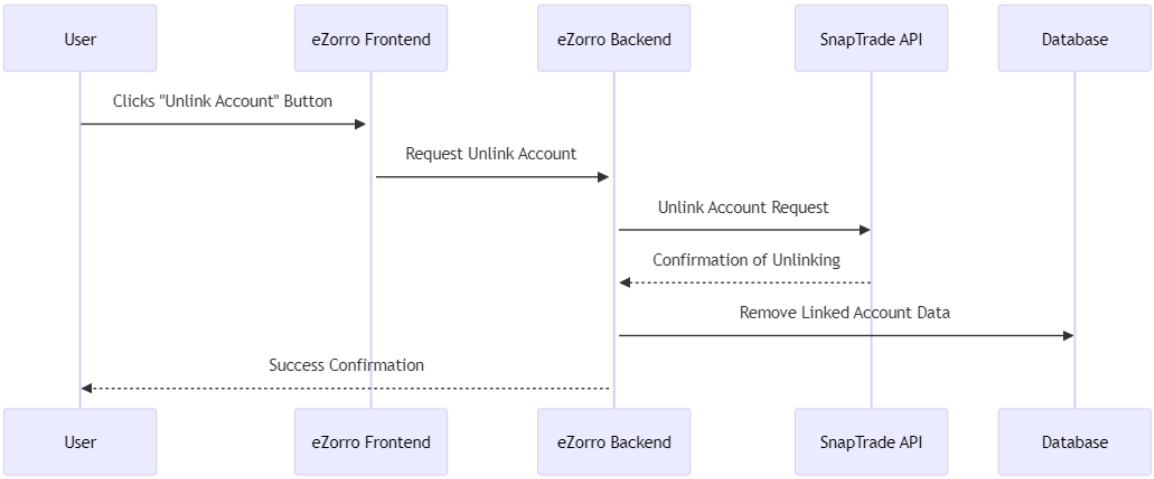
4. **Callback to eZorro**:
    - SnapTrade redirects the user back to eZorro with a **token** or **authorization code**.
5. **API Call to Retrieve Accounts**:
    - eZorro uses the received token to call SnapTrade's **Account Linking API** and retrieves linked account details.
6. **Store Linked Accounts**:
    - The linked accounts are securely stored in the eZorro database, associating them with the user's profile.



2. Unlinking an Account

Steps:

1. **User Action**:
    - The user clicks the **"Unlink Account"** button in the eZorro interface for a specific linked account.
2. **Send Unlink Request to SnapTrade**:
    - eZorro sends an unlink request to SnapTrade's **Account Management API**, specifying the account to be unlinked.
3. **Confirmation from SnapTrade**:
    - SnapTrade confirms the account is unlinked and stops accessing data from the brokerage.
4. **Update Database**:
    - eZorro updates its database to reflect the account's unlinked status and removes sensitive data associated with the account.
5. **Notify User**:
    - The user receives confirmation of the successful unlinking.



Chat with AI

This flow covers how the AI in eZorro answers user questions, generates and refines strategies, and proactively provides news, tips, and recommendations. The AI leverages real-time data from integrated APIs and intelligently interacts with users to enhance their trading experience.

1. User Interaction with AI

Steps:

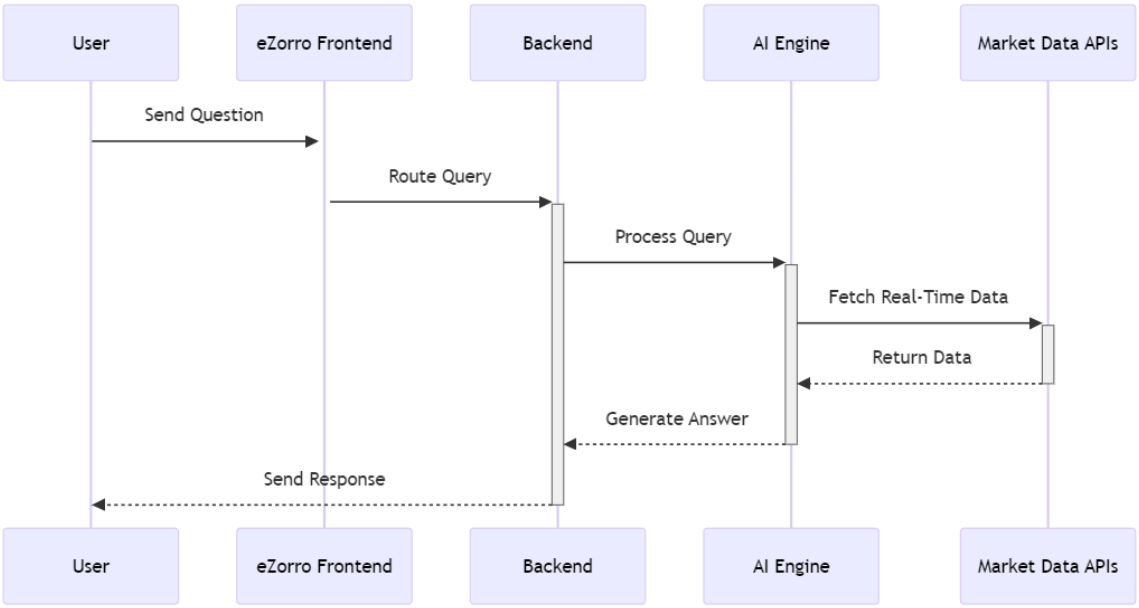
1. **User Initiates Chat**:
    - The user sends a message or query via the eZorro app or WhatsApp.
2. **AI Processes Query**:
    - The backend routes the query to the AI engine.
    - The AI determines the query's intent (e.g., question, strategy creation, strategy refinement).
3. **Retrieve Data (if required)**:
    - The AI fetches necessary data from relevant sources (e.g., market data APIs, strategy database).
4. **Generate Response**:
    - The AI compiles a response using real-time data, pre-defined logic, or user-specific context.
5. **Send Response to User**:
    - The user receives the AI-generated response in the chat interface.

2. Use Cases for Chat with AI

a. Answering User Questions

The AI can answer trading-related questions, such as:

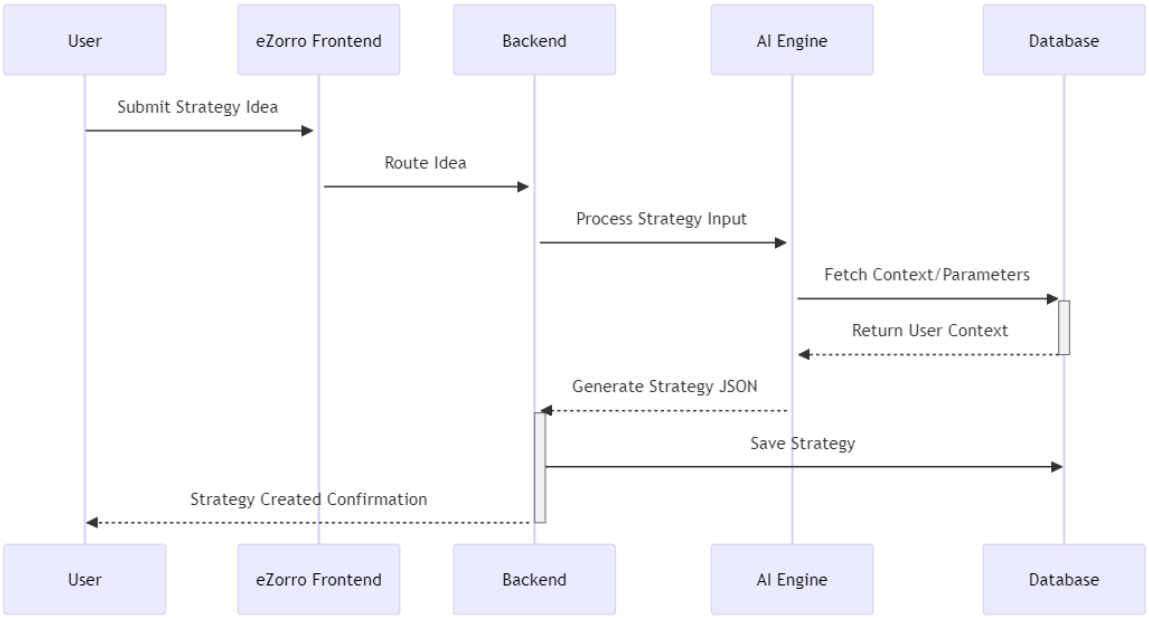
- Current stock/crypto prices.
- Technical indicator analysis (e.g., RSI, moving averages).
- Market sentiment analysis.
- Risk/reward calculations for specific trades.



b. Generating Strategies

The AI assists users in creating trading strategies based on natural language input:

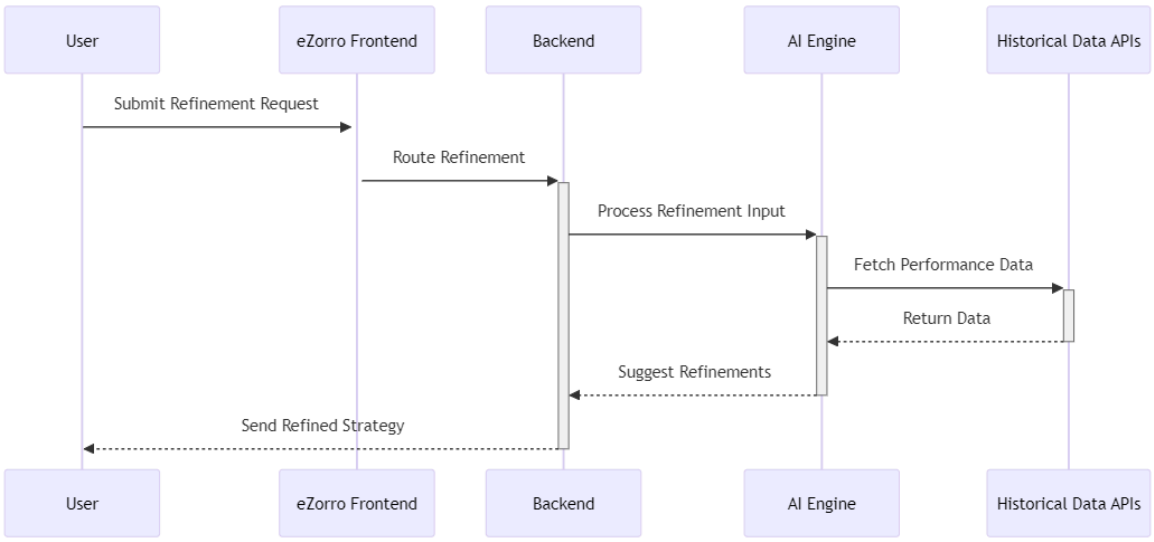
- Users describe their trading idea in plain English.
- The AI interprets the input and generates a structured strategy.
- The strategy is saved for further use.



c. Refining Strategies

The AI enables users to iteratively refine their strategies:

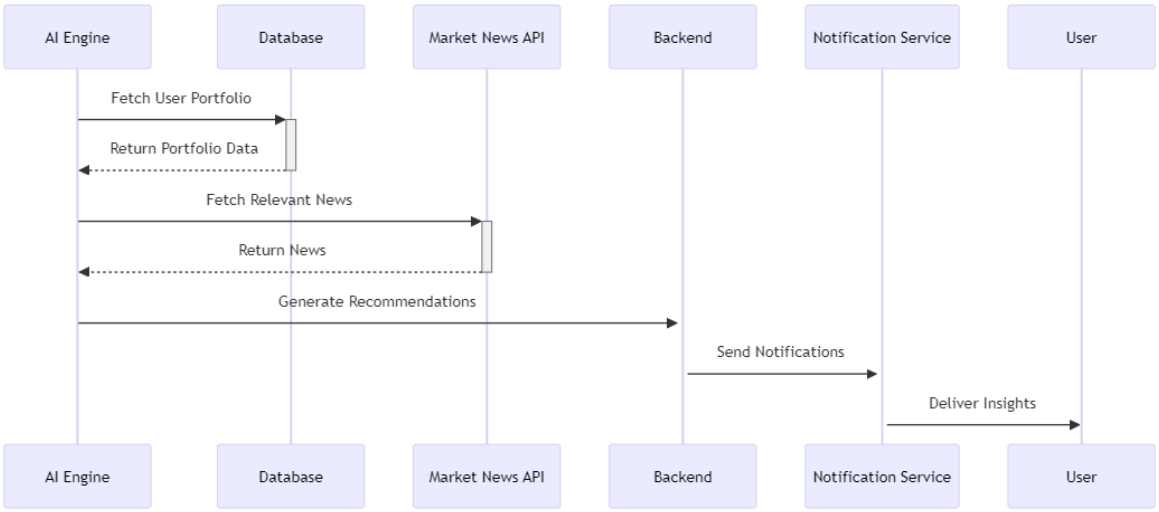
- Users provide feedback or request specific changes.
- The AI suggests modifications based on historical performance, risk factors, or user preferences.



d. Proactive Recommendations

The AI proactively sends personalized insights through notifications or WhatsApp, such as:

- News articles related to the user's portfolio.
- Trading tips based on market trends.
- Recommendations for portfolio adjustments or new strategies.



3. AI Functionality Details

a. Real-Time Data Retrieval

- **Sources**: APIs like Polygon, Alpaca, QuiverQuant, and SnapTrade.
- **Use Cases**:
    - Current stock/crypto prices and trends.
    - Sentiment analysis from alternative data (e.g., social media).
    - News articles and key events impacting specific sectors or assets.

b. Generating Strategies

- The AI uses NLP to parse user input and convert it into structured JSON defining the trading strategy.
- It ensures all parameters (e.g., assets, triggers, conditions) are logically complete.

c. Refining Strategies

- Historical performance is analyzed to identify weak points in strategies.
- AI suggests changes, such as:

    - Adjusting stop-loss limits.
    - Changing technical indicators.
    - Modifying timeframes for conditions.

d. Proactive Insights

- The AI monitors:
    - Portfolio performance.
    - Market trends.
    - Breaking news.
- Proactively sends alerts to help users make timely decisions.

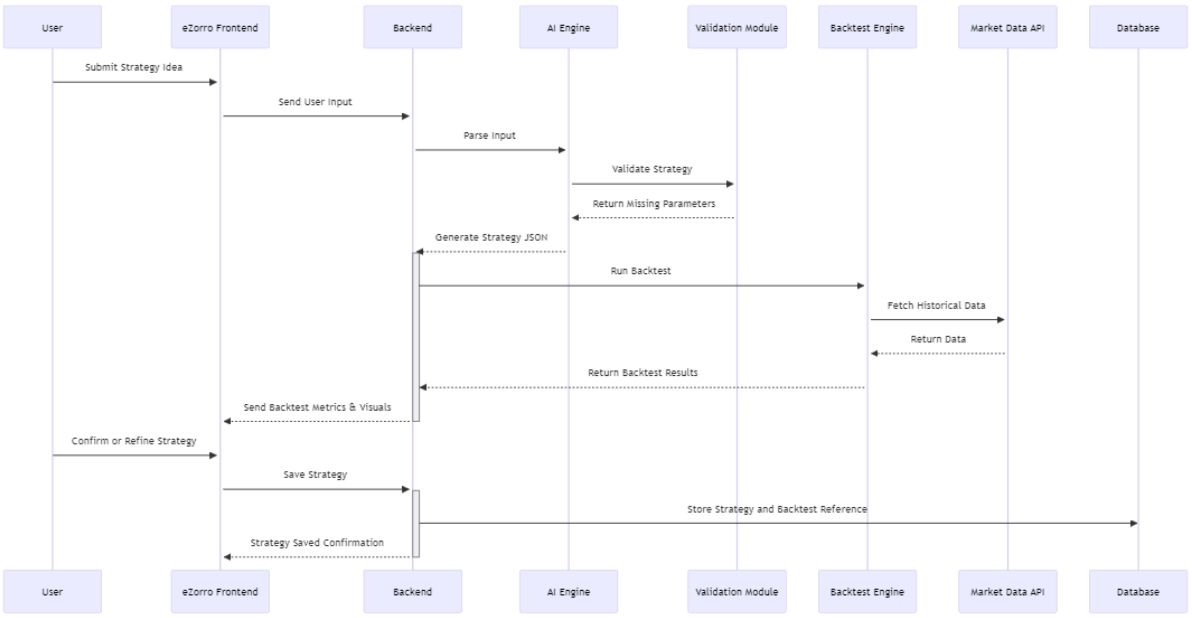
Creating a Strategy

The **Create Strategy** process in eZorro enables users to translate their trading ideas into actionable strategies through natural language input. This feature uses AI to understand user intent, validate parameters, and generate structured and backtested trading strategies that can be saved, tested, and executed.

Steps in the Flow

1. **User Input**:
    - The user submits a trading idea in natural language via the eZorro interface (eZorro chat or WhatsApp).
2. **Intent Extraction**:
    - The AI parses the input to identify key components like the asset, triggers, conditions, and actions.
3. **Validation**:
    - The AI checks for missing parameters and engages the user for clarification if necessary.
4. **Generate Strategy JSON**:
    - The AI creates a structured JSON representation of the strategy.
5. **Run Backtesting**:
    - The backend runs a backtest using historical data to evaluate the strategy's performance.
    - Key performance metrics (e.g., ROI, Sharpe Ratio, Total Trades) and visualizations (e.g., portfolio value plot) are generated.
6. **Present Results to User**:
    - The user receives backtesting results, including metrics, charts, and recommendations.
7. **Save Strategy**:

    - If the user is satisfied, the strategy is saved in the database with a reference to the backtest results.
8. **Confirmation**:
    - The user receives a success message and can proceed to further refine, schedule, or activate the strategy.



Backtest Integration Details

Trigger for Backtesting

- Automatically initiated after the strategy JSON is generated.
- Users are informed that backtesting will run and that it might take a few moments.

Inputs to the Backtest Engine

- **Strategy JSON**: Contains all triggers and actions for evaluation.
- **Historical Data**: Pulled from integrated APIs like Polygon or Alpaca.

Backtest Outputs

- Key Performance Metrics:
    - ROI (Return on Investment)

    - Sharpe Ratio
    - Max Drawdown
    - Total Trades
    - Win Rate
- Visualizations:
    - Portfolio value over time.
    - Trade signals plotted on historical price data.

User Interface Updates

1. **Backtesting Progress Indicator**:
    - Users see a loading animation or progress bar while the backtest is running.
2. **Backtesting Results Screen**:
    - Displays key metrics and plots in a user-friendly format.
    - Includes recommendations (e.g., "Consider adding a stop-loss to reduce drawdown").
3. **Action Buttons**:
    - **Save Strategy**: Saves the strategy and links it to the backtesting results.
    - **Refine Strategy**: Allows the user to modify triggers or actions and rerun the backtest.

Error Handling

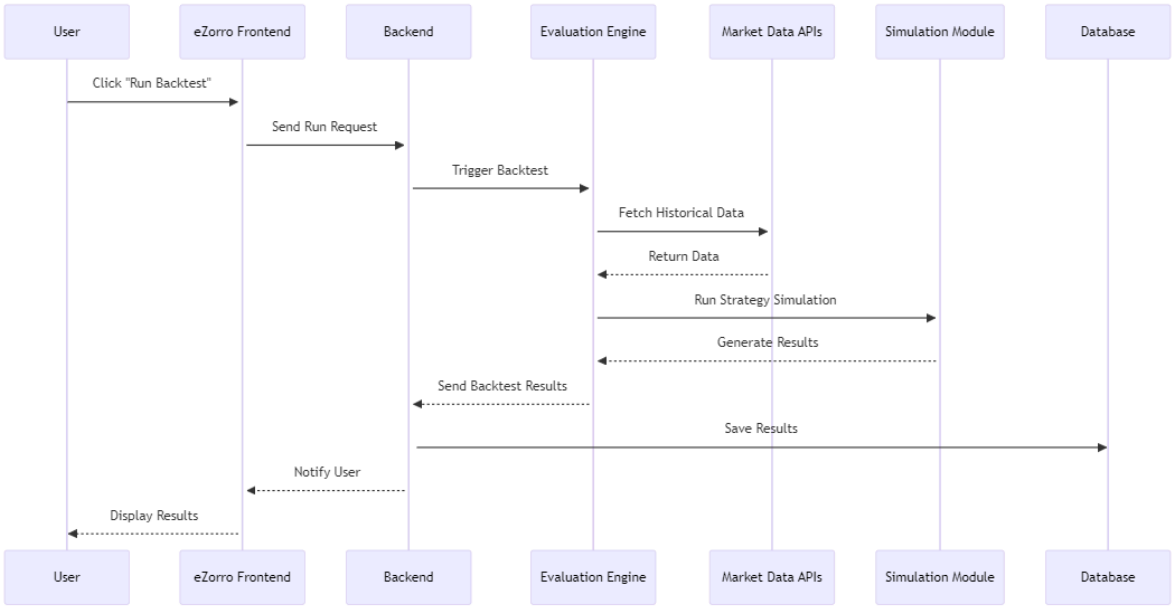
1. **Backtesting Errors**:
    - If the backtest fails (e.g., due to data unavailability), the system informs the user with a retry option.
2. **Incomplete Strategies**:
    - Users cannot run backtests on incomplete strategies.
    - The AI prompts for missing components.

Running a Backtest

The **Run Backtest** feature evaluates a user's strategy using historical market data. It calculates performance metrics, generates visualizations, and provides actionable insights to help users validate and refine their strategies before deploying them in live markets.

Steps in the Backtest Flow

1. **User Action**:
    - The user selects a strategy and clicks **"Run Backtest"** in the eZorro interface.
2. **Trigger Backtest**:
    - The frontend sends a request to the backend to initiate the backtest.
    - The backend identifies the strategy to be evaluated and sends a trigger request to the evaluation engine.
3. **Evaluation Engine Fetches Data**:
    - The evaluation engine retrieves the following directly from integrated APIs:
        - Historical market data (e.g., prices, volume).
        - Technical indicators if required.
        - User-specific context or portfolio data if relevant.
4. **Run Backtest**:
    - The evaluation engine simulates the strategy using the fetched data.
    - Performance metrics and visualizations are generated during the simulation.
5. **Return Results**:
    - Once the backtest completes, the evaluation engine sends the results back to the backend, including:
        - Key performance metrics.
        - URLs for visual assets (charts, graphs).
        - Alerts or notable events triggered during the backtest.
6. **Save Results**:
    - The backend saves the backtest results in the database, linking them to the corresponding strategy.
7. **Notify User**:
    - The frontend informs the user that the backtest is complete and displays the results.



Roles in the Backtest Flow

Frontend:

- Provides the user interface for initiating the backtest.
- Sends the backtest request to the backend.
- Displays results to the user once the backtest is complete.

Backend:

- Validates the user's request.
- Sends a trigger request to the evaluation engine with the strategy details.
- Receives and stores backtest results.
- Notifies the user upon completion.

Evaluation Engine:

- Executes the simulation logic:
    - Fetches historical data and indicators from external APIs.
    - Evaluates the strategy based on triggers and actions.
    - Generates performance metrics and visual assets.

- Sends results back to the backend.

Evaluation Engine Responsibilities

1. **Data Fetching**:
    - The engine pulls historical data from market data providers (e.g., Polygon, Alpaca) using its own integrations.
    - Retrieves any auxiliary data required for technical analysis (e.g., RSI, moving averages).
2. **Simulation Logic**:
    - Executes the strategy by processing the triggers and actions defined in the strategy JSON.
    - Simulates trade executions and portfolio adjustments over the historical data period.
3. **Result Generation**:
    - Performance metrics:
        - ROI, Max Drawdown, Total Trades, Sharpe Ratio, etc.
    - Visualizations:
        - Portfolio value over time.
        - Normalized price data with trade signals.
    - Triggered alerts (e.g., stop-loss, take-profit).
    - Stores visual assets (charts, graphs) in cloud storage and returns URLs.

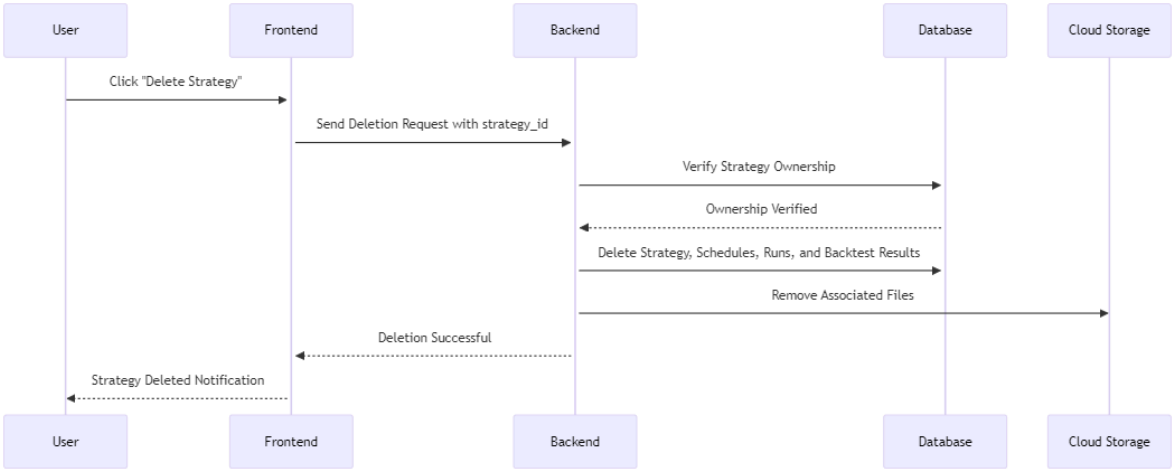
Delete Strategy

The **Delete Strategy** flow allows users to remove a strategy along with its associated schedules, backtest results, and runs. This ensures that the database remains clean and relevant to the user's needs.

Steps in the Flow

1. **User Action**:
    - The user selects a strategy and clicks the **"Delete Strategy"** button.
2. **Frontend Request**:
    - The frontend sends a deletion request to the backend with the `strategy_id`.
3. **Backend Processing**:
    - The backend verifies the user's authorization to delete the strategy.
    - Deletes the strategy and associated data:
        - **Schedules**: All schedules linked to the strategy.
        - **Runs**: Historical runs of the strategy.
        - **Backtest Results**: Any linked backtesting data.
    - Removes associated files from cloud storage (e.g., plots, reports).
4. **Confirmation**:
    - The backend sends a confirmation to the frontend.

- The frontend notifies the user that the strategy has been deleted.



Considerations

1. **Data Cascade**:
   - Deleting a strategy triggers a cascade to remove all related schedules, runs, and backtest results.
2. **Security**:
   - Ensure only the owner of the strategy can delete it.
3. **Error Handling**:
   - Strategy not found: Notify the user if the `strategy_id` is invalid.
   - Unauthorized action: Prevent deletion by unauthorized users.

Set Schedule

The **Set Schedule** flow enables users to assign periodic evaluations to their strategies. This ensures that strategies are executed automatically at defined intervals.

Schedules are managed by a **cron job** that evaluates which schedules are active and need to trigger strategy evaluations. This approach ensures periodic execution without overloading the system.

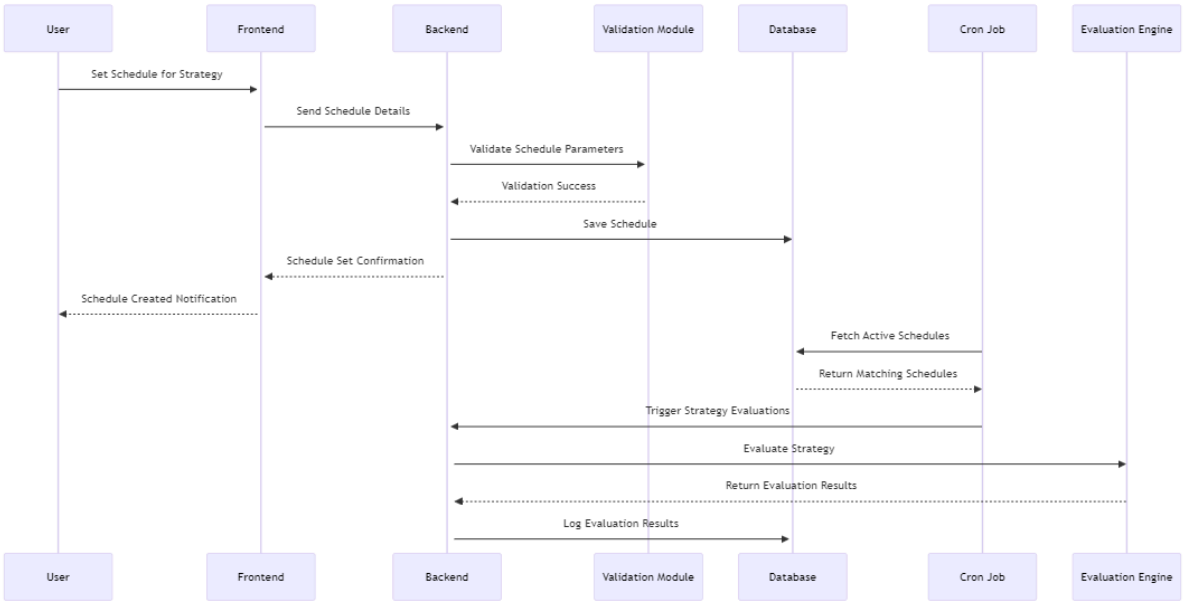
Steps in the Flow

1. User Sets a Schedule

1. **User Action**:
    - The user selects a strategy and clicks **"Set Schedule"**.
    - Inputs schedule parameters:
        - Frequency (e.g., 1m, 15m, daily).
        - Start and end times.
        - Optional date range.
2. **Frontend Request**:
    - Sends the `strategy_id` and schedule details to the backend.
3. **Backend Processing**:
    - Validates the parameters and saves the schedule to the database.
4. **Confirmation**:
    - The backend confirms the schedule creation, and the frontend notifies the user.

2. Cron Job Evaluation

1. **Cron Job Execution**:
    - Runs at fixed intervals (e.g., every minute or hour).
    - Checks the `schedules` table for active schedules that match the current cycle.
2. **Filter Active Schedules**:
    - Filters schedules based on:
        - **Interval**: Matches the current time against the schedule's interval.
        - **Date Range**: Ensures the schedule is within its valid date range.
        - **Time Range**: Confirms that the current time is within the defined start and end times.
3. **Trigger Strategy Evaluation**:
    - For each matching schedule:
        - Fetches the associated strategy.
        - Sends a trigger request to the evaluation engine.
4. **Log Execution**:
    - Logs the triggered evaluations for audit purposes.



Backend Responsibilities

Set Schedule

- Validates and saves schedule details to the database:
  - Ensures intervals are supported.
  - Checks for logical consistency (e.g., `end_time` > `start_time`).

Cron Job Execution

- Queries the `schedules` table to find matching schedules:

SQL-like query:
sql
Copy code

```
SELECT *
FROM schedules
WHERE interval = CURRENT_INTERVAL
  AND start_date <= CURRENT_DATE
  AND (end_date IS NULL OR end_date >= CURRENT_DATE)
```

```
  AND start_time <= CURRENT_TIME
  AND (end_time IS NULL OR end_time >= CURRENT_TIME);
```

    -
- For each match:
    - Fetches the associated strategy.
    - Sends a trigger to the evaluation engine.

Trigger Strategy Evaluation

Sends a request to the evaluation engine with the strategy details:
json
Copy code

```
{
  "strategy_id": "UUID",
  "strategy_json": {
    "triggers": [ ... ],
    "actions": [ ... ]
  }
}
```

Cron Job Flow

How the Cron Job Works

1. **Execution Frequency**:
    - Runs every minute or hour, depending on system configuration.
2. **Identify Active Schedules**:
    - Uses the database query to retrieve schedules that match the current time and conditions.
3. **Trigger Evaluation**:
    - For each active schedule, sends a request to evaluate the strategy.
4. **Error Handling**:
    - Logs failures if the evaluation engine is unreachable or returns errors.

Error Handling

1. **Invalid Schedule**:
    - If a user submits invalid parameters, the backend rejects the request with an appropriate error message.
2. **Cron Job Failures**:

    - If the cron job fails, logs are generated for debugging and the system retries in the next cycle.
3. **Evaluation Errors**:
    - If the evaluation engine fails to process a strategy, the backend logs the failure and retries for the next scheduled cycle.

Example Flow

User Input:

- Strategy: *"Buy TSLA when RSI < 30."*
- Schedule:
    - Interval: `15m`
    - Start Date: `2024-11-25`
    - End Date: `2024-12-01`
    - Time Range: `09:00 - 16:00`

Cron Job Execution:

1. **At 09:15**:
    - Matches the schedule.
    - Triggers evaluation for the strategy.
2. **Evaluation Engine**:
    - Fetches data and evaluates the strategy.
    - Logs results:
        - Alerts triggered: None.
        - Actions taken: Buy 100 TSLA.
3. **Result Logged**:
    - The backend saves the evaluation result in the database.

Advantages of Cron Job-Based Evaluation

1. **Efficiency**:
    - Evaluates only active schedules at the appropriate times.
2. **Scalability**:
    - Handles thousands of schedules with minimal resource consumption.
3. **Flexibility**:
    - Easily configurable intervals and triggers.

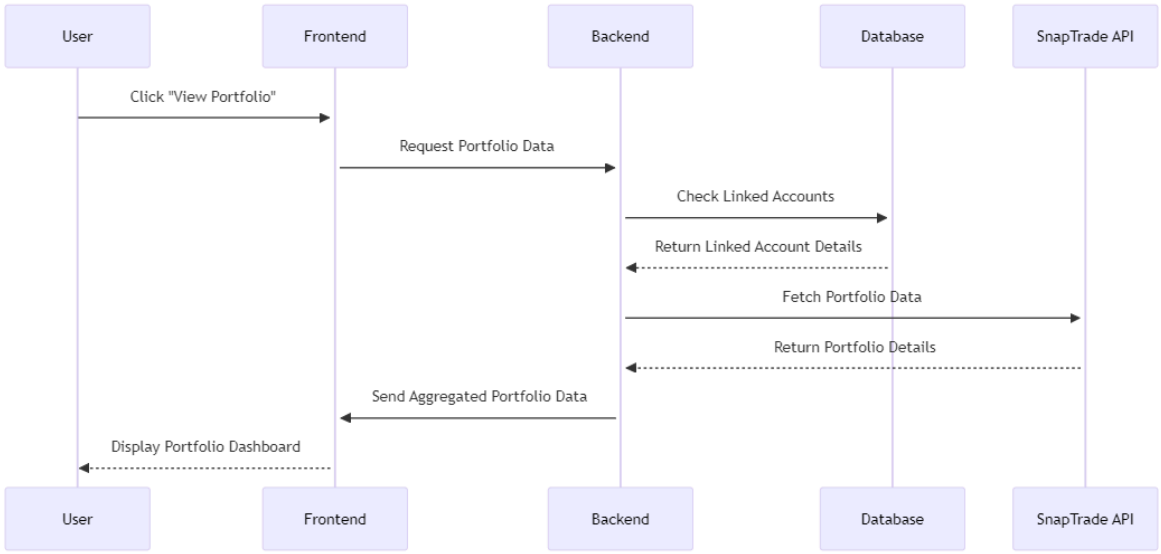
This approach ensures reliable and scalable strategy evaluations while keeping the system lightweight and maintainable.

View Portfolio

The **View Portfolio** feature provides users with a consolidated view of their investments across all linked brokerage accounts. Data is fetched from SnapTrade or similar APIs and displayed in an intuitive dashboard.

Steps in the Flow

1. **User Action**:
    - The user clicks **"View Portfolio"** in the eZorro interface.
2. **Frontend Request**:
    - Sends a request to the backend to fetch portfolio data.
3. **Backend Processing**:
    - Identifies linked brokerage accounts for the user.
    - Sends a request to SnapTrade or similar APIs to retrieve portfolio data.
4. **Fetch Portfolio Data**:
    - SnapTrade API fetches the portfolio details from linked brokerages and returns:
        - Holdings (assets, quantities, market values).
        - Account balances.
        - Performance metrics.
5. **Aggregate and Format Data**:
    - The backend aggregates data from multiple accounts into a unified format.
6. **Send Data to Frontend**:
    - The formatted data is sent to the frontend, which displays the portfolio to the user.



Error Handling

1. **No Linked Accounts**:
    - If no accounts are linked, notify the user: *"You have no linked brokerage accounts. Please link an account to view your portfolio."*
2. **API Errors**:
    - If SnapTrade fails, retry or notify the user: *"Unable to fetch portfolio data. Please try again later."*

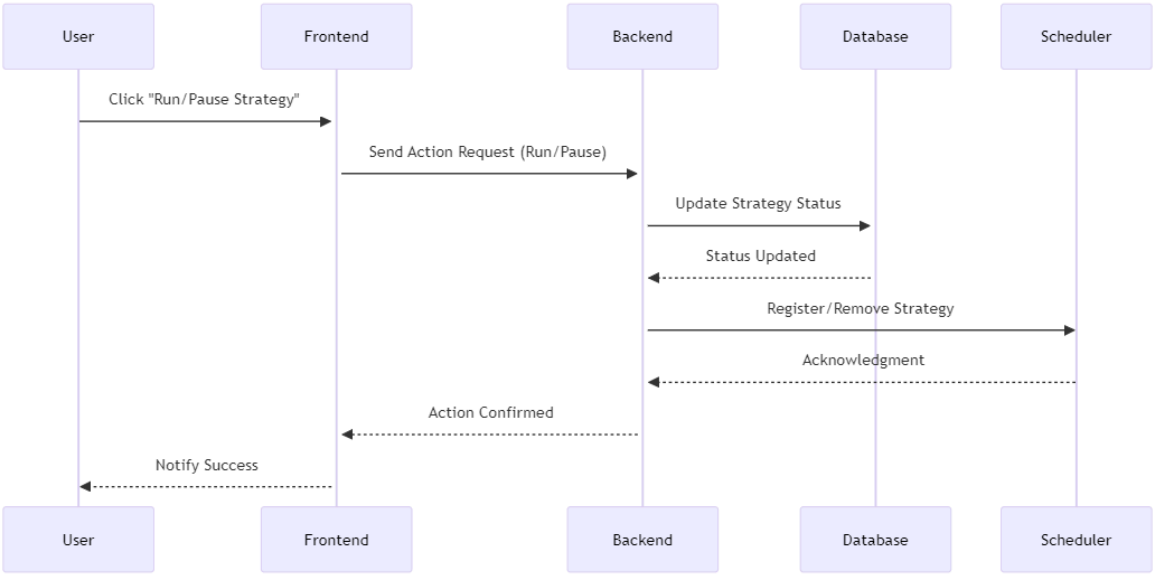
Run/Pause Strategy

The **Run/Pause Strategy** feature allows users to control the execution of their strategies. A running strategy continuously evaluates triggers, while a paused strategy stops evaluations temporarily.

Steps in the Flow

1. **User Action**:
    - The user selects a strategy and clicks **"Run"** or **"Pause"**.

2. **Frontend Request**:
    - Sends the request to the backend with the `strategy_id` and desired action (`run` or `pause`).
3. **Backend Processing**:
    - Validates the request and checks the current state of the strategy.
    - Updates the strategy's status in the database.
4. **Notify Scheduler**:
    - If the strategy is set to "Run," the backend registers the strategy with the task scheduler.
    - If set to "Pause," the backend removes the strategy from the scheduler.
5. **Confirmation**:
    - The backend sends a confirmation to the frontend, and the frontend notifies the user.



Scheduler Interaction

1. **Run Strategy**:
    - Registers the strategy with the task scheduler.
    - Ensures the scheduler evaluates the strategy based on its schedule.
2. **Pause Strategy**:

- Removes the strategy from the scheduler, halting further evaluations.

Error Handling

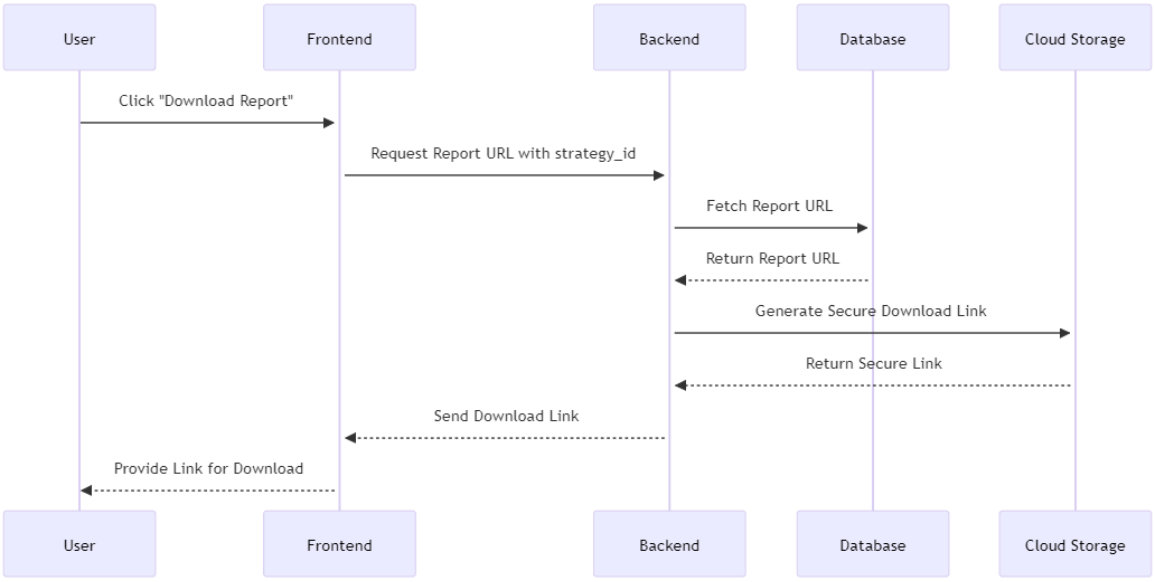
1. **Invalid State**:
   - If a paused strategy is already paused, notify the user: *"Strategy is already paused."*
2. **Scheduler Failure**:
   - If the scheduler fails to register/unregister the strategy, retry and log the error.
3. **Unauthorized Action**:
   - Ensure only the owner of the strategy can run/pause it.

Download Strategy Report

The **Download Strategy Report** feature allows users to retrieve a detailed PDF or document report summarizing the strategy, its parameters, performance, and backtest results. Reports are generated during backtesting or strategy evaluation and stored in cloud storage.

Steps in the Flow

1. **User Action**:
   - The user selects a strategy and clicks **"Download Report"**.
2. **Frontend Request**:
   - Sends a request to the backend with the `strategy_id` to fetch the report.
3. **Backend Processing**:
   - Validates the user's authorization to access the strategy.
   - Retrieves the `report_url` for the strategy from the database.
4. **Cloud Storage Interaction**:
   - The backend generates a secure, time-limited download link (if required) from the cloud storage system (e.g., AWS S3, Firebase).
5. **Send URL to Frontend**:
   - The backend sends the download link to the frontend.
6. **File Download**:
   - The user clicks the link, and the report is downloaded to their device.



View Runs and Download Run Report

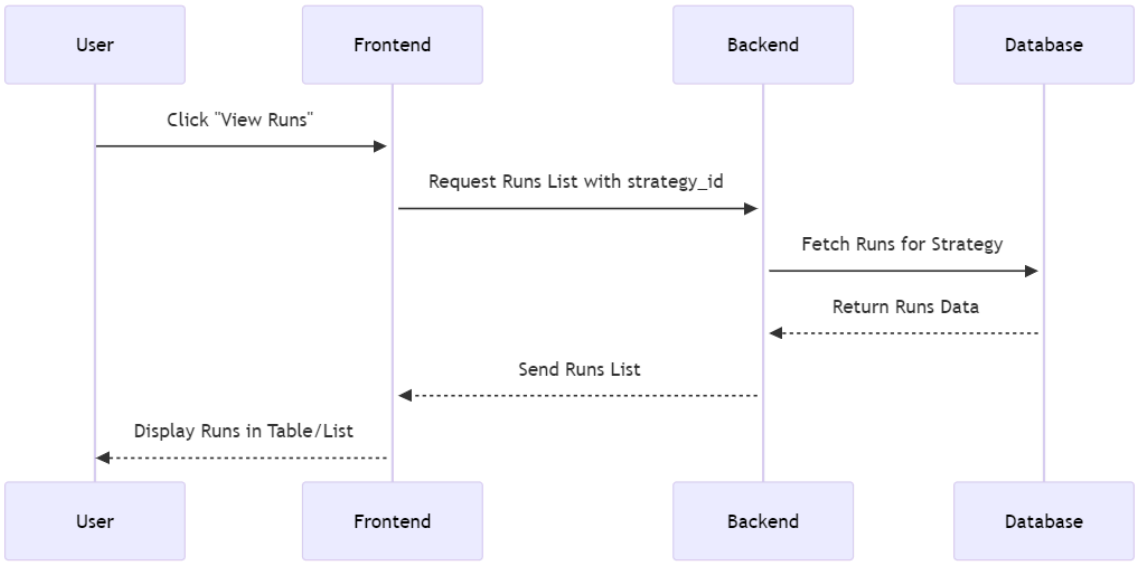
The **View Runs** feature provides users with a historical list of evaluations (runs) for a specific strategy, while the **Download Run Report** feature allows users to retrieve detailed reports for individual runs.

View Runs

Steps in the Flow

1. **User Action**:
   - The user selects a strategy and clicks **"View Runs"**.
2. **Frontend Request**:
   - Sends a request to the backend with the `strategy_id`.
3. **Backend Processing**:
   - Verifies the user's authorization to access the strategy.
   - Fetches the list of runs associated with the `strategy_id` from the database.
4. **Send Data to Frontend**:
   - The backend returns the run data to the frontend.

- The frontend displays a paginated list of runs, showing key details such as:
  - Run timestamp.
  - Result summary (e.g., ROI, total trades).
  - Any triggered alerts.



Frontend Display

The runs list includes:

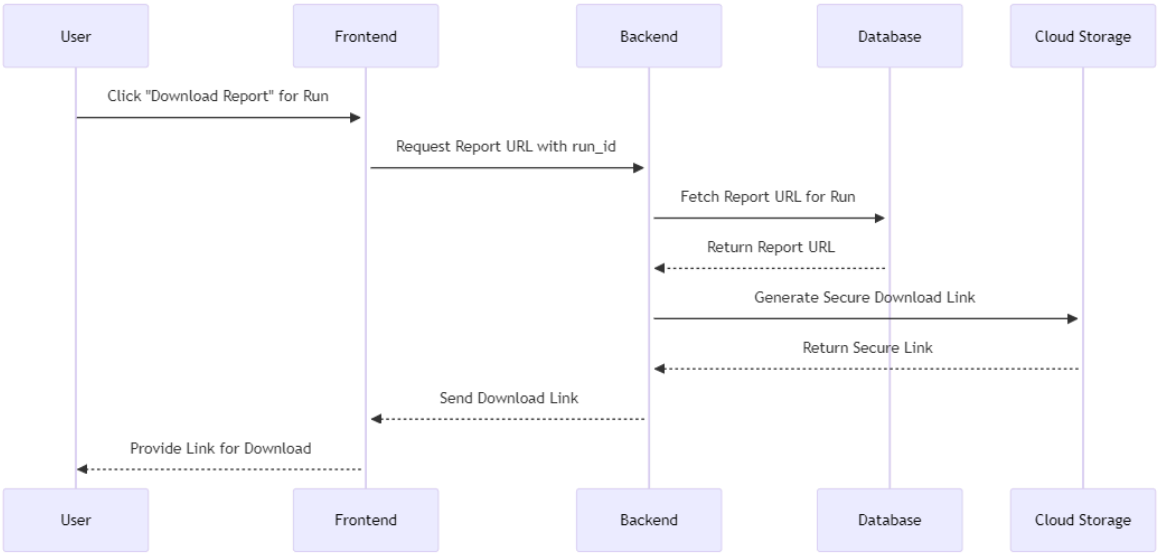
- Execution Time: When the run occurred.
- Key Metrics: ROI, total trades, win rate, etc.
- Action Buttons: A **"Download Report"** button for each run.

Download Run Report

Steps in the Flow

1. **User Action**:
   - The user selects a specific run and clicks **"Download Report"**.
2. **Frontend Request**:
   - Sends the request to the backend with the `run_id`.

3. **Backend Processing**:
    - Verifies the user's authorization to access the run.
    - Retrieves the `report_url` for the specified run from the database.
4. **Cloud Storage Interaction**:
    - The backend generates a secure, time-limited download link (if required) from the cloud storage.
5. **Send URL to Frontend**:
    - The backend sends the download link to the frontend.
6. **File Download**:
    - The user clicks the link, and the report is downloaded to their device.



Run Report Content

The report provides detailed insights into a single strategy run, including:

1. **Run Overview**:
    - Execution time.
    - Strategy summary.
2. **Performance Metrics**:
    - ROI, Sharpe ratio, drawdown, win rate, etc.

3. **Trade Analysis**:
    - Details of executed trades (entry/exit prices, quantities).
4. **Triggered Alerts**:
    - Conditions met during the run.
5. **Visualizations**:
    - Plots of portfolio value, trade signals, etc.

Advantages

View Runs:

- **Transparency**: Users can see historical evaluations and track their strategy performance over time.
- **Flexibility**: Easily access key metrics without downloading full reports.

Download Run Report:

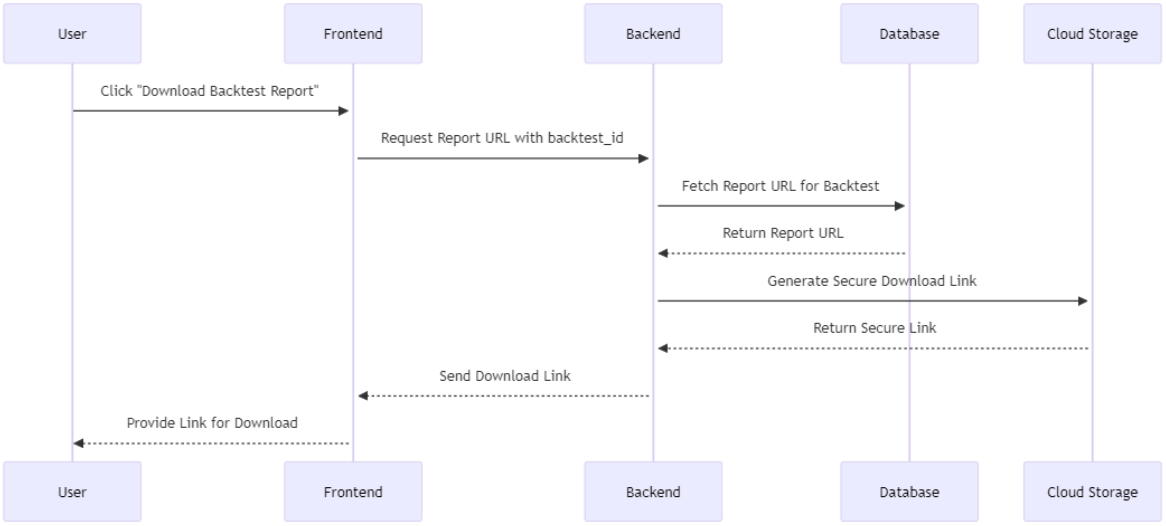
- **Rich Insights**: Provides a deep dive into individual run results.
- **Secure Access**: Ensures only authorized users can download reports.

Download Backtest Report

The **Download Backtest Report** feature allows users to retrieve a detailed report summarizing the backtest results for a strategy. Reports include performance metrics, triggered alerts, and visualizations.

Steps in the Flow

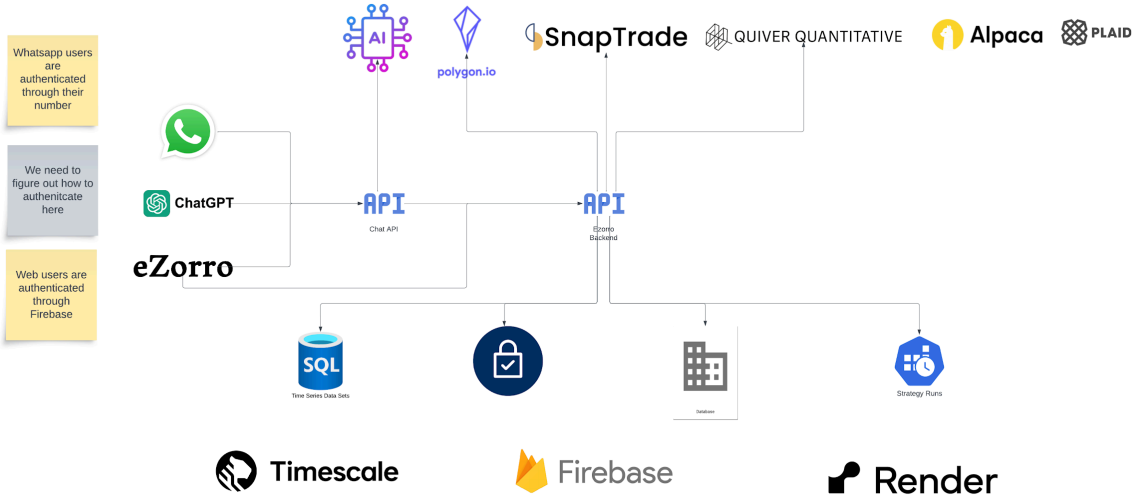
1. **User Action**:
    - The user selects a backtest and clicks **"Download Report"**.
2. **Frontend Request**:
    - Sends a request to the backend with the `backtest_id`.
3. **Backend Processing**:
    - Verifies the user's authorization to access the backtest.
    - Retrieves the `report_url` for the backtest from the database.
4. **Cloud Storage Interaction**:
    - The backend generates a secure, time-limited download link from cloud storage.
5. **Send URL to Frontend**:
    - The backend sends the download link to the frontend.
6. **File Download**:
    - The user clicks the link and downloads the report to their device.



eZorro Services

The following is a list of services currently supported or to be supported by eZorro:

1. Polygon.io to fetch market and fundamental data
2. QuiverQuant to fetch alternative data
3. SnapTrade to integrate broker accounts
4. Alpaca to offer accounts and trading for both US stocks and crypto currencies
5. Plaid for account funding and transactions
6. Firebase used to store NoSQL data
7. Timescale used to store time series data
8. Render used to host services and cron jobs


Whatsapp users are authenticated through their number
We need to figure out how to authenitcate here
Web users are authenticated through Firebase
AI
polygon.io
SnapTrade
QUIVER QUANTITATIVE
Alpaca
PLAID
ChatGPT
API
Chat API
API
Ezorro Backend
eZorro
SQL
Time Series Data Sets
Strategy Runs
Timescale
Firebase
Render