Project Scope: Automated Trading Bot

Project Overview

This document outlines the scope for developing an automated trading bot that integrates with TradingView and executes trades based on a custom RSI strategy – with the goal in mind to create a framework to be able to deploy bots dynamically. The bot will utilize webhooks for trade signals and dynamically configure trading parameters, enabling scalability across multiple accounts, charts, and trading rules.

Objectives

- Automate trading execution based on RSI-based strategies.
- Enable dynamic configuration for accounts, assets, and RSI parameters.
- Support dynamic trade scaling while adhering to maximum position limits.
- Provide comprehensive logging and notifications for trade executions.
- Ensure seamless integration with a brokerage APIs.
- Implement Celery workers for efficient background trade execution.
- Provide a real-time web dashboard for monitoring trade performance.

Key Features & Functionalities

1. TradingView Webhook Integration

- The bot will receive real-time trading alerts from TradingView via webhooks.
- Alerts will include relevant RSI values, asset details, and trade signals (LONG/SHORT).

2. Configuration Manager

- Supports multiple trading strategies per account.
- Allows configuration of:
    - Trading account (e.g., Herenya)
    - Chart symbol (e.g., Herenya, Herenya Capital)
    - Timeframe (e.g., 4H, 1H)
    - RSI settings (length, overbought/oversold levels)
    - Position sizing and scaling rules
    - Maximum number of open positions
- Configurations are stored in a PostgreSQL database for seamless updates.

3. Trade Execution Engine

- Executes trades via the CCXT library for unified exchange API integration.
- Ensures compliance with maximum position limits.
- Dynamically scales into trades if RSI conditions persist.
- Closes positions and reverses trades upon meeting opposite RSI conditions.
- Supports paper trading mode for strategy testing.
- Uses Celery workers and Redis for asynchronous trade execution to prevent blocking.

4. Monitoring & Notifications

- Logs all trade executions for audit purposes.
- Sends alerts via Telegram for trade updates, including execution success/failure.
- Implements error handling and retry mechanisms for API failures.
- Provides a real-time trade monitoring dashboard with execution logs and statistics.

5. Web Application & Dashboard

- A Flask-based web dashboard for real-time trade monitoring.
- Displays live trade logs, execution statuses, and performance analytics.
- Provides configuration management for adjusting trading parameters dynamically.

Technology Stack:

Component	**Technology/Service**
Webhook Handling	Flask / FastAPI
Trade Execution	Python (CCXT API)
Configuration Storage	PostgreSQL
Logging & Alerts	Telegram API
Deployment	VPS (Self-hosted)

Implementation Plan

Phase 1: Planning & Design (Weeks 1-2)

- Define detailed trading rules and webhook payload structure.
- Design system architecture, UML diagrams, and trading workflow.

Phase 2: Development (Weeks 3-6)

- Implement the Webhook Handler to receive TradingView signals.
- Develop the Configuration Manager for dynamic settings.
- Build the Trade Execution Engine with position management using CCXT.
- Implement Celery workers and Redis for background trade execution.
- Set up the Logging and Notification System.
- Develop the Web Dashboard for real-time monitoring.

Phase 3: Testing (Weeks 7-8)

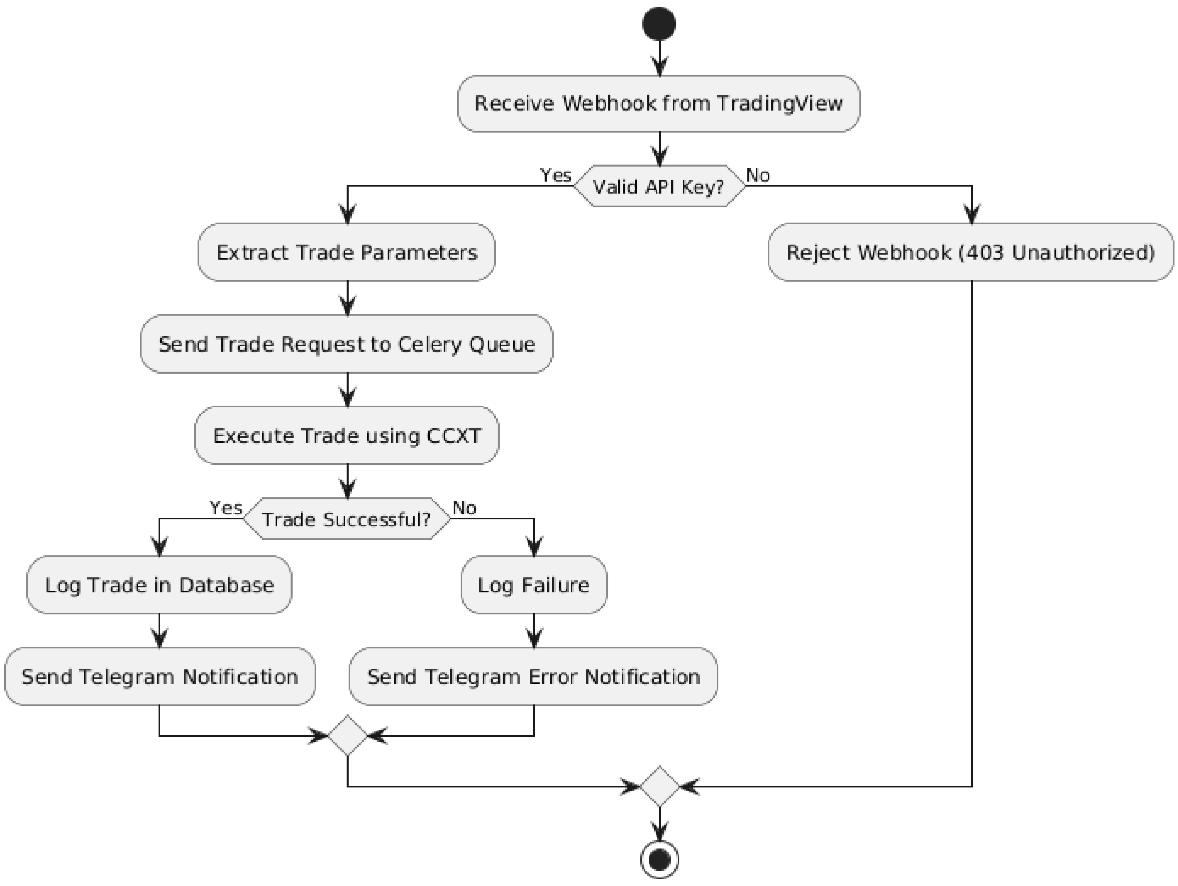
- Conduct backtesting using TradingView historical data.
- Run paper trading mode to simulate real trades.
- Validate trade execution, scaling behavior, and Celery worker efficiency.

Phase 4: Deployment & Monitoring (Weeks 9-10)

- Deploy the bot to a server (VPS).
- Configure security, API key management, and database persistence.
- Monitor performance, optimize parameters, and ensure fault tolerance.

System Workflow

This diagram represents the workflow when the system processes a trade.


Receive Webhook from TradingView
Valid API Key?
Yes
No
Extract Trade Parameters
Reject Webhook (403 Unauthorized)
Send Trade Request to Celery Queue
Execute Trade using CCXT
Trade Successful?
Yes
No
Log Trade in Database
Log Failure
Send Telegram Notification
Send Telegram Error Notification

Success Metrics

- Accurate execution of trades based on TradingView signals.
- Scalability across multiple trading strategies without requiring code modifications.
- Zero trade execution failures due to API issues.
- Efficient logging and real-time notifications for all trades.
- Smooth operation of Celery workers for handling large trade volumes.
- Intuitive web dashboard for monitoring and managing trading activities.