The Bot of Wall Street

Autonomous Stock Trading Bot System

2023 / 2024

Course: Software Engineering for Autonomous Systems

Professor: Davide Di Ruscio

Project GitHub Repository: https://github.com/ricardochavezt/the-bot-of-wall-street

Project created by:

Sherkhan Azimov	sherkhan.azimov@student.univaq.it
Daniel Ricardo Chavez Tapia	danielricardo.chaveztapia@student.univaq.it
Alex Montoya Franco	alex.montoyafranco@student.univaq.it

Table of Contents

Project Description 3
- Goals of the System 3
- Functional Requirements 3
- Non-Functional Requirements 4

System Components 4
- Managed Resources 4
- Sensors and Effectors 5
- Autonomic Manager 6
  - Architectural Pattern 6
  - Adaptation Goals 7
  - Decision Function 8
- MAPE-K 8
  - Monitor 8
  - Analyser 8
  - Planner 8
  - Executer 9
  - Knowledge 9

System Architecture 9
- MAPE-K Architecture Diagram 9
- Technologies 10

Instructions 11

Project Description

The automated stock trading bot aims to optimise investment returns through autonomous decision-making. Key objectives include efficient buy/sell order execution, real-time adaptation to market conditions, and a user-friendly interface for monitoring and configuration. Managed resources encompass financial portfolios and historical market data. The system employs a MAPE-K feedback control loop, actively monitoring market data, analysing trends through machine learning, formulating trading plans, executing orders, and updating a knowledge base for continuous improvement.

Goals of the System

The primary goal of the automated stock trading bot is to maximise returns on investment based on autonomous decision-making. Key objectives include:

- Efficiently execute buy/sell orders based on trading algorithms.
- Adapt to real-time market conditions to minimise risks and/or maximise profits.
- Provide an interface for monitoring and configuring the bot.

Functional Requirements

Functional Requirements of the Autonomous Stock Trading Bot		
Identifier	Name	Description
STB-FR001	Automated Trading	The system must be capable of automatically executing buy and sell orders based on predefined trading algorithms.
STB-FR002	Data Integration	The bot should integrate market data, including stock prices and trading volumes.
STB-FR003	Portfolio Management	Users should be able to configure and manage their financial portfolios by specifying their investment preferences.

STB-FR004	Adaptation to market conditions	The system must adapt to changing market conditions to maximise returns on investments and minimise risk.
STB-FR005	User Interface	Provide an interface for users to monitor the bot's activities, manage stocks, and receive performance reports.

Non-Functional Requirements

Non-Functional Requirements of the Autonomous Stock Trading Bot		
Identifier	**Name**	**Description**
STB-NFR001	Orders' Performance	The system should be able to execute buy and sell orders in less than 1 second.
STB-NFR002	Scalability	The system should be able to scale to an increasing number of stocks and new market data with minimum response time.
STB-NFR003	Usability	The system should provide a user interface with clear visualisations and notifications that let users understand their portfolio and trading decisions. The system shall let the user change preferences in an intuitive manner.

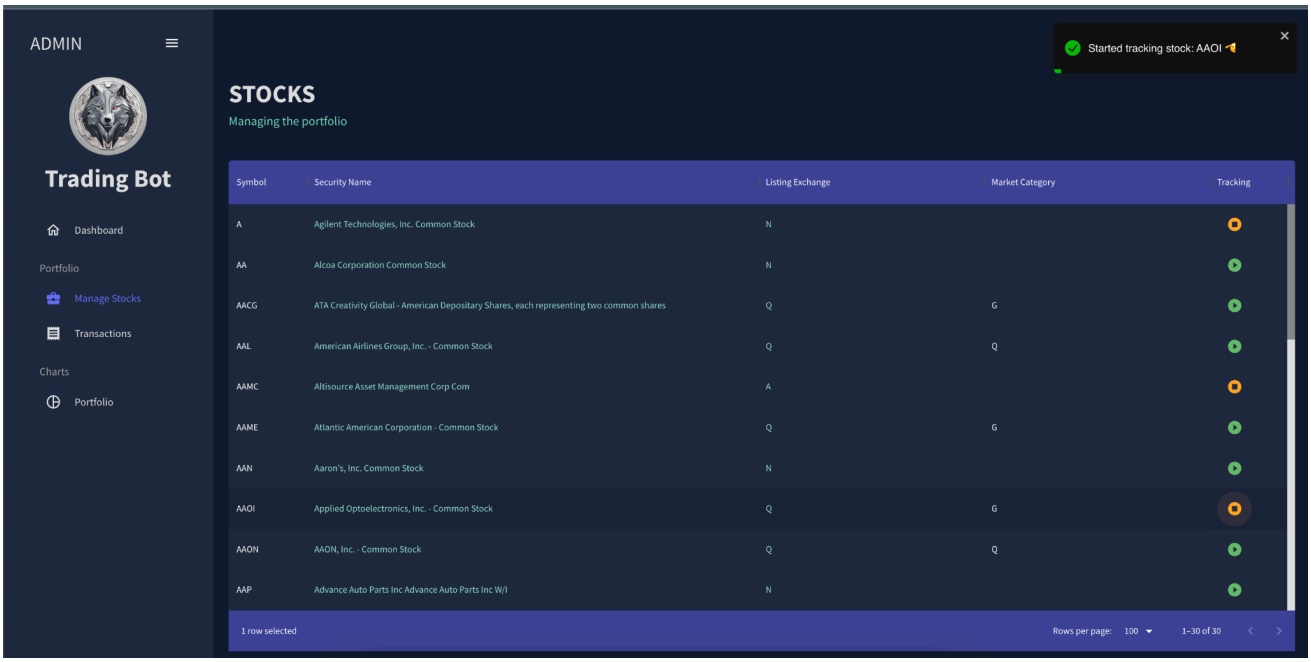
System Components

Managed Resources

The managed resources in the system include:

Financial Portfolio

- Stocks
    - Within the dashboard's 'Manage Stocks' section, illustrated in the figure below, users have the option to select from a range of 30 stocks to initiate/stop trading.



Market Data

- Real-time and historical data of the prices from financial markets.
    - The data was extracted from the following link: https://www.kaggle.com/datasets/jacksoncrow/stock-market-dataset/data
    - The data contains information on prices for up to 1 April 2020. The CSV files contain the following fields: Date, Open, High, Low, Close, Adj Close, and Volume.
    - In this project, we only used data from 30 stocks. The data is divided into two parts, where the first part is added in batch to represent historical data and train ML models. The second part is added incrementally to represent real-time data daily flow and update the model with online learning.

Sensors and Effectors

Market Data Sensor

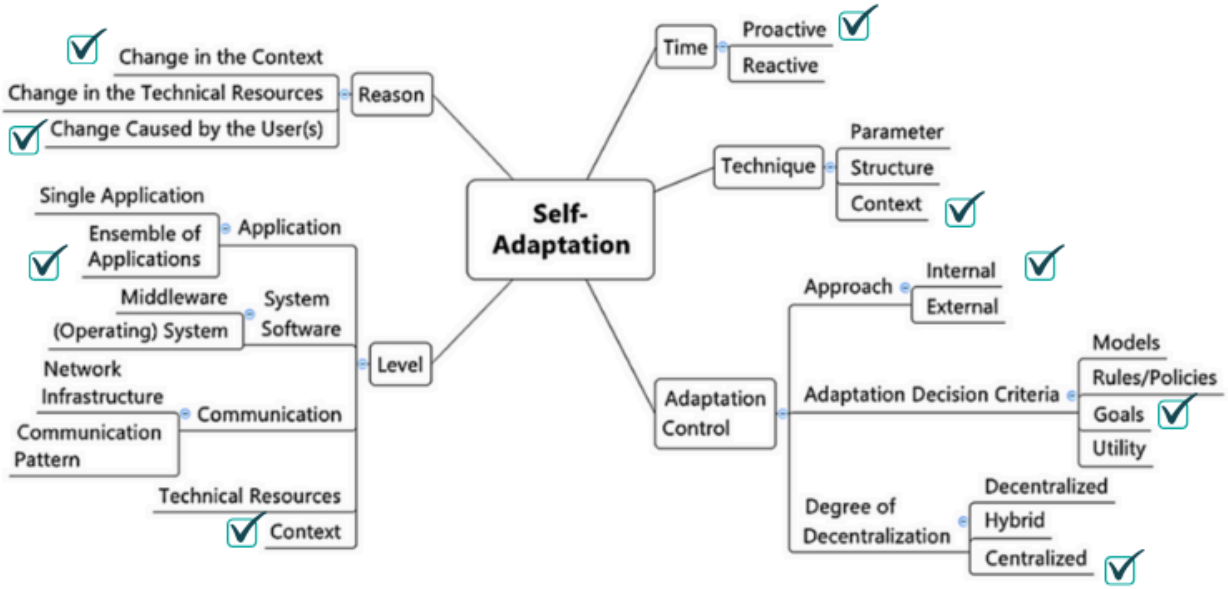
- Collects market data, including stock prices, trading volumes, and relevant financial indicators.

Execution Platform Effector

- Facilitates the execution of buy/sell orders in response to trading signals.

Autonomic Manager

Architectural Pattern



Self-Adaptation

- **Reason**
  - *Change in the Context:* Adaptation occurs when the forecasted stock price goes up or down in respect to the current price, triggering the system to execute a buy, hold, or sell action.
  - *Change caused by the user(s):* The user can start following a new stock causing the system to train a new forecasting model for that stock and start executing investment decisions. The user can also stop the investment on a previously followed stock causing the system to stop executing actions regarding that stock.
- **Level**
  - Application
    - *Ensemble of Applications:* The system is based on containerized microservices that communicate through a message broker and access a database server, meaning the system could easily be scaled to run on different machines.
  - *Context:* The system adapts to changes in market prices. For instance, limiting the number of stocks to buy depending on the price. If a stock price is higher than $100,

the system will only buy one stock, if the price is lower than $100, the system will buy the number of stocks that will amount up to $100.

- **Time**
  - *Proactive:* The analyzer module takes new incoming stock price data and forecasts the value of the next day for the planner to devise the best strategy.
- **Technique**
  - *Context:* The system keeps the state of the invested stocks and is continuously updated with the actual new prices in the stock market. The system will hold a specific stock if the price keeps going up (in order to maximise potential profit), and decides to sell once the forecasting of the price goes down.
- **Adaptation Control**
  - Approach
    - *Internal:* The adaptation logic is linked to the managed resources, with adaptation initiated in response to changes within these resources.
  - Adaptation Decision Criteria
    - *Goals:* Adaptation is driven by the goals of the autonomic manager.
  - Degree of Decentralization
    - *Centralised:* Adaptation logic is implemented on the planner module which takes forecasting values and devises the best strategy for the executor to move forward with.

Adaptation Goals

Adaptation Goals of the Autonomous Stock Trading Bot		
Goal	**Description**	**Evaluation metric**
Maximise Returns	Adjust trading strategies to maximise profit.	$PriceAtSelling - PriceAtBuying \geq (0 + ExpectedReturn)$
Risk Management	Implement adaptive measures to minimise potential losses.	$\lvert PriceAtSelling - PriceAtBuying \rvert \leq ExpectedLoss$
Accurate Market Predictions	Tune market predictions based on new available data.	$\lvert RealPrice_{t+1} - PredictedPrice_{t+1} \rvert \leq \lvert RealPrice_{t} - PredictedPrice_{t} \rvert$

Decision Function

The decision function of the autonomic manager is based on a hybrid approach that combines rule-based and AI-based methods. The rules define specific conditions for buying or selling, while machine learning algorithms analyse historical and real-time data to identify patterns and trends. This hybrid approach aims to combine the advantages of rule-based systems for transparency and interpretability with the learning capabilities of AI-based systems for adapting to dynamic market conditions.

MAPE-K

Monitor

Utilises the available data to track historical market prices. After adding the daily price entry from the CSV file to the database, the monitor will notify the analyser about the new entry.

Analyser

Processes data using machine learning algorithms to identify patterns, trends, and potential trading opportunities. After receiving a notification from the monitor, the analyser checks whether there is a machine-learning model for the specified stock. If a model exists, it retrieves the latest entry, updates the model, saves it, and informs the planner of the predicted stock price. In case the analyzer has not previously trained on the specific stock, it acquires all available data for the specific stock, conducts training, saves the model, and notifies the planner of the predicted price.

Planner

Formulates trading strategies through rule-based analysis, taking into account risk management and historical performance. After receiving a predicted price from the analyzer, the planner makes decisions based on the specified adaptation goal, as outlined in the table. After making a decision, which can be “buy”, “sell”, or “continue” the system passes it to the executor.

Executer

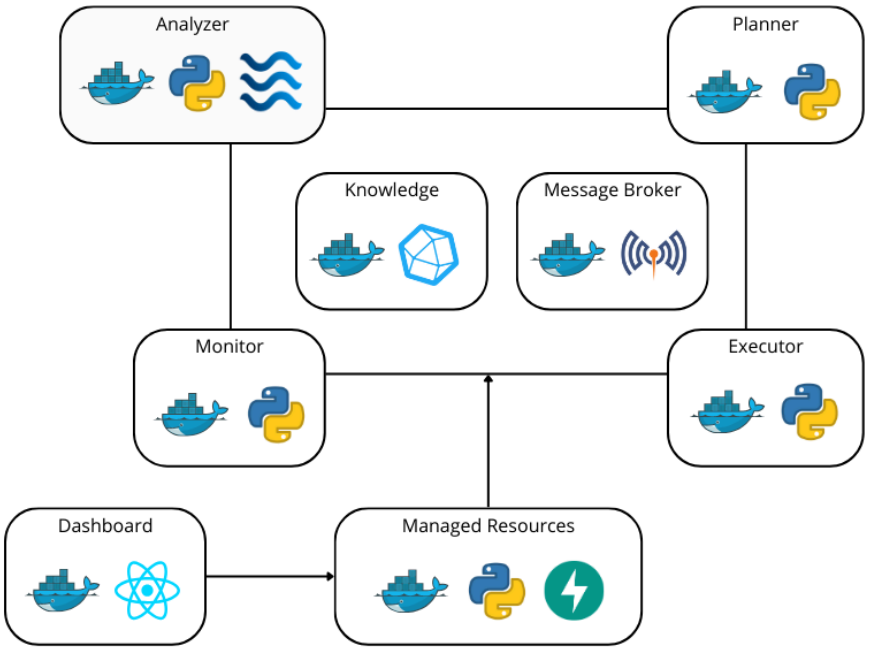
Initiates buy/sell orders based on the planned strategies. Upon receiving a decision from the planner, the executor executes the order and records the recent transaction. The executor buys some stocks up to $100. For example, if the stock price is $10, it will buy 10 stocks. If the price is higher than $100, then it will buy only one stock. If the action is to “sell”, then the executor sells all available stocks. In the case of “continue”, the executor doesn’t take any action. Finally, it notifies the monitor to obtain new data.

Knowledge

Reflects the knowledge base, incorporating insights gained from market analysis and execution outcomes.

System Architecture

MAPE-K Architecture Diagram



Within the dashboard, users select stocks for tracking, triggering a post-request to managed resources. These resources notify the monitor through specific topics:

- monitor/stock/added: Informs the monitor of the addition of a stock to the tracking list.
- monitor/stock/removed: Notifies the monitor of the removal of stock from the tracking list.

Consequently, the analyzer predicts the stock price and communicates with the planner using the following topic:

- planner/prediction/stock: Updates the planner with the predicted price for the specified stock.

The planner, guided by predefined goals, decides to "buy", "sell", or "continue" and communicates this decision to the executor through the topic:

- executor/action: Notifies the executor about the action to be taken.

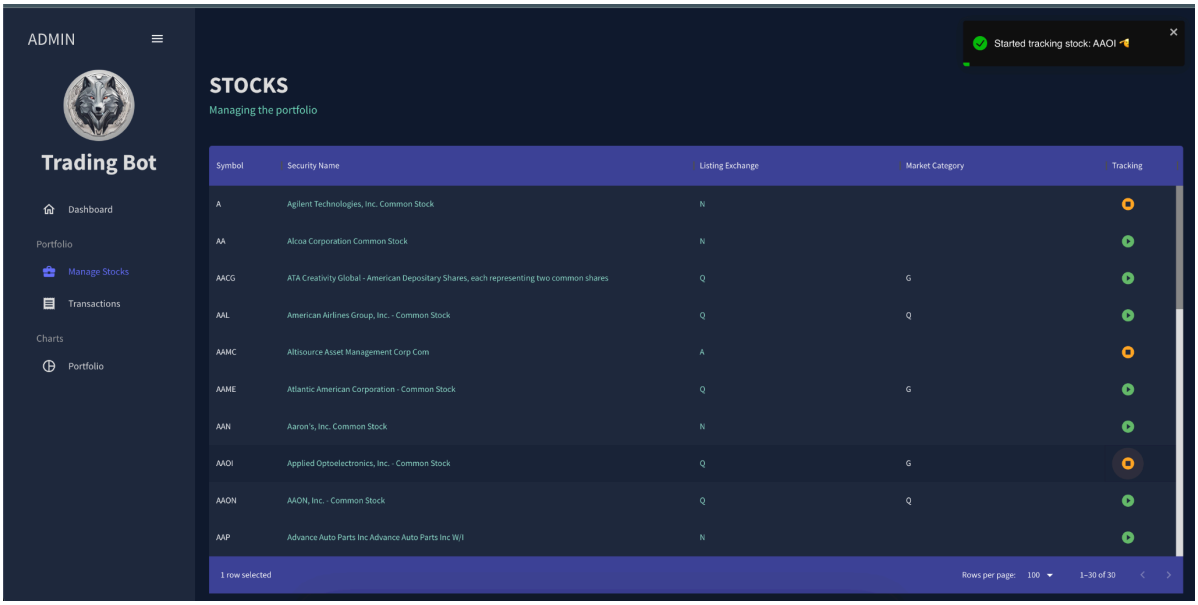
Upon execution, the executor notifies the monitor to obtain new data:

- monitor/completed: Notify the monitor to get the new data.

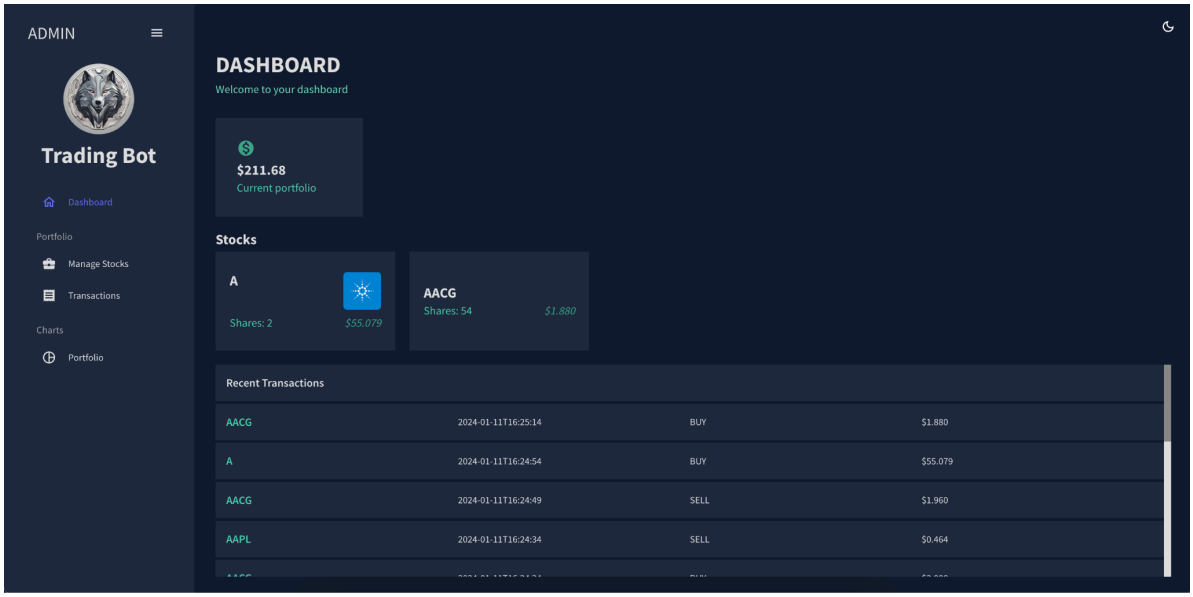
Technologies

Instructions

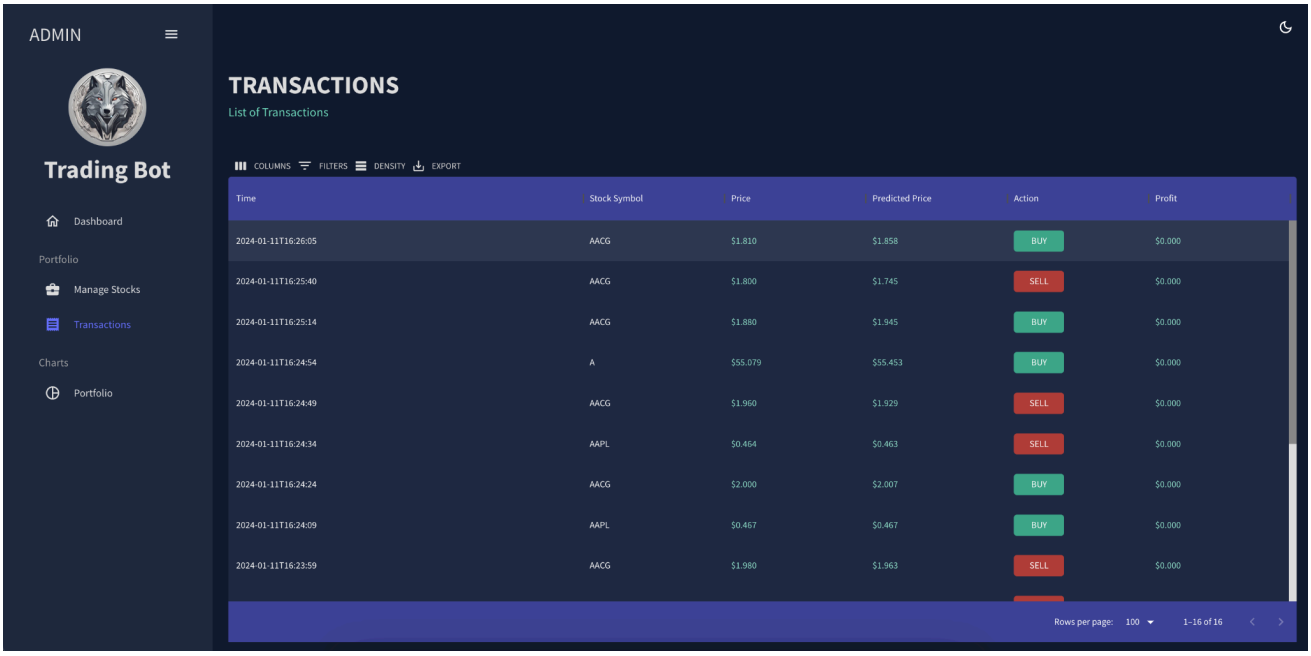
To start interacting with the trading bot, run the following command: docker-compose up. Then, navigate to http://localhost:5001/, where you can select the “Manage Stocks” section to see the table with the following data: Symbol, Security Name, Listing Exchange, Market Category, and Tracking. Also, you can start tracking the stock by clicking the button in the “Tracking” column. You will see the relevant notification if you started tracking or stopped tracking.



After selecting the stocks to track, in the “Dashboard” section, you can see the amount of your current portfolio, the stocks, and recent transactions.



In the “Transactions” section, you can find the table for the transactions with the following columns: “Time”, “Stock symbol”, “Price”, “Predicted Price”, “Action”, and “Profit”.



Also, the user has an option to visualise the current portfolio as a pie chart in the “Portfolio” section.