

Capstone Project Phase B

InvestSenseAI

**Project number** - 24-2-D-13

Supervisor: Mr. Alex Keselman

Raz Waiss - [WaissRaz27@gmail.com](mailto:WaissRaz27@gmail.com) - 315495622

Niv Dahan - [Niv2232@gmail.com](mailto:Niv2232@gmail.com) - 308215086

[Github](https://github.com/Raziloo/final.git)

[**Abstract 3**](#_oa977pqjooh4)

[**1 Introduction 3**](#_jyomcpqds1se)

[**2 System Objectives and Implementation 4**](#_nxhqhp4j7gtz)

[2.1 System Structure 4](#_5jmrmv7mfv6w)

[2.2 Target Users 4](#_fldqekyw6lx2)

[**3 System Architecture Overview 5**](#_30j0zll)

[3.1 Frontend (User Interface): 5](#_5fgd87khvkn7)

[3.2 Backend: 5](#_y8e4s1qatfx)

[3.3 ML Model: 5](#_htoqj3x3vvgu)

[3.4 Data Layer: 5](#_e761v5z1jz2o)

[3.5 AI & Analytics Engine: 5](#_wqg45slzx056)

[3.6 Integration & Deployment: 5](#_3fqdk1hmzaqc)

[**4 Research and System Development Approach 6**](#_gilc52qykrxp)

[4.1 Tools and Technologies Used 6](#_yimwskj98e12)

[4.2 Interaction with Users During Development 6](#_9abzpaqb6w2f)

[4.3 Challenges and Solutions 7](#_t3p67k2earcx)

[4.4 Results and Conclusions 7](#_h19rz5v6se54)

[4.5 Lessons Learned and Retrospective 8](#_ov4m6b47pjcc)

[**5. Our ML model 8**](#_30t8kthecody)

[5.1. Loading the Data 8](#_1ytl2ypdi8n7)

[5.2. Preprocessing the Data 9](#_qqmhbu1mqiq7)

[5.3. Building the Model 9](#_7pvfms7w3zl2)

[5.4. Training the Model 10](#_e9vp28utrbdo)

[5.5. Saving and Loading the Model 10](#_g9wi0x1yodeo)

[5.6. Preparing Test Data 10](#_920hw026s3hi)

[5.7. Making Predictions 10](#_w4mlf03wpqco)

[5.8. Predicting the Next Day’s Price. 11](#_gra47ejcjftq)

[5.9. Saving Predictions 11](#_3t0m04ffyh5g)

[**6 User guide 11**](#_m0cunytegqwg)

[**7 Maintenance Guide 11**](#_hcikh99sbmm4)

[7.1. System Environment 12](#_6dpl6pclofhk)

[7.2 Installation Guide 12](#_8om1lmj0nrtq)

[7.3 Frontend setup 12](#_dfherooll3fa)

[7.4 AI model setup 13](#_p2x2uk2dkgqx)

[7.5 System Updates & Maintenance 13](#_mbhvzrkeljte)

[**8 Troubleshooting & Common Issues 14**](#_29ik0f4e80f4)

[**9 Screenshots from the web app: 15**](#_dxj89dvtj8lp)

[**10 Activity diagram and package diagram: 17**](#_6963dc2wflqs)

[**Package diagram: 18**](#_rafywmx6mzeq)

[**11. References 19**](#_4uiuotiedlgj)

# 

# Abstract

Stock trading has evolved significantly with the advancement of artificial intelligence (AI) and data-driven analytics. This project aims to develop a web-based AI trading assistant that provides buy/sell recommendations based on a combination of technical indicators, fundamental analysis, and real-time stock data.;

The project utilizes Polygon.io for real-time stock data retrieval and MongoDB for data storage. The AI model processes historical data to generate actionable insights. The web application features user registration, login, and a dashboard that displays stock charts.

This study explores the effectiveness of AI-driven trading strategies and assesses their accuracy in real-world scenarios. By leveraging modern web technologies and AI models, this project seeks to empower traders with intelligent decision-making tools, ultimately enhancing trading efficiency and profitability.

# **1 Introduction**

Stock trading is a complex and dynamic field where investors continuously seek strategies to maximize returns while minimizing risks. Traditional trading strategies rely heavily on human intuition and manual analysis of financial data, which can be time-consuming and prone to biases. With the rapid advancements in artificial intelligence and data analytics, automated trading systems have emerged as powerful tools that enhance decision-making processes and optimize trading strategies.

This project aims to bridge the gap between AI technology and stock trading by developing a web-based AI trading assistant. The system integrates real-time stock data, technical indicators to provide traders with data-driven recommendations.

By leveraging AI and machine learning techniques, this project seeks to improve trading accuracy and efficiency. The web-based platform ensures accessibility for traders of all experience levels, from beginners to seasoned investors. The implementation of advanced data processing methods and AI-driven insights will contribute to a more informed and strategic approach to stock trading.

The following sections of this document will delve into the methodology, implementation, testing, and results of the AI trading assistant, providing a detailed analysis of its effectiveness in practical trading scenarios.

# 2 System Objectives and Implementation

The AI-driven trading assistant aims to provide intelligent, data-backed buy/sell recommendations to traders by analyzing real-time stock market data. The primary goal of the system is to predict the next day stock price, by integrating a mix of technical indicators. To achieve this, the system incorporates several key components: Data Aggregation: Utilizing Polygon.io for real-time stock data retrieval, ensuring that users have up-to-date market information.Technical Analysis: Applying indicators such as moving averages, and to identify potential trading signals. AI Model: Employing machine learning algorithms to analyze historical data, market trends, and sentiment to generate actionable insights. User Interaction: Providing a web-based interface where users can view AI-generated trading recommendations, customize their preferences, and track their investments.

## 2.1 System Structure

The system is structured into three main layers: Frontend: A web-based interface where users register, log in, and access their personalized trading dashboard. Backend: A server-side application handling user authentication, data processing, and AI-driven analysis. Database: A MongoDB-based storage system for user data, historical trading insights, and financial reports. Machine Learning: AI model in python.

## 2.2 Target Users

The system is designed for: Retail Traders: Individuals looking to optimize their trading strategies with AI assistance. Experienced Investors: Professionals seeking data-driven insights to enhance their decision-making. Financial Analysts: Those interested in leveraging AI models for in-depth market analysis. By integrating advanced AI methodologies with traditional trading strategies, the system empowers traders with an intelligent, efficient, and user-friendly platform for stock market analysis.

### 

# 

# 

# 3 System Architecture Overview

The AI-driven trading assistant is structured using a multi-layered architecture that ensures efficiency, scalability, and real-time data processing. The system integrates multiple components, each serving a distinct function to provide users with intelligent stock trading recommendations.

### 

### Key Architectural Components

## Frontend (User Interface):

* Developed as a web application to ensure accessibility across devices.
* Built with modern web technologies to offer a smooth and interactive user experience.
* Displays stock charts, AI-generated buy/sell recommendations.

## **Backend**:

* The backend allows users to register, log in, and retrieve their profile information.
* Uses JWT (JSON Web Token) for session management, ensuring secure authentication.
* Protects certain routes so only authenticated users can access them**.**

## ML Model:

* Implemented in Python to leverage advanced machine learning libraries.
* Processes stock market data and financial reports for accurate trade recommendations.
* Integrated with the backend, which is developed in both JavaScript and Python, ensuring seamless communication and data exchange.

## 3.4 Data Layer:

* Utilizes Polygon.io for real-time stock data streaming.
* Ensures data consistency and fast access for processing.

## AI & Analytics Engine:

* Employs machine learning models to analyze stock trends and generate trade recommendations.
* Uses of technical indicators
* Continuously learns and adapts based on market changes and user interactions.

## 3.6 Integration & Deployment:

* Deployed on cloud infrastructure for scalability and availability.
* Uses a microservices-based approach to separate concerns and ensure maintainability.
* Implements security measures, including encrypted user data and secure API communication.

### System Flow

1. The user logs into the platform.
2. The system fetches real-time stock data and processes it using AI models.
3. AI-generated recommendations are displayed on the user dashboard.
4. Users can analyze insights, adjust strategies, and take action based on AI predictions.

This architecture ensures a robust, data-driven, and user-centric approach to stock trading, making intelligent decision-making more accessible and efficient.

# 

# 4 Research and System Development Approach

To design and develop the AI-driven trading assistant, we followed a structured research and development process. This involved analyzing stock trading methodologies, selecting appropriate technologies, and iteratively building and refining the system to meet user needs.

## 4.1 Tools and Technologies Used

We utilized various tools and technologies to develop the system:

* Programming Languages: Python for AI model development, JavaScript (Node.js) for backend services, and React for frontend.
* Data Sources: Polygon.io for real-time stock data and financial reports.
* Database: MongoDB for structured storage and quick data retrieval.
* Machine Learning Libraries: TensorFlow and scikit-learn and Keras for AI model training and analysis.
* Version Control: Git and GitHub for collaboration and version management.

## 4.2 Interaction with Users During Development

Throughout the development cycle, we engaged with potential users, including traders and financial analysts, to gather feedback. This helped refine system features, UI/UX design, and improve recommendation accuracy.

## 

## 

## 4.3 Challenges and Solutions

1. Analytical Challenges:

* Challenge: Ensuring AI-generated recommendations were reliable and interpretable.
* Solution: We refined the AI model using multiple indicators and backtesting strategies to enhance accuracy.

1. Technical and Engineering Challenges:

* Challenge: Handling large-scale real-time stock data efficiently.
* Solution: Implemented optimized database queries and asynchronous data processing.
* Challenge: Ensuring seamless integration between Python-based AI models and the JavaScript backend.
* Solution: Used REST APIs and WebSockets for smooth communication.

1. Data Structure and Algorithmic Challenges:

* Challenge: Choosing the right data structures to store and process stock data efficiently.
* Solution: Implemented indexed storage in MongoDB and optimized querying techniques.

## 4.4 **Results and Conclusions**

* We successfully met the project objectives by developing an AI-powered trading assistant that provides accurate and actionable trading recommendations. The model demonstrated promising results in backtesting scenarios, validating its effectiveness.

### Key achievements:

* Integration of real-time stock data with AI-driven insights.
* Development of a user-friendly dashboard for easy interaction.
* Scalable system architecture supporting multiple users simultaneously.

### Decision-Making Considerations:

* We prioritized model accuracy over speed, leading to enhanced recommendation reliability.

## 4.5 Lessons Learned and Retrospective

1. Effectiveness of Development Approach:

* Our iterative approach allowed continuous refinement based on user feedback.
* Collaboration between AI and backend teams ensured smooth system integration.

1. Possible Improvements**:**
   * More extensive testing with live trading data to refine AI predictions further.
   * Enhancing the frontend experience with additional customization features.

3. Meeting Project Metrics:

* We successfully implemented the planned features within the project scope.
* The AI model's accuracy met our expectations in testing phases.

Overall, the project validated the potential of AI in trading, and future improvements will focus on real-time adaptation and user personalization.

# 5. Our ML model

## **5.1. Loading the Data**

#### The load\_data function utilizes the yfinance library to fetch historical stock price data for the specified company within a given date range. For this project, Tesla (TSLA) stock data was retrieved:

Input Parameters:

* **company**: Stock ticker symbol (e.g., ‘TSLA’).
* **start and end**: Start and end dates for data collection.
* **Purpose**: To provide the historical data required for training and testing the model.

## **5.2. Preprocessing the Data**

#### The preprocess\_data function prepares the raw stock data for the model:

**Steps**:

1. **Feature Scaling**: The closing prices are scaled between 0 and 1 using MinMaxScaler to improve model performance.
2. **Sequence Generation**: Training data is segmented into sequences of length prediction\_days (e.g., 60 days). Each sequence is used to predict the stock price of the next day.
3. **Reshaping Data**: The data is reshaped into the format (samples, timesteps, features) to fit the LSTM model’s input requirements.

**Output**: Scaled and reshaped training data (x\_train, y\_train) and the fitted scaler.

## 5.3. Building the Model

#### The build\_model function creates a neural network architecture:

**Architecture:**

* **LSTM Layers:** Three LSTM layers with 50 units each are stacked to learn temporal dependencies in the data.
* **Dropout Layers:** Dropout is applied to prevent overfitting by randomly deactivating 20% of the neurons**.**
* **Dense Layer:** A single neuron outputs the predicted stock price.

**Compilation:**

* **Optimizer:** Adam optimizer is used for faster convergence.
* **Loss Function:** Mean Squared Error (MSE) measures the prediction error.

**Output: A compiled model ready for training.**

The build\_model function creates a neural network architecture:

**Architecture**:

* **LSTM Layers**: Three LSTM layers with 50 units each are stacked to learn temporal dependencies in the data.
* **Dropout Layers**: Dropout is applied to prevent overfitting by randomly deactivating 20% of the neurons.
* **Dense Layer**: A single neuron outputs the predicted stock price.

**Compilation**:

* **Optimizer**: Adam optimizer is used for faster convergence.
* **Loss Function**: Mean Squared Error (MSE) measures the prediction error.

**Output**: A compiled model ready for training.

## 

## 5.4. Training the Model

#### The model is trained using the fit method:

**Training Parameters:**

* **Epochs:** 25 iterations through the entire training dataset.
* **Batch Size:** 32 samples per training batch.
* **Purpose:** To minimize the loss function and enable the model to learn from historical data.

## 

## 5.5. Saving and Loading the Model

#### The trained model and scaler are saved using:

* **Model:** Saved in .keras format using model.save.
* **Scaler:** Saved using joblib.dump.

If a saved model exists, it is loaded to avoid retraining.

## 

## 5.6. Preparing Test Data

**Steps:**

1. Combines training and test data to maintain continuity.
2. Scales the test data using the same scaler used for training.
3. Segments the test data into sequences of length prediction days.

**Output**: Prepared test data ('x\_test') and scaled inputs ('model\_inputs').

## 

## 5.7. Making Predictions

#### The model predicts stock prices for the test data:

* **Process:** The prepared test data is fed into the trained model using the predict method.
* **Postprocessing:** The predicted values are inverse-transformed to their original scale using the scaler.

**Output:** Predicted stock prices for the test dataset**.**

## 

## 5.8. Predicting the Next Day’s Price.

The 'predict\_next\_day\_price' function forecasts the stock price for the next day**:**

* **Input:** The last 'prediction\_days' worth of scaled data.
* **Output:** Predicted stock price for the next day in its original scale.

## 

## 5.9. Saving Predictions

#### 'The\_save\_prediction\_to\_file' function stores the predicted price and date in a JSON file:

* **Purpose:** To log the model’s output for future reference or integration with other system components.

# 

# 6 User guide

Getting started

### Accessing the System

* + Open a web browser and navigate to the application’s website.
  + Click on **Sign Up** to create a new account or **Log In** if you already have an account.

### Viewing Market Data

* + The homepage displays real-time stock data retrieved from **Polygon.io**.
  + Use the search bar to find specific stocks.

### Receiving AI Recommendations

* + The AI model analyzes stock data and provides **Buy/Sell** recommendations.
  + Recommendations are based on **technical indicators** (RSI, MACD, moving averages) and **fundamental analysis** (financial reports, 10-Q filings).

# 

# 7 Maintenance Guide

This document provides guidelines for maintaining the InvestSenseAI system, ensuring continuous usage after project completion. It covers software updates, system modifications, and improvements to extend the system’s lifecycle.

## 7.1. System Environment

The system requires the following software and hardware infrastructure:

• Operating System: macOS, Windows, or Linux

• Programming Languages: Python (for AI model), JavaScript/Node.js (for backend), React (for frontend)

• Database: MongoDB

• APIs & Data Sources: Polygon.io for real-time stock data

## 7.2 Installation Guide

Backend Setup

Install Node.js and npm (if not installed):

brew install node # macOS

Clone the backend repository:

git clone https://github.com/Raziloo/final/tree/main/backend.git

cd backend

Install dependencies:

npm install

## 7.3 Frontend setup

Clone the frontend repository:

git clone https://github.com/Raziloo/final/tree/main/frontend.git

cd frontend

Install dependencies

npm install

Start the frontend application:

npm start

## 7.4 AI model setup

Install Python 3.9+ and virtual environment tools:

brew install python # macOS

Clone the AI model repository:

git clone [https://github.com/Raziloo/final/tree/main/ml\_model.git](https://github.com/Raziloo/final/tree/main/frontend.git)

cd ml\_model

Create and activate a virtual environment:

python3 -m venv venv

source venv/bin/activate # macOS/Linux

venv\Scripts\activate # Windows

Install dependencies: 

pip install -r requirements.txt

Run the model API:

python app.py

## 7.5 System Updates & Maintenance

Updating Backend

* Pull the latest updates:

git pull origin main

npm install

npm start

Updating Frontend

* Pull latest UI updates:

git pull origin main

npm install 

npm start

Updating Ai Model

* Update Python dependencies:

pip install --upgrade -r requirements.txt

# 8 Troubleshooting & Common Issues

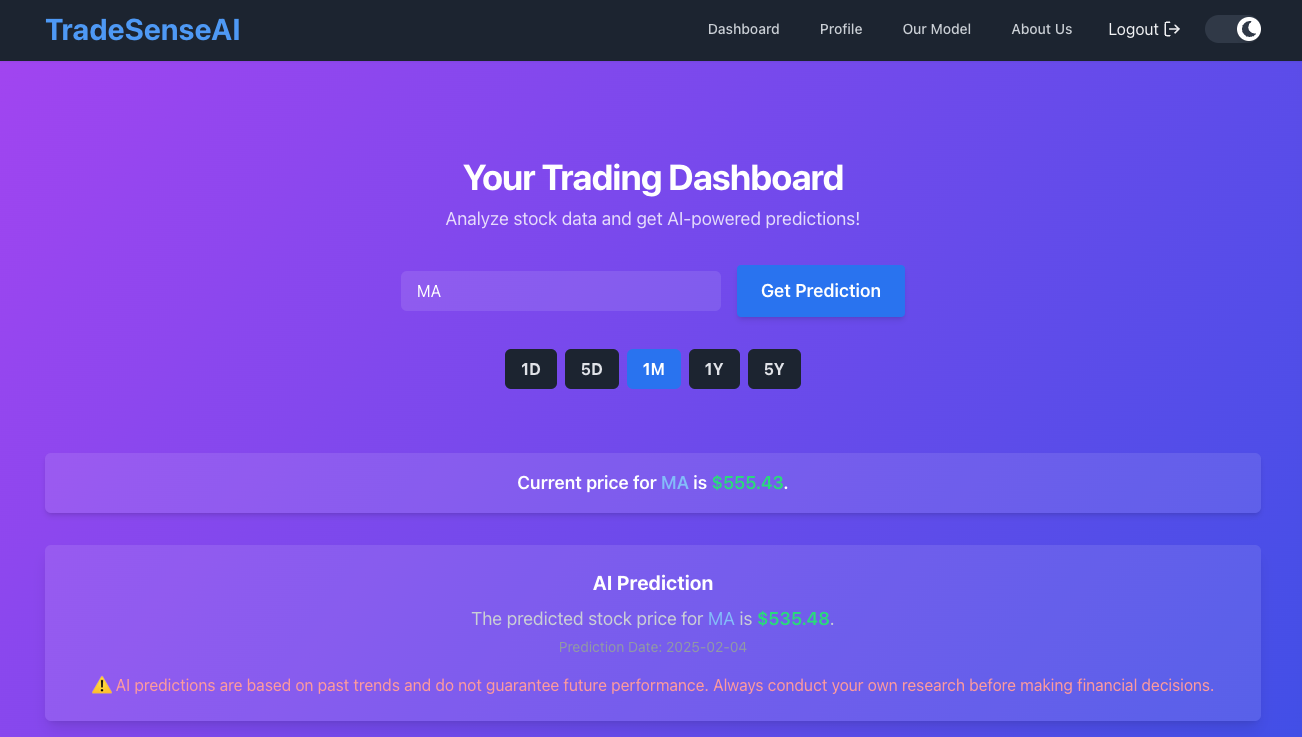
| Issue | Solution |
| --- | --- |
| Backend not starting | Check .env file and ensure MongoDB connection is valid |
| Frontend not loading | Ensure backend is running and CORS settings are correct |
| AI model errors | Verify Python dependencies and re-run train\_model.py |

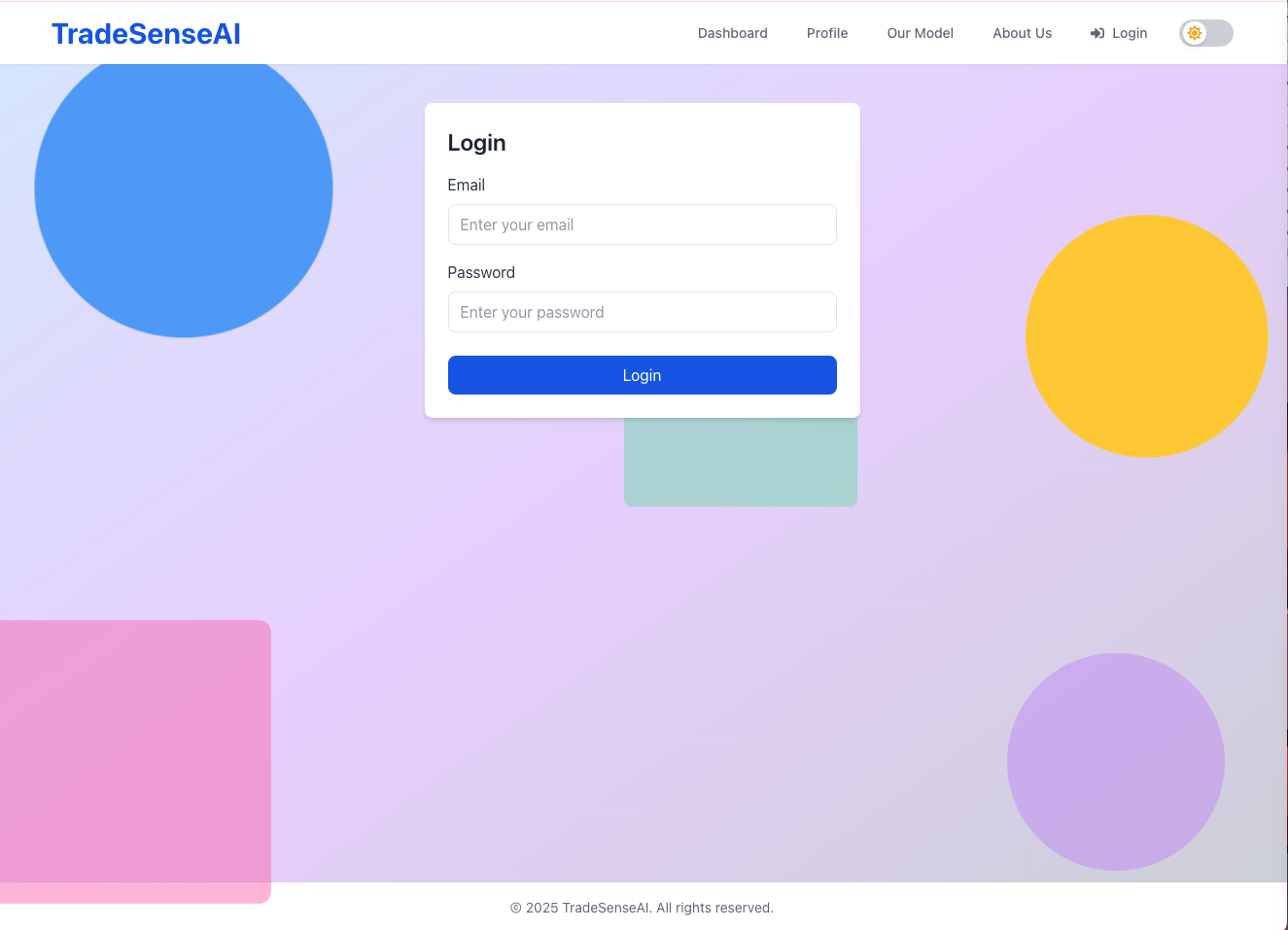
# 

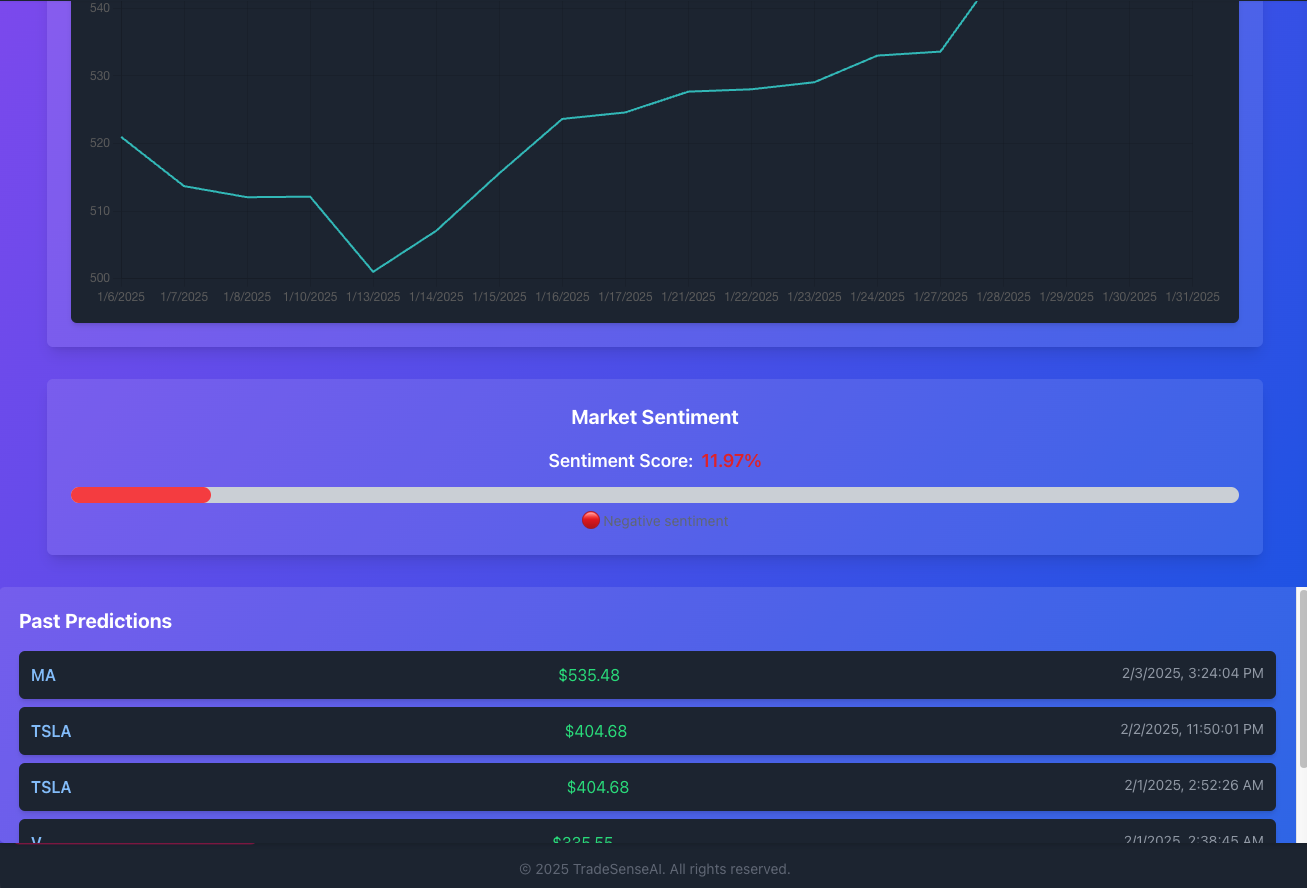
# 

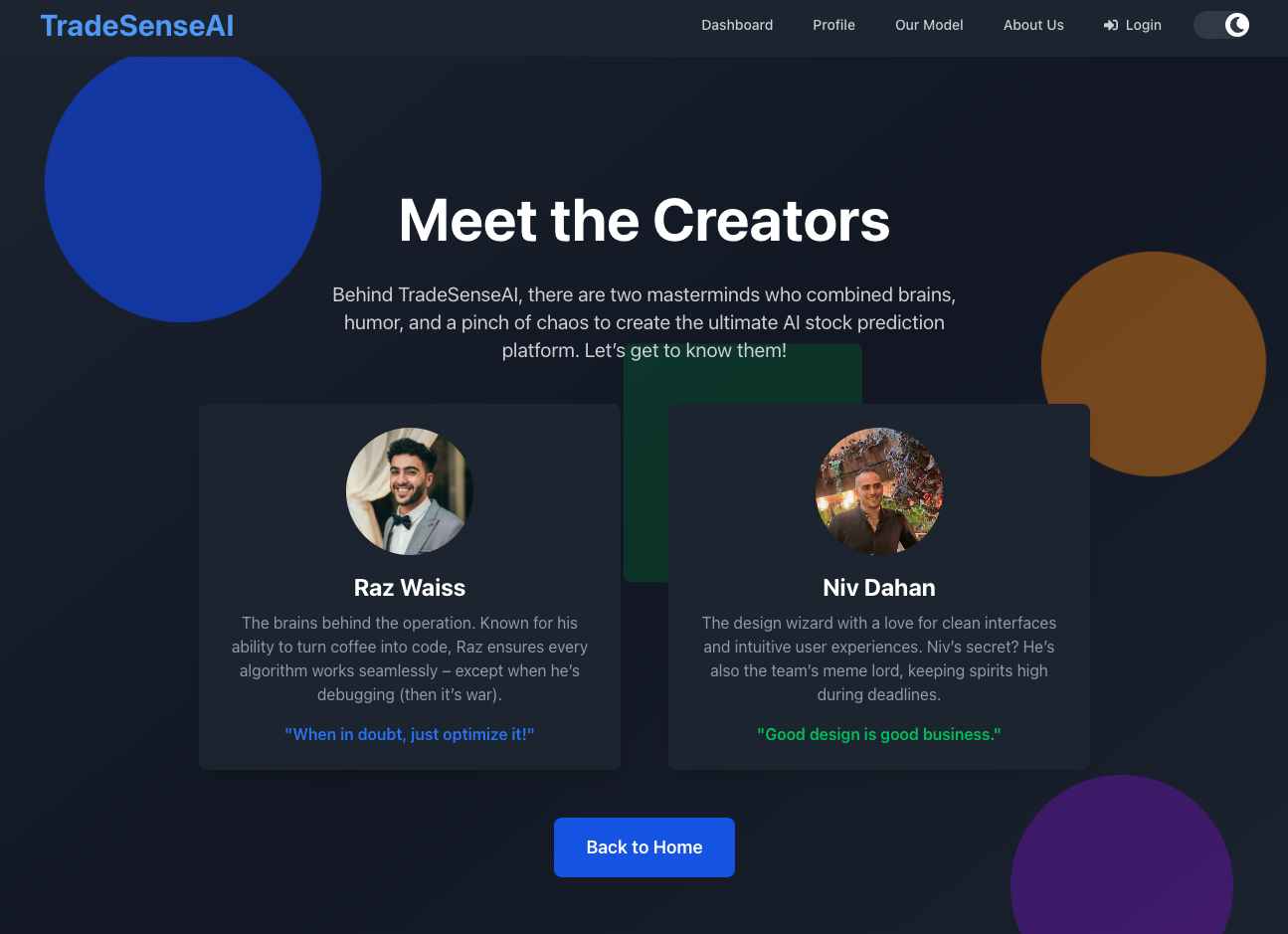
# 

# 9 Screenshots from the web app:

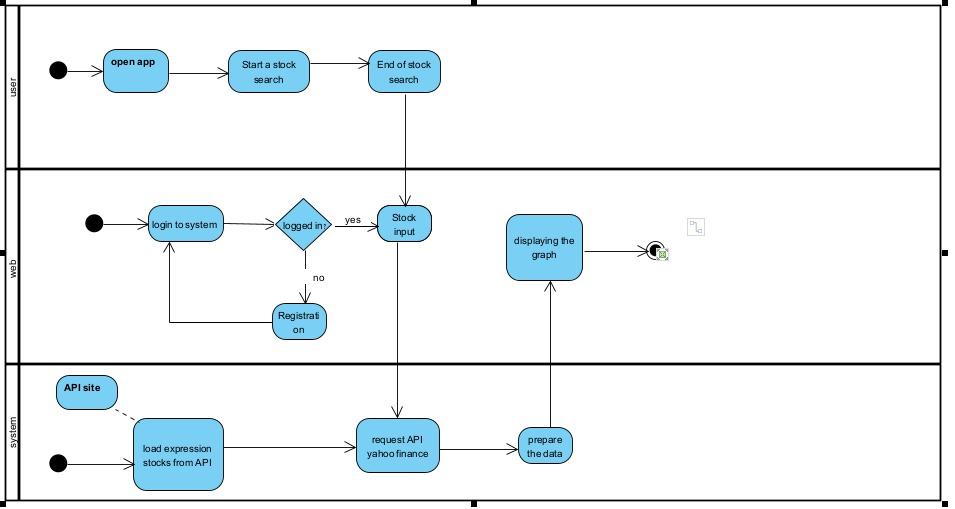








# 10 Activity diagram and package diagram:



**Sequence**

* The user launches or navigates to the app on their device/computer.
* Within the app, the user initiates a search (e.g., selects “Search Stock” or types a stock symbol).
* If the user is not already logged in, the system requests login credentials.
* If the user already has an account:

▪ They enter their username and password.

* If the user does **not** have an account:

▪ They register by providing the necessary details.

▪ The registration process completes, and the user can then log in successfully.

* After a successful login, the user enters or confirms the specific stock symbol to be searched (e.g., “AAPL” for Apple).
* Once the user confirms the stock symbol, the web layer calls the backend or external service to fetch stock data.
* This involves:

▪ Requesting available stocks or expressions from the API site.

▪ Submitting a specific request to an API provider like Yahoo Finance to retrieve the necessary stock information (historical data, current price, etc.).

* The API site processes the request, queries its data sources, and sends the response back to the web application.
* Before sending it to the front end, the system (server-side) processes or transforms the returned data. This might include:

▪ Formatting the data for a chart/graph.

▪ Calculating additional metrics or indicators if needed.

▪ With the processed data ready, the application’s front end (web layer) renders a stock chart or graph.

* The user sees the final visualization on their screen.
* The user’s stock search is complete. They can view the graph, possibly choose another stock, or log out.

# Package diagram:

The package diagram illustrates the high-level architecture of the system, highlighting its main components and their interactions. The system is divided into three primary layers: **backend**, **frontend**, and an **ML model** module.

1. **Backend**: This package includes the core server-side functionality, subdivided into:

* **Middleware**: Handles request/response processing, authentication, and other intermediary tasks.
* **Models**: Represents the database schema and manages data logic.
* **Routes**: Defines the API endpoints for interaction with the frontend and external systems.

1. **Frontend**: This package represents the user interface of the system, which interacts with users. It communicates with the backend to fetch data and display stock analysis and recommendations.
2. **ML Model**: This independent module is responsible for running machine learning algorithms to generate stock buy/sell recommendations. The **models** sub-package inside the ML model manages the training, prediction, and updating of ML algorithms. The communication between the **frontend**, **backend**, and **ML model** is seamless, ensuring efficient data flow for real-time stock analysis and decision-making.

# 11. References

Tensor flow - [https://www.tensorflow.org/api\_doc](https://www.tensorflow.org/api_docs)

Scikit-learn - <https://scikit-learn.org/stable>

Yahoo finance - <https://github.com/ranaroussi/yfinance>

polygon.io - <https://polygon.io/docs/stocks/getting-started>

Mongoose - <https://mongoosejs.com/docs/guide.html>

Express - https://expressjs.com/en/guide/routing.html

LTSM - <https://en.wikipedia.org/wiki/Long_short-term_memory>

Dense layers - <https://analyticsindiamag.com/topics/what-is-dense-layer-in-neural-network/>

Neural network - <https://en.wikipedia.org/wiki/Neural_network_(machine_learning)>

Tailwind css - <https://tailwindcss.com/docs/installation/using-vite>

Keras - <https://keras.io/api/>

Data preprocessing - <https://www.geeksforgeeks.org/data-preprocessing-machine-learning-python/>

Machine Learning for stock prediction - <https://neptune.ai/blog/predicting-stock-prices-using-machine-learning>

Mongo DB - <https://www.mongodb.com/docs/drivers/node/current/>

Hyperparameter optimization - <https://en.wikipedia.org/wiki/Hyperparameter_optimization>

Training, validation and test data sets - <https://en.wikipedia.org/wiki/Training,_validation,_and_test_data_sets>

React hooks - <https://react.dev/reference/react-dom/hooks>