Design Document

AI in Finance

By Group 8

Introduction

Project Overview

This project is about building an **AI system** to give users a competitive edge in financial markets. Our goal is to create a **Decision Support System** that uses smart computing techniques to identify new trends and recommend actions.

We are specifically focusing on a **Machine Learning** approach, which we believe is the best way to handle the complexity and speed of modern finance. Unlike optimisation models, our machine learning core will focus on **pattern recognition and prediction**. We're building a pipeline that not only handles prices but extracts important data from news and social feeds.

The final product will be a clean, accessible **web application** developed using Python for the machine learning and backend while standard frontend languages will be used such as HTML, CSS and Javascript for the user interface. This ensures the client can easily visualise the predictions and act on the system's recommendations.

Objectives

To achieve this, we have made the following objectives:

- **Robust Data Collection**: Design and implement a pipeline that can reliably collect, clean and merge lots of financial data.
- **Intelligent Prediction**: Develop and train Machine Learning models to accurately see future asset movements using combined financial and trend data.
- **Actionable Interface**: Create a responsive web application that visualises the model's predictions and translates them into straightforward BUY/SELL/HOLD recommendations.

Legal/Ethical Considerations

Projects that could potentially affect a user's quality of life require the right considerations to be made in order for the correct precautions to be implemented. Our project directly affects our users quality of life as we are handling their personal finances. Meaning the correct precautions must be taken to ensure the safety of potentially vulnerable or irresponsible individuals. Our application also interacts with financial markets so we must consider the corresponding regulatory rules.

Legal

- Application must contain a clear 'Not Financial Advice' disclaimer. If this is not included we can be legally treated as giving investment advice, which would come

with licensing requirements. Some high risk features such as specific buy/sell signals require specific licensing ; meaning those functions may be avoided.

- We must also consider Data Privacy & User Protection , in accordance with GDPR rules.

Ethical -

- With our project goal there comes a risk of misleading users. To avoid this we will avoid language conveying 100% confidence in our applications predictions.
- To protect users we will also not advise or support risky trading behaviour.
- Some users may also be too confident in the use of AI. To combat this we will provide a disclaimer explaining that these predictions are not 100% certain invest their money within reason.
- We will be sure to test the application thoroughly. Producing an acceptable and ethically valid end solution.

System Design & Architecture

Functional Requirements (Use Case)

The Use Case Diagram below shows the main interaction between the system and any external components. It defines the functional scope of the project from the perspective of the end user (Client/Investor) and the data sources needed for the machine learning pipeline.

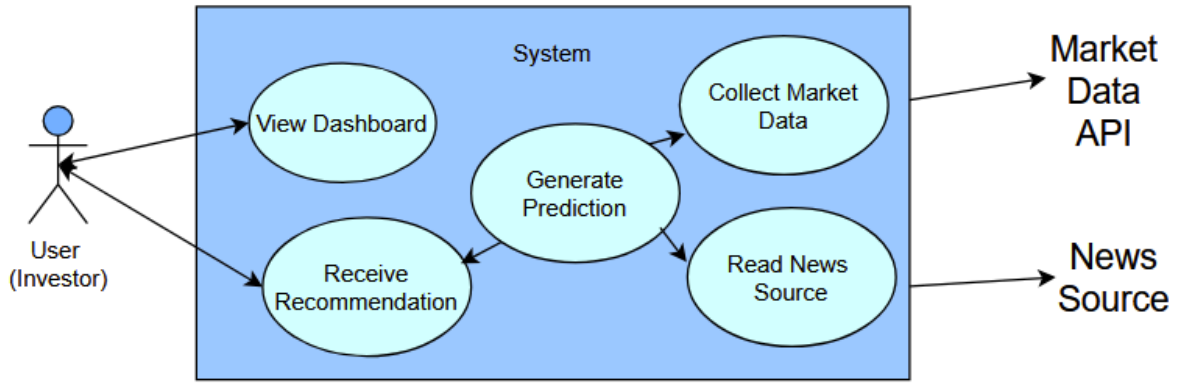
Actors:

- **User (Investor)**: The human actor who interacts with the web application to view predictions and decision support.
- **Market Data API**: The source providing historical and real time financial data.
- **News Source**: The source providing unstructured text data.

Use Cases:

- **View Dashboard**: The user accesses the main interface to see market trends and model outputs in a visualised way.
- **Generate Prediction**: The system processes data to see asset price movements.
- **Receive Recommendation**: The system provides a final action (BUY/SELL/HOLD) based on the model's output.

Use Case Diagram

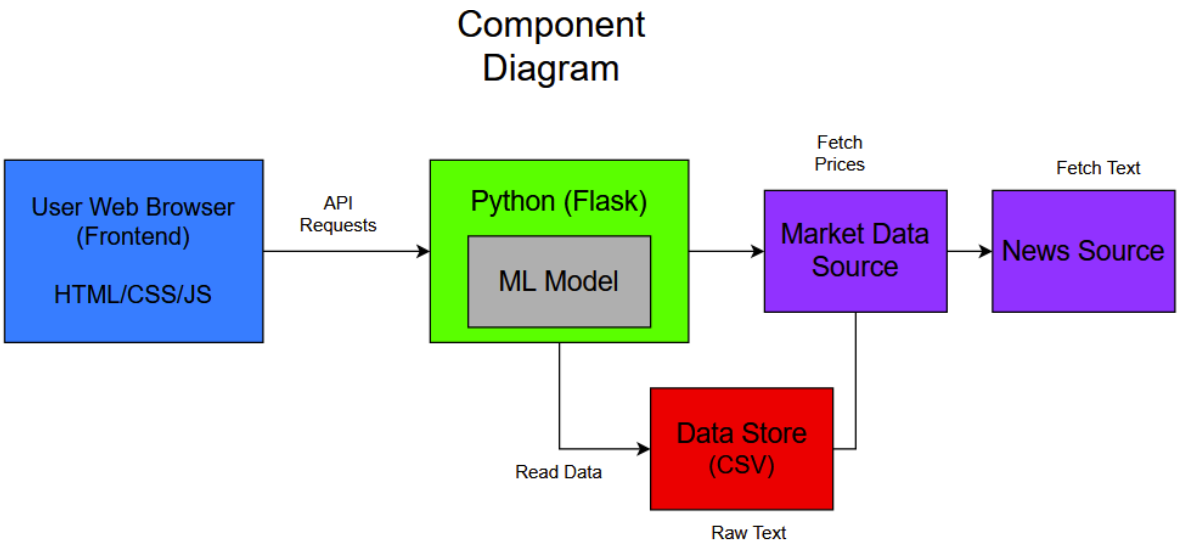


System Architecture (Component Diagram)

The Component Diagram shows the organisation of the systems structure. It shows how the project is split into modules to separate the responsibilities of data collection, machine learning processes and the user interface.

Components:

- **Data Preprocessing**: Python scripts used to fetch raw data from external APIs and perform cleaning.
- **Machine Learning Engine (Backend)**: The component hosted on a Python server (e.g. Flask). It has:
    - **Prediction Model**: Trained LSTM model.
    - **News Analyser**: The library that can look through text data.
- **Web Application (Frontend)**: The client side interface built with HTML, CSS and Javascript. Links to backend using RESTful API calls to display charts and recommendations.
- **Data Store**: A local storage solution (CSV) used to cache historical data and logged predictions for testing.

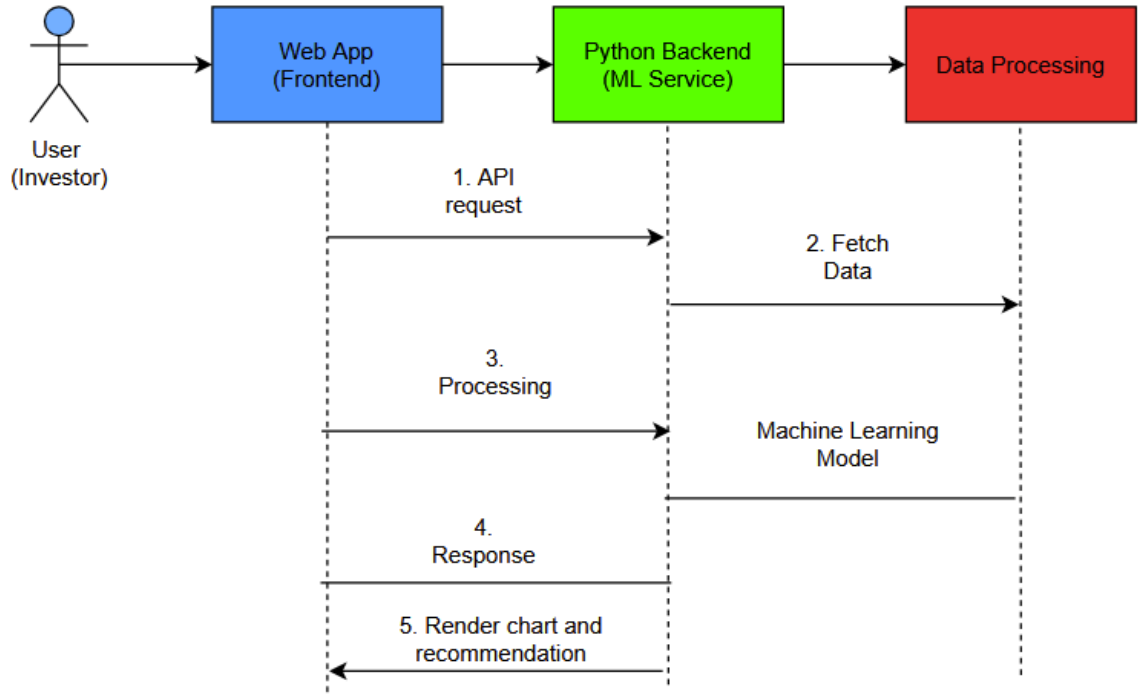


Interaction Flow (Sequence Diagram)

The sequence Diagram shows the behaviour of the system during a scenario where a user requests a **decision support analysis**. It maps the flow of messages between frontend, backend and data processing layers.

Workflow Description:

1. **Request**: The user initiates a request from the Web App Interface (e.g. by selecting 'Run Analysis').
2. **Data Fetch**: The Web App sends an API request to the Python Backend. The Backend starts the module that looks at the data to fetch the latest available market prices and news.
3. **Processing**: The data is fed into the Machine Learning Model.
4. **Response**: The model returns a prediction (e.g. 'Price up 2%') and how sure it is. The Backend uses the Decision Logic (how risky) and returns the final BUY signal to the Web App.
5. **Display**: The Web App renders the prediction chart and recommendation for the User.



Implemented Backend Logic and Evaluation

The backend has been extended beyond the initial design to include structured preprocessing, signal generation, and performance evaluation modules.

This area has four main components:

- Feature Engineering Module
- Prediction Module
- Backtesting Engine
- Benchmark Strategy Function

Feature Engineering Module

The feature engineering module transforms raw historical price data into structured inputs for the prediction engine.

Features include:

- Percentage returns
- Short term moving average
- Long term moving average
- Removal of incomplete rows

This makes sure there is consistent input format and reduces noise in the prediction process.

The separation of preprocessing logic improves maintainability and allows added indicators to be added without changing the prediction engine.

Prediction Module

The prediction module generates BUY, SELL, or HOLD signals based on processed features.

Currently it uses moving average comparison:

- If short term average exceeds long term average, the system outputs BUY
- If the short term average is below long term average, the system outputs SELL.
- Otherwise, the system outputs HOLD

The module also returns a confidence score

Separating this logic allows the future addition of more advanced machine learning models without affecting other parts.

Backtesting Engine

The backtesting engine evaluates the trading logic on historical data

The system simulates:

- Initial capital allocation
- Asset purchase based on signal
- Asset sale based on signal
- Final portfolio valuation

This allows performance measurement over a selected time period.

The engine calculates the final portfolio value after applying the strategy across historical data

This validates whether the prediction logic produces better results than a passive strategy.

Benchmark Strategy

A buy and hold benchmark function has been implemented to provide comparison

The system:

- Invests full capital at the start
- Holds the asset until the end of the period
- Calculates a final portfolio value

This creates a baseline performance metric.

The AI strategy is compared against this benchmark to determine added value.

Machine Learning Approach / Datasets

Initial Research

Our client presented us with two options on how to approach this project, machine learning or optimization.

Optimization focuses on finding the best possible solution given a set of constraints and objectives. This approach does not learn from the given data , instead searches for the best outcome based on a defined rule. Meaning optimization alone cannot predict stock prices as it does not learn from patterns within the historical data which we as a group would provide through researched datasets (mentioned in next section).

However machine learning models learn pattern recognition from historical data. Making generalisations from provided data allows them to predict new and yet to occur movements in stock price. As a result of this research we decided to choose the machine learning approach as it allowed us to focus on pattern recognition and prediction. Predicting movement in stock prices as a result of the provided historical data.

Datasets

Following our decision to use machine learning we then had to research appropriate datasets so that our intended system produced accurate and reliable predictions. To do this we used the website Kaggle which holds many datasets relating to stock price. First of all we picked out several datasets relating to cryptocurrency and the american stock market and screened each through google gemini allowing us to see which are the highest quality in both their accuracy of data and how that data is arranged to allow us to implement it in an ai model. As a result of this screening we decided on four datasets which were all of an excellent standard. Meaning our predictions would be based on extremely accurate data collected over several decades.

API Selection

**ML - Machine Leaning

Purpose

During this project we intend to use an API as an interface between the ML model and the user. The ML model will make the predictions and the chosen API will deliver these predictions to the user.

Flask

Our client recommended we use the Flask API. This is because flask is very easy to integrate with ML models as it is designed in python, which is also what most ML frameworks and data libraries are designed in. Meaning there is no need for cross language rappers.

Flask is also very lightweight , fitting perfectly with our project as stock prediction services are usually small, stateless and single purpose. Flask allows us to use no unnecessary architecture as it provides only what we need , Routes - Logic - Response.

Flask's roles in this project will be to

- Serve predictions via HTTP (GET/POST requests)
- Load and run the trained ML model
- Preprocess incoming data
- Return results in JSON
- Host endpoints

Targeted User Interface

React

For our User Interface we are using React. React is very Ideal when using real time charts, predictions , signal alerts and portfolio views. It is also very easy to integrate with our intended API Flask. React also produces very professional looking user interfaces. Making sure our project is perceived appropriately by users. React also uses the languages Javascript and HTML which all of our team members have used before during their first year projects. Meaning adjusting to a new coding language is not much of a problem.

Intended Look