Stock Genie

Submitted by	Qandeel Sajad
	Abdullah Irfan
	Tuba Ali
Roll Numbers	0100-BSCS-21
	0118-BSCS-21
	0049-BSCS-21
Session	2021–2025
Supervised by	Muhammad Hafeez

BS(HONS)
IN
COMPUTER SCIENCE

DEPARTMENT OF COMPUTER SCIENCE
GC UNIVERSITY LAHORE

Teachers Evaluation System

Submitted to GC University Lahore in partial fulfillment of the requirements for the award of degree of

BS(HONS)
IN
COMPUTER SCIENCE

Submitted by	Qandeel Sajad Muhammad Abdullah Tuba Ali
Roll Numbers	0100-BSCS-21 0049-BSCS-21 0118-BSCS-21
Session	2021–2025
Supervised by	Muhammad Hafeez

DEPARTMENT OF COMPUTER SCIENCE

GC UNIVERSITY LAHORE

Declaration

We, Qandeel Sajad, Muhammad Abdullah, Tuba Ali student of **BS(Hons)** in the subject of **Computer Science** session **2021-2025**, hereby declare that the matter printed in this thesis titled, **Stock Genie** is my own work and has not been printed, published and submitted as research work, thesis or publication in any form in any University, Research Institution etc in Pakistan or abroad.

Date: _____ **Signatures of Deponent**

Research Completion Certificate

It is certified that the research work contained in this thesis titled **Stock Genie** has been carried out by **Qandeel Sajad, Abdullah Irfan and Tuba Ali** Roll. No **0100-BSCS-21, 0049-BSCS-21 and 0118-BSCS-21** under my supervision.

Muhammad Hafeez
Department of Computer Science
GC University Lahore

Date: _____

Submitted Through

Prof. Dr. Muhammad Waqas Anwar	**Controller of Examination**
Chairperson	**GC University Lahore**
Department of Computer Science	
GC University Lahore	

Acknowledgements

I am grateful to the Almighty **Allah**, Who blessed me with health, wisdom, knowledge, thoughts and opportunity to make some contribution in the form of present effort. I offer my humblest thanks from the deepest core of my heart to the **Holy Prophet Muhammad (Peace be upon him)**, the most perfect and excelled among and ever born on the surface of earth.

The research work embodied in this dissertation was accomplished under the able guidance and affectionate supervision of **Muhammad Hafeez**, Assistant Professor, GC University, Lahore. I will always remember his moral encouragement, skillful guidance, positive criticism and valuable advice throughout the course of my study.

I also express my feelings of love and respect for my beloved parents that are the real asset of my life and gave me confidence and my friends also deserve my thanks for their loving encouragement and prayers for my success.

Dedication

My research work is dedicated to my family and my honorable teacher Muhammad Hafee who encouraged and helped me to complete my research in the area of formal modeling and formal verification related to computer science.

Abstract

Stock Genie is an AI-powered web application designed for the Pakistan Stock Exchange (PSX) to assist investors with intelligent Buy and Sell signals. It analyzes six months of historical stock data, computing technical indicators like RSI, SMA-30, and price change, which are then processed by an ensemble of machine learning models—Random Forest, SVM, and Logistic Regression—for prediction through majority voting. The platform also integrates sentiment analysis by scraping financial news from Dawn and Brecorder, classifying market tone using NLP techniques. Additional features include a gainers/losers module and interactive visualizations using Chart.js. Built with Python, Flask, scikit-learn, and BeautifulSoup, Stock Genie delivers a lightweight, modular, and user-friendly solution tailored for PSX investors.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction

The Pakistan Stock Exchange (PSX) can be considered the backbone of the capital market in Pakistan, where thousands of investors, traders, and institutions are involved in the daily routine of purchasing and selling listed securities. Although retail investors form the backbone of the economy, they often face challenges in accessing impactful analytical tools and insights needed to make informed decisions. Compared to markets worldwide, where efficient data platforms and high-quality advisory services are found, the PSX lacks locally oriented solutions that combine technical indicators and data-based forecasts. Retail investors in Pakistan often rely on gut feelings, rumors, or outdated strategies because data visualization tools, market forecasting, and guidance on interpreting market sentiment are not readily available. This tends to result in speculative trading, which can lead to significant losses or missed opportunities. It is due to this gap that our project comes in to help empower PSX investors with a contemporary (and, to boot) AI decision support system that incorporates two potent elements: **technical analysis** and **market sentiment analysis**. StockGenie is an online, web-based service that allows access to real-time information and predictive trading signals, as well as interactive visualization of a stock from a PSX company, enabling analysis of

the same stock in various ways. Most importantly, the idea is to allow investors to make more intelligent, quicker, and more informed trading decisions rather than simply automating analysis. StockGenie offers the impossible to end users through the smarts of its backend and the ease of its uncluttered front facade; it delivers contemporary financial smarts to a historically underserved local investor community.

1.1.1 Background of the Problem

Stock Genie is built to address critical challenges faced by investors in the Pakistan Stock Exchange (PSX) by combining three intelligent modules: technical analysis, machine learning-based prediction, and sentiment analysis. While global markets have long adopted AI-driven trading platforms, PSX investors—especially retail participants—are still limited to basic tools that lack predictive intelligence and contextual awareness. The absence of localized, accessible, and automated decision-support systems results in poor investment outcomes and a lack of trust in data-driven trading strategies. The key problems faced by investors in this context include:

- **Lack of AI-powered local platforms** tailored to PSX for predictive analysis and signal generation.
- **No integration of local news** sentiment into decision-making tools, despite its strong influence on stock behavior.
- **Dependence on raw or static data** without actionable insights or interpretation.
- **High cost and complexity of global platforms** like Bloomberg or TradingView, which are not optimized for the Pakistani market.
- **Emotion-driven and speculative decisions** due to unreliable tips, rumors, and the absence of credible automated analysis.

Stock Genie was envisioned to solve these issues through a modular, AI-enhanced architecture that empowers investors with accurate signals, contextual market sentiment, and interactive visual insights.

1.1.2 Background of the System

Retail and institutional traders around the world are utilizing tools such as TradingView, Bloomberg Terminal, and MetaStock to access real-time data, complex charts, AI signal generators, and even social sentiment reading programs. However, they are usually subscription-based systems or prohibitively costly and/or are too expensive in Western markets to be effective, or rather entirely irrelevant for stock traders in Pakistan. The PSX, however, continues to lack localized, AI-based services that portray real-time signals and market interpretations in a contextualized manner. What investors in Pakistan tend to do is the following:

- Manual chart interpretation using basic tools that lack analytical intelligence or real-time insights.
- Poor quality of information provided by communities of brokers or on social media.
- Poor attempts at trial and error with limited historical knowledge.

In addition, technical indicators such as RSI or SMA are mathematically efficient measures but fail to pick up the emotional and psychological drivers behind the market (a policy change, political rumblings, or macroeconomic changes), all of which are reflected in the content of the news. Thus, relying solely on price trends is not sufficient.

StockGenie was designed to eliminate these shortcomings by providing a more holistic approach that understands both the price and sentiments, using an AI-integrated tool. The system presents a more intelligent solution by directly gleaning information from PSX (and Pakistani financial news sites) and augmenting

it with robust machine learning models to provide localized intelligence that is relevant, timely, and actionable.

StockGenie is specifically designed to fit the market dynamics of Pakistan, unlike other generic finance tools. It permits simple enlargement and enhancement of its backend (e.g., adding new indicators, models, or sentiment providers) and guarantees ease of accessibility to both new and experienced investors. Through this, the system is introducing a greatly needed innovation into the Pakistani digital finance ecosystem.

1.2 Objectives of the System

The primary objective of Stock Genie is to bridge the gap between Pakistani stock investors and intelligent financial tools by offering a localized, AI-powered platform tailored for the PSX. The system is designed to solve key challenges faced by both novice and experienced traders by combining machine learning, technical indicators, and sentiment analysis in a unified decision-support environment. The specific objectives of the system are:

- **Provide Data-Driven Trading Signals:**
  To reduce dependence on speculation and guesswork, the system delivers Buy, Sell, or Neutral signals based on historical stock data processed through trained ML models (Random Forest, SVM, Logistic Regression).
- **Offer Interpretable Technical Insights:**
  To eliminate the need for manual chart reading using outdated software, the platform automatically computes and displays key technical indicators such as RSI, SMA-30, and price change in a visual, user-friendly format.
- **Incorporate Market Sentiment from Local Sources:** To overcome the limitations of emotionless numerical indicators, the system integrates real-time sentiment analysis by scraping news headlines from trusted Pakistani financial portals, such as *Dawn* and *Brecorder*.

- **Make Advanced Tools Accessible to All Users:** To address the inaccessibility of expensive global platforms, Stock Genie provides a lightweight, web-based interface that is free to use and optimized for local market needs.
- **Support Context-Aware Decision Making:** By combining technical analysis with news-driven sentiment, the system allows investors to consider both price trends and external market-moving events such as political shifts or economic announcements.
- **Enable Real-Time Analysis and Visualization:** To facilitate rapid decision-making, the platform presents all outputs—including predictions, indicators, and sentiment—in an interactive dashboard using dynamic charts and visual cues.

These objectives ensure that Stock Genie serves not only as a forecasting tool but also as an educational and practical resource for investors looking to make smarter, informed decisions in the PSX environment.

1.3 Significance of the System

StockGenie is poised to become a valuable tool, empowering retail investors to conduct advanced stock analysis without requiring advanced or technical knowledge. It makes decisions easier by integrating machine learning forecasts with live sentiment reading, allowing its users to stop trading with an emotional or speculative approach. The system also utilizes headlines from local financial media to ensure that its intelligence is grounded in the real scenario of the Pakistani market, both economically and politically.

Besides automating the calculation of technical indicators such as RSI and SMA, StockGenie provides interactive renderings to its users, thereby avoiding the time and effort required for manually creating a chart. It is also educational, as a person with no prior experience may learn how technical indicators and market sentiment affect stock prices, making it both advisory and educational.

1.4 Literature Review

Numerous studies have demonstrated the power of applying machine learning algorithms to stock market prediction. The indicators that Logistic Regression, SVM, and Random Forest models were found to be superior to are RSI, SMA, and price change. Patel et al. (2015) and Kara et al. (2011) emphasized the predictive capabilities of this type of model in new markets in their research.

Financial forecasting has also been found helpful in this process of sentiment analysis. Bollen et al. (2011) found a significant correlation between stock index trends and people's moods. However, Loughran and McDonald (2011) proposed a financial sentiment dictionary to enhance the approach's classification accuracy in the field of economics. These books reaffirmed the fact that the sentiment on the market, as prevalent in the news and media, can significantly influence the behavior of stocks.

A combination of technical indicators and sentiment inputs, such as hybrids, has proven to be more effective. Nguyen et al. (2015) and Akita et al. (2016) have developed systems that combine the two types of data to enhance prediction accuracy. Nonetheless, the majority of these tools are targeted at international markets and do not focus on local trading, such as the Pakistan Stock Exchange (PSX).

Currently, local investors lack suitable platforms for conducting PSX-specific technical and sentiment analysis. StockGenie fills this gap through locally scraped financial data and headlines, published indicators, the implementation of ensemble machine learning models, and real-time buy/sell signals, along with sentiment analysis. A convenient web interface powers all of this.

1.5 Project Scope

The web-based application StockGenie is an intelligent and scalable solution created to help Pakistani investors make data-driven decisions in the stock market.

It focuses on PSX-listed firms, combining technical analysis and sentiment analysis, and utilizes natural language processing to generate investment alerts. The system automatically fetches historical stock price data, calculates common-sense technical indicators, and provides predictions tailored to the needs of various users, including entry-level investors, finance students, and established traders.

It delivers the correct information at the right time through pre-trained machine-learning models. At the same time, it also examines the sentiment of local financial news and determines the overall market's mood. StockGenie aims to provide intuitive stock insights in a timely, intelligent, and localized format.

1.6 Project Description

1.6.1 Project Perspective

The architecture of StockGenie involves three basic functional modules that carry out a unique task in the analysis pipeline:

- **Technical Analysis Module:** This module is used to calculate essential indicators, such as the Relative Strength Index (RSI), Simple Moving Average (SMA_30), and Percentage Price Change. These measures reflect stock strength, trend, and volatility.
- **Ensemble Prediction Module:** Utilizes three machine learning models trained on PSX data, including Logistic Regression, Support Vector Machine (SVM), and Random Forest. These models determine the behavior of a stock as Buy, Sell, or Neutral based on its technical characteristics.
- **Sentiment Analysis Module:** Scrapes up-to-date headlines from well-known Pakistani news websites, such as Dawn and Brecorder. They are fed into a sentiment classifier, which labels each headline according to its sentiment category (Positive, Negative, or Neutral) to measure the market's sentiment with respect to the headline.

1.6.2 Project Functionality

StockGenie can assist with the following end-user functionality:

- **Ticker Input:** To find any stock listed on PSX, the user can query it by its symbol (e.g. OGDC, HBL, LUCK).
- **Technical Indicator Calculation:**The calculations of RSI, SMA_30, and the percent change in price over the past six months are automatically loaded and calculated.
- **ML-Based Signal Prediction:** Indicators identified are fed into a group of models to provide a proposed trading signal.
- **Financial News Integration:** Local news is retrieved from Pakistani business news websites.
- **Sentiment Classification:** Applying NLP to headlines would enable the establishment of market sentiment (Positive, Negative, Neutral).
- **Visual Reporting:** The findings are also presented in interactive charts, as well as structured sections on the front end, allowing for easy analysis of the results.

1.6.3 Users and Characteristics

User Type	**Description**
Retail Investors	Small time investors that lacks the financial or technical information on the market.
Traders	Active traders that follow short-term possibilities and cues.
Students	Academic students interested in the field of financial analytics, stock behaviour, and indicators.

Table 1.1: Charecteristics of users

1.6.4 Operating Environment

StockGenie can be installed to be used by any modern browser and can be installed on a local server, or a hosting cloud environment. It uses:

- **Backend:** Web Python 3.10+ with Flask route processing and logical partitioning.
- **Frontend:** HTML, CSS, JavaScript and Chart.js or Plotly.js to render responsive charts.
- **Data and News Parsing:** Python libraries, such as the requests and BeautifulSoup to scrape PSX and news websites.
- **ML Runtime:** Prior-trained models through joblib on the scikit-learn framework.
- **Deployment:** Deployable with PythonAnywhere, Render, Vercel (through API endpoint connections), and localhost.

Chapter 2

Requirement Specification

2.1 Functional Requirements

- **FR1: Stock Input**

  The user is allowed to enter any valid PSX-listed stock ticker (e.g., OGDC, HBL) into the system.

- **FR2: Technical Indicator Computation**

  The system automatically calculates:

  - Relative Strength Index (RSI)
  - Simple Moving Average (SMA_30)
  - Percentage Price Change using the past 6 months of stock data.

- **FR3: Machine Learning Prediction**

  The calculated indicators are passed to three pre-trained models:

  - Logistic Regression
  - Support Vector Machine (SVM)
  - Random Forest

  Each model returns an individual prediction (Buy, Sell, or Neutral).

- **FR4: Historical Data Scraper**

  The system includes a scraper module that performs the following:

  - Sends HTTP requests to the PSX historical data portal.
  - Retrieves and parses stock data for the last 6 months.
  - Structures data into a usable format for indicator calculation.

  This module automates the collection and preprocessing of technical data.

- **FR5: Ensemble Voting**

  Three of the models present weighted results, and a majority vote determines the final prediction.

- **FR6: News Scraper**

  Scraps recent news of financial headlines on Pakistani sources such as including:

  - Dawn News
  - Brecorder
  - Pakistan Today

- **FR7: Sentiment Classification**

  The listed headlines obtained with the help of scraping are passed through an NLP model to classify as a whole, how does market sentiment look like:

  - Positive
  - Neutral
  - Negative

- **FR8: Data Visualization**

  The frontend displays:

  - Interactive 6-month price chart.
  - Computed indicators

  - Predicted signal
  - Market sentiment
  - List of headlines with links.

2.2 Non-Functional Requirements

- **Performance**
  - The system's response to user input requires no more than 5 seconds.
  - The execution of prediction models and scrapers should be relatively fast (2-3 seconds).
  - There must be the ability to display charts in less than 1 second after computation.
- **Usability**
  - The navigation system must be straightforward, and non-technical personnel should be able to operate it easily and comfortably.
  - All the findings are logically divided and should be marked according to their type (technical, sentiment, prediction).
- **Compatibility**
  - It should be cross-browser compatible with Chrome, Firefox, and Edge, in particular.
  - It should be available on both desktops and phones.
- **Scalability**

  The architecture should support:
  - The introduction of new stock indicators.
  - Training additional models.

  - Adding the incorporation of additional news or APIs.

- **Maintainability**

  - Its codebase should be modular, utilizing Flask Blueprints and reusable modules to enable growth and maintainability.
  - Any new model, route, or scraping logic should be convenient to apply and not require any changes to the entire system.

2.2.1 Performance Requirements

To ensure a smooth, responsive, and enjoyable experience, StockGenie is developed to meet rigorous performance thresholds in all critical system functionalities. These limits ensure that there is minimal delay between the time a user makes a query and the time it takes to receive reasonable recommendations and graphics feedback.

- **API Performance**

  The backend endpoints of the system, as well as the endpoints involved in retrieving stock data from the PSX and producing model predictions, must respond within 2-3 seconds. This ensures that users are not at the receiving end of delays related to requesting analysis of any stock. Multi-threading is performed efficiently, and data pipelines are optimized to handle concurrent user loads in real time.

- **Chart Rendering Performance**

  After the backend finishes preparing the data, the frontend should be able to display technical charts (price trend, SMA, RSI, etc.) in a 1-second time frame. This method is used to deploy a convenient user interface without the need to compensate between links, as each component offers smooth transitions without lag. It ensures interactive and snappy rendering on any device by utilizing lightweight charting frameworks, such as chart.js or Plotly.

- **News & Sentiment Processing**

  To quickly gather the latest headlines and apply sentiment classification to every chosen source of financial news (e.g., Brecorder, Dawn), **the time limit for each source should be 2 seconds**. This includes:

  - Sending HTTP requests.
  - Parsing HTML for headlines.
  - The use of sentiment models on the text recovered.
  - The rapid rotation ensures that the mood on the board accurately reflects the latest market developments and can be factored into investment decisions in real time.

- **Model Execution Time**

  To support the fast ensemble voting, the predictions made by each of these machine learning models (Logistic Regression, SVM, and Random Forest) must run and make predictions **within 500 milliseconds**. Model files are loaded into memory as a precaution against cold-start.

- **Total System Response Time** Under normal conditions, the end-to-end time (including input into stock and the eventual output) should not exceed 5 seconds. These are data scraping, preprocessing, prediction, sentiment analysis, and visualization. Such performance criteria are vital in maintaining users' trust and engagement, as they will be dealing with these applications in a financial context where timing and responsiveness will directly impact their decision-making.

Chapter 3

Project Design

3.1 Methodology

The development of the StockGenie system adhered to a formal and disciplined process to ensure clarity, traceability, and high-quality deliverables. Due to the technical complexity of the project — integrating real-time data scraping, financial analysis, natural language processing, and machine learning — a methodology was required that would allow for systemic planning and provide checkpoints at every critical stage. The waterfall Model was utilized due to its linear and phase-based workflow, which is conducive to the sequential development needs of this project.

3.1.1 Waterfall Method

The Waterfall Model is among the earliest and most widely used methodologies for software development, and it is particularly effective for projects with clearly defined requirements and limited scope for mid-project changes. In the StockGenie project, the Waterfall approach provided a robust framework where each phase was completed and validated before the next phase began. This was particularly required for the complex modules, such as stock data processing, model-based

prediction logic, and frontend integration. The following phases are followed in order:

- Software Requirements
- Analysis
- Design
- Coding
- Testing
- Maintenance

These stages ensure a sequential development cycle, where each phase must be completed before moving on to the next. The phases involved in the Waterfall Model used for this project are as follows:

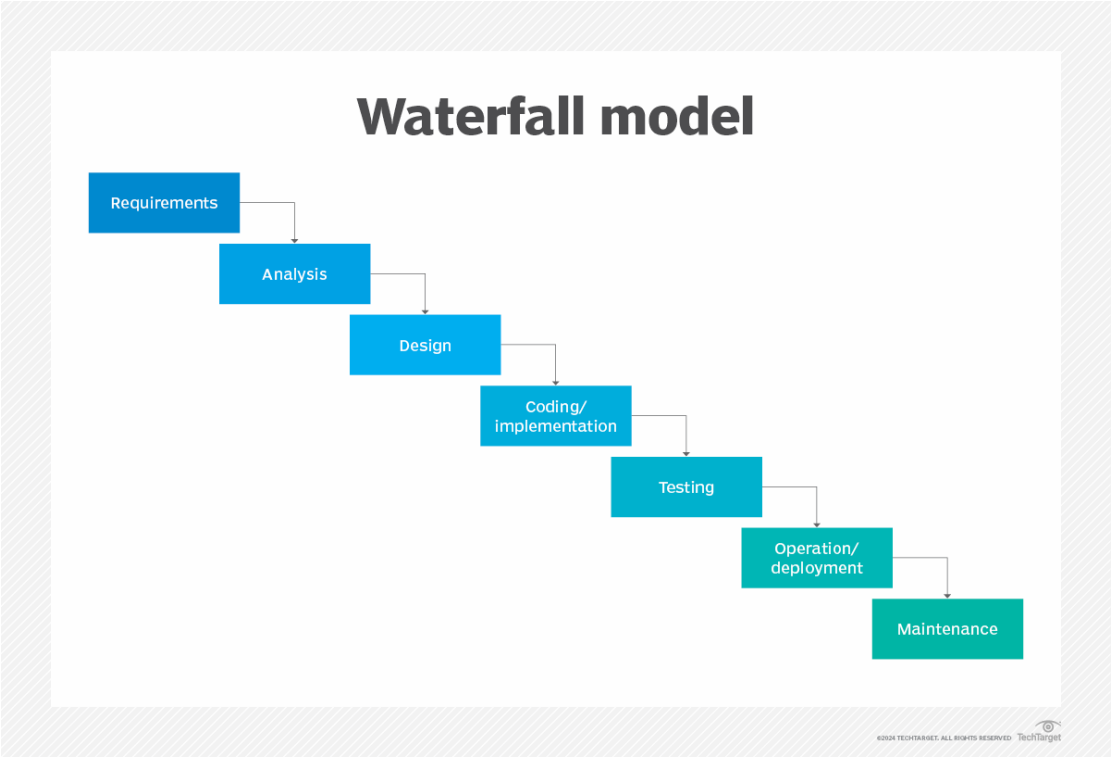

FIGURE 3.1: Waterfall Methodology

- **Phase 1: Requirements**
  - Determine the needs
  - Get information
  - Testing performance
  - Prepare the document
- **Phase 2: Analysis**
  - Analyze requirements
  - Analyze the needs of the data
- **Phase 3: Design**
  - Analyze results
  - Analyze reports
  - Gather requirements
  - Prepare for the input
  - Data analysis should be performed first
- **Phase 4: Coding**
  - Develop backend
  - Develop frontend
- **Phase 5: Testing**
  - Perform quality assurance testing
  - Test and validate

- **Phase 6: Maintenance**
  - Addressing user feedback
  - Fixing bugs
  - Providing ongoing support to ensure the software continues to function effectively
- **Phase 7: Deployment (Addition)**
  - Deploy app
  - Integrate app

3.2 Architectural Overview

3.2.1 Introduction

StockGenie's system architectural design is based on modularity, scalability, and maintainability. The system is structured into separate layers to separate the functionality and responsibilities of various system elements, including data acquisition, machine learning logic, sentiment analysis, and the end-user interface. The layered design facilitates loose coupling and high cohesion, allowing each system component to be developed, tested, and improved individually. The architectural design also enhances fault isolation, facilitates easy scalability, and enables the addition of future improvements with ease. Whether to enhance model accuracy or implement an additional source of data, the system is versatile enough to evolve, maintaining both system performance and stability.

3.2.2 High-Level Architecture

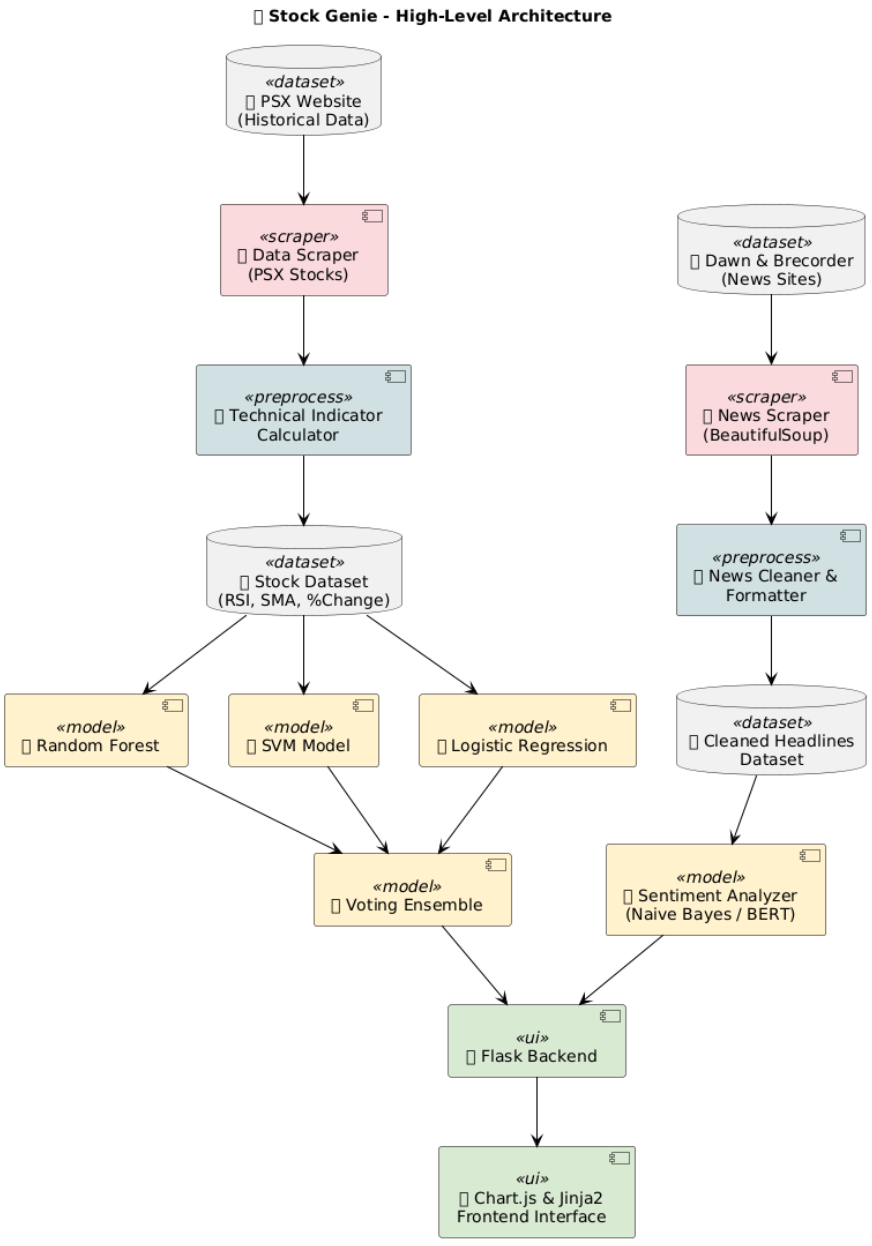

FIGURE 3.2: High-Level Architecture

3.2.2.1 Presentation Layer

The **Presentation Layer** is the user-facing side of the StockGenie application — essentially, it's what users see and interact with. Here, users can input PSX stock tickers, such as HBL, OGDC, or BOP, and trigger analysis functions with a simple button click. Once submitted, users receive real-time outputs that include

technical indicators, machine learning predictions, and overall market sentiment. These results are dynamically rendered using **Chart.js** and embedded **HTML blocks**, presenting six-month stock trends with visual markers, including the RSI (Relative Strength Index) and SMA (Simple Moving Average). These charts help users quickly grasp market movements and trends. The front end is built with **HTML** and **CSS**, and it utilizes **Jinja2 templates** within the **Flask framework** to render content from the backend. This ensures a responsive and intuitive layout that's easy to navigate, whether you're a beginner or a seasoned investor. Although this layer doesn't handle any heavy data processing or business logic, it plays a vital role in making the data digestible and interactive.

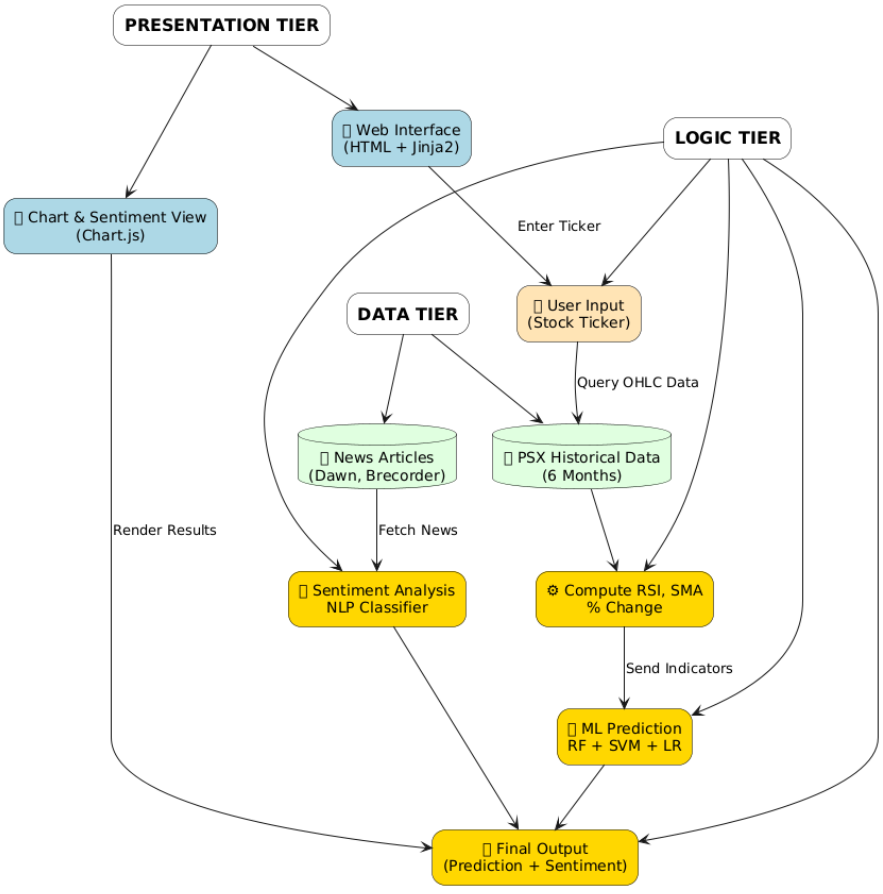

FIGURE 3.3: Presentation Layer

3.2.2.2 Application Layer

The **Application Layer** is the engine room of the system. It acts as the central coordinator, processing user inputs, handling backend logic, and managing communication between the different layers of the application. Built on **Flask**, this layer utilizes **Flask Blueprints** to organize core functionalities into clean, modular sections, such as technical analysis and sentiment analysis. It's responsible for computing technical indicators (such as RSI, SMA_30, and price change), running pre-trained machine learning models, scraping financial news, and classifying sentiment. Think of it as the system's "controller" — it takes in user requests, runs the necessary computations, and passes the results back to the front end for visualization. This clear separation ensures users experience smooth, responsive interactions, even while complex tasks are handled quietly in the background.

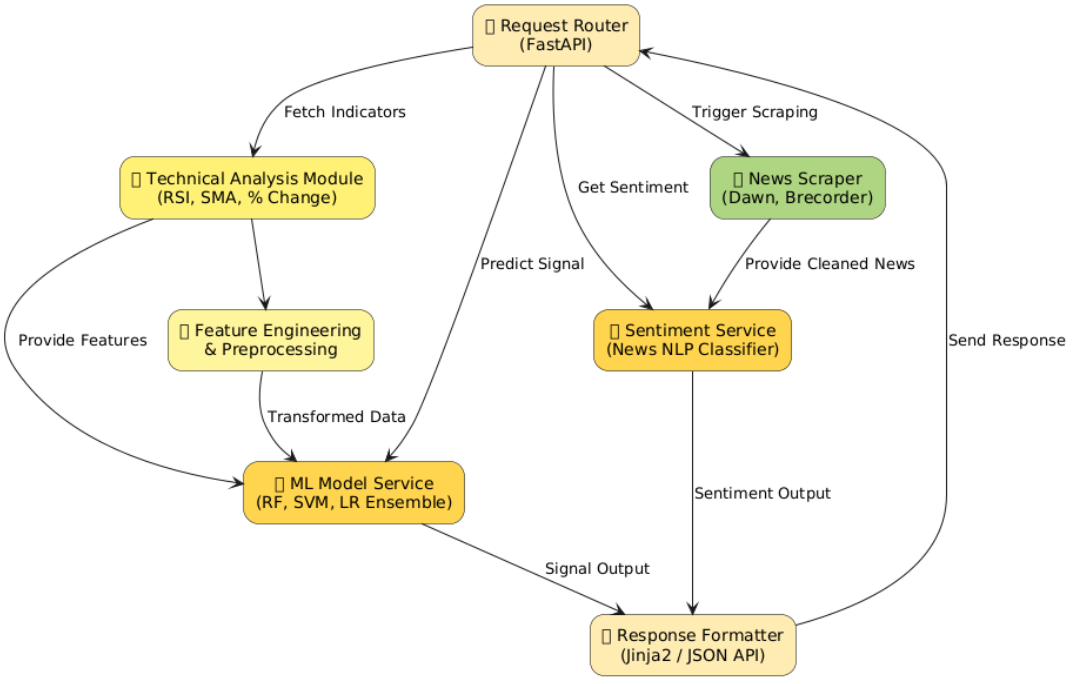

FIGURE 3.4: Application Layer

3.2.2.3 Integration Layer

The **Integration Layer** handles all interactions with external data sources. This includes real-time scraping of stock information directly from the **Pakistan Stock**

Exchange (PSX) website and news headlines from reputable financial sources, such as **Dawn** and **Brecorder**. Behind the scenes, libraries like **requests** and **BeautifulSoup** handle data extraction and HTML parsing. This layer also loads and applies machine learning models saved in **.joblib** format using **scikit-learn**. These models are then used to generate predictions based on user-supplied inputs. It also integrates **Natural Language Processing (NLP)** models to analyze and classify the sentiment behind financial headlines. Thanks to its modular architecture, parts of this layer — such as scrapers or ML models — can be updated independently without disrupting the entire system.

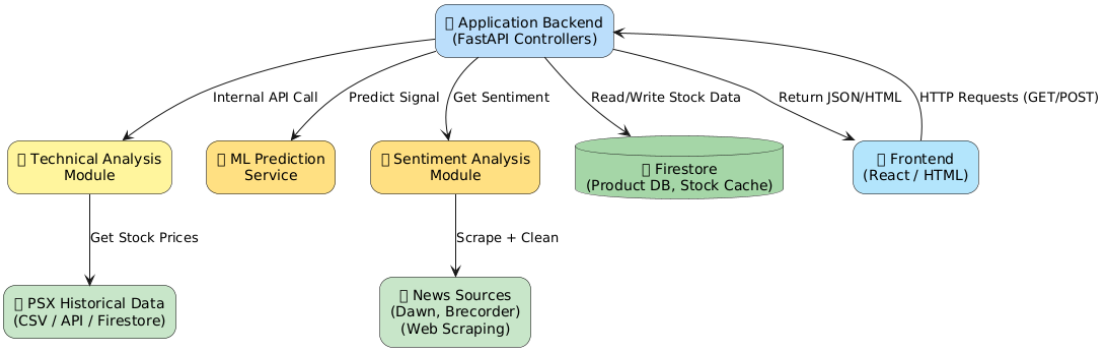

FIGURE 3.5: Integration Layer

3.2.2.4 Data Layer

The **Data Layer** is responsible for cleaning, structuring, and preparing all incoming data. Stock data pulled from PSX is formatted into **pandas DataFrames**, where missing values are handled, and date-time indices are correctly set. From there, calculations for RSI, SMA_30, and price change are performed. This layer also constructs feature vectors for the machine learning models. These are selected and normalized using libraries such as **NumPy** and **pandas**, which are known for their power in managing large and complex datasets. Once the data is fully processed, it's handed off to the Application Layer, which then triggers predictions and visualizations. The goal here is to keep data preprocessing separate from the system's core logic and front end, ensuring accuracy and consistency.

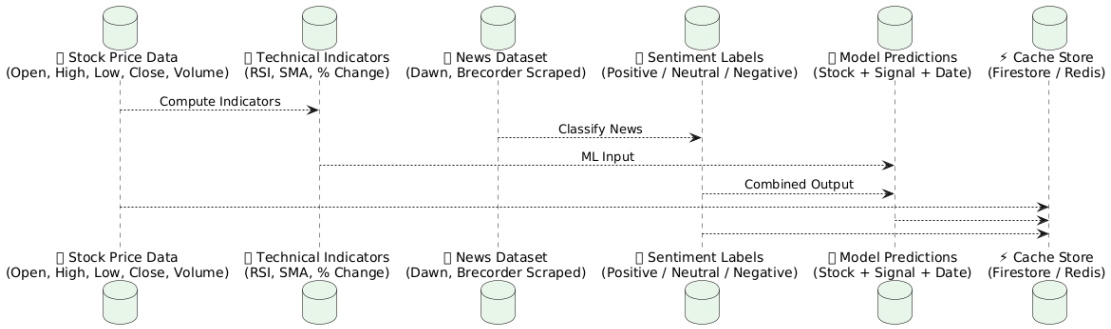

FIGURE 3.6: Data Layer

3.2.2.5 Security Layer

Security is a top priority in StockGenie. To maintain system integrity, the backend validates all user input, rejecting any invalid or suspicious stock symbols that could pose a threat.

Timeout settings in the scraping modules prevent endless loops if an external site goes down. When loading ML models, the app uses verified paths and secure joblib loading methods. The use of server-side rendering means sensitive logic and data are never exposed to the browser, reducing the risk of injection attacks or unauthorized access.

Though StockGenie currently doesn't support user accounts, plans for future enhancements include secure authentication, rate limiting, and API key access for controlled usage.

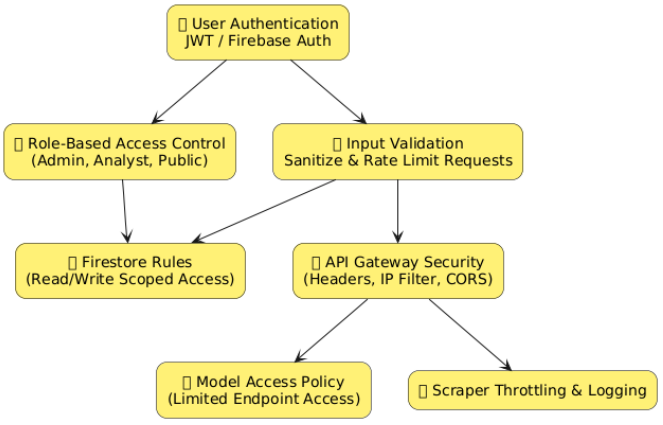

FIGURE 3.7: Security Layer

3.2.3 Key Features and Components

- The system fetches real-time historical stock data from PSX based on user input, ensuring up-to-date analysis.
- Technical indicators like RSI, SMA_30, and price change are computed automatically from the fetched data.
- Predictions are made using an ensemble of Logistic Regression, SVM, and Random Forest with majority voting logic.
- News headlines from local sources are scraped and analyzed for sentiment to reflect market tone around PSX stocks.
- Interactive charts display 6-month stock trends and indicators, improving data interpretation through visualization.
- Flask Blueprints structure the backend into separate modules for clean, scalable, and maintainable architecture.

3.2.4 Scalability and Performance

StockGenie is built with a modular architecture, allowing it to scale efficiently as the system grows. Key components, such as **data scraping**, **machine learning**, and **sentiment analysis**, are designed to function independently, making it easy to upgrade or enhance each part without disrupting the rest of the application. To boost performance, the platform utilizes **thread pools** that simultaneously fetch stock data and financial news, significantly reducing overall processing time. Its **flexible machine-learning pipeline** also makes it simple to add new models without modifying the existing structure. On the frontend, **lightweight visualizations** are employed to ensure fast loading times and smooth performance, even on devices with limited resources. Altogether, this thoughtful design delivers both **high performance** and **long-term adaptability**, making StockGenie a reliable and future-ready solution.

3.2.5 Conclusion

The architecture of StockGenie is carefully crafted to strike a balance between performance, modularity, and ease of use. By structuring the system into distinct layers ranging from the user interface to backend logic, data handling, integration, and security, the platform ensures long-term maintainability and seamless operation. Each layer functions independently yet works cohesively with the others, allowing the system to evolve without significant disruptions. With real-time stock analysis, AI-driven predictions, and localized news sentiment integration, StockGenie offers a solid, intelligent foundation for investors looking to navigate the PSX with data-backed confidence.

3.3 Design Description

StockGenie is designed using the **Model-View-Controller (MVC)** architecture. This widely adopted pattern separates the application into three key components: data processing (Model), user interface (View), and system control logic (Controller). This clear division makes the system easier to maintain, test, and scale as new features are added.

3.3.1 Model

The **Model** is the brain of the application. It takes care of all the heavy lifting behind the scenes — from scraping stock data and calculating technical indicators like RSI, SMA, and percentage price change to running machine learning predictions and analyzing sentiment in news headlines. Once this processing is complete, the Model prepares clean, structured results that the Controller can use to decide what to display to the user.

3.3.2 Viewer

The **Viewer** (or View) is what users see and interact with. Built using **HTML**, **CSS**, **Jinja2**, and **Chart.js**, it includes everything from input forms for stock tickers to interactive charts that display prediction results and sentiment analysis. The goal of the Viewer is to present complex data in a way that's simple, clean, and easy to understand — giving users an intuitive and engaging experience.

3.3.3 Controller

The **Controller** acts as the middleman between the user and the system's internal logic. When a user submits input (such as selecting a stock symbol), the Controller receives the request, sends it to the corresponding part of the Model for processing, and then passes the resulting data back to the Viewer. It ensures that everything flows smoothly, connecting user actions with the correct backend functions.

3.4 UML Diagrams

3.4.1 Use Case Diagram

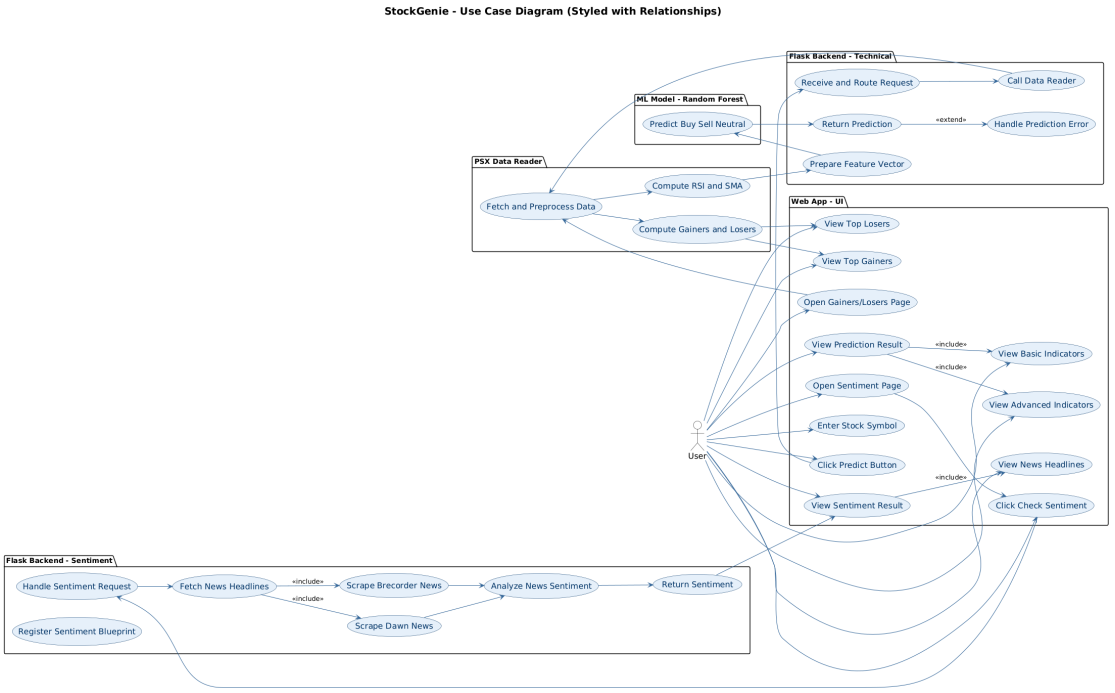

FIGURE 3.8: Use Case

3.4.2 Use Case Descriptions

3.4.2.1 Authentication Module

Use Case ID	**UC-01**
Use Case Name	User Authentication (Login & Signup)
Actors	User, Flask Backend
Description	Allows users to register and log in to access personalized features like saving predictions or tracking activity, Ensures that only authorized users access user-specific functions.
Precondition	User must visit the login or signup page.
Post Condition	A valid session or token is created upon successful login, allowing user access to personalized features.
Scenario	1. User visits the login/signup page. 2. For new users, signup form collects name, email, and password. 3. For existing users, login form collects email and password. 4. Backend validates credentials or stores new user securely. 5. If valid, user is redirected to dashboard with session active.
Exception	1. Invalid credentials “Incorrect email or password” message shown. 2. Duplicate email during signup prompt user to log in instead. 3. Backend error show retry or fallback message.
Priority	Medium (High if login-restricted features exist)
Frequency of Use	Moderate to High (depending on user-based features)

Table 3.1: Authentication Module UC

3.4.2.2 Technical Analysis Prediction

Use Case ID	**UC-02**
Use Case Name	Predict Buy/Sell Signal.
Actors	User
Description	User enters a PSX stock ticker and receives a prediction with supporting technical indicators.
Precondition	Precondition User must access the website and input a valid stock ticker.
Post Condition	Technical indicators and prediction are shown to the user.
Scenario	1. User enters a valid stock ticker. 2. System fetches historical data. 3. Technical indicators (RSI, SMA, % change) are calculated. 4. ML models make predictions. 5. Result is displayed.
Exception	1. User enters an invalid ticker system shows an error. 2. PSX data not available user is notified. 3. If model fails, a fallback or error message is shown.
Priority	High
Frequency of Use	High

Table 3.2: Technical Analysis Prediction UC

3.4.2.3 Sentiment Analysis

Use Case ID	**UC-03**
Use Case Name	Analyze Market Sentiment
Actors	User
Description	System fetches financial news and classifies sentiment as Positive, Neutral, or Negative.
Precondition	User opens the sentiment analysis section.
Post Condition	Headline list and sentiment result are displayed.
Scenario	1. User clicks "Check Sentiment". 2. System scrapes headlines from Dawn & Brecorder. 3. Headlines are passed to NLP model. 4. Sentiment is classified and displayed.
Exception	1. Network error during scraping user is informed. 2. No headlines fetched message is shown. 3. Sentiment model fails default fallback message shown.
Priority	Medium
Frequency of Use	Moderate

Table 3.3: Sentimental Analysis UC

3.4.2.4 Gainers and Losers Viewer

Use Case ID	**UC-04**
Use Case Name	View Top Gainers and Losers
Actors	User
Description	User views a list of stocks with the highest and lowest performance based on recent price movement.
Precondition	User clicks the "Gainers/Losers" tab.
Post Condition	Top gainer and loser stocks are shown with price change percentages.
Scenario	1. User navigates to Gainers/Losers section. 2. System fetches and processes stock data. 3. Stocks are sorted by gain/loss. 4. Lists are displayed to the user.
Exception	1. Stock data is unavailable or corrupt error shown. 2. Internet issue display fallback message. 3. Sorting fails empty list with retry option.
Priority	Medium
Frequency of Use	Moderate

Table 3.4: Gainers & Losers UC

3.4.2.5 Random Forest Model Prediction

Use Case ID	**UC-05**
Use Case Name	Predict Signal using Random Forest
Actors	Flask Backend, ML Engine
Description	The backend passes processed stock features to the Random Forest model to predict a Buy, Sell, or Neutral signal.
Precondition	Feature vector must be prepared from technical indicators.
Post Condition	Prediction result is returned by the model.
Scenario	1. System computes RSI, SMA_30, and price change. 2. Features are fed into the Random Forest classifier. 3. Model outputs one of three signals. 4. Signal is passed back to the controller.
Exception	1. Input vector is incomplete raise error. 2. Model fails to load trigger fallback logic. 3. Model outputs unexpected class log and return "Neutral".
Priority	High
Frequency of Use	High (used for every prediction request)

Table 3.5: Random Forest Model Prediction UC

3.4.2.6 Support Vector Machine (SVM) Model Prediction

Use Case ID	**UC-06**
Use Case Name	Predict Signal using SVM
Actors	Flask Backend, ML Engine
Description	The SVM model predicts the trading signal using technical features as input.
Precondition	Cleaned and scaled feature vector must be available.
Post Condition	SVM returns a class label representing the trading decision.
Scenario	1. Feature vector is passed to the SVM model. 2. Model performs binary/multi-class classification. 3. Result is returned to the backend. 4. Added to ensemble voting.
Exception	1. Improper input shape return error. 2. Kernel computation fails return neutral fallback. 3. SVM predicts out-of-range class log issue.
Priority	Medium
Frequency of Use	High

Table 3.6: SVM Model Prediction UC

3.4.2.7 Logistic Regression Prediction

Use Case ID	**UC-07**
Use Case Name	Predict Signal using Logistic Regression
Actors	Flask Backend
Description	The backend uses the Logistic Regression model to classify a trading signal (Buy, Sell, Neutral) based on precomputed technical indicators like RSI, SMA_30, and price change percentage.
Precondition	Feature vector must be prepared and validated.
Post Condition	A valid class label (Buy/Sell/Neutral) is returned or a fallback (Neutral) is applied.
Scenario	1. Backend prepares and validates the feature vector. 2. Model is loaded from file (e.g., lr_model.pkl). 3. Features are passed into the model. 4. Model returns a class label. 5. Prediction is forwarded to the ensemble voting system.
Exception	1. If input vector is malformed return error. 2. If model fails to load log error and fallback to default. 3. If model returns undefined class log and return "Neutral."
Priority	Medium
Frequency of Use	High – called on every prediction request

Table 3.7: Logistic Regression prediction UC

3.4.2.8 Ensemble Prediction using Majority Voting

Use Case ID	**UC-08**
Use Case Name	Generate Final Signal via Ensemble Voting
Actors	Flask Backend
Description	Final prediction is computed by combining the outputs of Logistic Regression, SVM, and Random Forest using majority voting.
Precondition	Individual model predictions must be successfully returned.
Post Condition	Final Buy/Sell/Neutral signal is shown to the user.
Scenario	1. Predictions from RF, SVM, and Logistic models are collected. 2. Voting mechanism tallies each class. 3. Most frequent class is selected as the final signal. 4. Result is sent to the frontend.
Exception	1. One or more models fail continue with remaining models. 2. Tie in votes fallback to Random Forest or default to Neutral. 3. All models fail prediction is not shown.
Priority	High
Frequency of Use	High

Table 3.8: Ensemble Prediction with majority voting UC

3.4.2.9 Technical Analysis Scraper Module

Use Case ID	**UC-09**
Use Case Name	Scrape PSX Historical Stock Data
Actors	Flask Backend – Technical Module
Description	The module scrapes 6-month historical price data for a given stock ticker from the official PSX website to prepare input for technical analysis.
Precondition	User must input a valid PSX stock symbol and trigger prediction.
Post Condition	Cleaned stock data is returned for indicator computation.
Scenario	1. User enters a stock ticker and clicks "Predict". 2. Backend calls the scraper module. 3. Scraper sends HTTP request to PSX historical data page. 4. Data is parsed into a structured DataFrame. 5. Output is returned to technical indicator calculator.
Exception	1. Invalid or delisted stock symbol display error. 2. PSX site down or slow retry or notify user. 3. Data format change log error and show fallback response.
Priority	High
Frequency of Use	High (invoked for every stock prediction request)

Table 3.9: Technical Analysis Scraper UC

3.4.2.10 Sentimental Analysis Scraper Module

Use Case ID	**UC-10**
Use Case Name	Scrape Financial News Headlines
Actors	Flask Backend – Sentiment Blueprint
Description	The system scrapes the latest business headlines from Dawn and Brecorder to gather market-related news content for sentiment analysis.
Precondition	User must trigger the sentiment analysis feature through the frontend.
Post Condition	A list of cleaned and relevant headlines is passed to the sentiment classifier.
Scenario	1. User clicks "Check Sentiment" on the UI. 2. System activates the scraper module. 3. Dawn and Brecorder business pages are requested using HTTP. 4. HTML is parsed to extract news headlines. 5. Headlines are cleaned and returned for classification.
Exception	1. If Dawn/Brecorder website structure changes return "source unavailable" message. 2. If no headlines found notify user with fallback message. 3. Request timeout or parsing failure skip source or retry once.
Priority	Medium
Frequency of Use	Moderate (whenever user requests sentiment check)

Table 3.10: Sentimental Analysis Scraper UC

3.4.2.11 Technical Indicator Calculation Module

Use Case ID	**UC-11**
Use Case Name	Compute Technical Indicators
Actors	Flask Backend – Technical Engine
Description	After retrieving historical stock data, this module computes essential technical indicators including RSI, SMA_30, and price change percentage. These indicators are used as input features for machine learning models.
Precondition	Clean historical stock data (with valid Close prices and Dates) must be available.
Post Condition	Computed indicator values are passed to the ML prediction module.
Scenario	1. Scraped data is passed to the calculation module. 2. RSI is computed using closing prices. 3. SMA_30 is calculated over the 30-day window. 4. Price change percentage is calculated between oldest and latest values. 5. Result is structured into a feature vector.
Exception	1. Missing or NaN values indicators defaulted or dropped. 2. Less than 30 days of data system returns "insufficient data". 3. Calculation logic fails prediction skipped, error logged.
Priority	High
Frequency of Use	High (used on every prediction cycle)

Table 3.11: Technical Calculation UC

3.4.2.12 NLP Sentiment Classification Module

Use Case ID	**UC-12**
Use Case Name	Classify Sentiment using NLP
Actors	Flask Backend – Sentiment Engine
Description	This module takes cleaned financial headlines as input and uses a Natural Language Processing model to classify their sentiment into Positive, Neutral, or Negative. The result provides context to market mood around the selected stock.
Precondition	A valid list of headlines must be scraped and cleaned.
Post Condition	A sentiment label is returned for each headline, and an overall sentiment is calculated.
Scenario	1. Headlines from Dawn and Brecorder are passed into the classifier. 2. Each headline is tokenized and transformed using the NLP pipeline. 3. Model predicts sentiment class for each entry. 4. A majority vote or score summary determines the overall market sentiment. 5. Sentiment is returned to the frontend.
Exception	1. Input text is empty system returns "Neutral". 2. Model not loaded properly fallback message shown. 3. Text preprocessing fails log error and skip headline.
Priority	Medium
Frequency of Use	Moderate (used only when sentiment is requested)

Table 3.12: NLP Sentimental Classification UC

3.4.2.13 Handle Invalid Input

Use Case ID	**UC-13**
Use Case Name	Handle Invalid or Missing Stock Input
Actors	User, Flask Backend
Description	This module validates the user's stock ticker input. If the ticker is missing, malformed, or does not exist on PSX, the system prevents further processing and returns a meaningful error message.
Precondition	User is on the homepage and attempts to enter or submit a stock ticker for prediction.
Post Condition	User is either allowed to continue or shown an appropriate message if the input is invalid.
Scenario	1. User submits the form with a stock ticker. 2. Backend checks input validity (non-empty, valid PSX symbol format). 3. If invalid, user is notified with an error message. 4. If valid, prediction process proceeds.
Exception	1. Empty input system displays "Please enter a stock symbol." 2. Ticker not found in PSX list show "Invalid or unlisted stock." 3. Backend crash or timeout fallback message shown.
Priority	High
Frequency of Use	High (invoked on every user input)

Table 3.13: Handling Invalid Input UC

3.4.2.14 Visualize Prediction Result

Use Case ID	**UC-14**
Use Case Name	Visualize Prediction Result
Actors	User, Frontend (Chart.js), Flask Backend
Description	This module graphically presents the predicted Buy/Sell/Neutral signal along with technical indicators (RSI, SMA, Price) over a 6-month timeline using interactive charts.
Precondition	The prediction result and technical indicators must be computed and available.
Post Condition	The user sees an interactive chart showing the stock's price trend with technical overlays and an icon or label indicating the prediction.
Scenario	1. Prediction and indicators are returned from the backend. 2. Frontend receives data as a JSON response. 3. Chart.js renders the stock price line graph. 4. RSI and SMA curves are plotted. 5. Prediction label (Buy/Sell/Neutral) is shown at the latest data point.
Exception	1. Chart.js fails to render default error message or empty chart shown. 2. Incomplete or corrupted data frontend disables visualization section. 3. Prediction missing user prompted to retry.
Priority	High
Frequency of Use	High (for every valid prediction request)

Table 3.14: Visualizing Predictions UC

3.4.3 DataFlow Diagrams

3.4.3.1 Authentication

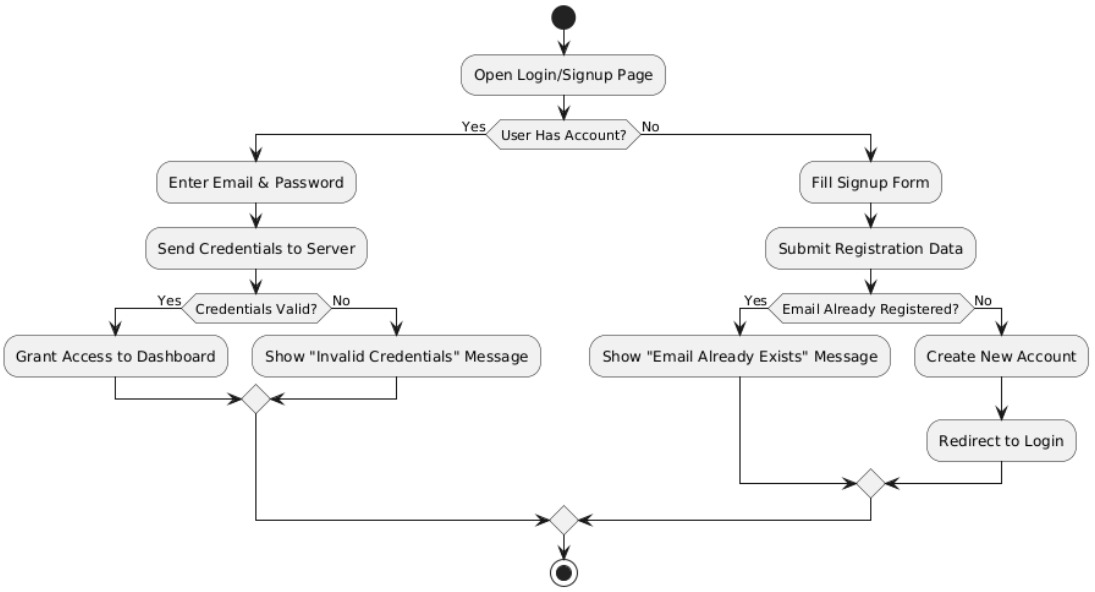

FIGURE 3.9: Authentication Module

3.4.3.2 Gainers/Losers

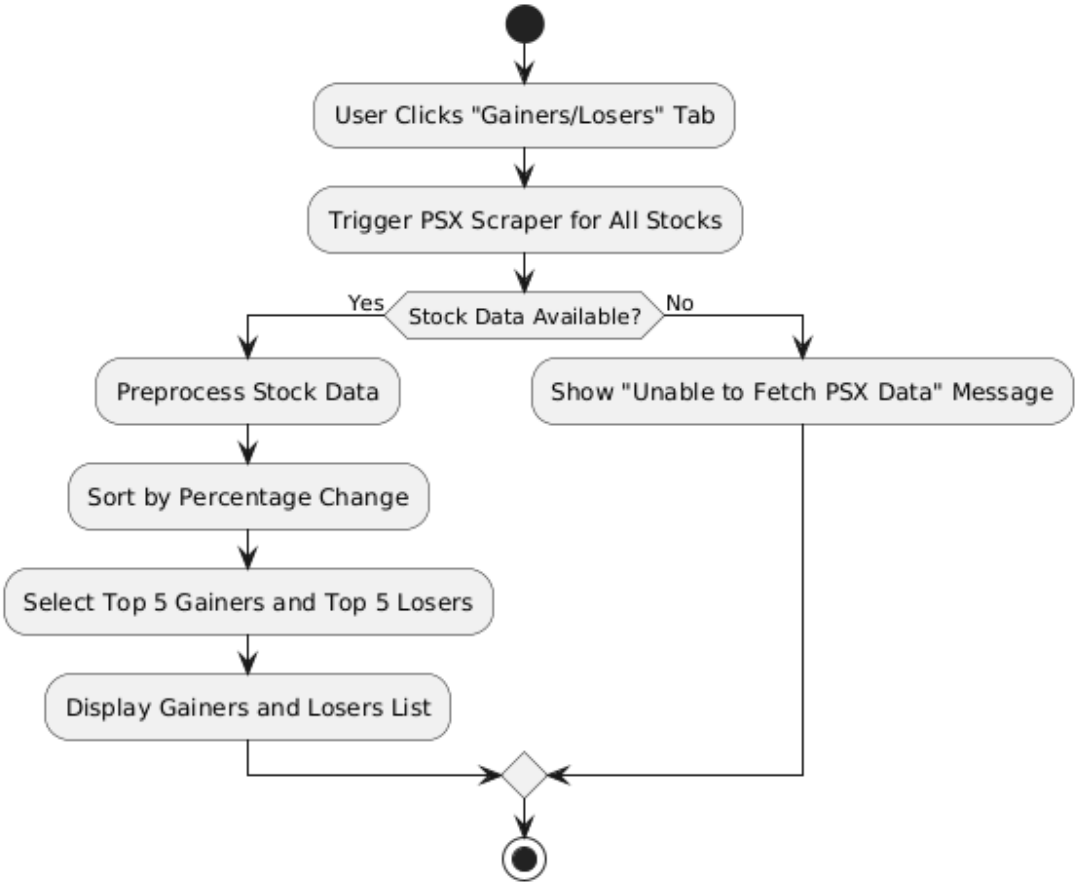

FIGURE 3.10: Gainers/Losers Module

3.4.3.3 Sentimental Analysis

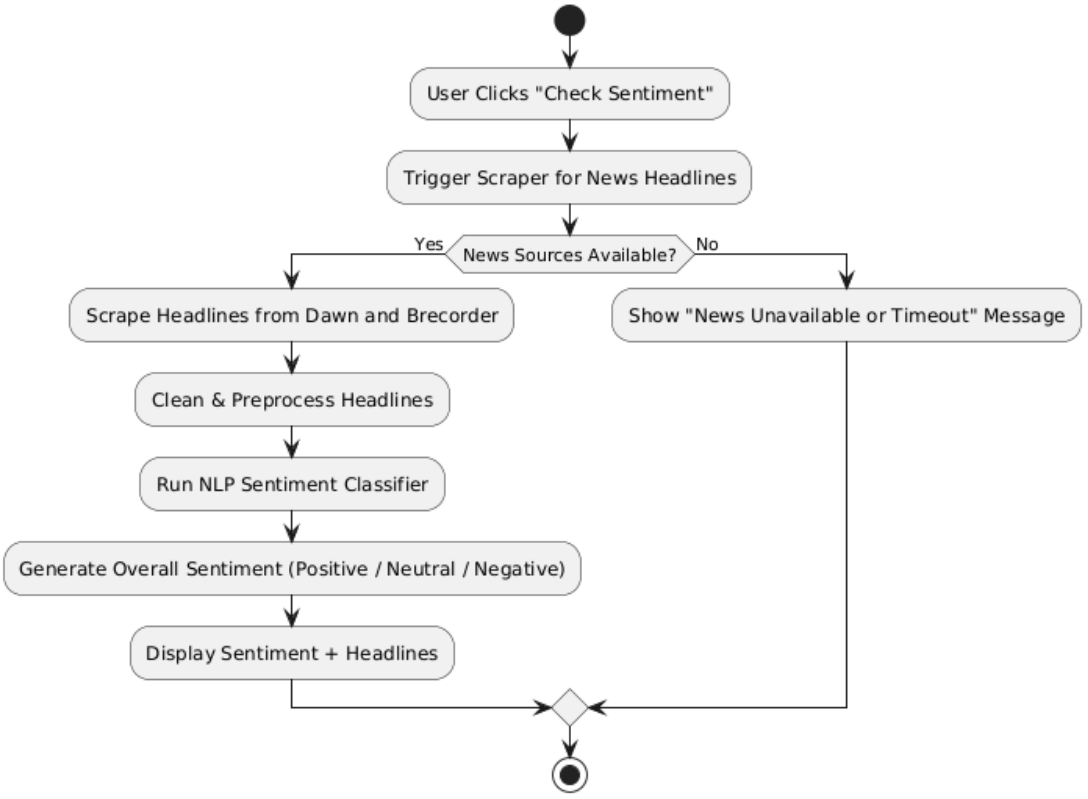

FIGURE 3.11: Sentimental Analysis Module

3.4.3.4 Technical Analysis

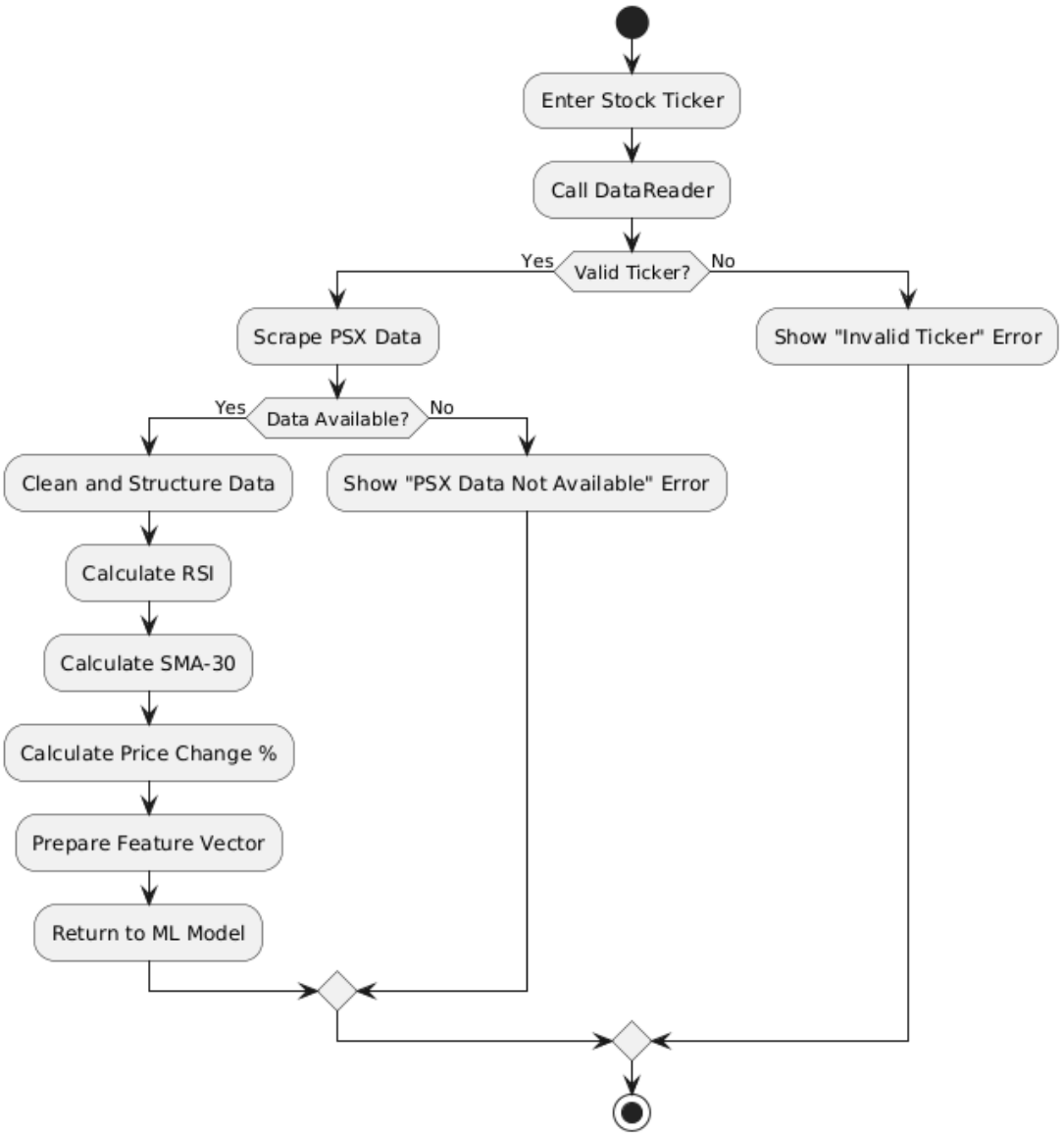

FIGURE 3.12: Technical Analysis Module

3.4.4 ER Diagram

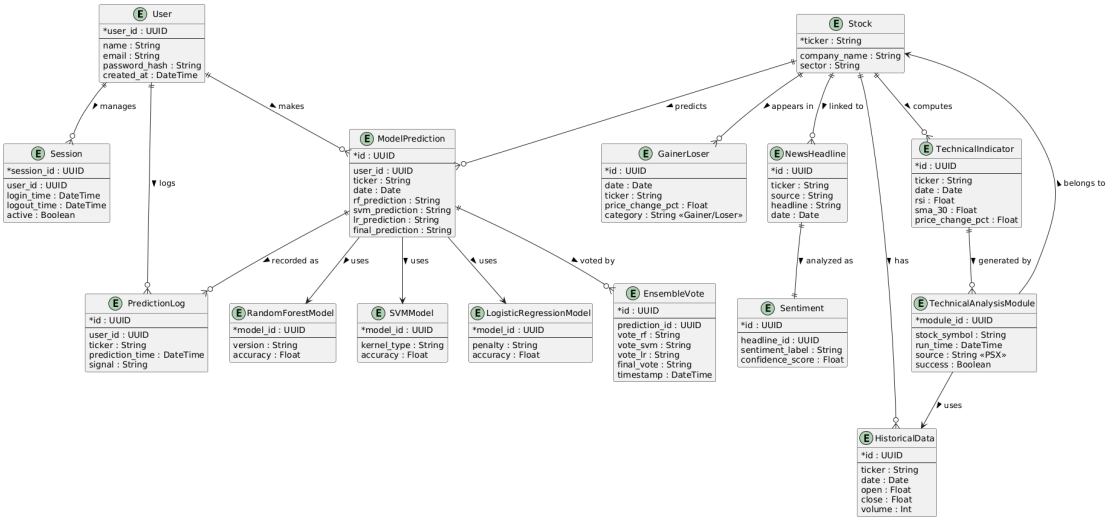

FIGURE 3.13: ER Diagram

3.4.5 Class Diagram

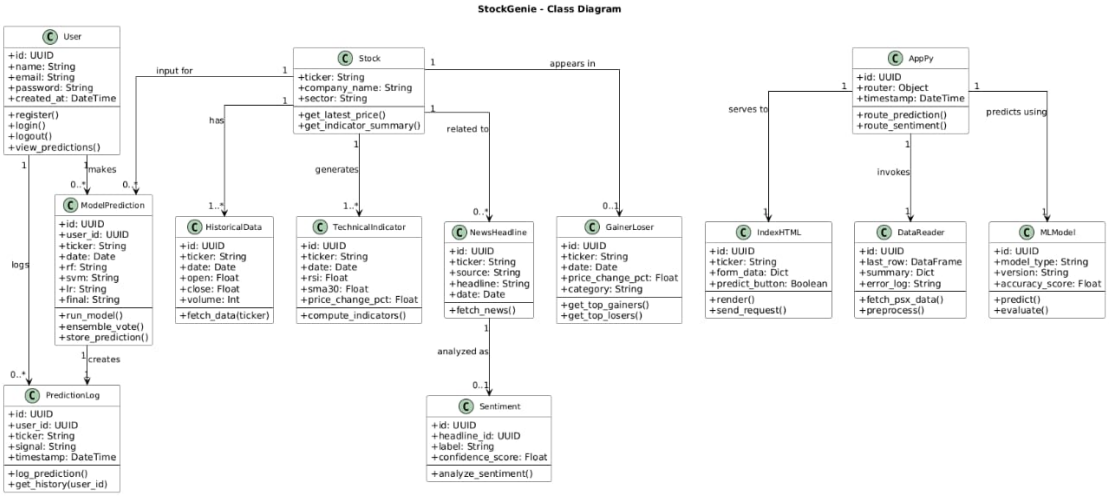

FIGURE 3.14: Class Diagram

3.4.6 Sequence Diagrams

3.4.6.1 Authentication

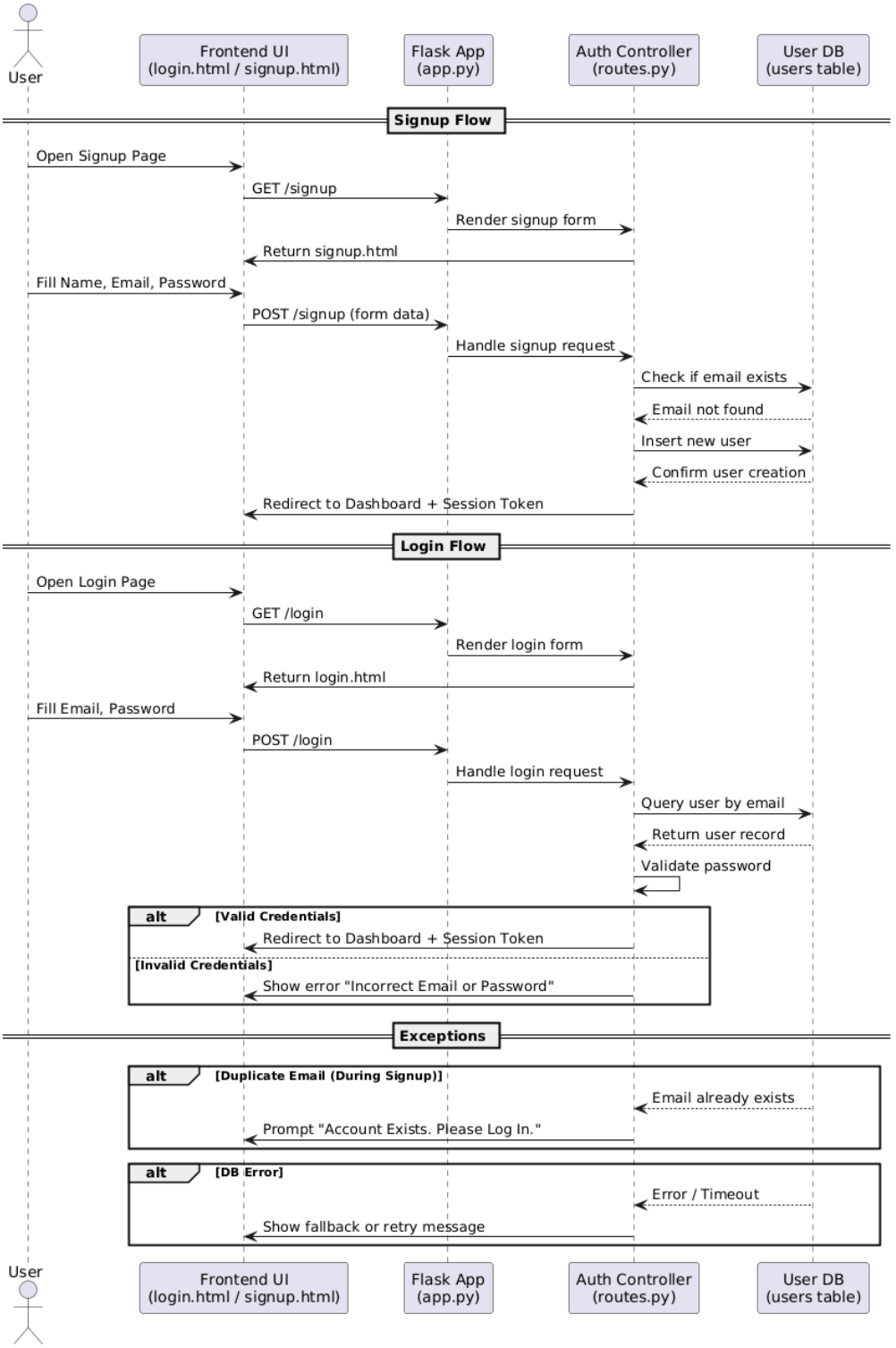

FIGURE 3.15: Sequence Diagram: Authentication

3.4.6.2 Technical Analysis

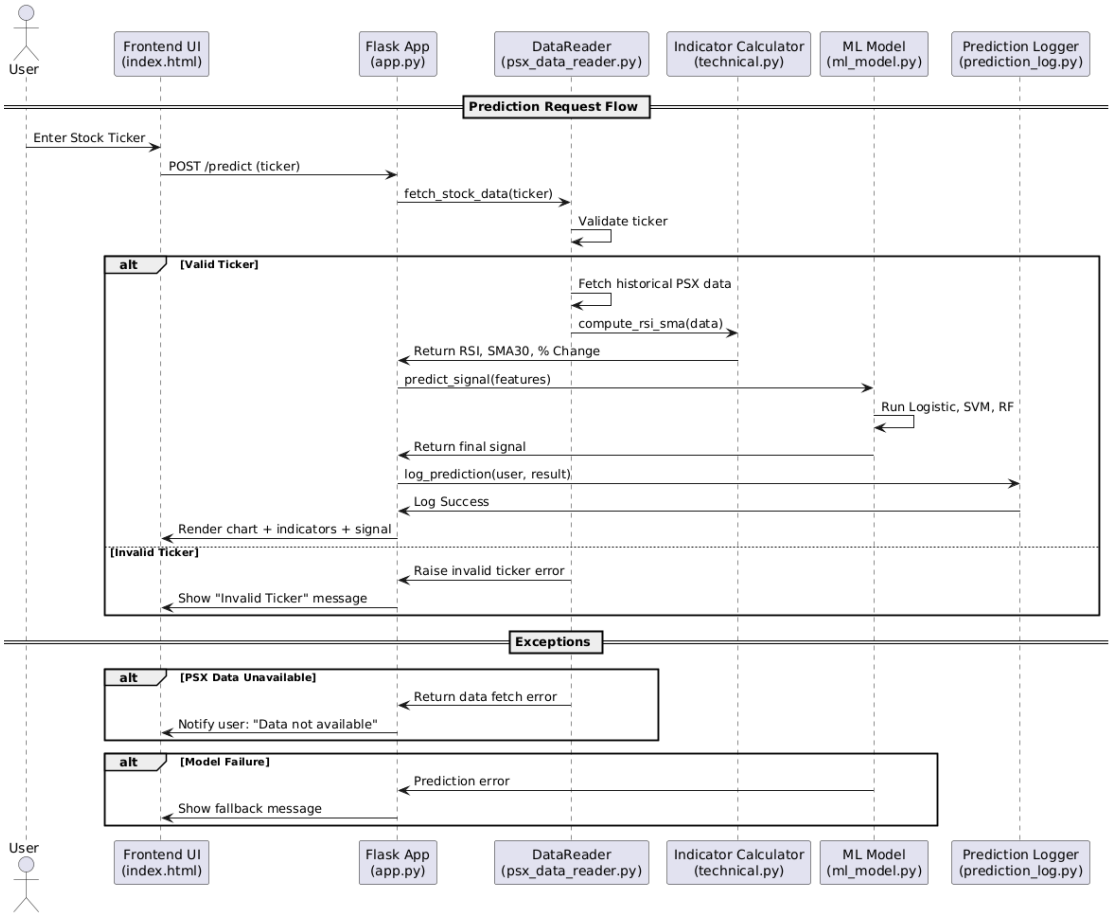

FIGURE 3.16: Sequence Diagram: Technical Analysis

3.4.6.3 Sentiment Analysis

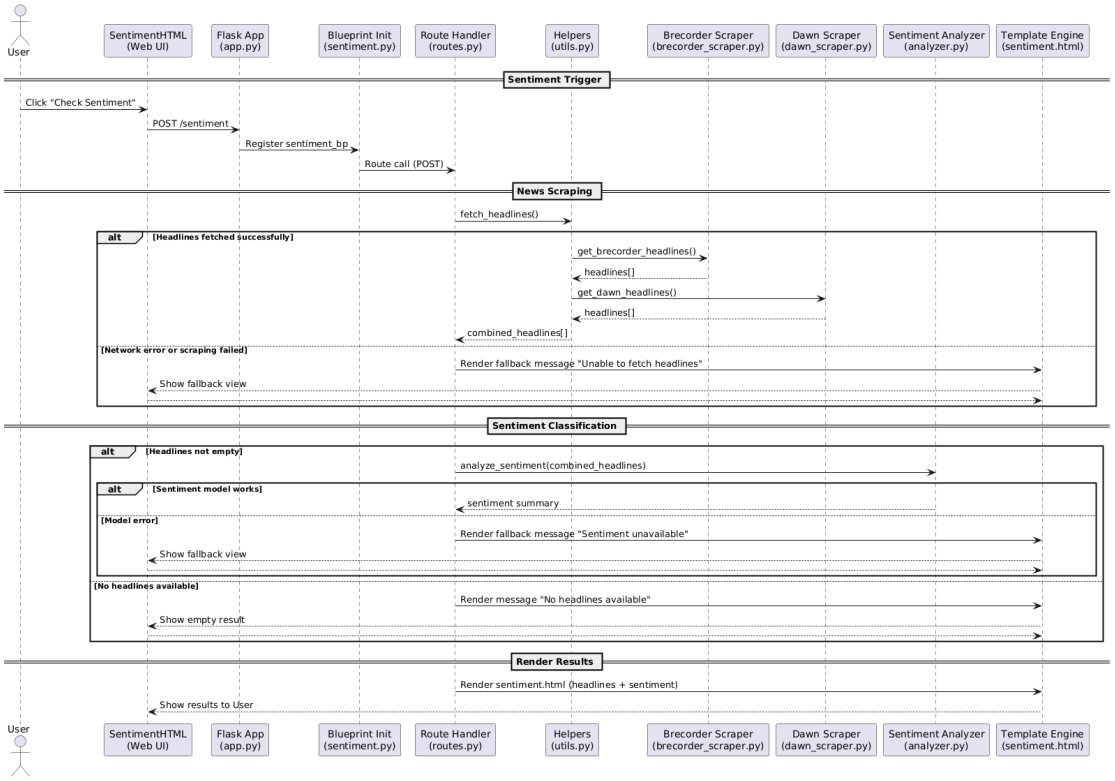

FIGURE 3.17: Sequence Diagram: Sentimental Analysis

3.4.6.4 Gainers and Losers

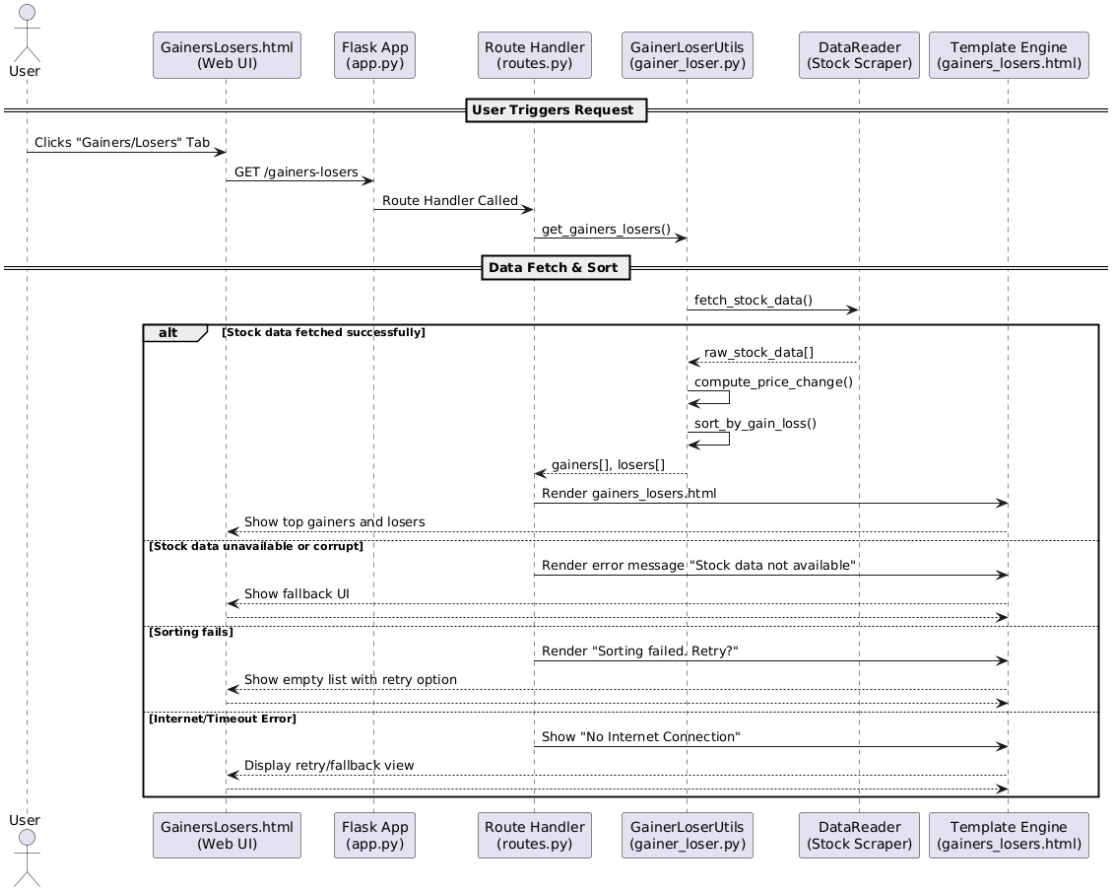

FIGURE 3.18: Sequence Diagram: Gainers and Losers

3.4.6.5 Random Forest Model Prediction

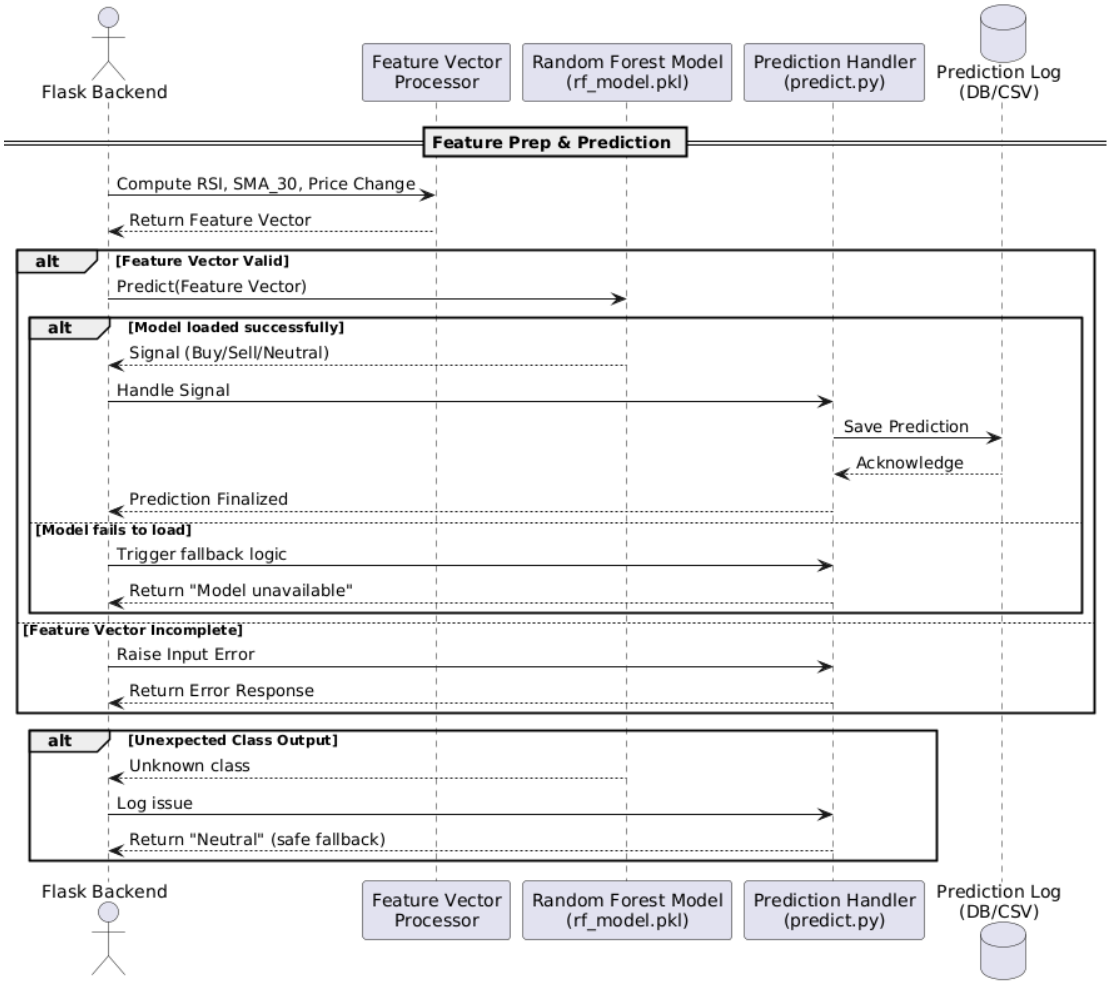

FIGURE 3.19: Sequence Diagram: Random Forest Model Prediction

3.4.6.6 Support Vector Machine (SVM) Model Prediction

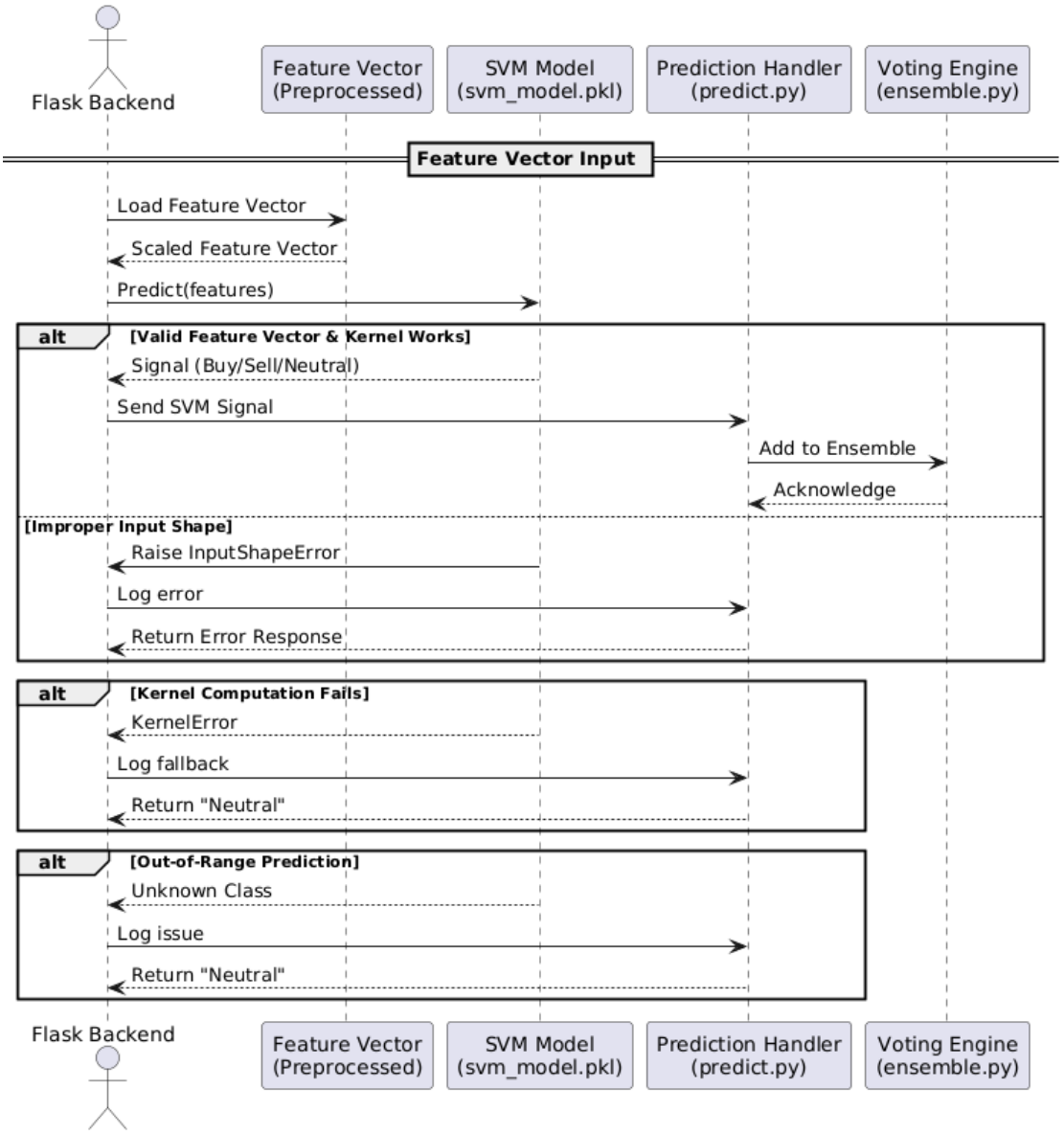

FIGURE 3.20: Sequence Diagram: Support Vector Machine (SVM) Model Prediction

3.4.6.7 Logistic Regression Prediction

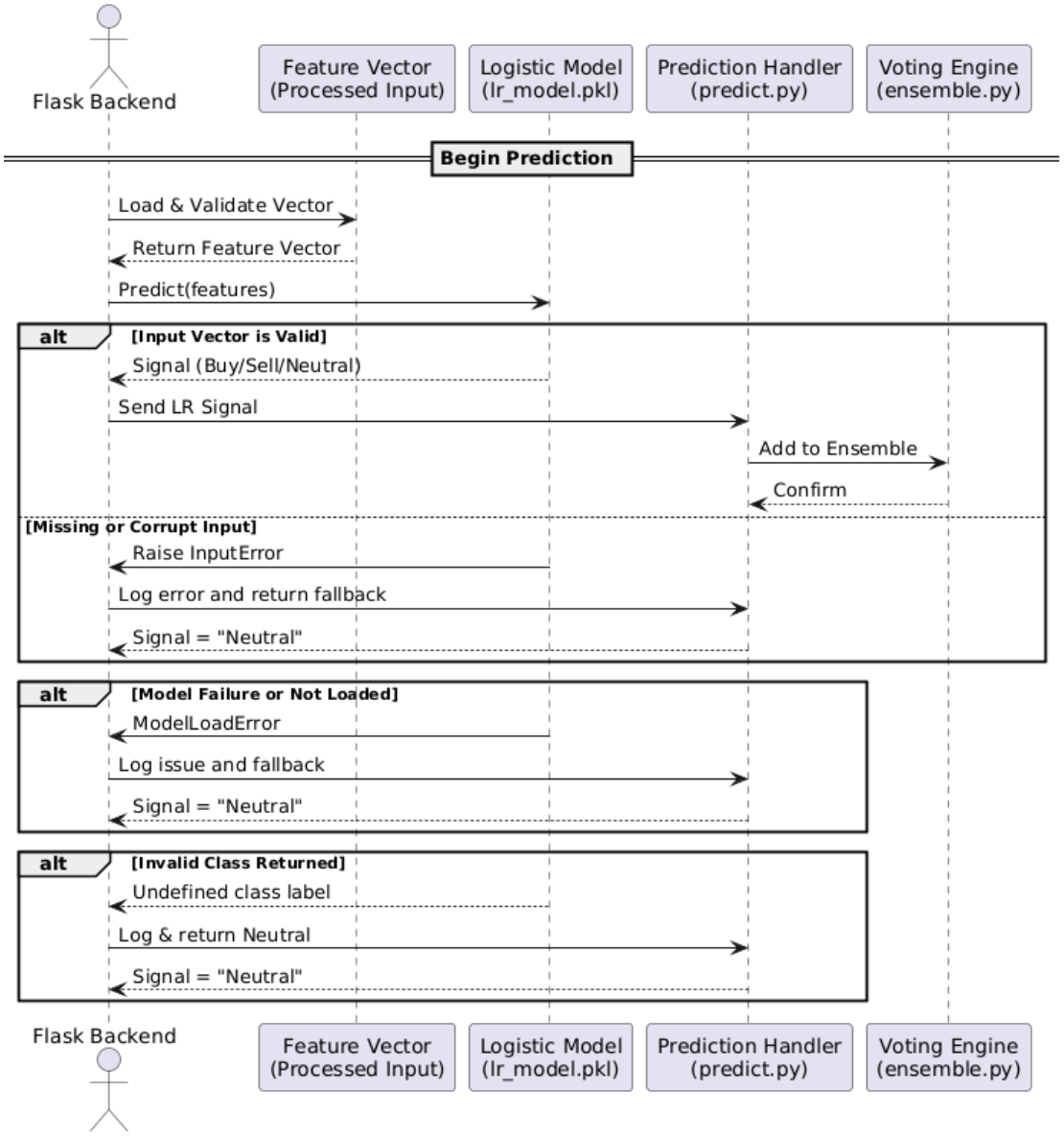

FIGURE 3.21: Sequence Diagram: Logistic Regression Prediction

3.4.6.8 Ensemble Prediction using Majority Voting

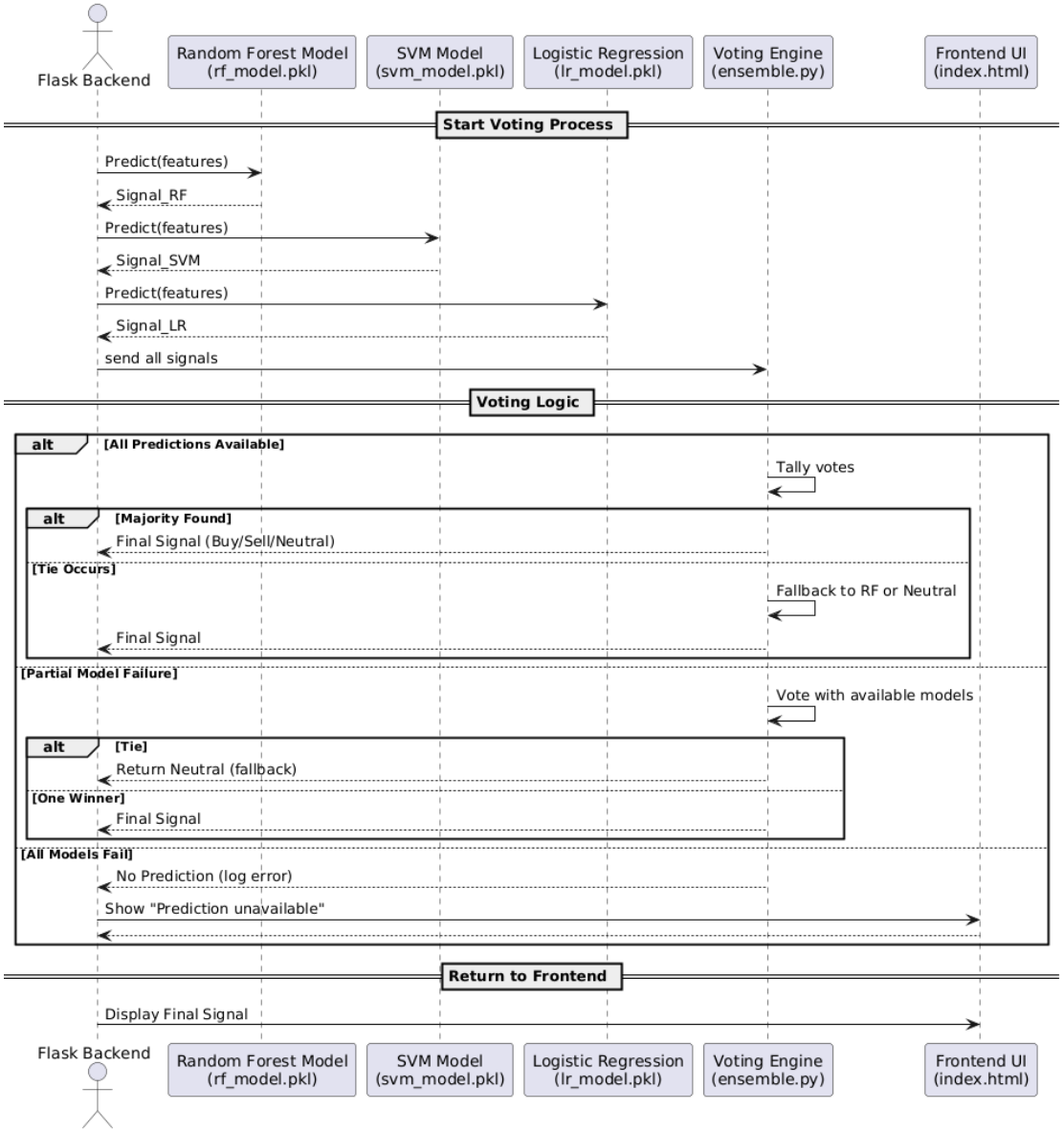

FIGURE 3.22: Sequence Diagram: Ensemble Prediction

3.4.6.9 Technical Analysis Scraper Module

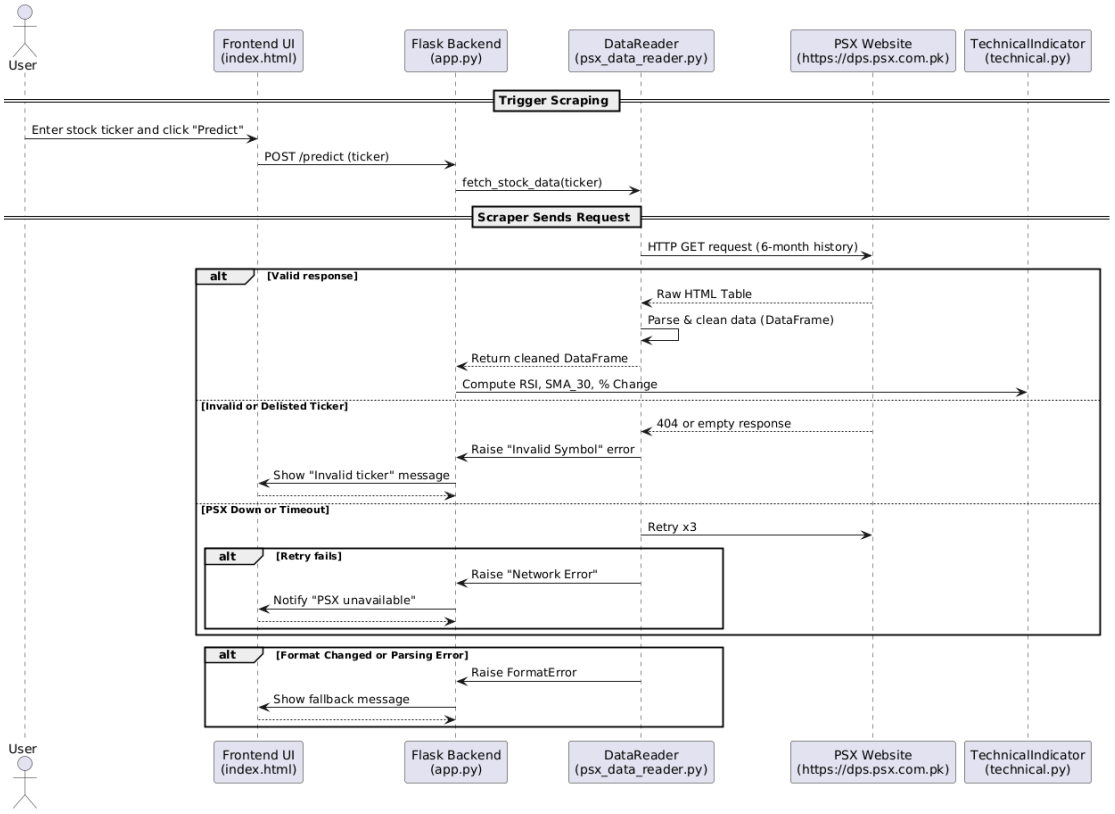

FIGURE 3.23: Sequence Diagram: Scraper Module(Technical)

3.4.6.10 Sentimental Analysis Scraper Module

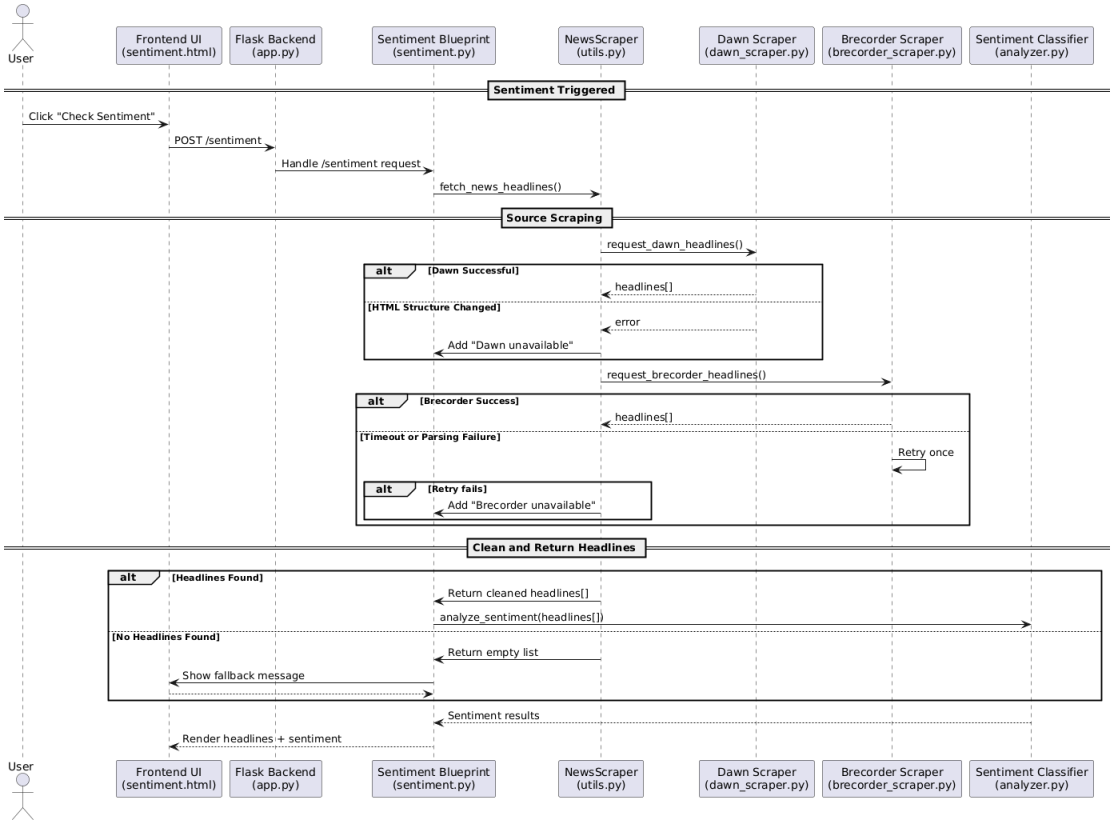

FIGURE 3.24: Sequence Diagram: Scraper Module(Sentimental)

3.4.6.11 Technical Indicator Calculation Module

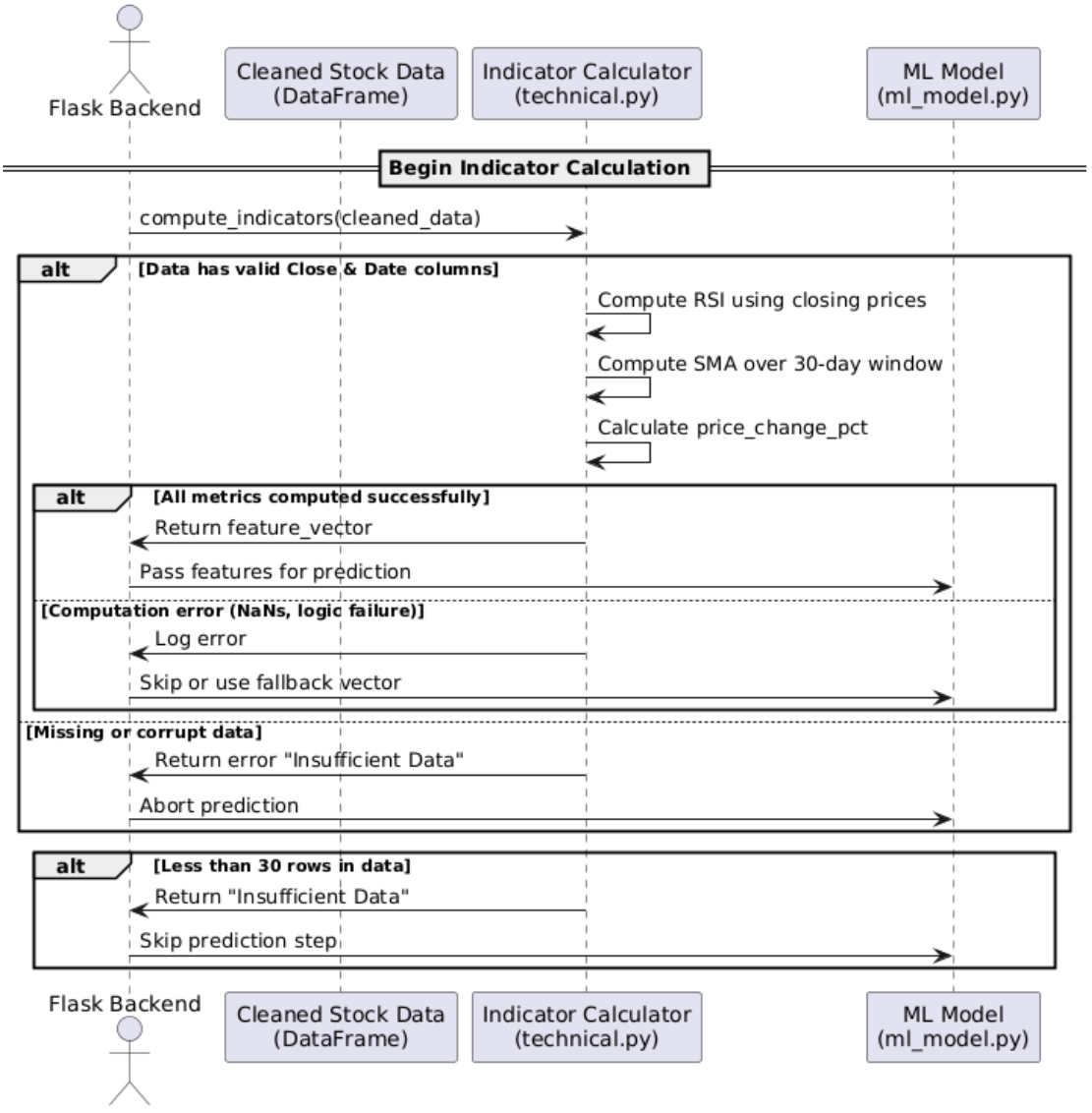

FIGURE 3.25: Sequence Diagram: Technical Indicator Calculation

3.4.6.12 Visualize Prediction Result

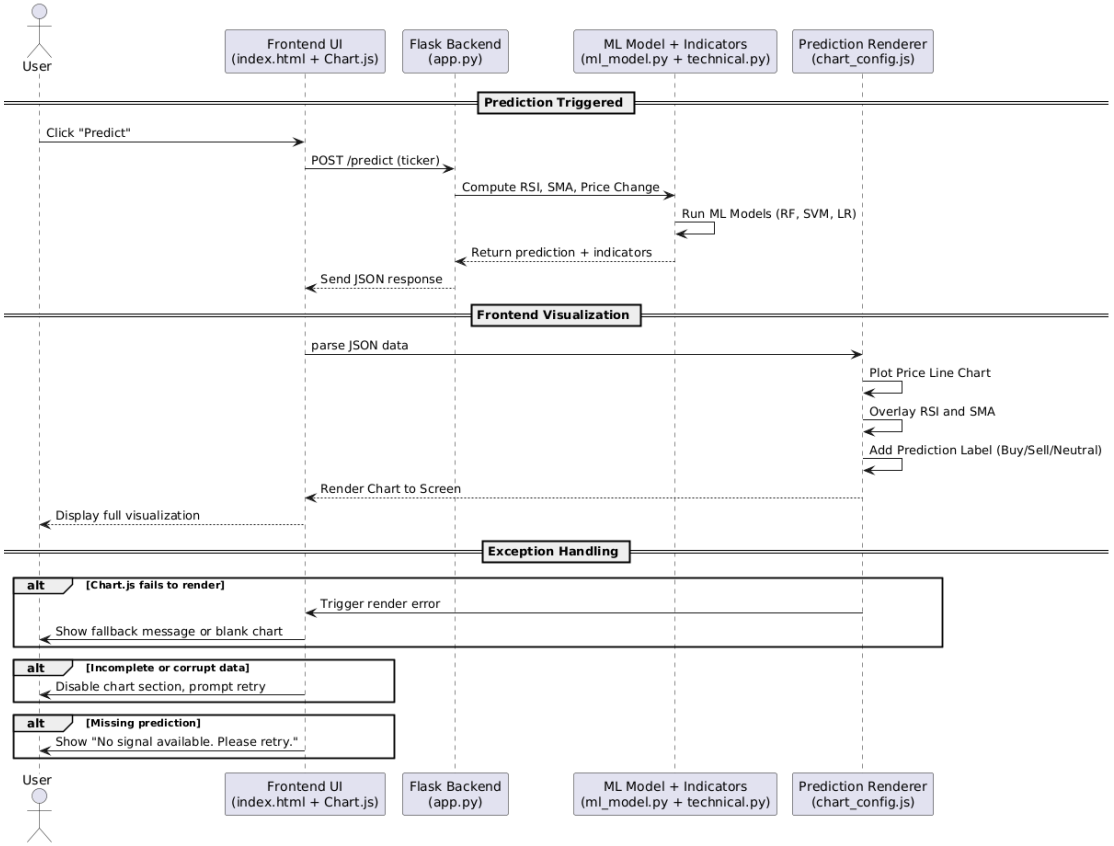

FIGURE 3.26: Sequence Diagram: Visualize Prediction

Chapter 4

Implementation and Evaluation

4.1 Development Stages

The development stage of *Stock Genie* involved a series of carefully planned steps starting from gathering resources to the final implementation of technical modules. This section outlines the journey from ideation to execution with details on tools, logic, and integration points.

4.1.1 Gathering Information

The first step in the development of *Stock Genie* was to gather and organize relevant information to establish a strong conceptual and technical base. We began by analyzing data from the **Pakistan Stock Exchange (PSX)** to understand how stock tickers, historical price movements, and trading volumes are structured. This was critical for developing accurate scrapers and reliable input for technical indicators. For the sentiment analysis module, we explored reputable **Pakistani financial news sources**, specifically *Dawn* and *Brecorder*, to extract market-relevant headlines that could reflect investor sentiment. Additionally, a wide range of **machine learning resources**—including research papers, online tutorials, and public repositories—were studied to guide the implementation of

classification models such as Random Forest, Support Vector Machine (SVM), and Logistic Regression. To further align the project with real user needs, we conducted informal surveys and analyzed popular platforms like *TradingView* and *Investing.com*. This helped us understand the expectations of local investors and identify gaps in existing tools. The insights obtained during this stage laid the foundation for defining the system's architecture, features, and evaluation criteria.

4.1.2 Initialization

During the initialization phase, the core development setup and architecture of *Stock Genie* were established. Key activities included:

- Git and GitHub were configured for tracking changes and collaboration across team members.
- A Python venv was created to manage dependencies and avoid conflicts with system-level packages.
- The project was divided into clear modules—scrapers, ML models, routes, templates, and static assets—to ensure maintainability and scalability.
- Initial ERDs, use case diagrams, and data flow diagrams were prepared to guide development and ensure functional alignment.
- CI/CD pipeline was optionally explored using GitHub Actions (future scope).

4.1.3 Implementation

The core implementation phase is divided into several sub-parts to explain the logic and architecture of *Stock Genie*.

4.1.3.1 Overview and Main Features

Stock Genie is an AI-driven web application built for the Pakistan Stock Exchange (PSX), designed to assist users in making informed stock decisions through technical, predictive, and sentiment-based insights. The system integrates multiple intelligent modules into a unified platform with the following core features:

- **Technical Analysis:** Computes key indicators including **RSI**, **SMA_30**, and **Price Change %** using six months of historical PSX data.
- **ML-Based Signal Prediction:** Utilizes an ensemble of **Random Forest**, **SVM**, and **Logistic Regression** models to predict **Buy**, **Sell**, or **Neutral** signals.
- **Ensemble Voting Logic:** Combines model outputs using majority voting to generate a more reliable final prediction.
- **Sentiment Analysis:** Scrapes and classifies headlines from **Dawn** and **Brecorder** using NLP techniques to detect **Positive**, **Neutral**, or **Negative** market sentiment.
- **Gainers and Losers Module:** Displays top-performing and worst-performing stocks based on latest percentage change.
- **Interactive Charting:** Uses **Chart.js** to visually present stock price trends, prediction signals, RSI/SMA overlays, and sentiment tags on an interactive graph.
- **Prediction Logging:** Maintains a log of user predictions for tracking, future evaluation, or personalization.
- **Authentication System:** Allows user **registration**, **login**, and **session handling** for accessing saved features.

4.1.3.2 Technical Explanations

The technical flow is broken down into multiple modules:

- **Data Reader Module:** Scrapes historical stock data using requests and BeautifulSoup, and converts it into DataFrames using pandas.
- **Technical Indicators:** RSI is calculated using exponential moving averages, SMA over 30-day windows, and % price change is derived using the first and last data points.
- **ML Models:** Three supervised classifiers (Random Forest, SVM, Logistic Regression) are trained on labeled stock signal data.
- **Ensemble Voting:** Majority voting is used to select the final prediction output.
- **News Scraper & Sentiment Analyzer:** Headlines are scraped and passed to a fine-tuned transformer or rule-based sentiment classifier.
- **Prediction Logging:** Results and indicators are stored in backend logs for potential feedback loops.

4.1.3.3 Tools and Technology

The following technologies were used in the development of Stock Genie, each chosen for its effectiveness in handling specific aspects of the system:

Tool / Technology	**Purpose**
Python 3.10+	Core language for backend logic and data processing
Flask	Serves both backend APIs and renders frontend pages (e.g., prediction, sentiment, gainers/losers)
Pandas, NumPy	Used for computing technical indicators (RSI, SMA, price change) and preparing feature vectors
scikit-learn	Handles training and prediction with ML models (Random Forest, SVM, Logistic Regression)
BeautifulSoup	Web scraping tool for fetching data from PSX, Dawn, and Brecorder websites
Chart.js	Generates interactive graphs for price trends, indicators, and prediction display
Firebase	Supports storing and retrieving gainers/losers data dynamically (in extended versions)

Table 4.1: Tools and Technologies Used in StockGenie

4.2 System Integration

System integration in Stock Genie involved seamlessly connecting all modules—including technical analysis, machine learning predictions, sentiment analysis, and frontend visualization—into a unified Flask-based web application. Each component communicates through structured data pipelines, ensuring smooth execution from user input to final prediction display.

4.3 User Interface

The user interface of StockGenie is designed for simplicity and ease of use, allowing users to input stock symbols and instantly receive predictions and insights. Interactive charts, sentiment indicators, and model results are presented in a clean and intuitive layout.

4.3.1 Home page

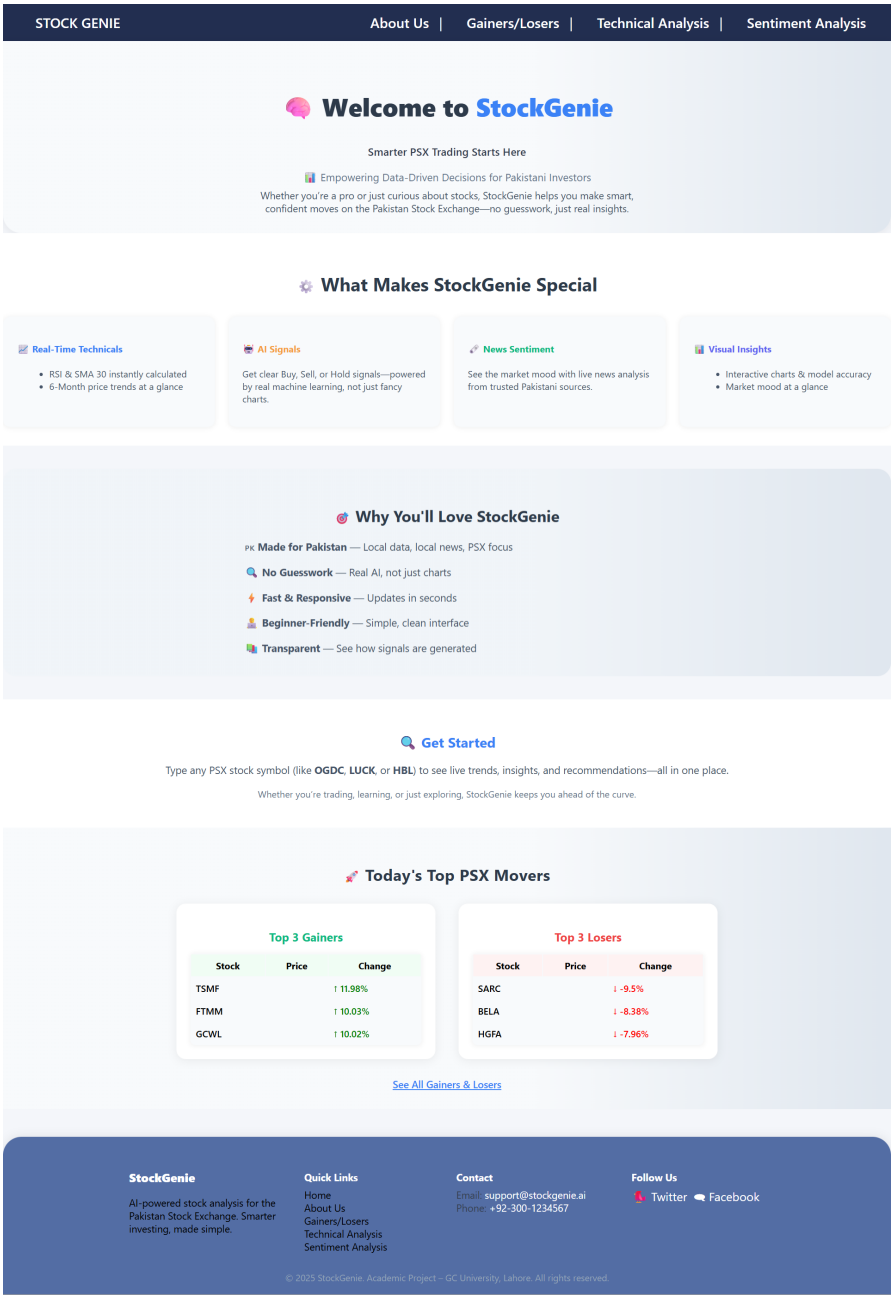

FIGURE 4.1: Home Page

4.3.2 About Us

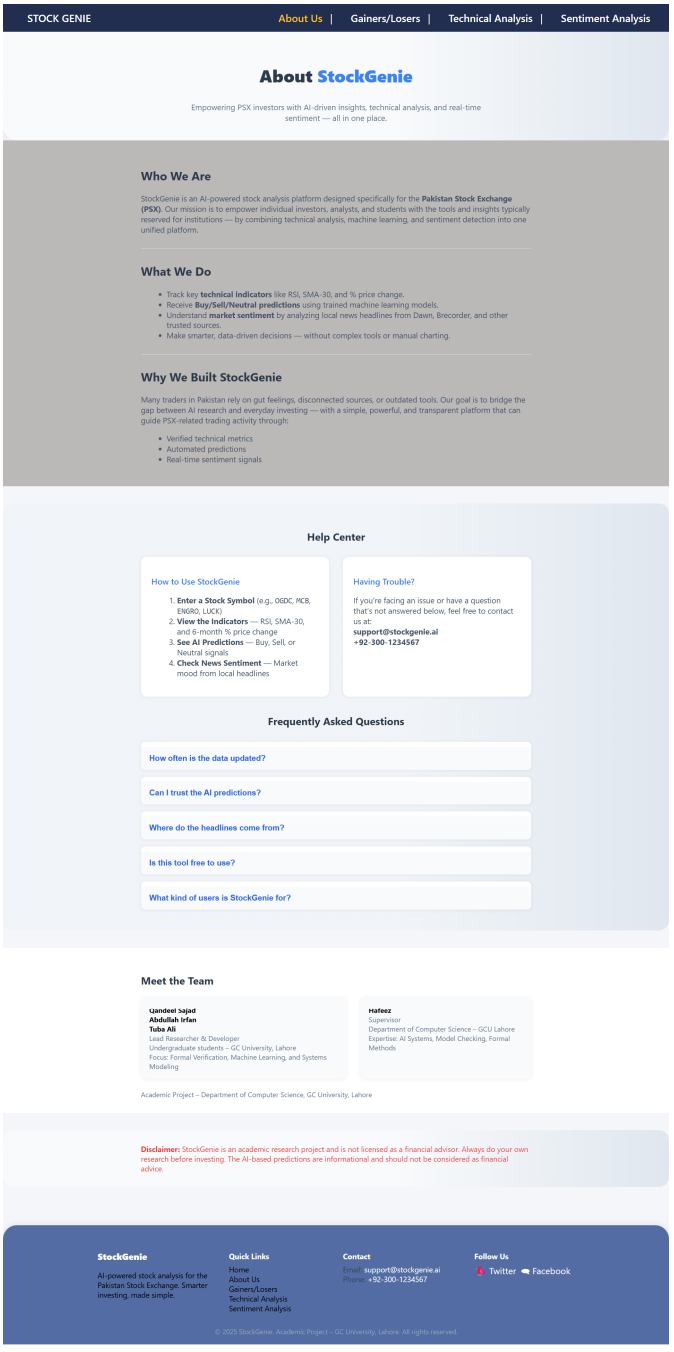

FIGURE 4.2: About Us

4.3.3 Register Page

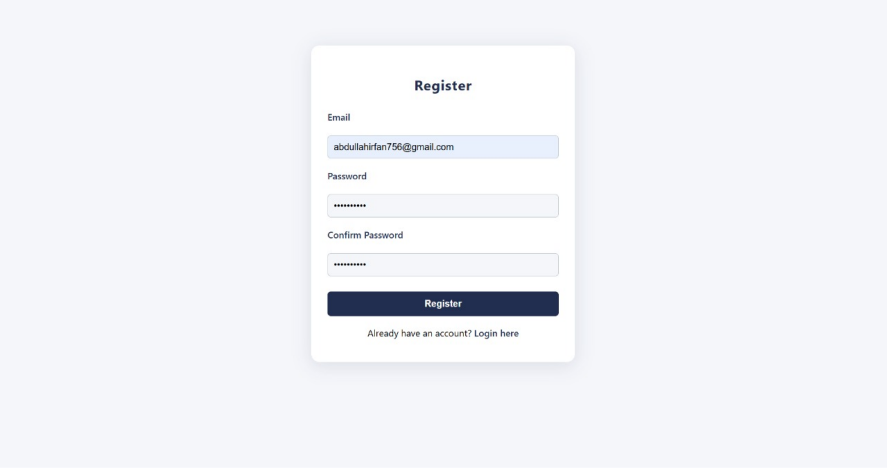

FIGURE 4.3: Registering Page

4.3.4 Login Page

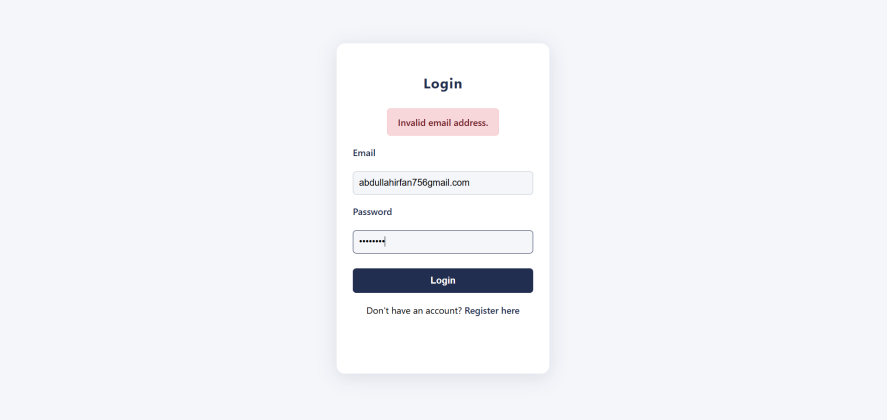

FIGURE 4.4: Invalid credential

4.3.5 Gainers/ Loser Page

FIGURE 4.5: Gainers & Losers

4.3.6 Technical Analysis Overview

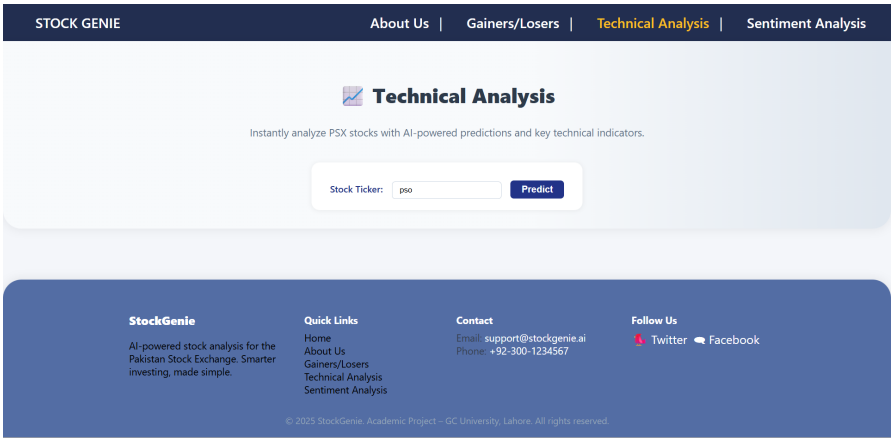

FIGURE 4.6: Technical Analysis

4.3.7 Technical Analysis Prediction

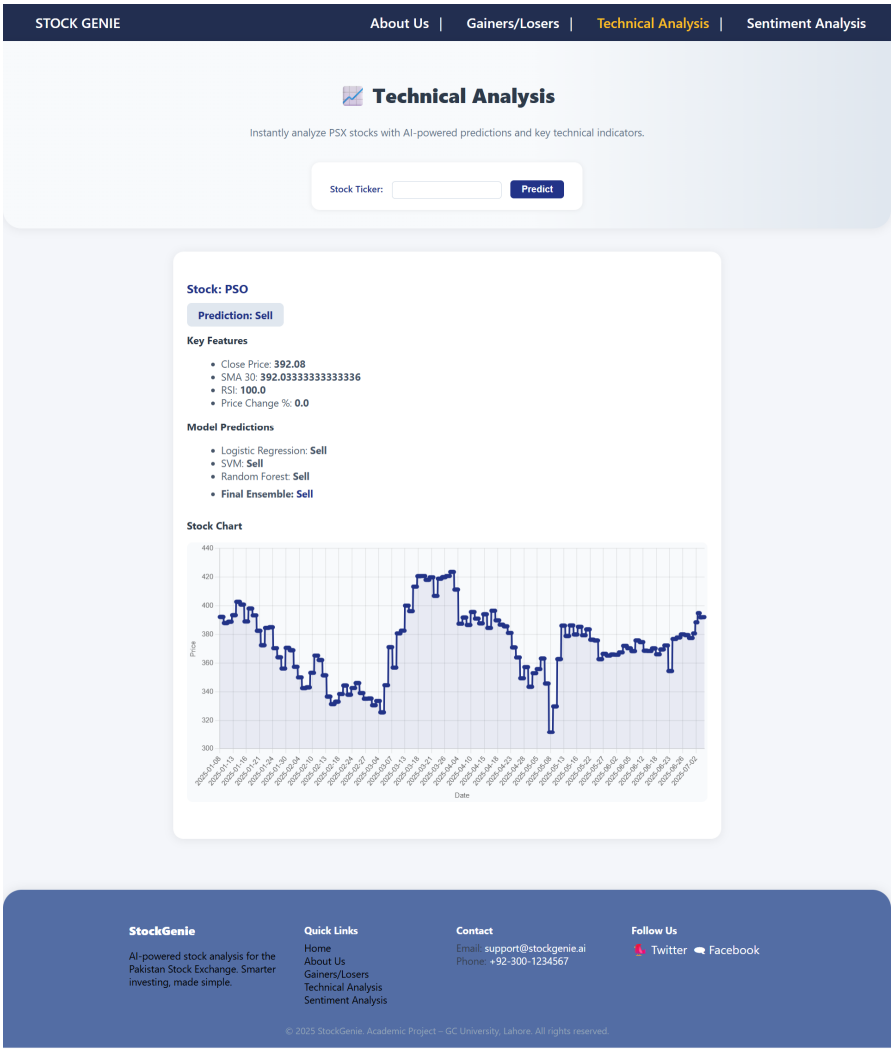

FIGURE 4.7: Technical Analysis Prediction

4.3.8 Sentimental Analyzer

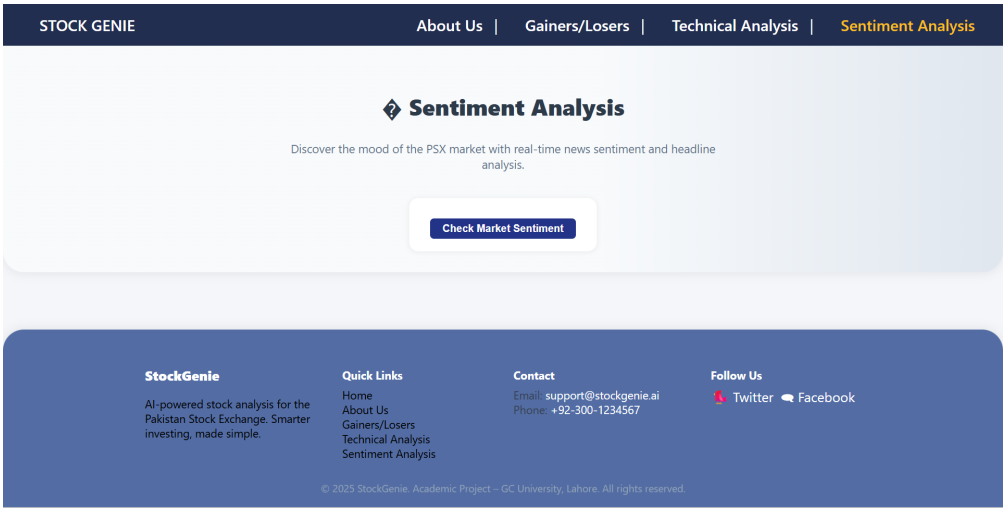

FIGURE 4.8: Sentimental Analyzer

4.3.9 Sentiment of the Market

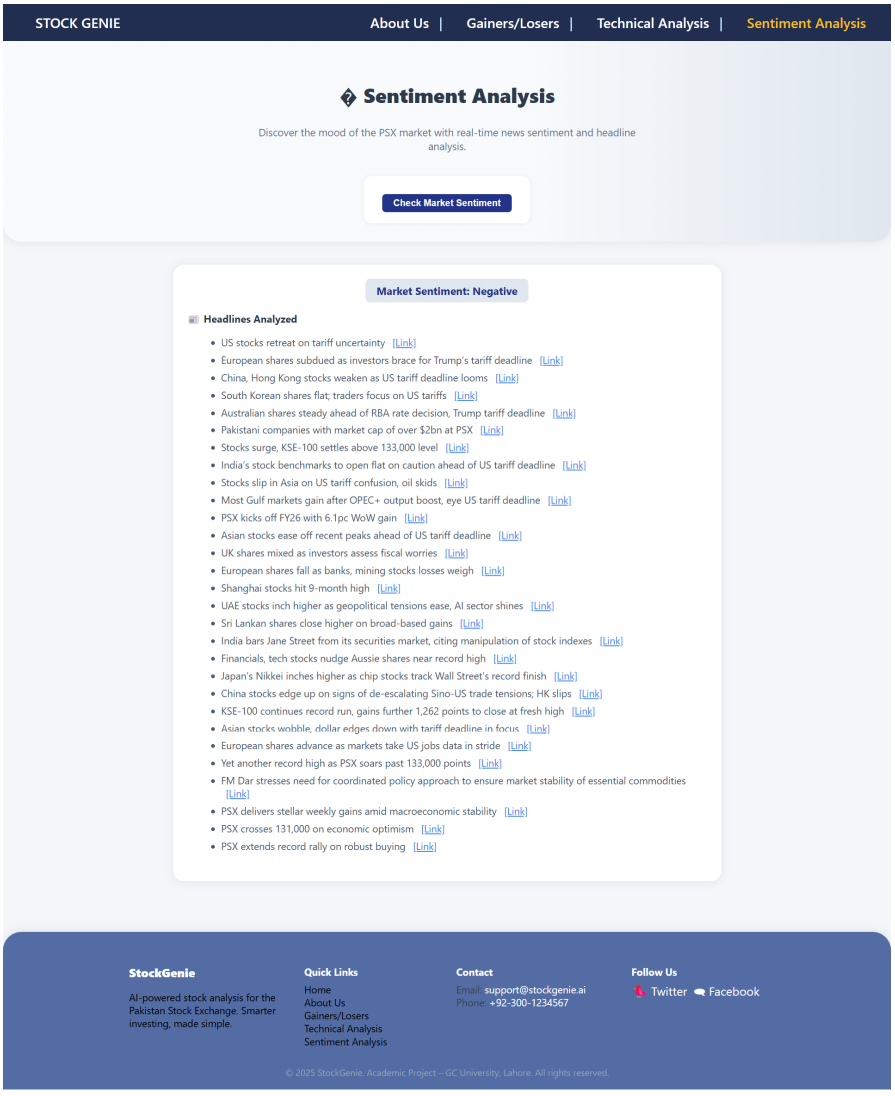

FIGURE 4.9: Sentiment of the Market

4.4 Hardware Interface

Here are minimal requirements of hardware to run this website:

- Pentium 4 1.97GHz
- 4GB RAM (Recommended)
- 1GB RAM (Minimum) for our website

4.5 Software Interface

- Windows OS XP / VISTA / 7 / 8 / 8.1 / 10 / 11
- Mac OS year-2010 or above
- Web browsers like Google Chrome, Mozilla Firefox, Opera, Safari etc.

4.6 Unit Testing

Unit testing in Stock Genie plays a crucial step to ensure that each individual component of the system performs its intended functionality accurately and reliably. The development team implemented test cases for core modules including the technical indicator calculator, machine learning model prediction logic (Random Forest, SVM, Logistic Regression), and the sentiment classifier.

These tests helped validate that computations like RSI, SMA-30, and price change percentage produced correct results even when provided with edge cases or missing data. For the machine learning layer, unit tests verified model loading, input vector formatting, and correct prediction class returns. Similarly, the sentiment analysis module was tested to confirm the proper parsing of news headlines and classification output under various text conditions.

In addition, the web scraping scripts (PSX data, Dawn, and Brecorder scrapers) were tested independently to ensure they could extract data even if minor changes occurred in page structure. Flask backend route handlers and response structures were also validated to ensure correct API behavior. Overall, unit testing played a vital role in detecting bugs early, improving code quality, and guaranteeing the stability of Stock Genie during integration and deployment stages.

4.7 Functional Testing

Functional testing of **Stock Genie** is conducted to ensure that all system features performed according to the specified requirements. Each module — including technical analysis, machine learning prediction, sentiment classification, visualization, and gainers/losers display — was validated based on its expected input/output behavior and real-world usage scenarios. Test cases were designed for each functional requirement.

For example, when a user enters a valid PSX stock symbol and clicks "Predict," the system was expected to return a buy/sell/neutral signal along with technical indicators like RSI and SMA. This was tested for multiple tickers to ensure consistent results. Similarly, the sentiment analysis feature was tested by triggering the scraping process and verifying that headlines were fetched from Dawn and Brecorder, cleaned, and classified correctly.

The gainers and losers module was also tested by simulating different data conditions to ensure that it correctly sorted stocks based on price movement and displayed the top performers. Input validation, such as handling of invalid or delisted tickers, was also verified to ensure the system returned appropriate error messages and fallback responses. Overall, functional testing confirmed that Stock Genie behaves as expected across all modules, providing reliable and meaningful output for users in various scenarios.

4.7.1 Testing Requirements

The testing requirements for **Stock Genie** were carefully defined to ensure that each module performed reliably, accurately, and efficiently. The system needed to validate both user input and internal logic for technical accuracy. Stock tickers entered by users were tested for valid formatting, while incorrect or malformed inputs triggered appropriate error messages. The technical indicator calculations—such as RSI, SMA-30, and percentage price change—were tested for accuracy against known benchmarks to ensure consistency across repeated executions.

Model prediction outputs were also a focus area, with the Random Forest, SVM, and Logistic Regression models tested individually and as part of the ensemble voting mechanism. The system had to handle model failures or conflicting outputs and return a final decision signal without crashing. Similarly, sentiment analysis was tested to ensure that the system could reliably fetch news headlines from Dawn and Brecorder, parse them, and classify sentiment using NLP models.

Any failure in scraping or classification was logged and handled gracefully. Furthermore, visualization components such as interactive graphs generated via Chart.js were tested across devices and browsers to confirm that the predicted signals, RSI, and SMA were rendered correctly. Backend responsiveness, endpoint stability, and overall system behavior under load were also assessed to meet performance expectations. These requirements formed the basis for comprehensive unit, functional, and integration testing throughout the development lifecycle of Stock Genie.

4.7.1.1 Welcome Page

Purpose	**To verify that the welcome page loads correctly with all interactive sections and displays accurate stock and feature content.**
Preconditions	User visits the Stock Genie homepage via a supported browser.
Test Data	No input required (static and dynamic content based on backend).
Steps	1. Navigate to the website's root URL (e.g., stockgenie.ai). 2. Observe all sections and visuals.
Expected Result	Welcome message, "What Makes StockGenie Special," gainers/losers data, and navigation bar should render properly.
Actual Result	All elements successfully loaded and displayed with proper formatting.
Status	**Pass** (Page displays correctly and gainers/losers module dynamically updates.)

Table 4.2: Home Page

4.7.1.2 About Us

Purpose	**To verify that the About Us page loads all its content sections correctly, including mission, features, help center, FAQs, and team profiles.**
Preconditions	User is logged in or visiting the public version of the website and clicks on the "About Us" link in the navbar.
Test Data	No user input required.
Steps	1. Click the "About Us" link from the top navigation bar. 2. Scroll through and validate all sections.
Expected Result	All subsections like "Who We Are," "What We Do," "Help Center," "FAQs," and "Meet the Team" should load properly.
Actual Result	All content was loaded successfully. No layout or functionality issues were observed.
Status	**Pass**

Table 4.3: About Us

4.7.1.3 Registration

Purpose	**To verify that the user registration process in Stock Genie functions correctly and successfully stores user credentials.**
Preconditions	The user is on the Stock Genie registration page.
Test Data	Email, Password, Confirm Password
Steps	1. Navigate to the registration page. 2. Enter the test data in the input fields. 3. Click the 'Register' button.
Expected Result	User should be registered and redirected to the login page.
Actual Result	Registration successful.
Status	Account registered successfully.

Table 4.4: Registration

4.7.1.4 Login

Purpose	**To verify that the login functionality correctly validates user credentials and grants access to authenticated users.**
Preconditions	The user must be registered and navigated to the login page.
Test Data	Email, Password
Steps	1. Enter the email and password. 2. Click the 'Login' button.
Expected Result	If credentials are valid, user is redirected to the dashboard. If invalid, an error message is shown.
Actual Result	Invalid email address message appears.
Status	**Fail** (invalid email format used; expected format includes '@' and domain)..

Table 4.5: Login

4.7.1.5 Gainer/Losers

Purpose	**To verify that the Gainers and Losers stocks list loads correctly with accurate values and formatting.**
Preconditions	User is logged in or on the public site and clicks on the "Gainers/Losers" link from the navbar.
Test Data	No input is required; system fetches data automatically from backend/database.
Steps	1. Navigate to the "Gainers/Losers" page. 2. Check if both tables load. 3. Verify symbols, last close, and % change.
Expected Result	Both tables (Gainers and Losers) should be populated with real-time data, sorted by percentage change.
Actual Result	Tables loaded successfully. Data sorted correctly. Visual formatting of positive (green) and negative (red) values applied.
Status	**Pass**

Table 4.6: Gainer/Losers

4.7.1.6 Technical Analysis Overview

Purpose	**To verify that the technical analysis interface accepts stock ticker input and initiates prediction workflow correctly.**
Preconditions	User is on the Technical Analysis page.
Test Data	Stock Ticker: PSO, OGDC, LUCK, etc.
Steps	1. Enter a valid PSX stock ticker in the input field. 2. Click the "Predict" button. 3. Observe system response.
Expected Result	Page should transition to display technical indicators and prediction result for the entered ticker.
Actual Result	The system correctly fetched historical data, computed indicators (RSI, SMA, % change), and displayed predictions.
Status	**Pass**

Table 4.7: Technical Analysis Overview

4.7.1.7 Technical Analysis Prediction

Purpose	**To verify that the system correctly displays technical indicators, model predictions, and stock chart after a valid ticker input.**
Preconditions	A valid stock ticker (e.g., PSO) has been submitted via the technical analysis form.
Test Data	Stock Ticker: PSO
Steps	1. Enter a valid ticker in the input field. 2. Click "Predict". 3. System fetches data, computes results, and displays them.
Expected Result	The user should see: 1. Key features (RSI, SMA-30, Price Change). 2. Model-wise predictions (RF, SVM, LR). 3. Final signal 4. Stock chart rendered with Chart.js.
Actual Result	All indicators were computed, model predictions generated, and final ensemble decision shown along with an accurate stock chart.
Status	**Pass**

Table 4.8: Technical Analysis Prediction

4.7.1.8 Sentimental Analyzer

Purpose	**To verify that clicking "Check Market Sentiment" triggers backend scraping and analysis of financial headlines.**
Preconditions	User must be on the Sentiment Analysis page of the website.
Test Data	Button Click: Check Market Sentiment
Steps	1. Navigate to the Sentiment Analysis page. 2. Click the "Check Market Sentiment" button.
Expected Result	System should start scraping news from Dawn and Brecorder, perform sentiment analysis, and return results.
Actual Result	Sentiment analysis triggered successfully, analysis completed, and next result view shown.
Status	**Pass**

Table 4.9: Sentimental Analyzer

4.7.1.9 Sentiment of the Market

Purpose	**To verify that after sentiment analysis is triggered, headlines and sentiment labels are accurately displayed.**
Preconditions	Sentiment scraping and classification process must complete successfully.
Test Data	Financial headlines and classification results from Dawn and Brecorder.
Steps	1. Click "Check Market Sentiment". 2. Wait for analysis to complete. 3. Observe returned sentiment and headlines.
Expected Result	Sentiment label (e.g., Positive, Negative, Neutral) and headline list with links should be shown clearly.
Actual Result	Market Sentiment: Negative label displayed along with a list of 30+ headlines and source links.
Status	**Pass**

Table 4.10: Sentiment of the Market

4.8 Algorithms

4.8.1 Authentication Module

4.8.1.1 Authentication System

Algorithm 1.1: Authentication System

Input: Transition system of each member $\{T_1, T_2, T_3\}$

Output: Combined Transition System of members that is **T**

1: Let $m = 1, 2, 3$
2: $q_T^0 = q_m^0$
3: Recursive search on **T** starting from initial state (q_T^0)
4: $q_m \in q$
5: The transition of T_m is defined as $\rightarrow_m$, where $\rightarrow_m = (q, q')$
6: **if** $\rightarrow_m \in \tau$ **then** ▷ τ: set of possible transitions
7: $\omega \leftarrow$ minimum weight of $\rightarrow_m$ when the robot finds some region
8: Find new state of transition system q'; if q' doesn't exist:
9: Insert state q' into **T**
10: Insert transition into δ_T with weight ω ▷ δ_T is the set of transitions
11: Continue search from q'
12: **else**
13: **if** no transition exists for (q, q') **then**
14: Insert transition (q, q') into δ_T with weight ω
15: **end if**
16: **end if**

4.8.1.2 User Registration

Algorithm 1.2: User Registration

Input: {username, email, password, password_confirmation}

Output: New user account or registration error

1: Validate input fields:

- Verify username meets length and character requirements
- Verify email format
- Verify password complexity
- Check password matches confirmation

2: Check if username or email exists in database

3: **if** exists **then**

4: Return to q_0 with duplicate account error

5: **end if**

6: Generate password hash using secure hashing algorithm

7: Create new user record with:

- Username
- Email
- Password hash
- Registration timestamp
- Default user role

8: Store user record in database

9: **if** storage successful **then**

10: Transition to q_2 (authenticated state)

11: Create user session

12: Return success message

13: **else**

14: Transition to q_3 (authentication failed state)

15: Return system error message
16: **end if**

4.8.1.3 User Login Authentication

Algorithm 1.3: User Login Authentication

Input: {username/email, password}

Output: Authentication success or failure with session token

1: Transition from q_0 to q_1 (credentials submitted)
2: Query database for account matching username/email
3: **if** user not found **then**
4: Transition to q_3 with "Invalid credentials" message
5: **return** failure
6: **end if**
7: Retrieve stored password hash for the user
8: Apply same hashing algorithm to input password
9: Compare generated hash with stored hash
10: **if** match **then**
11: Transition to q_2 (authenticated)
12: Generate session token
13: Store session data with expiration time
14: Return success with session token
15: Update last login timestamp
16: **else**
17: Transition to q_3 (authentication failed)
18: Log failed attempt
19: Return "Invalid credentials" message
20: **end if**

4.8.2 Technical Analysis

Algorithm 2: Technical Analysis Prediction

Input: Stock ticker symbol, Time period for analysis

Output: Buy/Sell/Hold signal with supporting technical indicators

Define system states $Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$
    q_0: Initial, q_1: Data fetch, q_2: Feature engineering,
    q_3: Model prediction, q_4: Signal integration, q_5: Error
Initialize in q_0
Validate ticker symbol
if invalid **then**
    Transition to q_5, return "Invalid ticker"
else
    Continue
end if
Transition to q_1: fetch OHLCV data
if unavailable **then**
    Transition to q_5, return "Data unavailable"
end if
Transition to q_2: Engineer features
Calculate SMA, EMA (periods 9, 20, 50, 200)
Calculate RSI, MACD, Stochastic
Compute Bollinger Bands, ATR
Extract OBV, Volume MA
Derive ratios/divergences
Normalize using standard scaler
Transition to q_3: Model predictions
Load Model 1 (RF), get prediction_1
Load Model 2 (SVM), get prediction_2
Load Model 3 (LR), get prediction_3

Transition to q_4: Ensemble output
Combine via weighted sum or majority vote
if score ¿ threshold **then**
    Signal = BUY
else if score ¡ threshold **then**
    Signal = SELL
else
    Signal = HOLD
end if
Return signal, indicator values, and chart data

4.8.3 Sentimental Analysis

Algorithm 3: Sentiment Analysis

Input: User request to check sentiment

Output: Overall market sentiment and supporting headlines

Define states $Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$
    q_0: Initial, q_1: Scraping, q_2: Preprocessing,
    q_3: Classification, q_4: Display, q_5: Error
Start at q_0
On "Check Market Sentiment" click, transition to q_1
Scrape headlines:
Initialize empty `headlines_collection`
Try Dawn and Brecorder
if network fails **then**
    Transition to q_5, return "Scraping error"
else if no headlines **then**
    Transition to q_5, return "No headlines"
else
    Add headlines to collection
end if
Transition to q_2: Preprocess headlines
Remove special characters, lowercase
Remove stopwords, apply lemmatization
Transition to q_3: Classify sentiment
Load NLP model
for all headline in collection **do**
    Predict sentiment (Positive / Neutral / Negative)
end for
Aggregate scores to determine overall sentiment
if model fails **then**

Transition to q_5, return "Model error"
else
Transition to q_4
end if
Display:
Show sentiment (Positive/Neutral/Negative)
Show headlines + their classifications
Link to original sources
On error (q_5): Show message, allow retry

4.8.4 Gainers & Losers

Algorithm 4: Gainers and Losers Analysis

Input: Stock market data from Firebase

Output: Two sorted lists of stocks (Gainers and Losers) with performance metrics

Define states $Q = \{q_0, q_1, q_2, q_3, q_4\}$

q_0: Initial, q_1: Data acquisition, q_2: Processing, q_3: Display, q_4: Error

Start in q_0

On user navigation to Gainers/Losers section:

Transition to q_1

In q_1: Fetch data from Firebase

Connect to Firebase

Request all current PSX stock data

if connection fails **then**

    Transition to q_4, return "Internet issue"

else if data is empty or corrupt **then**

    Transition to q_4, return "Stock data unavailable or corrupt"

else

    Transition to q_2

end if

In q_2: Process stock data

Initialize empty lists: `gainers`, `losers`

for all stock in retrieved data **do**

    Extract: `symbol`, `last_close`, `previous_close`

    Compute: `change_pct` $= \frac{(last\_close - previous\_close)}{previous\_close} \times 100$

    Add stock object (symbol, last_close, change_pct) to list

end for

Sort list by `change_pct`

if sorting fails **then**

    Transition to q_4, return "Sorting failed"

else

Extract top N gainers (positive change) to `gainers`
Extract top N losers (negative change) to `losers`
Transition to q_3
end if
In q_3: Display results
Round change percentages to 2 decimal places
Apply green to gainers, red to losers
Render display with gainers and losers
Return formatted list to user
In q_4: Error state
Show relevant error message
Provide retry option
If cached data exists, display it as fallback

4.8.5 Random Forest Model Prediction

Algorithm 5: Random Forest Model Prediction

Input: Stock ticker symbol

Output: Trading signal (Buy, Sell, or Neutral)

1: Define states $Q = \{q_0, q_1, q_2, q_3, q_4\}$

q_0: Initial, q_1: Feature preparation, q_2: Model prediction, q_3: Result processing, q_4: Error

2: Start in q_0 and transition to q_1

3: In q_1 (Feature Preparation):

4: Calculate technical indicators:

- RSI (14-day)
- SMA (30-day)
- Price change percentage (1-day)

5: Normalize features

6: **if** feature vector is incomplete **then**

7: Transition to q_4, return error

8: **else**

9: Transition to q_2

10: **end if**

11: In q_2 (Model Prediction):

12: Load pretrained Random Forest model:

- Try "rf_stock_signal_model.joblib"
- Try fallback models if primary fails

13: Feed feature vector to model

14: Generate prediction

15: Transition to q_3

16: In q_3 (Result Processing):

17: Map numerical output:

- $0 \rightarrow$ "Sell"
- $1 \rightarrow$ "Buy"
- $2 \rightarrow$ "Neutral"

if output is unexpected **then**
    Default to "Neutral"
end if
Forward result to ensemble system
Return final signal
In q_4 (Error State):
Log error details
Return "Neutral" as fallback
Report error to system

4.8.6 Random Forest Model Prediction

Algorithm 5: Random Forest Model Prediction

Input: Stock ticker symbol

Output: Trading signal (Buy, Sell, or Neutral)

1: Define states $Q = \{q_0, q_1, q_2, q_3, q_4\}$

q_0: Initial, q_1: Feature preparation, q_2: Model prediction, q_3: Result processing, q_4: Error

2: Start in q_0 and transition to q_1

3: In q_1 (Feature Preparation):

4: Calculate technical indicators:

- RSI (14-day)
- SMA (30-day)
- Price change percentage (1-day)

5: Normalize features

6: **if** feature vector is incomplete **then**

7: Transition to q_4, return error

8: **else**

9: Transition to q_2

10: **end if**

11: In q_2 (Model Prediction):

12: Load pretrained Random Forest model:

- Try "rf_stock_signal_model.joblib"
- Try fallback models if primary fails

13: Feed feature vector to model

14: Generate prediction

15: Transition to q_3

16: In q_3 (Result Processing):

17: Map numerical output:

- 0 → "Sell"
- 1 → "Buy"
- 2 → "Neutral"

if output is unexpected **then**
    Default to "Neutral"
end if
Forward result to ensemble system
Return final signal
In q_4 (Error State):
Log error details
Return "Neutral" as fallback
Report error to system

4.8.7 Support Vector Machine (SVM) Model Prediction

Algorithm 6: SVM Model Prediction

Input: Stock ticker symbol

Output: Trading signal (Buy, Sell, or Neutral)

1: Define states $Q = \{q_0, q_1, q_2, q_3, q_4\}$

q_0: Initial, q_1: Feature preparation, q_2: Model prediction, q_3: Result processing, q_4: Error

2: Start in q_0 and transition to q_1

3: In q_1 (Feature Preparation):

4: Calculate technical indicators:

- RSI (14-day)
- SMA (30-day)
- 1-day price change

5: Clean and standardize features

6: **if** feature shape is invalid **then**

7: Transition to q_4, return error

8: **else**

9: Transition to q_2

10: **end if**

11: In q_2 (Model Prediction):

12: Load pretrained SVM model:

- Try "svm_model_buy_sell_final.pkl"
- Try fallback if primary fails

13: Generate prediction using `model.predict()`

14: **if** kernel computation fails **then**

15: Default to "Neutral"

16: **end if**

17: Transition to q_3

18: In q_3 (Result Processing):

19: Check that prediction $\in \{0, 1, 2\}$

20: Map prediction:

- $0 \rightarrow$ "Sell"
- $1 \rightarrow$ "Buy"
- $2 \rightarrow$ "Neutral"

21: Forward result to ensemble voting system

22: Return final signal

23: In q_4 (Error State):

24: Log error details

25: Return "Neutral" as fallback

26: Report error to system

4.8.8 Logistic Regression Prediction

Algorithm 7: Logistic Regression Prediction

Input: Stock ticker symbol, Historical price data

Output: Trading signal (Buy, Sell, or Neutral)

Define states $Q = \{q_0, q_1, q_2, q_3, q_4\}$

q_0: Initial, q_1: Feature preparation, q_2: Model prediction, q_3: Result processing, q_4: Error

Start in q_0 and transition to q_1

In q_1 (Feature Preparation):

Calculate RSI (14-day)

Calculate SMA (30-day)

Calculate price change percentage (1-day)

Validate feature vector format

if malformed **then**

    Transition to q_4 and return error

else

    Transition to q_2

end if

In q_2 (Model Prediction):

Try to load logistic_regression_model_buy_sell.pkl

if failed **then**

    Try fallback logistic_regression_model2.pkl

    if also failed **then**

        Transition to q_4

    end if

end if

Pass features to model and generate prediction

Transition to q_3

In q_3 (Result Processing):

if prediction = 0 **then**
    Return "Sell"
else if prediction = 1 **then**
    Return "Buy"
else if prediction = 2 **then**
    Return "Neutral"
else
    Return "Neutral"
end if
Forward result to ensemble voting system
In q_4 (Error):
Log error
Return "Neutral" as fallback
Report error to calling system

4.8.9 Ensemble Prediction using Majority Voting

Algorithm 8: Ensemble Prediction with Majority Voting

Input: Feature vector derived from stock technical indicators

Output: Final trading signal (Buy/Sell) based on majority vote

Define states $Q = \{q_0, q_1, q_2, q_3, q_4\}$
q_0: Initial state (no ensemble prediction)
q_1: Individual model prediction
q_2: Vote tallying
q_3: Result determination
q_4: Error handling
Start in q_0 and transition to q_1
In q_1:
Verify feature vector format
Attempt predictions from all models
$pred_{log} \leftarrow$ Logistic Regression prediction
$pred_{svm} \leftarrow$ SVM prediction
$pred_{rf} \leftarrow$ Random Forest prediction
if all predictions succeed **then**
    Transition to q_2
else if at least one succeeds **then**
    Log model failures
    Continue with available predictions
    Transition to q_2
else
    Transition to q_4
end if
In q_2:
Collect all available predictions
Count occurrences of each prediction (0 = Buy, 1 = Sell)

Transition to q_3
In q_3:
Determine the most common vote
if tie occurs **then**
    if Random Forest prediction available **then**
        Use $pred_{rf}$ as tiebreaker
    else
        Default to "Neutral"
    end if
end if
if final prediction = 0 **then**
    Return "Buy"
else if final prediction = 1 **then**
    Return "Sell"
else
    Return "Neutral"
end if
Also return individual model predictions and consensus signal
In q_4:
Log ensemble system failure
Return appropriate error message
Hide prediction result from user

4.8.10 Technical Analysis Scraper

Algorithm 9: Technical Analysis Scraper

Input: Stock ticker symbol, Start date, End date

Output: Cleaned and structured historical stock data with technical indicators

Define states $Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$
  q_0: Initial, q_1: Connection, q_2: Data Retrieval, q_3: Parsing, q_4: Preprocessing, q_5: Error
Start in q_0
Validate input parameters:
  Check if ticker symbol is present and formatted
  Check if date range is valid
if invalid **then**
  Transition to q_5 with error
else
  Transition to q_1
end if
In q_1 (Connection):
  Initialize HTTP session to PSX site
  Generate date list from start to end date
if connection successful **then**
  Transition to q_2
else
  Transition to q_5 with "PSX site down" error
end if
In q_2 (Data Retrieval):
for each date in date list **do**
  Create worker thread with ThreadPoolExecutor
  Send POST request to PSX with ticker, month, and year

23: Collect response
24: **end for**
25: **if** all data retrieved **then**
26: Transition to q_3
27: **else**
28: Transition to q_5 with HTTP error
29: **end if**
30: In q_3 (Parsing):
31: **for** each response **do**
32: Parse HTML using BeautifulSoup
33: Extract table rows and columns
34: Convert date strings to datetime
35: Structure into DataFrame
36: Set TIME column as index
37: **end for**
38: Combine all dataframes
39: **if** successful **then**
40: Transition to q_4
41: **else**
42: Transition to q_5 with parsing error
43: **end if**
44: In q_4 (Preprocessing):
45: Sort DataFrame by date
46: Rename columns
47: Remove commas in numeric fields
48: Convert strings to float
49: Compute indicators: SMA_30, RSI, Price_Change_Pct
50: Drop rows with missing values
51: Return final preprocessed DataFrame
52: In q_5 (Error):
53: Log error details

```
54:     Return error message
55:     Trigger retry mechanism if needed
```

4.8.11 Sentimental Analysis Scraper

Algorithm 10: Sentimental Analysis Scraper

Input: User request to check sentiment

Output: Collection of cleaned financial news headlines

Define states $Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$

q_0: Initial, q_1: Dawn Scrape, q_2: Recorder Scrape, q_3: HTML Parse, q_4: Clean, q_5: Error

Start in q_0

On user action "Check Sentiment", transition to q_1

In q_1 (Dawn scraping):

    Initialize empty headlines_collection

    Create HTTP session with headers

    Send GET request to Dawn business section

if successful **then**

        Store response and transition to q_2

else

        Log "Dawn website unavailable" and continue to q_2

end if

In q_2 (Business Recorder scraping):

    Create HTTP session with headers

    Send GET request to Business Recorder news

if successful **then**

        Store response and transition to q_3

else if Dawn data available **then**

        Transition to q_3

else

        Transition to q_5 with "All sources unavailable"

end if

In q_3 (HTML parsing):

24: **for** each valid response **do**
25: Parse HTML with BeautifulSoup
26: Extract headlines using CSS selectors
27: Store headlines with source info
28: **end for**
29: **if** headlines found **then**
30: Transition to q_4
31: **else**
32: Transition to q_5 with "No headlines found"
33: **end if**
34: In q_4 (Cleaning headlines):
35: **for** each headline **do**
36: Remove HTML tags
37: Strip whitespace
38: Remove special characters
39: Filter by financial keywords
40: **end for**
41: Return cleaned headlines collection
42: In q_5 (Error):
43: Log detailed error info
44: Attempt retry once if failure due to timeout
45: Return error message to system
46: If partial headlines exist, return with warning

4.8.12 Technical Indicator Calculation

Algorithm 11: Technical Indicator Calculation

Input: Historical stock data (OHLCV data)

Output: Feature vector containing computed technical indicators

Define states $Q = \{q_0, q_1, q_2, q_3, q_4\}$

q_0: Initial, q_1: Data validation, q_2: Indicator calculation, q_3: Feature construction, q_4: Error

Start in q_0 and transition to q_1

In q_1 (Data validation):

Check if DataFrame has columns: Date, Open, High, Low, Close, Volume

Verify at least 30 days of data

if insufficient **then**

Transition to q_4 with error "insufficient data"

end if

Check for NaN values in Close price

if missing **then**

Apply handling strategy:

Drop rows **or** fill with previous value

end if

If validation passes, transition to q_2

In q_2 (Indicator calculation):

RSI calculation:

Compute price changes

Separate gains and losses

Average gain/loss over 14-day window

$RS = \frac{\text{avg gain}}{\text{avg loss}}$

$RSI = 100 - \left(\frac{100}{1+RS}\right)$

SMA_30 calculation:

30-day rolling mean of Close prices
Price Change Percentage:
Price_Change_Pct = Close.pct_change()
if calculation fails **then**
    Log error and transition to q_4
else
    Transition to q_3
end if
In q_3 (Feature construction):
    Extract latest RSI, SMA_30, Price Change
    Combine into vector: $features = [RSI, SMA30, ChangePct]$
    Validate vector is complete
    Return feature vector
In q_4 (Error):
    Log error details
    Return error message
    Notify prediction module of failure

4.8.13 NLP Sentiment Classification

Algorithm 12: NLP Sentiment Classification

Input: Collection of cleaned financial news headlines

Output: Overall market sentiment classification (Positive, Neutral, or Negative)

Define states $Q = \{q_0, q_1, q_2, q_3, q_4\}$
    q_0: Initial, q_1: Preprocessing, q_2: Classification, q_3: Aggregation, q_4: Error
Start in q_0 and transition to q_1 if input is valid
Validate input:
    Check if headlines list exists and is not empty
if empty **then**
    Transition to q_4 with error "Input text is empty"
else
    Transition to q_1
end if
In q_1 (Preprocessing):
for each headline in collection **do**
    Tokenize text into words
    Remove stop words
    Apply stemming or lemmatization
    Transform into model input format
end for
if preprocessing successful **then**
    Transition to q_2
else
    Log failure, skip invalid headlines
    if all headlines fail **then**
        Transition to q_4 with error "Text preprocessing fails"

```
    end if
end if
In q2 (Classification):
Try loading NLP sentiment model
if model fails then
    Transition to q4 with error "Model not loaded"
end if
for each preprocessed headline do
    Feed headline into model
    Retrieve sentiment: Positive, Neutral, or Negative
    Store result with headline
end for
Transition to q3
In q3 (Aggregation):
Count number of Positive, Neutral, and Negative headlines
Apply majority voting or weighted scoring
if Positive is majority then
    Set overall sentiment to "Positive"
else if Negative is majority then
    Set overall sentiment to "Negative"
else
    Set overall sentiment to "Neutral"
end if
Format result with:
  Overall sentiment
  Distribution counts
  Headlines with classifications
Return result to frontend
In q4 (Error):
Log detailed error info
Set sentiment to "Neutral" as fallback
```

54: Return error to calling system

4.8.14 Handle Invalid Input

Algorithm 13: Handle Invalid Input

Input: User-submitted stock ticker symbol

Output: Validation result (valid/invalid) with appropriate message

Define states $Q = \{q_0, q_1, q_2, q_3, q_4\}$

q_0: Initial input, q_1: Format validation, q_2: Existence validation, q_3: Valid, q_4: Error

Start in q_0

Check if input exists

if input is null or empty **then**

    Transition to q_4 with error "Please enter a stock symbol"

else

    Transition to q_1

end if

In q_1 (Format Validation):

    Remove whitespace from input

    Check alphanumeric pattern

    Validate length (3–5 characters)

if format invalid **then**

    Transition to q_4 with error "Invalid ticker format"

else

    Transition to q_2

end if

In q_2 (Existence Validation):

    Load valid PSX symbols list

    Check if ticker exists in list

if not found **then**

    Transition to q_4 with error "Invalid or unlisted stock"

else

Transition to q_3
end if
In q_3 (Valid Result):
    Mark ticker as valid
    Allow prediction to proceed
    Return success with validated symbol
In q_4 (Error):
    Prepare user-friendly error message
    Block prediction logic
    Return error status
    Preserve user input in form
    Log validation error

Chapter 5

Conclusion & Future Work

5.1 Discussion

The development and deployment of the Stock Genie platform represent a significant step toward integrating artificial intelligence with local stock market analysis—specifically for the Pakistan Stock Exchange (PSX). The project effectively combined sentiment analysis, technical indicators, and machine learning predictions to provide clear Buy/Sell/Neutral signals for investors. Through the incorporation of ensemble models (Random Forest, Logistic Regression, and SVM), the system achieved a balanced and intelligent decision-making process tailored for a market often underserved by AI-driven tools.

Moreover, Stock Genie prioritized user-centric design by offering an interactive web interface powered by Flask and Chart.js. This allowed both novice and seasoned investors to visualize predictions, trends, and market sentiment in an intuitive format. The modular architecture ensured the project's scalability and maintainability. At the same time, the integration of local news sources, such as Dawn and Brecorder, added contextual relevance to the sentiment output—something global tools often lack. In essence, the project demonstrates how AI and NLP can be leveraged locally to facilitate more informed financial decision-making.

5.2 Limitations

Despite its successful implementation, Stock Genie faced several limitations during development and testing. Firstly, the reliance on static scraping techniques for PSX data and financial news introduces fragility—any change in site structure could break the modules. Secondly, the sentiment analysis model is trained on general-purpose NLP techniques rather than domain-specific financial corpora, which may slightly affect classification accuracy for niche headlines.

Moreover, real-time performance is currently constrained by the local execution of scraping and prediction logic, which may not scale well under high usage without cloud infrastructure. The system also assumes the availability of six months of historical data for every stock, which may not always be the case for newly listed or illiquid stocks. Lastly, authentication is basic and may require further hardening for production environments, especially if sensitive user data or investment decisions are tied to the platform.

Chapter 6

Discussion and Conclusion

6.1 Conclusion

The development of Stock Genie marks a significant step toward empowering investors in the Pakistan Stock Exchange (PSX) with AI-driven decision support. By integrating machine learning models, technical indicators, and sentiment analysis, the platform delivers clear and actionable Buy and Sell signals in a user-friendly web interface. Stock Genie automates complex stock evaluation processes by scraping PSX historical data, calculating financial indicators like RSI and SMA, and analyzing market news sentiment—all presented visually through dynamic charts.

The project successfully achieved its goal of simplifying stock research for both novice and experienced investors by merging financial computation, machine learning, and local news sentiment into one centralized system. Throughout development, emphasis was placed on modularity, transparency, and responsiveness—making Stock Genie a reliable academic prototype with real-world relevance for the Pakistani stock market.

6.2 Future Work

While Stock Genie effectively meets its current objectives, several enhancements are planned for future iterations. A significant area for improvement involves replacing static scraping with live PSX API integrations to enhance reliability and minimize data-fetching errors. Additionally, the sentiment analysis module could be improved by training custom NLP models on finance-specific datasets in Urdu and English for higher contextual accuracy.

Other future enhancements include implementing user dashboards to track saved predictions, enhancing authentication with Firebase or OAuth2, enabling real-time updates with Firebase or WebSockets, and deploying the platform on cloud infrastructure to support concurrent users. Further, integrating portfolio recommendations, voice-based search, or WhatsApp chatbot access are possible innovations to make Stock Genie even more accessible and intelligent.

Bibliography

Index