



SUPERVISOR'S RECOMMENDATION

I hereby recommend that the report prepared under my supervision by Nilesh Nath (TU Exam Roll No. 26370/077), Nishchal Karki (TU Exam Roll No. 26371/077), Rajan Shrestha (TU Exam Roll No. 26376/077) entitled “**STOCK PRICE PREDICTION SYSTEM USING LSTM**” in partial fulfilment of the requirements for the degree of B.Sc.Computer Science and Information Technology be processed for evaluation.

.....
Hemanta Mehta,
Department of CSIT
Orchid International College
Bijayachowk, Gaushala



CERTIFICATE OF APPROVAL

This is to certify that this project prepared by Nilesh Nath, Nishchal Karki and Rajan Shrestha entitled "Stock Price Prediction System Using LSTM" in partial fulfilment of the requirements for the degree of B.Sc.Computer Science and Information Technology has been well studied. In our opinion it is satisfactory in the scope and quality as a project for the required degree.

<p>.....</p> <p>Hemanta Mehta</p> <p>Supervisor,</p> <p>Orchid International College</p> <p>Bijayachowk, Gaushala</p>	<p>.....</p> <p>Er. Dhiraj Kumar Jha</p> <p>Head of Deptment,</p> <p>Department of IT</p> <p>Orchid International College</p> <p>Bijayachowk, Gaushala</p>
<p>.....</p> <p>Mr. Jagdish Bhatta</p> <p>External Examiner</p> <p>Central Department of Computer Science and IT,</p> <p>Tribhuvan University,</p> <p>Kirtipur, Nepal</p>	

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to everyone who has assisted us on our journey to completion of this project. We are deeply thankful to Hemanta Mehta, our project supervisor, for his invaluable guidance, support, and encouragement throughout the course of this project. His exceptional expertise and mentorship have been immensely helpful to us in overcoming many challenges.

We would also like to extend our appreciation to the Department of Computer Science and Information Technology, Orchid International College for providing the necessary resources and a helpful environment for developing this project.

Lastly, we would like to express our thanks to the teachers, mentors and our colleagues for their support, understanding, and encouragement throughout this endeavour.

Sincerely,

Nilesh Nath (26370/077),

Nishchal Karki (26371/077),

Rajan Shrestha (26376/077)

ABSTRACT

Stock price forecasting is a difficult but critical undertaking in financial markets, allowing investors to make informed decisions. This project provides a stock price prediction system based on Long Short-Term Memory (LSTM), a form of Recurrent Neural Network (RNN) that can detect long-term dependencies in time-series data. The system takes past stock price data, normalizes it, and trains an LSTM model to forecast future stock values. To test the model's performance, many error metrics have been applied, including Mean Squared Error (MSE), Root Mean Squared Error (RMSE) and Mean Absolute Percentage Error (MAPE). The results reveal that the LSTM model accurately captures stock price movements. This study highlights the importance of deep learning in predictive analytics by demonstrating how LSTM models may provide useful insights into stock market behavior using advanced pattern recognition and trend analysis.

Keywords: Finance, LSTM, MSE, MAPE, RMSE, RNN

TABLE OF CONTENTS

SUPERVISOR’S RECOMMENDATION	i
CERTIFICATE OF APPROVAL	ii
ACKNOWLEDGEMENT.....	iii
ABSTRACT.....	iv
TABLE OF CONTENTS	v
LIST OF FIGURES.....	vii
LIST OF TABLES	viii
LIST OF ABBREVIATIONS	ix
CHAPTER 1 INTRODUCTION.....	1
1.1 Introduction	1
1.2 Problem Statement	1
1.3 Objectives.....	1
1.4 Scope and Limitations	2
1.5 Development Methodology	2
1.6 Report Organization	4
CHAPTER 2 BACKGROUND STUDY AND LITERATURE REVIEW.....	6
2.1 Background Study.....	6
2.2 Literature Review.....	6
CHAPTER 3 SYSTEM ANALYSIS.....	8
3.1 System Analysis	8
3.1.1 Requirement Analysis.....	8
3.1.1.1 Functional Requirements	8
3.1.1.2 Non Functional Requirements.....	11
3.1.2 Feasibility Analysis	12
3.1.2.1 Technical Feasibility	12
3.1.2.2 Operational Feasibility	12
3.1.2.3 Schedule Feasibility	12
3.1.3 Analysis.....	14
CHAPTER 4 SYSTEM DESIGN.....	19
4.1 Design... ..	19
4.1.1 Refinement of Class Diagram	26

4.1.2 Refinement of Activity Diagram.....	28
4.1.3 Refinement of Sequence Diagram	29
4.1.4 Component Diagram	30
4.1.5 Deployment Diagram	31
4.2 Algorithm Details	32
4.2.1 The Problem LSTM Solves	32
4.2.2 LSTM Architecture	32
4.2.3 How LSTM Works in Stock Price Prediction:	34
CHAPTER 5 IMPLEMENTATION AND TESTING	35
5.1 Implementation.....	35
5.1.1 Tools used... ..	35
5.1.2 Implementation Details	36
5.2 Testing.....	38
5.2.1 Unit Testing.....	38
5.2.2 Integration Testing	40
5.2.3 System Testing.....	41
5.3 Result Analysis.....	43
5.3.1 Mean Squared Error (MSE).....	43
5.3.2 Root Mean Squared Error (RMSE).....	43
5.3.3 Mean Absolute Percentage Error (MAPE).....	44
CHAPTER 6 CONCLUSION AND FUTURE RECOMMENDATION	46
6.1 Conclusion.....	46
6.2 Future Recommendation	46
REFERENCES.....	47
APPENDIX	48

LIST OF FIGURES

Figure 1.1 Incremental Model.....	3
Figure 3.1 Use Case Diagram.....	9
Figure 3.2 Work Breakdown Structure.....	13
Figure 3.3 Gantt Chart.....	13
Figure 3.4 Class Diagram.....	15
Figure 3.5 Sequence Diagram.....	16
Figure 3.6 Activity Diagram.....	17
Figure 4.1 High-Level System Design.....	19
Figure 4.2 Overview of the Dataset of AHPC.....	20
Figure 4.3 Information on the Data of AHPC.....	21
Figure 4.4 Stock Close Price Over Time of AHPC.....	22
Figure 4.5 Moving Averages of AHPC.....	23
Figure 4.6 Correlation Matrix of AHPC.....	24
Figure 4.7 Refined Class diagram.....	27
Figure 4.8 Refined Activity diagram.....	29
Figure 4.9 Refined Sequence diagram.....	30
Figure 4.10 Component Diagram.....	31
Figure 4.11 Deployment diagram.....	31
Figure 4.12 Architecture of LSTM.....	34
Figure 5.1 Implementation Process.....	36
Figure 5.2 Convergence graph of LSTM Model.....	45

LIST OF TABLES

Table 3.1 Search stock ticker.....	9
Table 3.2 Predict stock price.....	10
Table 3.3 Return prediction results.....	10
Table 3.4 Visualize results.....	10
Table 3.5 View Precious Predictions.....	11
Table 5.1 Tools used.....	35
Table 5.2 External modules.....	35
Table 5.3 Performance metrics.....	38
Table 5.4 Test get_stock_data function.....	39
Table 5.5 Test mse function.....	39
Table 5.6 Test rmse function.....	39
Table 5.7 Test mape function.....	40
Table 5.8 Test for valid ticker.....	40
Table 5.9 Test /get_available_tickers.....	41
Table 5.10 Test integration of stockform component with API.....	41
Table 5.11 End-to-end test for stock prediction	42
Table 5.12 Test application health check.....	42
Table 5.13 Test user interface for error handling	42
Table 5.14 Result Analysis on AHPC	44

LIST OF ABBREVIATIONS

ANN	Artificial Neural Networks
ARIMA	Autoregressive Integrated Moving Average
CASE	Computer Aided Software Engineering
IDE	Integrated Development Environment
LSTM	Long Short Term Memory
MAPE	Mean Absolute Percentage Error
MSE	Mean Squared Error
NEPSE	Nepal Stock Exchange
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Network
SPA	Single Page Application
UI	User Interface
UML	Unified Modeling Language

CHAPTER 1 INTRODUCTION

1.1 Introduction

Stock price prediction is a popular topic in financial markets because effective forecasting can help investors make informed trading decisions and reduce risk. However, stock prices are very volatile and impacted by a variety of factors, including market movements, economic indicators, and investor emotions, making forecasting difficult. Traditional statistical models frequently fail to capture the nonlinear dependencies and long-term trends in stock price movements.

In this project, LSTM was used to create a Stock Price Prediction System that forecasts future prices by utilizing historical stock price data. To make sure the model discovers significant patterns, the data is subjected to preprocessing procedures. Metrics like Mean Squared Error (MSE), Root Mean Squared Error (RMSE) and Mean Absolute Percentage Error (MAPE) are used to evaluate the accuracy and dependability of the LSTM model.

1.2 Problem Statement

The stock market, known for its volatility and complex dynamics, presents a significant challenge for investors aiming to predict future stock prices. With the advent of advanced machine learning techniques, particularly Long Short-Term Memory (LSTM) networks, there is potential to improve the accuracy of these predictions significantly. LSTM, a type of recurrent neural network (RNN), excels in capturing temporal dependencies and patterns, making it an ideal choice for time-series forecasting tasks such as stock price prediction. This project, titled "Stock Market Prediction Using LSTM," aims to leverage the power of LSTM networks to forecast stock prices more accurately. By developing a model from scratch, this project has provided insights of stock market behavior and offered a robust predictive tool for investors and financial analysts.

1.3 Objectives

The primary objectives of the project are:

- To develop a stock price prediction system using Long Short-Term Memory (LSTM) for accurate forecasting.
- To train and evaluate the LSTM model using performance metrics like MSE, RMSE, and MAPE.
- To compare LSTM predictions with actual stock prices and assess model accuracy.

1.4 Scope and Limitations

The stock price prediction model, particularly one based on Long Short-Term Memory (LSTM) networks, can be applied to various other areas beyond stock market predictions.

They are as follows:

- LSTM models can analyze historical weather data to predict future weather conditions, such as temperature, rainfall, and humidity.
- LSTM networks can be used for tasks like language translation, sentiment analysis, and text generation by learning from sequences of words or characters.
- Businesses can use LSTM to predict future sales based on past sales data, helping with inventory management and demand planning.

The limitations of the system are as follows:

- Depends on the quality of historical data, and missing data can affect predictions.
- Does not include real-time market factors like news or sentiment.
- May not generalize well to other stocks or different time periods without retraining.

1.5 Development Methodology

Software was developed using the incremental development methodology, which builds a system gradually in digestible chunks called increments. Each increment enhances the current system with new features or functionality rather than planning and building the full system at once. Each step starts with developers planning and designing particular features. They then go to the next increment of the system after building and testing that portion. The new functionality is assessed at each level, and input is obtained to enhance subsequent iterations. This method provides flexibility by enabling quicker delivery of functional elements and allowing for adjustments to be made at any point in response to feedback. By concentrating on smaller, more manageable components of the system, it also lowers risk by guaranteeing that any problems may be fixed early on, producing a more dependable end result.

The following figure shows the different phases of incremental development:

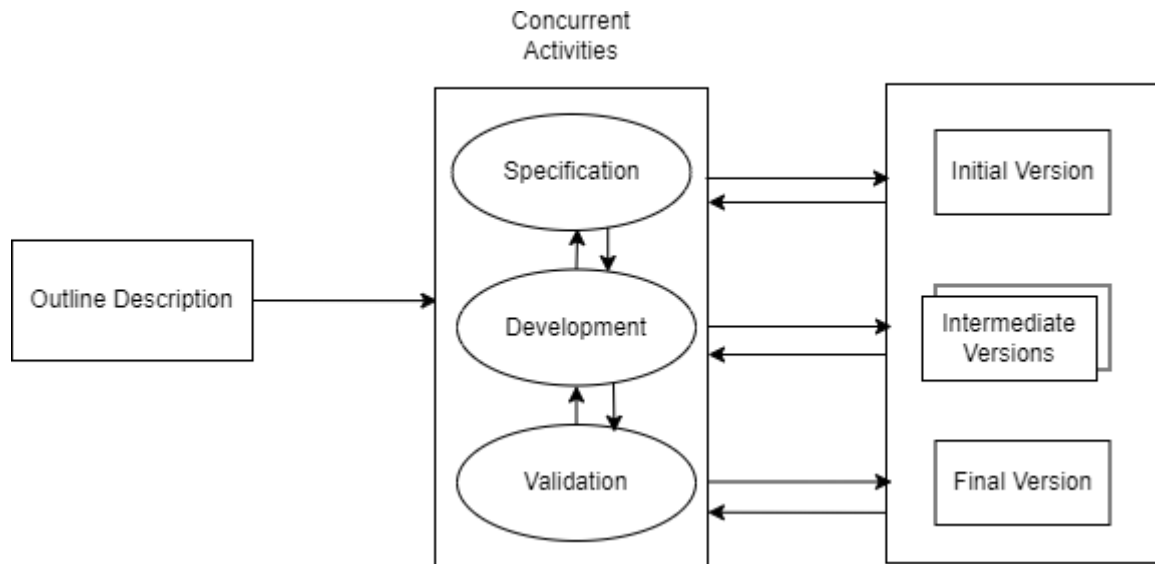


Figure 1.1 Incremental Model

- **Iteration 1:**

The project began with setting up the development environment, which involved selecting appropriate technologies like React for the front-end and FastAPI for the back-end. Configuring tools and libraries, such as Axios for API calls. A challenge faced during this phase was ensuring compatibility between different libraries and frameworks, which sometimes led to version conflicts. Additionally, establishing a clear project structure was crucial to facilitate collaboration among team members and maintain code organization.

- **Iteration 2:**

In the second increment of the project, data was sourced from a publicly accessible GitHub repository containing stock information for NEPSE. A manual verification process was implemented to ensure the dataset's reliability and accuracy. Before utilizing the data for analysis, it was thoroughly inspected for errors and inconsistencies. The dataset included multiple stock price records with various attributes such as open, high, low, and close prices, among others. Additionally, Exploratory Data Analysis (EDA) was performed to identify trends, detect anomalies, and gain insights into the stock market patterns.

- **Iteration 3:**

The development of the LSTM model for stock price prediction involved training the model on the collected data to learn patterns. A significant challenge was selecting the right hyperparameters, such as learning rate and batch size, which required extensive experimentation. Overfitting was another concern, as the model performed well on

training data but struggled with unseen data. To address this, we implemented techniques like dropout and early stopping. Additionally, ensuring that the model could handle the time-series nature of the data effectively required careful consideration of input shapes and sequences.

- **Iteration 4:**

The front-end was built incrementally, creating components like forms and dashboards to display predictions. A challenge during this phase was ensuring a user-friendly design. Integrating the front-end with the back-end API also posed difficulties, particularly in managing asynchronous data fetching and handling loading states. We had to implement effective error handling to provide users with clear feedback in case of issues, ensuring a smooth user experience.

- **Iteration 5:**

Each part of the application was tested thoroughly to ensure functionality and performance. A challenge faced was ensuring comprehensive test coverage, as some edge cases were initially overlooked. We had to balance between implementing new features and fixing bugs, which sometimes led to delays in the development timeline. Continuous integration practices helped streamline testing and deployment. Additionally, ensuring that the application performed well under various conditions required extensive testing and optimization.

- **Iteration 6:**

In the final phase, all components were integrated to ensure seamless interaction between the front-end and back-end. A significant challenge was ensuring that data flowed correctly between the two layers, which required thorough debugging and validation of API responses. We also faced issues with performance optimization, as the initial implementation sometimes resulted in slow response times. To address this, we optimized API calls and improved data handling on the front-end. Ensuring that the application was robust and user-friendly required adjustments before the final deployment.

1.6 Report Organization

The report consists of six chapters which are organized as follows:

Chapter 1: Introduction - A comprehensive project introduction, project-related issue statements, the project's goals, scope, and constraints, as well as the techniques used to construct the system, are all included in this chapter.

Chapter 2: Background Study and Literature Review - An overview of previous studies on the topic is provided in this chapter. In order to execute the project, it covers a variety of studies pertaining to data processing and machine learning approaches.

Chapter 3: System Analysis - Both the functional and non-functional needs of the system are examined in this chapter. A feasibility study that looked at the system's task breakdown structure and operation is also included.

Chapter 4: System Design - This chapter contains the algorithm specifics and the system's thorough design.

Chapter 5: Implementation and Testing - The software tools, dependencies, and hardware tools required to finish the project are covered in this chapter. This chapter describes a number of project implementation steps and a variety of test cases.

Chapter 6: Conclusion and Future Recommendations - The results of the model's implementation are examined in this chapter. It also suggests other things that can be done to make the project better.

CHAPTER 2 BACKGROUND STUDY AND LITERATURE REVIEW

2.1 Background Study

Since a variety of factors, including investor sentiment, corporate performance, and economic conditions, affect market behavior, predicting stock prices is a challenging undertaking. It was difficult for conventional techniques, such as linear regression and ARIMA, to identify the non-linear trends in stock prices. In order to address these issues, models such as Artificial Neural Networks (ANNs) and Recurrent Neural Networks (RNNs) have been developed with the rise of machine learning and deep learning. Since LSTM networks can identify long-term dependencies in time-series data, they have shown themselves to be the most successful of these in stock price prediction. In general, the application of LSTM and deep learning has greatly improved stock price prediction, increasing its accuracy and efficacy in identifying market patterns.

2.2 Literature Review

The potential influence of stock price prediction on financial markets and investing strategies has made it a major research topic. Stock price forecasting has been done using conventional statistical techniques like linear regression and ARIMA. Nevertheless, these techniques frequently fail to capture the intricate, non-linear patterns present in time series data related to finance. The development of deep learning methods, especially Long Short-Term Memory (LSTM) networks, has made it possible to model and forecast stock prices more accurately.

Recurrent neural networks (RNNs) of the LSTM network type are ideal for time series forecasting since they are made to identify long-term dependencies in sequential data. The effectiveness of LSTM models in forecasting stock prices has been shown by research. To illustrate its potential in financial forecasting, one study, for example, suggested an LSTM-based model to analyze future stock values [1].

The use of LSTM networks in stock market analysis and prediction has been investigated in more detail. One such study used historical data to create and assess an LSTM-based prediction model with a focus on key technology stocks [2]. Another study highlighted LSTM's ability to capture temporal dependencies in stock data and presented a reliable

approach for precise stock price prediction [3].

Some literatures have suggested hybrid models that mix LSTM with other methods in order to improve prediction accuracy. To address the nonlinear and extremely unpredictable nature of stock price changes, for instance, a novel variation of LSTM was created with the goal of enhancing stability and minimizing overfitting [4].

In conclusion, LSTM networks have emerged as a key component of contemporary stock price prediction techniques, providing notable enhancements over conventional statistical techniques. These models are still being improved by ongoing research, which incorporates a variety of strategies to improve their predictive capabilities.

CHAPTER 3 SYSTEM ANALYSIS

3.1 System Analysis

3.1.1 Requirement Analysis

A critical first step in creating a stock price prediction system is requirement analysis, which outlines the essential elements, features, and limitations of the system. Non-functional requirements define system properties like performance, security, and usability, while functional requirements explain the essential functionalities.

3.1.1.1 Functional Requirements

Any software system's development must include functional requirements since they specify the precise features, functionalities, and capacities that the system must have in order to satisfy user demands. Throughout the development process, these requirements serve as a guide for designers, developers, and testers to make sure the system functions as planned.

The functional requirements of the system are as follows:

- To allow users to input stock symbols for prediction.
- To use Long Short-Term Memory (LSTM) networks for stock price prediction.
- To allow users to compare actual and predicted stock prices.

The following Use Case Diagram for the Stock Price Prediction System illustrates the interactions between users and the system:

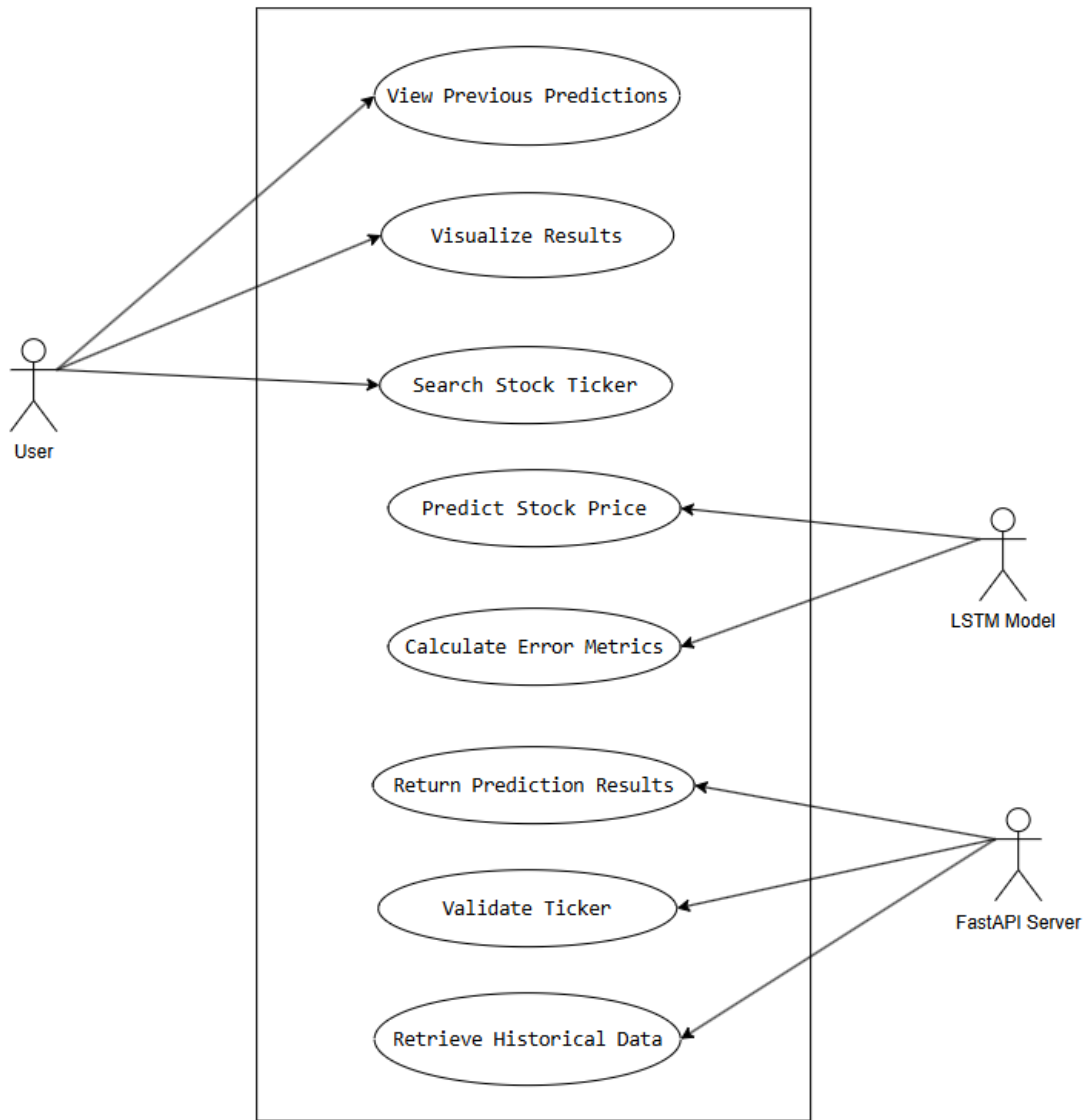


Figure 3.1 Use case diagram of stock price prediction system

Table 3.1: Search Stock Ticker

Use case ID:	UC001
Use case Name:	Search Stock Ticker
Description:	User searches for a specific stock ticker symbol in the application's search bar
Primary Actor:	User
Secondary Actor:	FastAPI Server
Preconditions:	<ul style="list-style-type: none"> - Application is running - User has internet connection - User knows the stock ticker symbol

Postconditions:	<ul style="list-style-type: none"> - Ticker symbol is validated - Historical stock data is retrieved - Prediction process is initiated
-----------------	---

Table 3.2: Predict Stock Price

Use case ID:	UC002
Use case Name:	Predict Stock Price
Description:	Generate future stock price predictions using LSTM machine learning model
Primary Actor:	LSTM Model
Secondary Actor:	None
Preconditions:	<ul style="list-style-type: none"> - Historical stock price data is preprocessed - LSTM model is trained and loaded
Postconditions:	<ul style="list-style-type: none"> - Price predictions are generated - Prediction confidence is calculated - Error metrics are computed

Table 3.3: Return Prediction Results

Use case ID:	UC003
Use case Name:	Return Prediction Results
Description:	Transfer prediction data and error metrics from server to frontend
Primary Actor:	FastAPI Server
Secondary Actor:	Frontend Application
Preconditions:	<ul style="list-style-type: none"> - Predictions and error metrics are generated - Server connection is established
Postconditions:	<ul style="list-style-type: none"> - Prediction data is packaged into JSON - Response is sent to frontend - Frontend is ready to visualize results

Table 3.4: Visualize Results

Use case ID:	UC004
Use case Name:	Visualize Results
Description:	Display stock price predictions and performance metrics in

	user interface
Primary Actor:	User
Secondary Actor:	Frontend Application
Preconditions:	<ul style="list-style-type: none"> - Prediction data received from server - Visualization components are loaded
Postconditions:	<ul style="list-style-type: none"> - Price prediction chart is rendered - Error metrics are displayed - User can interact with visualization

Table 3.5: View Previous Predictions

Use case ID:	UC005
Use case Name:	View Previous Predictions
Description:	Display Previous price predictions made by user
Primary Actor:	User
Secondary Actor:	Frontend Application
Preconditions:	<ul style="list-style-type: none"> - User should predict prices - Prediction data received from server
Postconditions:	<ul style="list-style-type: none"> - User should be able to see previous predictions made by them

3.1.1.2 Non Functional Requirements

Non-functional requirements outline how a system should accomplish its functions rather than the operations themselves. These requirements are centered on characteristics like as performance, security, usability, and maintainability, which ensure that the system functions effectively, efficiently, and can be maintained over time.

The non-functional requirements of the **Stock Price Prediction System** are as follows:

1. **Performance:** The system should respond to user requests within a reasonable time frame, ideally under a minute for fetching predictions. This ensures a smooth user experience, especially when dealing with large datasets.
2. **Usability:** The system is designed with a simple, responsive, and easy-to-navigate interface. It ensures that both technical and non-technical users can use the system without difficulty.

3. **Maintainability:** The system is developed using an object-oriented approach, which makes it easy to modify and update as new requirements arise. Proper documentation is provided to aid in future maintenance and updates.

3.1.2 Feasibility Analysis

Feasibility analysis evaluates whether the proposed system can be successfully developed and implemented based on technical, operational, and financial aspects. The feasibility study completed for the project is listed below:

3.1.2.1 Technical Feasibility

The Stock Price Prediction System is theoretically possible since it builds the LSTM model with widely used tools and technologies. However, data collection and verification proved to be difficult. Gathering accurate and dependable stock market data from external sources necessitated meticulous validation to confirm its authenticity, as inconsistent or fraudulent data could have an impact on the accuracy of predictions.

3.1.2.2 Operational Feasibility

The system is operationally practicable since it allows several users (investors, analysts, and administrators), make prediction requests, and view data. The user-friendly interface ensures that users of all technical levels may easily estimate stock values. Users can search for equities on NEPSE (Nepal Stock Exchange) and compare anticipated to real prices. This feature addresses the user's needs for data visualization and prediction accuracy, allowing them to make informed decisions based on real-time stock performance.

3.1.2.3 Schedule Feasibility

Schedule feasibility is the process of determining if a proposed plan or project can be completed realistically within a given timeframe. It entails evaluating a variety of factors, including resource availability, time constraints, task dependencies, and potential dangers. Given below is the WBS and Gantt chart, which depicts the entire schedule of the project. As the project is successfully completed within the appointed time, it is schedule feasible.

		Task Mode ▾	Task Name ▾	Duration ▾	Start ▾	Finish ▾	Predecessors ▾
0			Stock Market Prediction	79 days	Thu 7/25/24	Tue 11/12/24	
1			Planning	7 days	Thu 7/25/24	Fri 8/2/24	
2			Requirements gathering	3 days	Thu 7/25/24	Mon 7/29/24	
3			Requirement documenting	4 days	Tue 7/30/24	Fri 8/2/24	2
4			Analysis	6 days	Mon 8/5/24	Mon 8/12/24	1
5			Resource Analysis	3 days	Mon 8/5/24	Wed 8/7/24	
6			Feasibility Analysis	3 days	Thu 8/8/24	Mon 8/12/24	5
7			Design	10 days	Tue 8/13/24	Mon 8/26/24	4
8			Architectural Design	6 days	Tue 8/13/24	Tue 8/20/24	
9			UI/UX Design	4 days	Wed 8/21/24	Mon 8/26/24	8
10			Implementation	45 days	Tue 8/27/24	Mon 10/28/24	7
11			Frontend Development	10 days	Tue 8/27/24	Mon 9/9/24	
12			Backend Development	22 days	Tue 9/10/24	Wed 10/9/24	11
13			Algorithm Development	25 days	Thu 9/12/24	Wed 10/16/24	11
14			Algorithm Integration	8 days	Thu 10/17/24	Mon 10/28/24	12,13
15			Testing	8 days	Tue 10/29/24	Thu 11/7/24	10
16			Unit Testing	2 days	Tue 10/29/24	Wed 10/30/24	
17			Integration Testing	2 days	Thu 10/31/24	Fri 11/1/24	16
18			System Testing	4 days	Mon 11/4/24	Thu 11/7/24	17
19			Documentation	3 days	Fri 11/8/24	Tue 11/12/24	15

Figure 3.2 Work breakdown structure of stock price prediction system

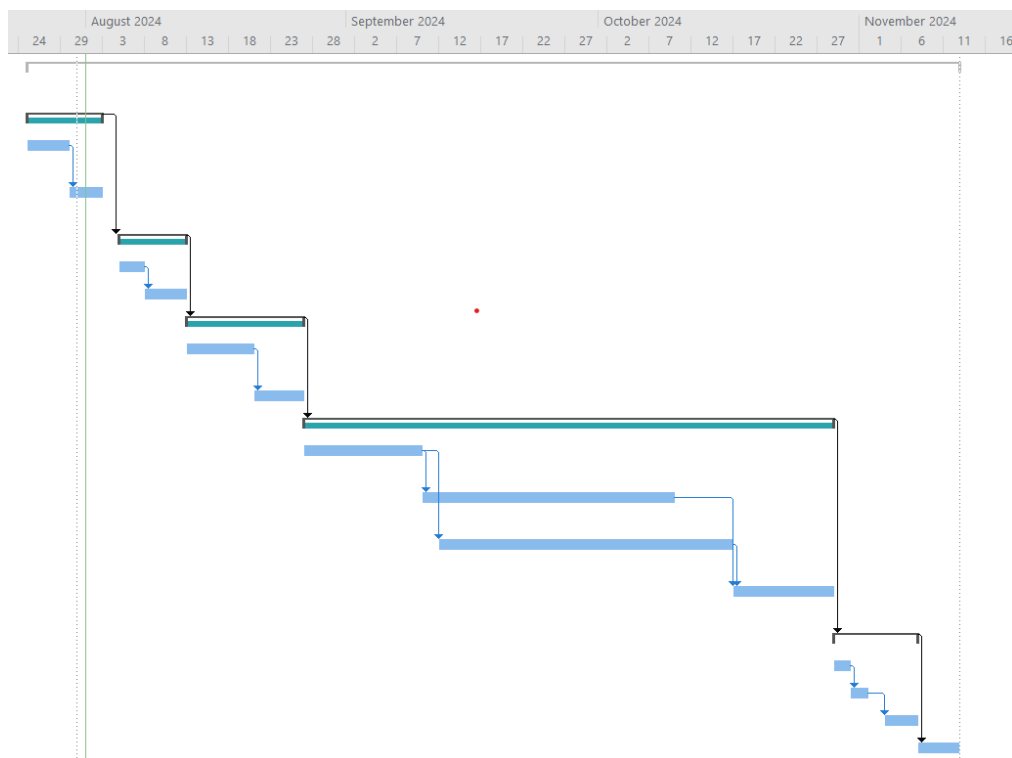


Figure 3.3 Gantt chart of stock price prediction system

3.1.3 Analysis

During the analysis phase, system requirements are structured. Requirements are structured using an object-oriented approach.

3.1.3.1 Object modelling using Class Diagram

A class diagram in UML is a blueprint that shows how different parts of a system are organized. It displays classes with their properties and functions, and shows how these classes connect and relate to each other. Think of it as a map showing how all the pieces of the system fit together.

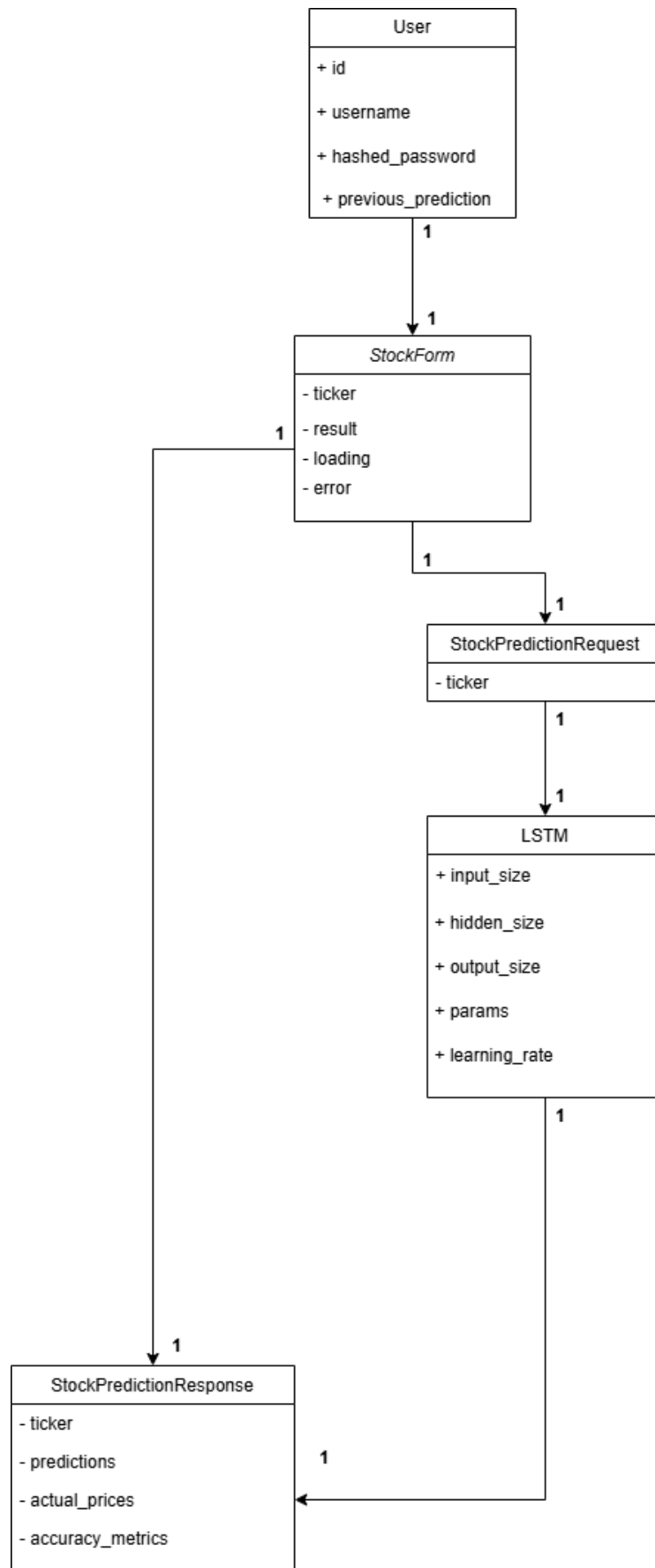


Figure 3.4 Class diagram of stock price prediction system

This class diagram illustrates the core structure of the Stock Price Predictor system through five interconnected classes. Starting with the User class that stores basic user information (ID, username, password, and previous predictions), it connects to the StockForm class which handles the input interface with ticker symbol, results, loading state, and error management. The StockForm links to StockPredictionRequest, which simplifies the request to the ticker symbol. This feeds into the LSTM class, containing essential machine learning model parameters like input size, hidden size, output size, and learning rate. Finally, the StockPredictionResponse class captures the prediction outputs including the ticker, predictions, actual prices, and accuracy metrics. All these classes are connected through one-to-one relationships, establishing a clear flow from user input through to final prediction output.

3.1.3.2 Dynamic modelling using Sequence Diagram

A sequence diagram shows how different parts of a system communicate with each other over time, using vertical lines to represent objects and arrows to show the messages passed between them in order.

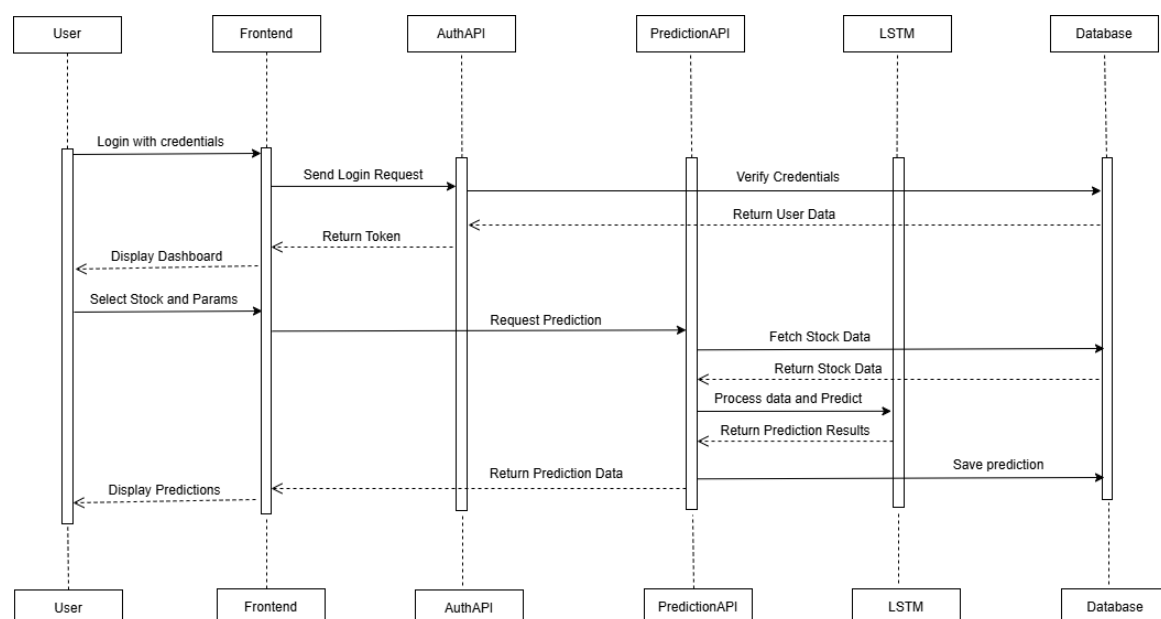


Figure 3.5 Sequence diagram of stock price prediction system

This sequence diagram shows the step-by-step flow of the Stock Price Predictor system. First, the user logs in through the frontend, which verifies credentials via the AuthAPI and database. After successful authentication, the user sees the dashboard and can select a stock and parameters. This selection triggers the PredictionAPI to fetch data and pass it to the

LSTM model for processing. The LSTM generates predictions, which are then saved to the database and displayed back to the user through the frontend. The diagram uses solid arrows for requests and dashed arrows for responses, clearly showing how data flows between the components of the system.

3.1.3.3 Process Modeling using Activity Diagram

An activity diagram in UML is like a flowchart that shows how a system works step by step. Think of it as a recipe or instruction manual that maps out the path of actions, including different choices (like if-then decisions), repeated steps (loops), and tasks that can happen at the same time. It helps visualize how work flows through a system from start to finish.

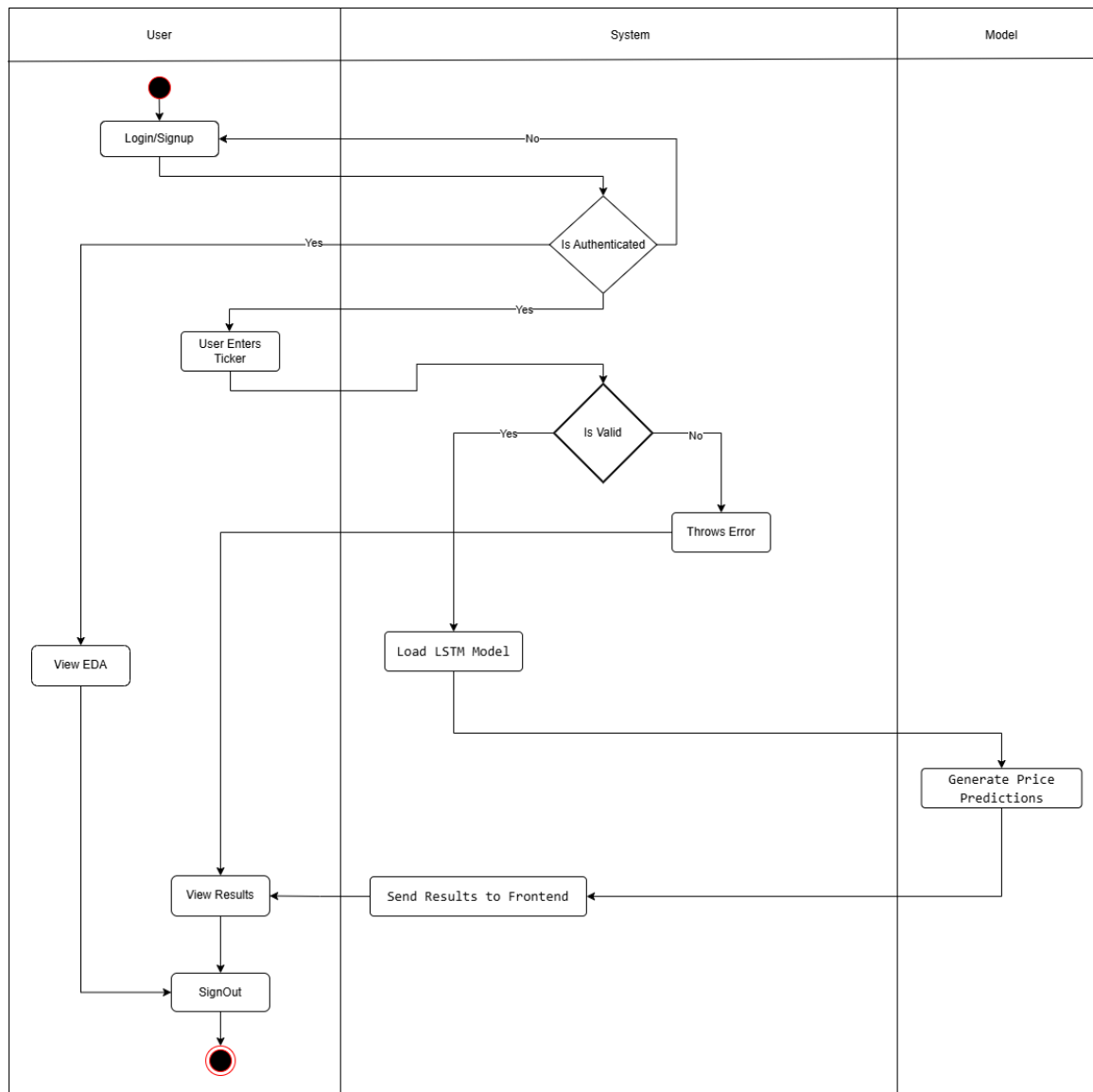


Figure 3.6 Activity diagram of stock price prediction system

This activity diagram shows the complete workflow of the Stock Price Predictor system divided into three sections (User, System, and Model). The process starts with user login/signup, followed by authentication checks. If authenticated, users can either view EDA or enter a stock ticker. For valid tickers, the system loads the LSTM model to generate price predictions. The results are sent to the frontend for user viewing. Invalid tickers trigger error messages. Users can sign out when finished, ending the session. The diagram uses decision points (diamonds) to show different paths based on authentication and ticker validation, making the flow of actions clear from start to finish.

CHAPTER 4 SYSTEM DESIGN

4.1 Design

The design phase is vital because it plans and defines the general structure and architecture of the Stock Price Prediction System. The major purpose of this phase is to convert the system's requirements into a thorough design that will serve as the basis for developing the actual software.

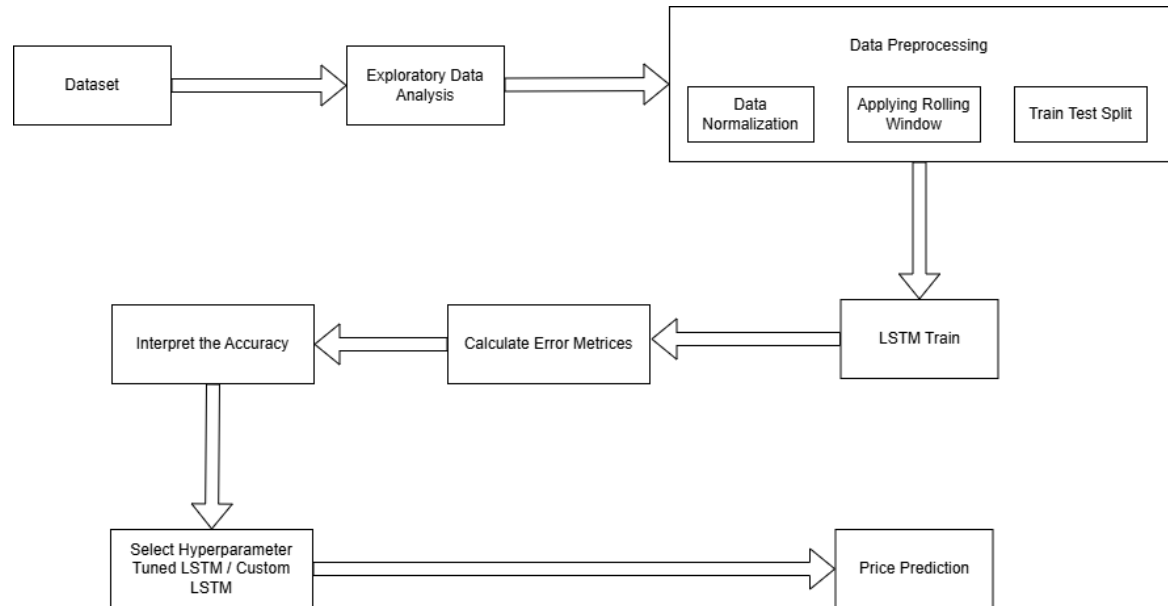


Figure 4.1 High-Level System Design of Stock Price Prediction

The high level system design of Stock Price Prediction is shown in the above figure. The following steps were carried out throughout the project development:

1. Data Collection

The dataset for the Stock Price Prediction project was collected from a GitHub repository and manually checked for its reliability and up-to-date data.

The screenshot shows a Jupyter Notebook interface. The top part contains Python code to load a CSV file named 'AHPC.csv' into a DataFrame 'df'. The code sorts the DataFrame by 'published_date' in ascending order. Below the code, the DataFrame 'df' is displayed as a table. The table has 10 columns: 'published_date', 'open', 'high', 'low', 'close', 'per_change', 'traded_quantity', 'traded_amount', 'status', and 'Category'. The first five rows of data are shown, followed by an ellipsis indicating more rows, and then rows 3282 through 3286. The status of all rows is 0 or 1, and the category is 'Hydropower' for all.

```
# Load dataset
df = pd.read_csv("../stock_data_csv/AHPC.csv")
df["published_date"] = pd.to_datetime(df["published_date"])
df.sort_values("published_date", inplace=True)

df
```

	published_date	open	high	low	close	per_change	traded_quantity	traded_amount	status	Category
0	2009-11-25	408.0	790.0	790.0	790.0	NaN	10.0	0.0	0	Hydropower
1	2009-12-03	790.0	770.0	740.0	740.0	-6.33	50.0	0.0	0	Hydropower
2	2009-12-06	740.0	726.0	687.0	687.0	-7.16	50.0	0.0	0	Hydropower
3	2009-12-07	687.0	674.0	661.0	661.0	-3.78	20.0	0.0	0	Hydropower
4	2009-12-08	661.0	701.0	648.0	700.0	5.90	3230.0	0.0	0	Hydropower
...
3282	2024-08-11	257.5	277.7	255.0	277.7	9.98	1392573.0	375484936.1	1	Hydropower
3283	2024-08-12	283.2	305.4	281.3	302.8	9.04	1404188.0	412855574.8	1	Hydropower
3284	2024-08-13	302.8	305.0	286.0	295.0	-2.58	1415664.0	420012598.5	-1	Hydropower
3285	2024-08-14	298.0	317.0	286.0	312.0	5.76	1347043.0	402093869.6	1	Hydropower
3286	2024-08-15	318.2	340.5	317.6	334.0	7.05	1448716.0	482083982.8	1	Hydropower

3287 rows × 10 columns

Figure 4.2 Overview of the Dataset of AHPC

From the above figure, it is seen that there are a total of 3287 rows and 10 columns in the dataset. The columns named 'published_date', 'open', 'high', 'low', 'close', 'per_change', 'traded_quantity', 'traded_amount', 'status', and 'Category' are the independent variables. However, the column 'close' can be considered the dependent or target variable. The target variable depends on each of these independent variables, meaning any changes in the independent variables may also affect the target variable.

2. Exploratory Data Analysis

In simple terms, insights from the data were collected by performing an Exploratory Data Analysis (EDA). This process involved using descriptive statistics and visualizations to better understand the data's patterns, trends, and key features.

```
# Print results
print("Dataset Overview:")
print(df.info())
✓ 0.0s

Dataset Overview:
<class 'pandas.core.frame.DataFrame'>
Index: 3287 entries, 0 to 3286
Data columns (total 10 columns):
#   Column          Non-Null Count  Dtype
---  -
0   published_date   3287 non-null   datetime64[ns]
1   open             3287 non-null   float64
2   high             3287 non-null   float64
3   low              3287 non-null   float64
4   close            3287 non-null   float64
5   per_change       3286 non-null   float64
6   traded_quantity  3287 non-null   float64
7   traded_amount    3287 non-null   float64
8   status           3287 non-null   int64
9   Category         3287 non-null   object
dtypes: datetime64[ns](1), float64(7), int64(1), object(1)
memory usage: 282.5+ KB
None
```

Figure 4.3 Information on the Data of AHPC

The columns include trading information like published_date (datetime format), open, high, low, close prices (all float64), per_change, traded_quantity, traded_amount (float64), status (int64), and Category (object). The data quality appears strong with almost no missing values (only one null in per_change). This clean, well-structured dataset is ready for stock market analysis without needing extensive data cleaning.

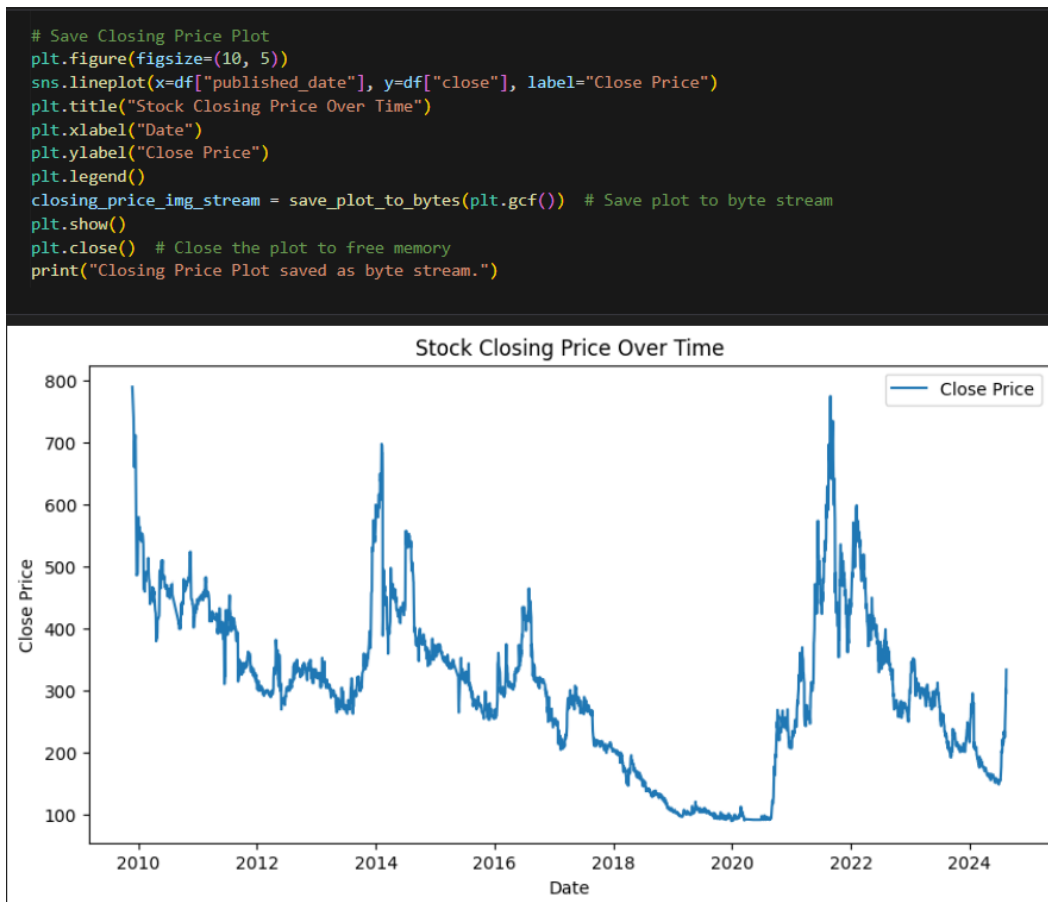


Figure 4.4 Stock Close Price Over Time of AHPC

The above visualization shows the Stock Closing Price Over Time from 2010 to 2024, presenting a clear historical trend of the stock's performance. The graph reveals significant price volatility throughout this period, with major peaks reaching around 800 in both 2010 and 2022. A notable low point occurs around 2020 where the price dropped to approximately 100. The recent trend from 2022 to 2024 shows a general decline in stock price, providing crucial historical context for price prediction.

We conducted an exploratory data analysis (EDA) to understand the trends in the moving average (MA), explore the correlation between different variables, and analyze how the closing price of the stock has changed over the years.

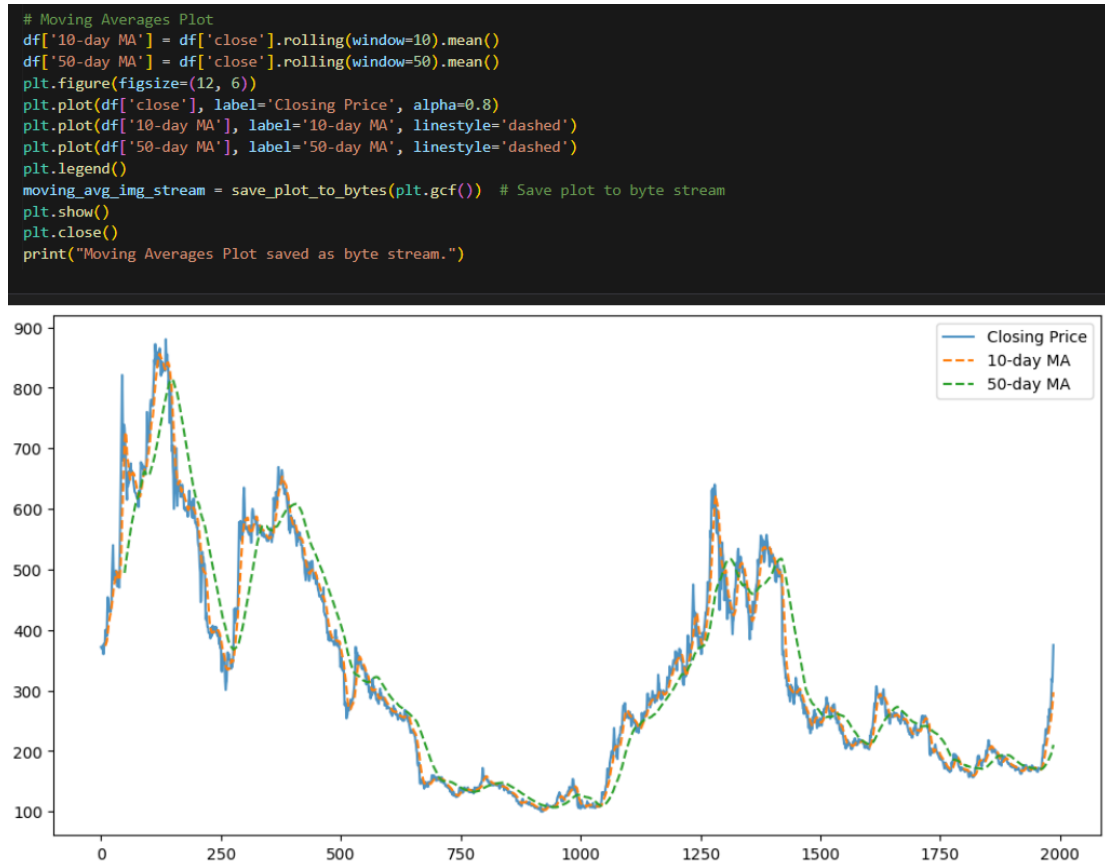


Figure 4.5 Moving Averages of AHPC

The above visualization features Moving Averages Analysis, which overlays two key trend indicators on the stock price data. The actual closing price is shown in blue, while the 10-day moving average (orange dashed line) and 50-day moving average (green dashed line) help identify both short-term and long-term trends. These moving averages smooth out price fluctuations and help identify potential trend reversals and market momentum, making it easier to understand the stock's overall direction at different time scales.

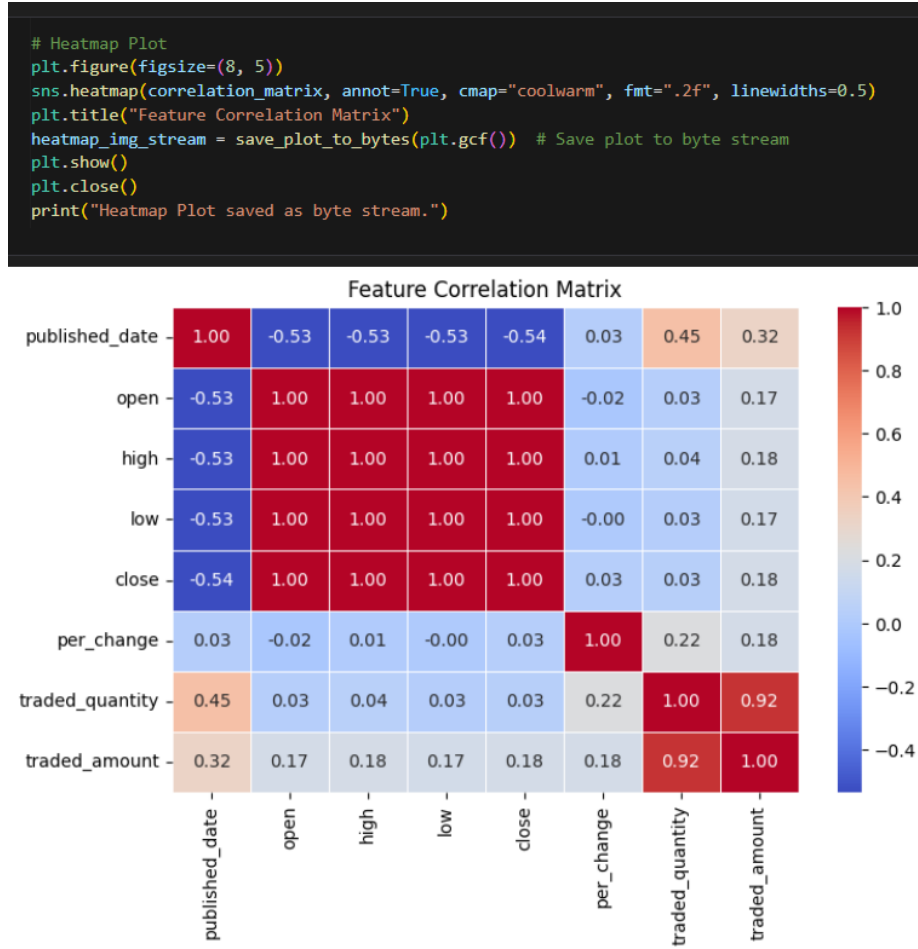


Figure 4.6 Correlation Matrix of AHPC

The above visualization presents a Feature Correlation Matrix through a heatmap, revealing the relationships between different stock metrics. The color coding ranges from dark red (strong positive correlation of 1.0) to dark blue (negative correlation). Notable relationships include the strong positive correlation between open, high, low, and close prices, and the significant correlation (0.92) between traded quantity and traded amount. The published date shows a moderate negative correlation (-0.53) with price metrics, while the percentage change demonstrates weak correlations with most other features, indicating its relative independence.

3. Data Preprocessing

The data preprocessing for the Stock Price Prediction project involved three essential steps to prepare the stock price data for the LSTM model. First, data normalization was performed using MinMaxScaler to transform all stock prices to a range between -1 and 1, ensuring all features contribute equally and helping the LSTM model learn more

effectively. Next, a rolling window approach was implemented with a window size of 6 days, creating sequences where each set of 6 consecutive days of stock data is used to predict the following day's price. For example, the data from Days 1-6 would be used to predict Day 7's price, allowing the model to learn temporal patterns in stock price movements. Finally, the dataset was split into training and testing sets using an 80/20 ratio, where 80% of the data was used for training the model to learn patterns, and the remaining 20% was reserved for testing to evaluate the model's performance on unseen data. This comprehensive preprocessing pipeline ensures the data is appropriately formatted for the LSTM model while maintaining the time-series nature of stock prices and enabling proper model evaluation.

4. LSTM Train

The LSTM Train shows how the model learns from the preprocessed stock data. It takes the normalized data with 6-day windows and learns price patterns during training. The model's performance is then measured through error metrics, which help determine if the model needs hyperparameter adjustments before making final price predictions.

5. Calculate Error Metrics

The Stock Price Prediction project implemented three key error metrics to evaluate the LSTM model's performance. Mean Squared Error (MSE), Root Mean Square Error (RMSE), and Mean Absolute Percentage Error (MAPE) were calculated by comparing the model's predicted stock prices against actual market values. These calculations were performed on the test dataset to ensure unbiased evaluation of the model's predictions. The implementation of these metrics provides a comprehensive assessment of the model's accuracy and helps in understanding how well the LSTM model captures stock price patterns and makes predictions.

6. Interpret the Accuracy

The model's accuracy was interpreted by analyzing MSE, RMSE, and MAPE values together. Lower values in these metrics indicated better prediction accuracy. By comparing these metrics across different versions of the LSTM model, we could identify which configuration performed best and understand the model's overall prediction reliability.

7. Select Hyperparameter Tuned LSTM/Custom LSTM

The Stock Price Prediction system provides flexibility in model selection through two LSTM options. The Hyperparameter Tuned LSTM comes with pre-optimized settings

that have been tested for best performance, making it ready for immediate use. Alternatively, the Custom LSTM allows users to input their own parameters such as learning rate, batch size, and epochs, enabling experimentation with different settings. This approach accommodates both users who prefer a reliable pre-tuned model and those who want to explore custom configurations for their predictions.

8. Price Prediction

The Price Prediction represents the final output of the Stock Price Predictor system. Using either the Hyperparameter Tuned LSTM or Custom LSTM model, the system generates future stock price predictions based on the processed historical data. The predictions are displayed to users through visualizations and numerical values, helping them understand potential future price movements of their selected stock.

4.1.1 Refinement of Class Diagram

This class diagram shows the detailed structure of the Stock Price Predictor system with both attributes and methods. The User class contains basic user information (id, username, hashed_password) and prediction history. The StockForm class handles the user interface with attributes for ticker, result, loading, and error states, plus methods fetchPrediction() and handleSubmit() for form processing. The StockPredictionRequest class simplifies requests to just the ticker symbol. The LSTM class is the most detailed, containing model parameters (input_size, hidden_size, output_size, params, learning_rate) and essential methods for machine learning operations (test(), forward(), error(), backpropagation(), train()). Finally, the StockPredictionResponse class structures the output with ticker, predictions, actual_prices, and accuracy_metrics. All classes are connected through one-to-one relationships, creating a logical flow from user input through model processing to final prediction output.

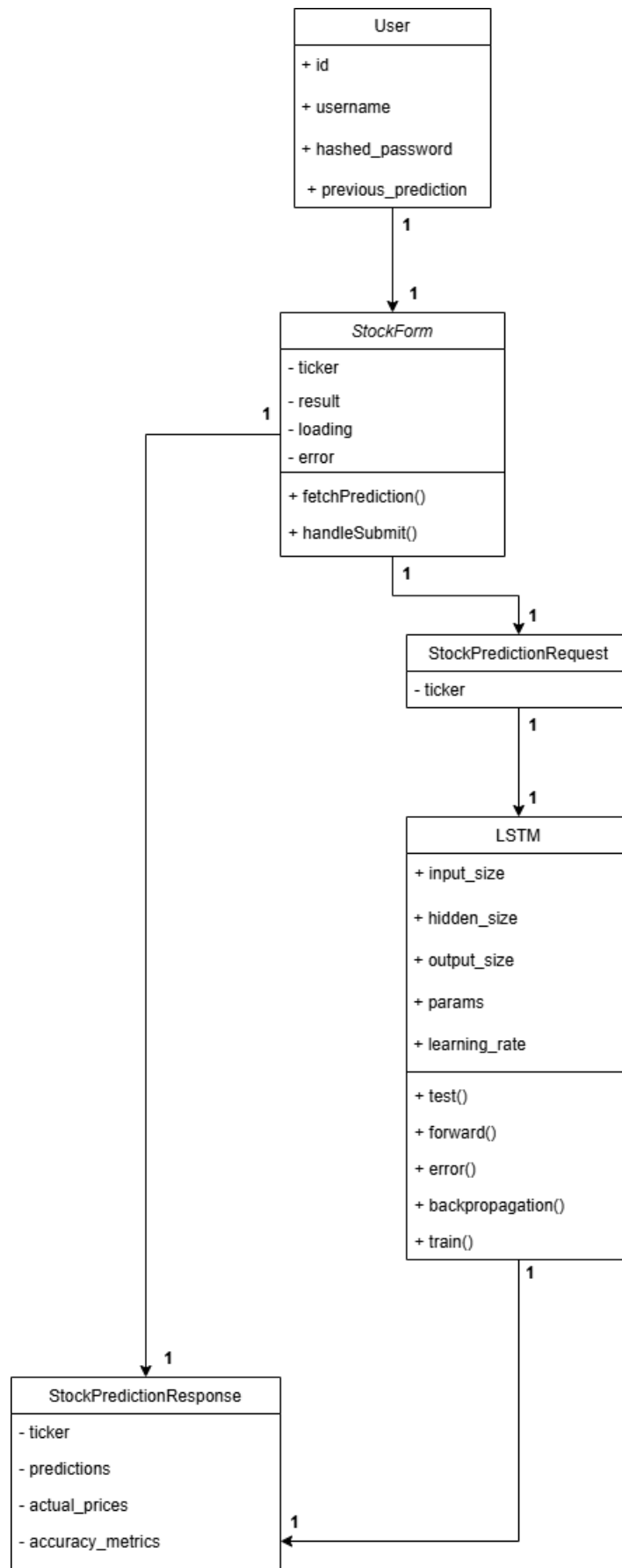


Figure 4.7 Refined Class diagram of stock price prediction system

4.1.2 Refinement of Activity Diagram

This activity diagram illustrates the complete workflow of the Stock Price Predictor system, starting with user authentication through login/signup. Once a user attempts to log in, the system checks their authentication status. If not authenticated, they're shown an error and prompted to retry. If authenticated, users can either view EDA (Exploratory Data Analysis) or proceed with stock prediction by entering a ticker symbol. When a valid ticker is entered, the system follows a sequential process: retrieving historical data, preprocessing it, loading the LSTM model, generating price predictions, calculating error metrics, and finally sending results to the frontend for visualization. If an invalid ticker is entered, an error message is displayed. All paths eventually converge to an end state, whether through successful prediction visualization, error messages, or EDA viewing. This diagram effectively shows how the system handles both successful and error scenarios while maintaining a logical flow from user input to final output.

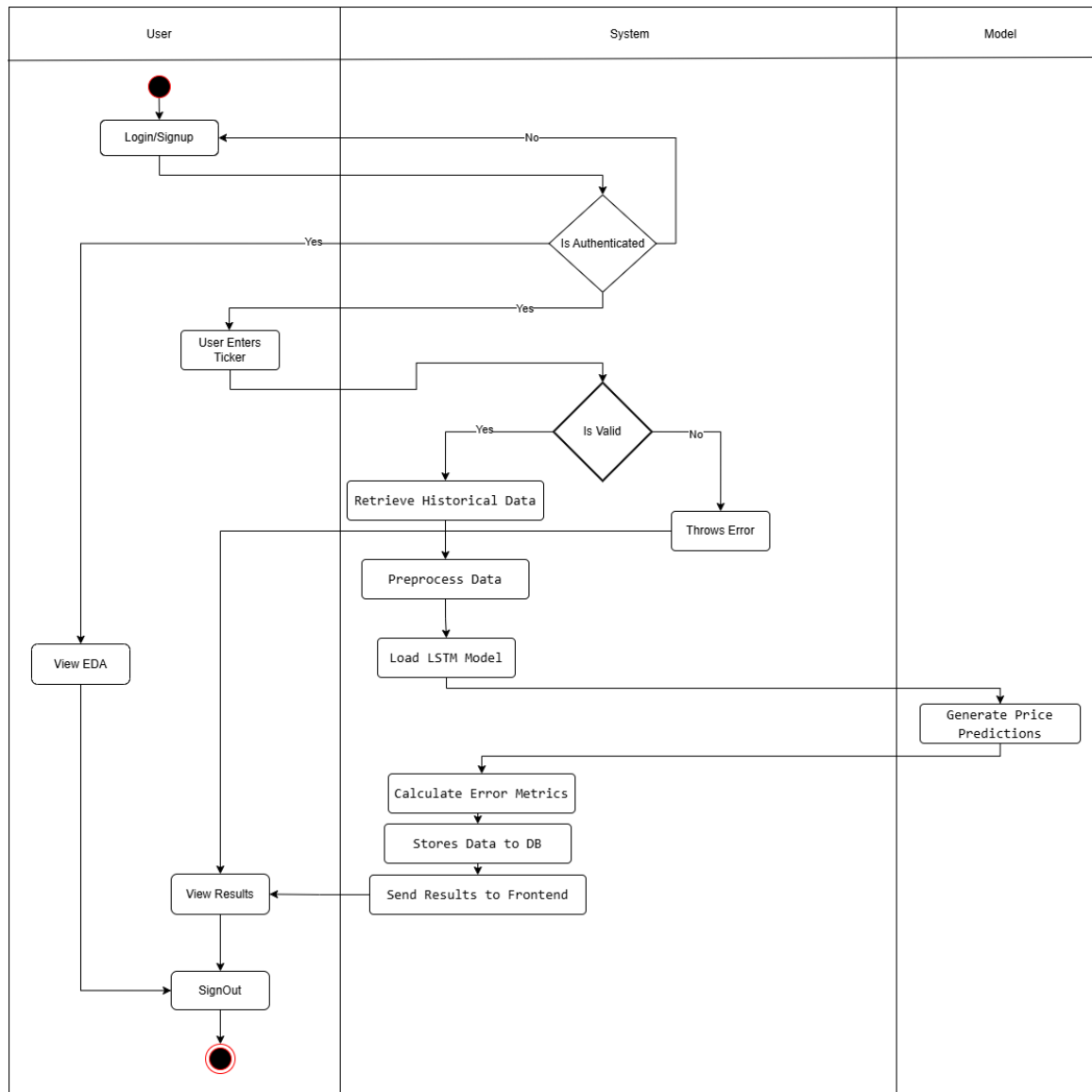


Figure 4.8 Refined Activity diagram of stock price prediction system

4.1.3 Refinement of Sequence Diagram

This sequence diagram shows the step-by-step interaction between different components of the Stock Price Predictor system. The flow begins when a user logs in with their credentials through the frontend. The frontend sends these credentials to the AuthAPI, which verifies them with the database and returns a token. Once authenticated, the user sees the dashboard where they can select a stock and set prediction parameters. This selection triggers a request to the PredictionAPI, which then interacts with the LSTM model. The LSTM model fetches historical stock data from the database, processes it, and generates predictions. These prediction results are sent back through the PredictionAPI to the frontend, which displays them to the user, while also being saved in the database.

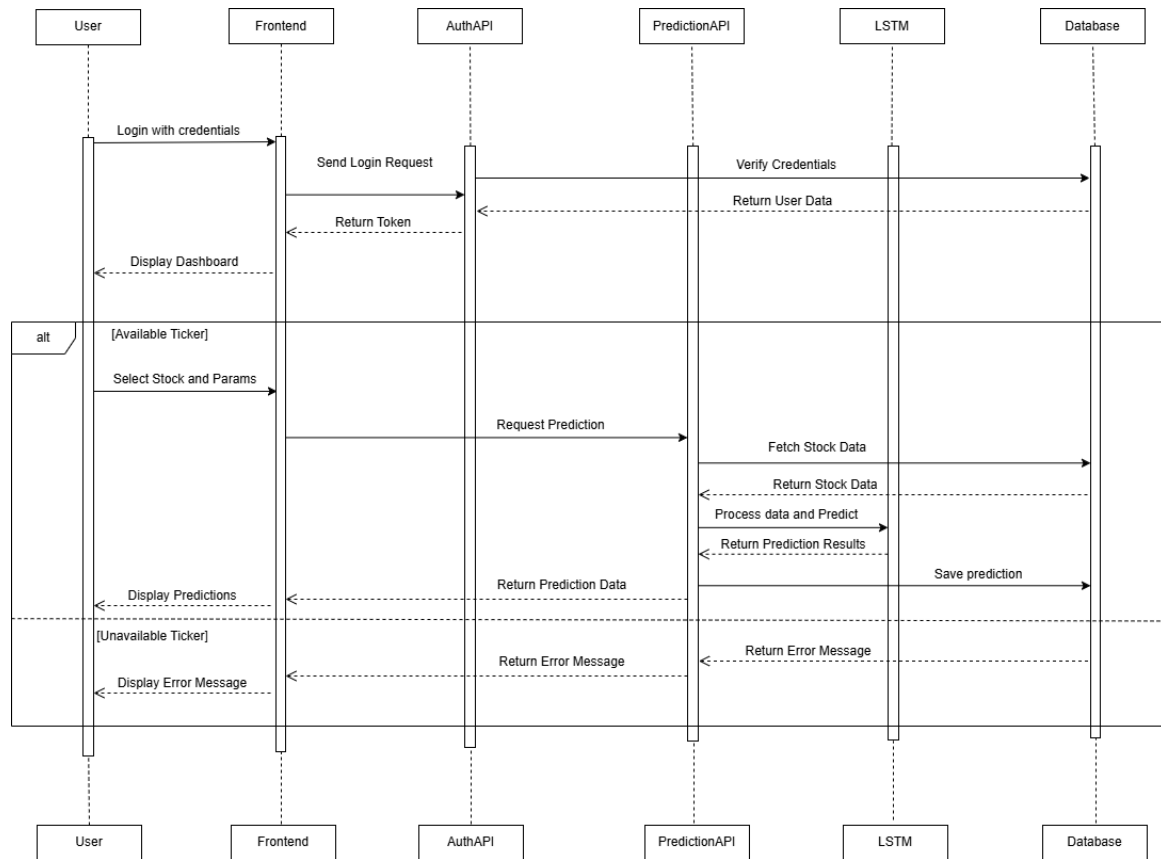


Figure 4.9 Refined Sequence diagram of stock price prediction system

4.1.4 Component Diagram

The component diagram for the Stock Price Predictor system illustrates a well-structured architecture with five key components interconnected through the FastAPI Server as its central hub. At one end, we have the User component representing the frontend interface where users interact with the system, while at the other end, we have the Data Source component that stores historical stock data. The FastAPI Server sits at the center, acting as the main coordinator that handles all communications and business logic. It connects to an SQLite Database component for storing user information and prediction history, and to the LSTM Model component which performs the actual stock price predictions. This architecture ensures that each component has a specific responsibility while maintaining organized data flow throughout the system, from user input to final prediction output.

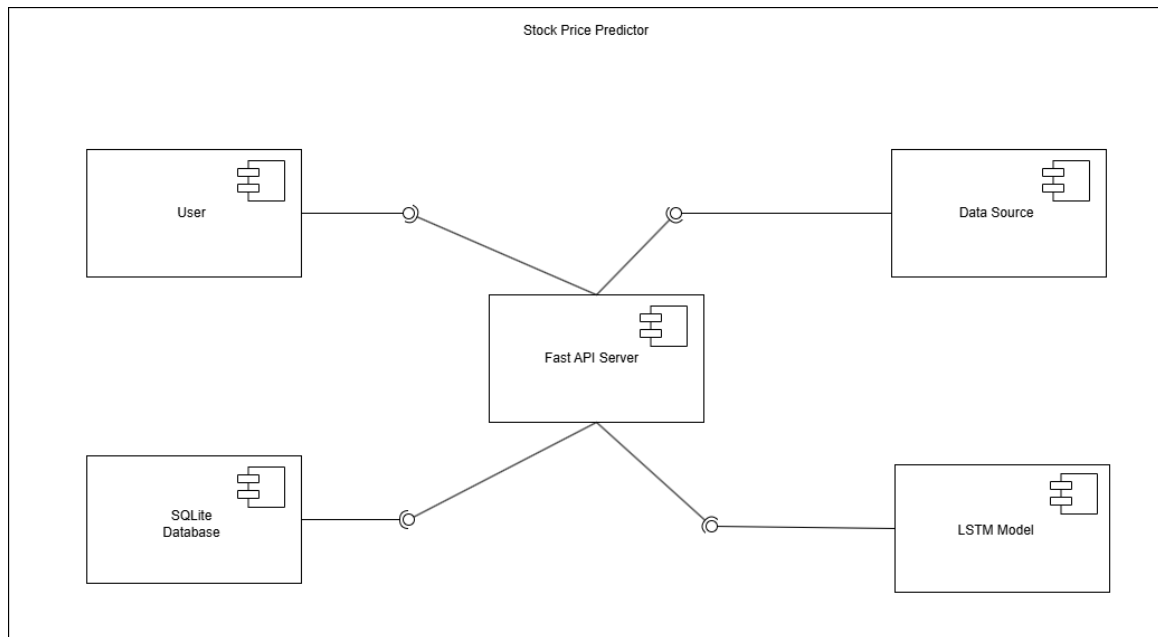


Figure 4.10 Component diagram of stock price prediction system

4.1.5 Deployment Diagram

This deployment diagram shows the architecture of the Stock Price Predictor system across three main devices. The Desktop device hosts the client-side application running on localhost, which communicates with the Server device through a FastAPI connection. The Server, in turn, connects to the Database device using SQLite protocol. Each device is represented as a node with its specific artifacts running on localhost. This simple three-tier architecture separates the user interface (Desktop), application logic (Server), and data storage (Database), making the system modular and maintainable.

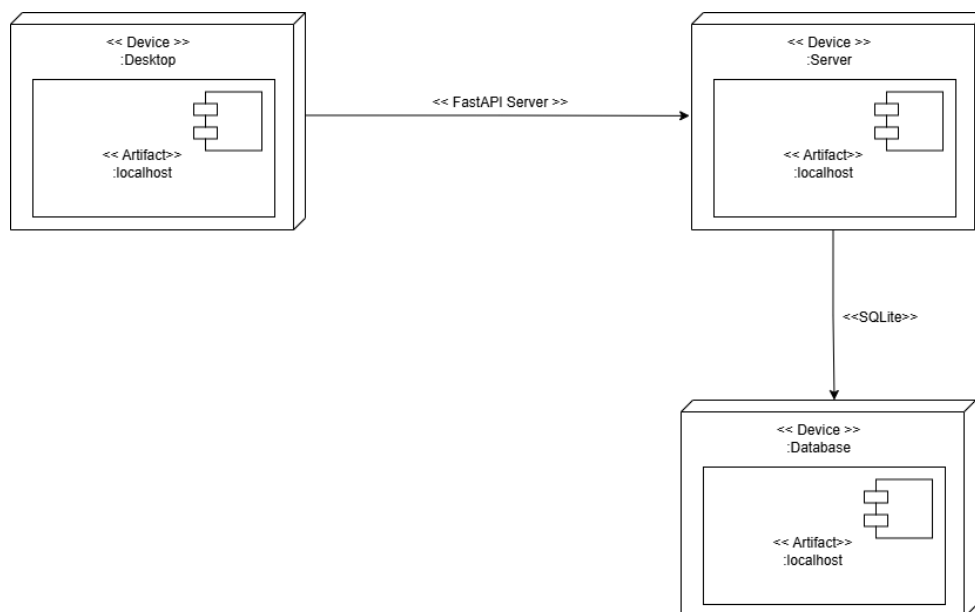


Figure 4.11 Deployment diagram of stock price prediction system

4.2 Algorithm Details

Long Short-Term Memory (LSTM) is a type of Recurrent Neural Network (RNN), designed to learn and predict sequences of data, making it particularly effective for time-series problems such as stock price prediction. Unlike traditional neural networks, LSTMs can retain information over longer periods, overcoming the limitations of standard RNNs, which struggle with the "vanishing gradient problem" in long sequences. Below is a detailed explanation of how LSTM works:

4.2.1 The Problem LSTM Solves

- Standard RNNs are effective for sequence prediction, but the vanishing gradient problem makes it difficult to capture long-term dependencies. This means that as the sequence grows longer, it becomes more difficult for the RNN to recall information from previous time steps.
- LSTMs are intended to address this issue by employing a more sophisticated architecture that enables the model to "remember" relevant information across long periods.

4.2.2 LSTM Architecture

An LSTM is made up of multiple components, the most important of which is the memory cell, which can store information for extended periods of time. The LSTM architecture consists of the following fundamental parts:

- a. Memory Cell:** The memory cell stores information and informs the LSTM model on what to remember and forget at each time step.
- b. Gates:** The LSTM employs three gates (forget, input, and output) to govern information flow into and out of memory cells. These gates are implemented as neural networks, which receive inputs and determine which information is meaningful.
- 1. Forget Gate:** The forget gate determines whether information from prior memories should be erased or maintained. This is based on both the prior output and the present input. The forget gate produces a value between 0 and 1, with 0 representing "forget everything" and 1 representing "keep everything."

Equation:

$$f_t = \sigma (w_f \cdot [h_{t-1}, x_t] + b_f)$$

The forget gate output is represented by: f_t

σ is the sigmoid activation function.

h_{t-1} represents the preceding hidden state.

x_t represents the current input.

The weights and bias for the forget gate are represented by w_f and b_f , respectively.

2. Input Gate: The input gate controls what new information is added to the memory cell. It decides which values to update the memory cell with, by combining the current input and the previous hidden state.

Equation:

$$i_t = \sigma (w_i \cdot [h_{t-1}, x_t] + b_i)$$

Where: i_t is the input gate output.

w_i and b_i are the weights and bias for the input gate.

3. Output Gate: The output gate determines what information is output from the memory cell. The information in the memory cell is processed through a tanh function to compress it between -1 and 1, and then multiplied by the output gate's result to get the final output.

Equation:

$$o_t = \sigma (w_o \cdot [h_{t-1}, x_t] + b_o)$$

Where: o_t is the output gate output.

The weights and bias for the output gate are represented by w_o and b_o respectively.

c. Cell State (c_t): The network's memory. At each time step, the forget gate determines what fraction of the prior memory should be retained, while the input gate determines what new information should be added to memory. The forget and input gates govern the final memory at each time step, which is a blend of prior and new information.

Cell state update equation:

$$c_t = f_t * c_{t-1} + i_t * \tilde{c}_t$$

Where: c_{t-1} is the preceding cell state.

\tilde{c}_t The candidate cell state (t) represents fresh information created from the current input.

The output of the forget gate is f_t .

The input gate output is denoted by the symbol i_t .

d. Hidden State (h_t): The LSTM cell's output, the hidden state, is either transferred to the following time step or used to make predictions.

The hidden state equation is:

$$h_t = O_t * \tanh(ct)$$

where the hidden state (final output) is represented by: h_t

The output gate is denoted by t.

The current cell state is denoted by ct .

4.2.3 How LSTM Works in Stock Price Prediction:

LSTMs are very useful for stock price prediction because of their capacity to handle data sequences, such as stock prices over time. This is how the procedure operates:

1. Data Preparation:

After gathering historical stock price data, the system pre-processes it by organizing it into sequences of input features (such as historical stock prices, volume, and technical indicators) and normalizing the data.

2. Model Training:

The information is utilized to train the LSTM model, which analyzes the patterns in the past price data to learn how to forecast future stock prices. The LSTM will pick up on long-term dependencies in the stock prices, including trends, seasonal patterns, and other cyclical activity.

3. Prediction:

Using the patterns it has learned, the LSTM model can forecast future stock prices after it has been trained. For example, given the historical data of a certain stock, the LSTM will predict the next day's stock price or future prices for a specific period.

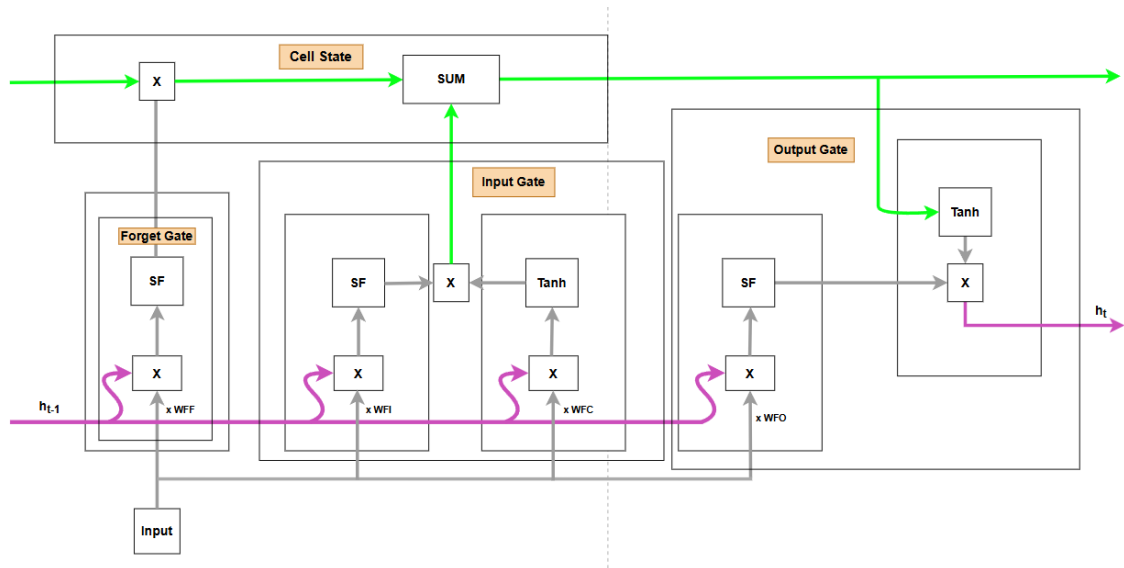


Figure 4.12: Architecture of LSTM

CHAPTER 5 IMPLEMENTATION AND TESTING

5.1 Implementation

Implementation is the process of creating a working system by effectively utilizing tools and technology that are suited for the project at hand. This procedure turns preexisting designs into a workable system.

5.1.1 Tools used

The following are the tools utilized in the implementation of project and the making of document:

Table 5.1: Tools used

Category	Tool/Technology	Description
Frontend	React	The front-end of the system was constructed in this project using React
Backend	Python	Python is utilized in this project to manage data processing, incorporate machine learning models.
Backend	FastAPI	FastAPI is a fast web framework for Python API development.
IDE	Visual Studio Code	An open-source, cross-platform, and lightweight IDE.

Dependencies

The libraries needed to execute and run a project are known as dependencies. The external libraries and packages created by the community or third-party developers are the dependencies used in the project.

The project uses the external modules or dependencies listed below:

Table 5.2: External Modules

S.No.	Modules	Description
1	fastapi (0.104.1)	Used for building the backend API.
2	uvicorn (0.24.0)	ASGI server for running the FastAPI app.

3	numpy (1.24.3)	Used for numerical computations.
4	pandas (2.0.3)	Used for handling and processing data.
5	pandas-datareader (0.10.0)	Helps fetch financial data from external sources.
6	matplotlib (3.7.1)	Used for data visualization.
7	react (18.3.1)	Front-end JavaScript library for UI development.
8	recharts (2.15.0)	Used for creating interactive charts in React.
9	@mui/material (6.3.0)	Material UI library for React components.
10	tailwindcss (3.4.17)	Utility-first CSS framework for styling.
11	Autograd	Automatic differentiation for gradient computation.

5.1.2 Implementation Details

The overall project is divided into three different components, front-end, back-end, and algorithm implementation for development. The following figure shows the implementation process of the algorithm in detail.

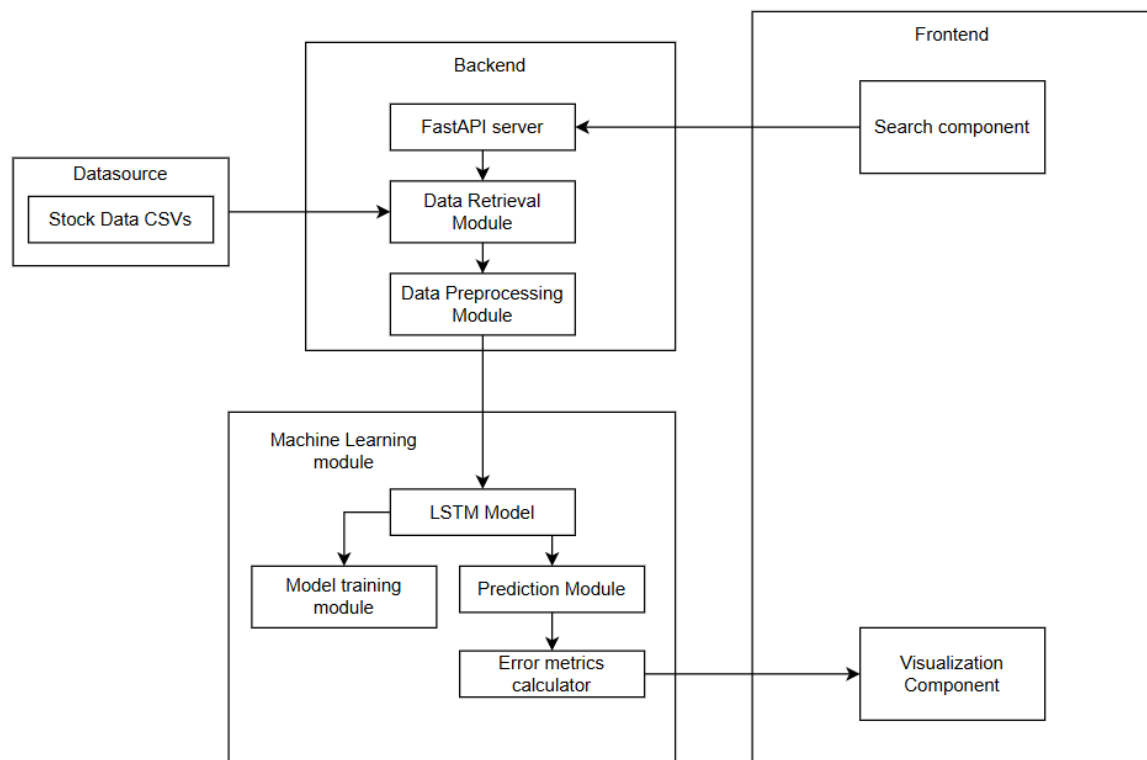


Figure 5.1 Implementation Process of stock price prediction system

5.1.2.1 Data Collection

This project's data came from a publicly accessible GitHub repository. A manual verification procedure was used to guarantee the dataset's dependability and accuracy. Before utilizing the data to train the LSTM model, it was necessary to thoroughly inspect it for mistakes or inconsistencies. Several stock price records with different attributes were included in the dataset; these were processed and examined to provide precise forecasts.

5.1.2.2 Data Preparation

To guarantee that the model can learn from the data in an efficient manner, data preparation is crucial. The data preparation stage of your stock price prediction project entails the following crucial steps:

1. Data Cleaning:

Handling Missing Values: Any missing data points are identified and addressed, either by filling them with appropriate values or removing the affected rows.

Removing Duplicates: Duplicate entries in the dataset are identified and removed to ensure data integrity.

2. Data Transformation:

Normalization: The raw data is normalized (scaled) to a specific range to ensure that all features contribute equally to the model training.

3. Data Splitting:

The dataset is divided into training and testing sets. Typically, a portion of the data (e.g., 80%) is used for training the model, while the remaining data (e.g., 20%) is reserved for testing its performance.

4. Data Formatting:

The data is formatted into the appropriate structure required by the model. This may involve reshaping the data into arrays that the LSTM model can process.

5. Data Validation:

The prepared data is validated to ensure that it meets the necessary criteria for analysis. This includes checking for consistency, accuracy, and relevance to the prediction task.

6. Final Review:

A final review of the prepared data is conducted to ensure that it is ready for model training. This step may involve visualizing the data to identify any remaining issues or patterns. By following these steps, the project ensures that the data used for analysis and

model building is accurate, complete, and relevant, ultimately leading to better model performance and more reliable predictions.

5.1.2.3 Model Optimization

Monitoring performance metrics during model training is essential for evaluating how well the model is learning and identifying areas for improvement. Root Mean Squared Error (RMSE) offers a similar perspective but expresses the error in the same units as the target variable, making it more interpretable. Mean Squared Error (MSE) calculates the average absolute differences between predictions and actual values, giving a straightforward measure of prediction accuracy. Mean Absolute Percentage Error (MAPE) expresses the error as a percentage, allowing for easy comparison across different datasets. By continuously monitoring these metrics throughout the training process, adjustments can be made to optimize the model, ensuring it generalizes well to unseen data and improves its predictive accuracy.

Table 5.3: Performance metrics on AHPC

	MSE	RMSE	MAPE
Custom LSTM	1036.7489	32.1986	13.17%
Hyperparameter tuned LSTM	298.4512	17.2757	5.45%

The results of the custom LSTM model were recorded using specific settings. The batch size was set to 64, the learning rate was 0.02, and the model was trained for 15 epochs. Additionally, a window size of 7 was used during training.

5.2 Testing

Testing is the procedure used to confirm the Stock Price Prediction System's correctness and dependability by examining its performance, quality, and usefulness. It is carried out at every stage of the development process to find and address any mistakes or defects. To verify the system's efficacy, it is assessed in relation to predetermined standards and specifications. To verify the overall operation of this project, including stock search, prediction accuracy, and data display, system testing was conducted in addition to unit testing, which was used to examine individual components. The system's smooth operation and accurate stock price predictions are ensured by these tests.

5.2.1 Unit Testing

Unit testing is a software testing methodology in which every component is examined

separately from the system as a whole. Unit testing's primary objective is to confirm that every unit operates as intended and error-free.

Test Cases

Table 5.4: Test get_stock_data function

Test Case ID	UT-001
Test Case Name	Test get_stock_data function for existing CSV file
Objectives	Verify that the get_stock_data function correctly loads stock data from an existing CSV file.
Pre-condition	A CSV file for the given stock ticker must exist.
Test Data	Ticker symbol for a stock with an existing CSV file.
Expected output	DataFrame containing the stock data loaded from the CSV file.
Post-condition	The function should return a DataFrame containing the stock data.
Status	Pass

Table 5.5: Test mse function

Test Case ID	UT-002
Test Case Name	Test mse function
Objectives	Verify that the mse function correctly calculates the Mean Squared Error (MSE).
Pre-condition	The function should receive valid actual and predicted stock prices.
Test Data	Actual prices and predicted prices.
Expected output	Correct MSE value.
Post-condition	The function should return the correct MSE value.
Status	Pass

Table 5.6: Test rmse function

Test Case ID	UT-003
Test Case Name	Test rmse function
Objectives	rmse function correctly calculates the Root Mean Squared Error (RMSE).
Pre-condition	The function should receive valid actual and predicted stock prices.
Test Data	Actual prices and predicted prices.

Expected output	Correct RMSE value.
Post-condition	The function should return the correct RMSE value.
Status	Pass

Table 5.7: Test mape function

Test Case ID	UT-004
Test Case Name	Test mape function
Objectives	Verify that the mape function correctly calculates the Mean Absolute Percentage Error (MAPE).
Pre-condition	The function should receive valid actual and predicted stock prices.
Test Data	Actual prices and predicted prices.
Expected output	Correct MAPE value.
Post-condition	The function should return the correct MAPE value.
Status	Pass

5.2.2 Integration Testing

Integration testing examines the interoperability of various system modules or components. It guarantees proper data flow between integrated components.

Test Cases

Table 5.8: Test for valid ticker

Test Case ID	IT-001
Test Case Name	Test /predict endpoint for valid ticker
Objectives	Verify that the /predict endpoint returns predictions and accuracy metrics for a valid ticker.
Pre-condition	The API must be running and contain historical data for the requested ticker.
Test Data	POST request to /predict with a valid ticker.
Expected output	JSON response containing predictions and accuracy metrics.
Post-condition	The response should contain predictions and accuracy metrics in JSON format.
Status	Pass

Table 5.9: Test /get_available_tickers

Test Case ID	IT-002
Test Case Name	Test /get_available_tickers endpoint
Objectives	Verify that the /get_available_tickers endpoint returns a list of available tickers.
Pre-condition	The API must be running and contain available tickers.
Test Data	GET request to /get_available_tickers.
Expected output	JSON response containing a list of available tickers.
Post-condition	The response should return a JSON object containing a list of available tickers.
Status	Pass

Table 5.10: Test integration of StockForm component with API

Test Case ID	IT-003
Test Case Name	Test integration of StockForm component with API
Objectives	Verify that the StockForm component correctly interacts with the API and displays prediction results.
Pre-condition	The frontend application must be running and properly connected to the API.
Test Data	User submits a valid ticker in the form.
Expected output	The component displays the prediction results and accuracy metrics.
Post-condition	The component should display prediction results and accuracy metrics.
Status	Pass

5.2.3 System Testing

System testing assesses the complete application to ensure that it satisfies the criteria. The system's performance, security, and functionality are all tested.

Test Cases

Table 5.11: End-to-End Test for Stock Prediction

Test Case ID	ST-001
Test Case Name	End-to-End Test for Stock Prediction
Objectives	Verify that the application correctly processes user input, interacts with the API, and displays stock predictions along with accuracy metrics.
Pre-condition	The application and API must be running with historical stock data available.
Test Data	User enters a valid ticker and submits the form.
Expected output	The application displays the predicted stock prices and accuracy metrics.
Post-condition	The application should display the predicted stock prices and accuracy metrics.
Status	Pass

Table 5.12: Test Application Health Check

Test Case ID	ST-002
Test Case Name	Test Application Health Check
Objectives	Verify that the API health check endpoint responds correctly and indicates the application's health status.
Pre-condition	The API must be running.
Test Data	GET request to /health.
Expected output	JSON response with {"status": "healthy"}.
Post-condition	The response should return {"status": "healthy"} in JSON format.
Status	Pass

Table 5.13: Test User Interface for Error Handling

Test Case ID	ST-003
Test Case Name	Test User Interface for Error Handling
Objectives	Verify that the application correctly handles invalid input and provides a user-friendly error message.
Pre-condition	The application and API must be running.
Test Data	User enters an invalid ticker and submits the form.

Expected output	The application displays an error message.
Post-condition	The application should display an appropriate error message indicating the issue.
Status	Pass

5.3 Result Analysis

The step of result analysis is when a trained model's performance is examined and analyzed to determine how well it produces correct predictions. In order to determine whether a model achieves the desired outcomes and how effectively it generalizes unknown data, result analysis is essential.

The model's performance was assessed using the following evaluation metrics:

5.3.1 Mean Squared Error (MSE)

MSE measures the average of the squares of the errors, which are the differences between predicted and actual values. It provides a sense of how far off predictions are from the actual outcomes. MSE is sensitive to outliers because it squares the errors, which means larger errors have a disproportionately large effect on the metric. A lower MSE indicates better model performance.

Formula:

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \tilde{y}_i)^2$$

MSE = Mean squared error

y_i = observed value

\tilde{y}_i = predicted value

n = number of data points

5.3.2 Root Mean Squared Error (RMSE)

RMSE is the square root of the Mean Squared Error. It provides a measure of error in the same units as the target variable, making it easier to interpret. RMSE is particularly useful for understanding the model's performance in practical terms, as it reflects the average distance between predicted and actual values. Like MSE, lower RMSE values indicate better performance.

Formula:

$$RMSE = \sqrt{\frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{n}}$$

RMSE = Root mean squared error

y_i = observed value

\hat{y}_i = predicted value

5.3.3 Mean Absolute Percentage Error (MAPE)

MAPE expresses the error as a percentage of the actual values, providing a relative measure of accuracy that is easy to interpret. MAPE is particularly useful for comparing the accuracy of predictions across different datasets or scales, as it normalizes the error relative to the actual values. A lower MAPE indicates better model performance.

Formula:

$$MAPE = \frac{1}{n} \sum_{t=1}^n \left| \frac{A_t - F_t}{A_t} \right|$$

MAPE = mean absolute percentage error

n = number of times the summation iteration happens

A_t = actual value

F_t = forecast value

Table 5.14: Result Analysis on AHPC

Evaluation metrics	Results
Mean Squared Error	298.4512
Root Mean Squared Error	17.2757
Mean Absolute Percentage Error	5.45 %

The results were calculated using a hyperparameter tuned LSTM model. The optimized values used were a learning rate of 0.0014, 15 epochs, a batch size of 15, and a window size of 6.

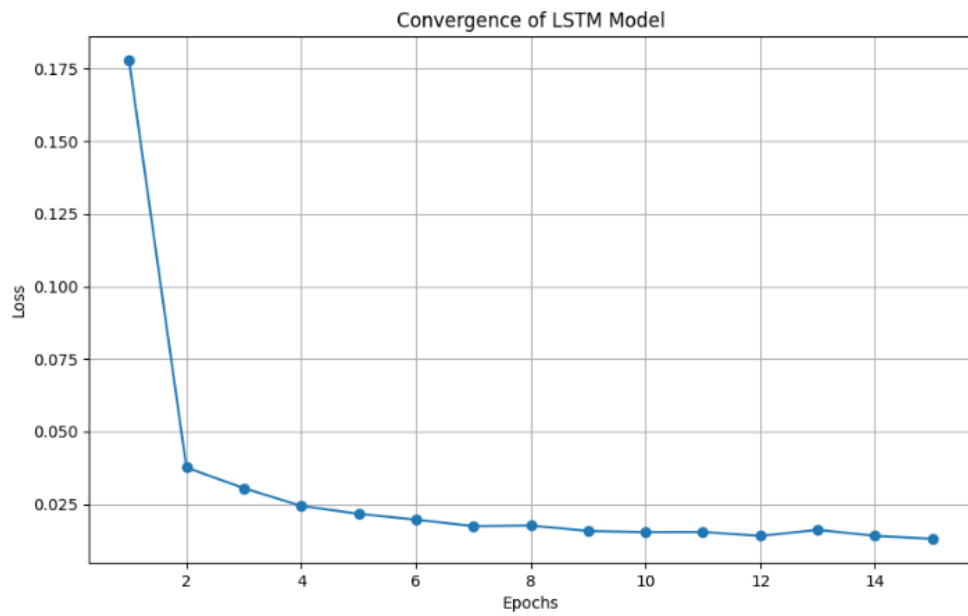


Figure 5.2 Convergence graph of LSTM Model

This graph shows how the LSTM model's loss decreases over epochs, helping to find the best number of epochs for training.

CHAPTER 6 CONCLUSION AND FUTURE RECOMMENDATION

6.1 Conclusion

In conclusion, the Stock Price Prediction System using LSTM has successfully demonstrated the potential of machine learning to predict stock prices based on historical data. By leveraging Python, FastAPI, and other tools, the system provides accurate predictions and a user-friendly interface for investors and analysts. The project highlights the effectiveness of LSTM models for time-series forecasting and offers valuable insights into the stock market. Overall, the system is a useful tool for making informed investment decisions and has room for further enhancement in terms of data sources and model accuracy.

6.2 Future Recommendation

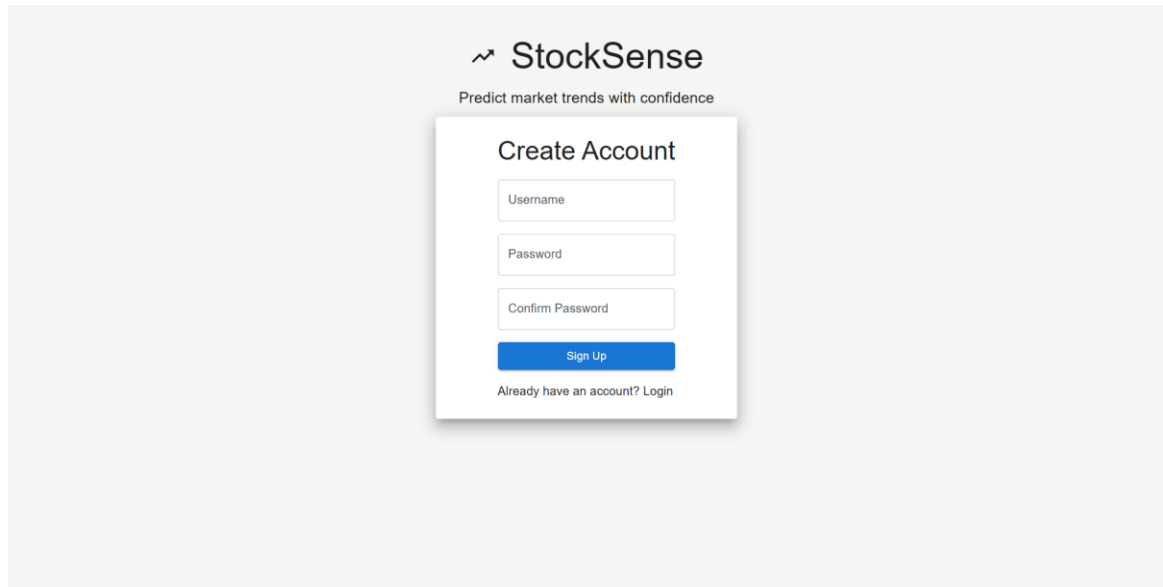
Future enhancements to the stock price prediction system can include adding more data sources, such as news sentiment and social media, utilizing sophisticated machine learning models, like Transformers, and integrating real-time data for the most recent forecasts. Other worthwhile avenues include investigating deep reinforcement learning for trading techniques, integrating risk management and portfolio optimization, and improving model interpretability. Accessibility and dependability would be further enhanced by a more user-friendly interface, cloud deployment enabling scalability, and guaranteeing regulatory compliance. The system may become more precise, responsive, and helpful for analysts and investors as a result of these developments.

REFERENCES

- [1] S. Selvin, R. Vinayakumar, E. A. Gopalakrishnan, V. K. Menon, and K. P. Soman, "Stock price prediction using LSTM, RNN and CNN-sliding window model," in Proc. Int. Conf. Adv. Comput. Commun. Inform. (ICACCI), Sep. 2017, pp. 1643–1647.
- [2] Y. Chen, Y. Zhou, B. Dai, and H. Chen, "A LSTM-based method for stock returns prediction: A case study of China stock market," in Proc. 2015 IEEE Int. Conf. Big Data (Big Data), Oct. 2015, pp. 2823–2824.
- [3] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with LSTM," *Neural Comput.*, vol. 12, no. 10, pp. 2451–2471, Oct. 2000.
- [4] X. Ding, Y. Zhang, T. Liu, and J. Duan, "Deep learning for event-driven stock prediction," in Proc. 24th Int. Conf. Artif. Intell. (IJCAI), Jul. 2015, pp. 2327–2333.
- [5] T. Bhardwaj and S. Rao, "STOCK PRICE PREDICTION USING LSTM," ResearchGate. [Online].
- [6] Jyoti Prakash Behura, Sagar Dhanraj Pande, and V. Naga, "Stock Price Prediction using Multi-Layered Sequential LSTM," *ICST Transactions on Scalable Information Systems*, Dec. 2023, doi: <https://doi.org/10.4108/eetsis.4585>.

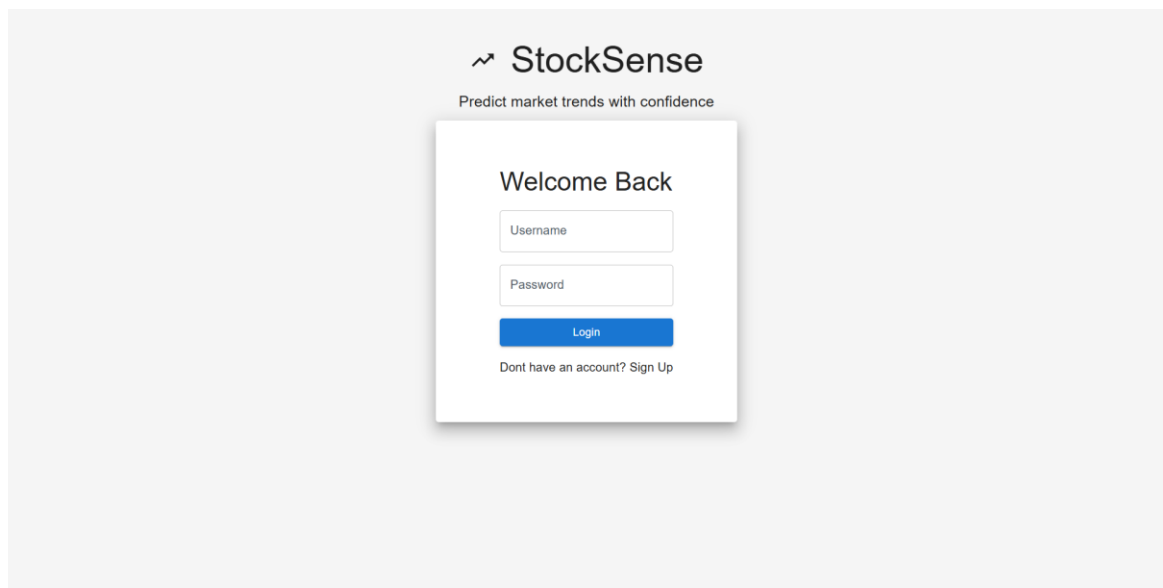
APPENDIX

UI Screenshots:



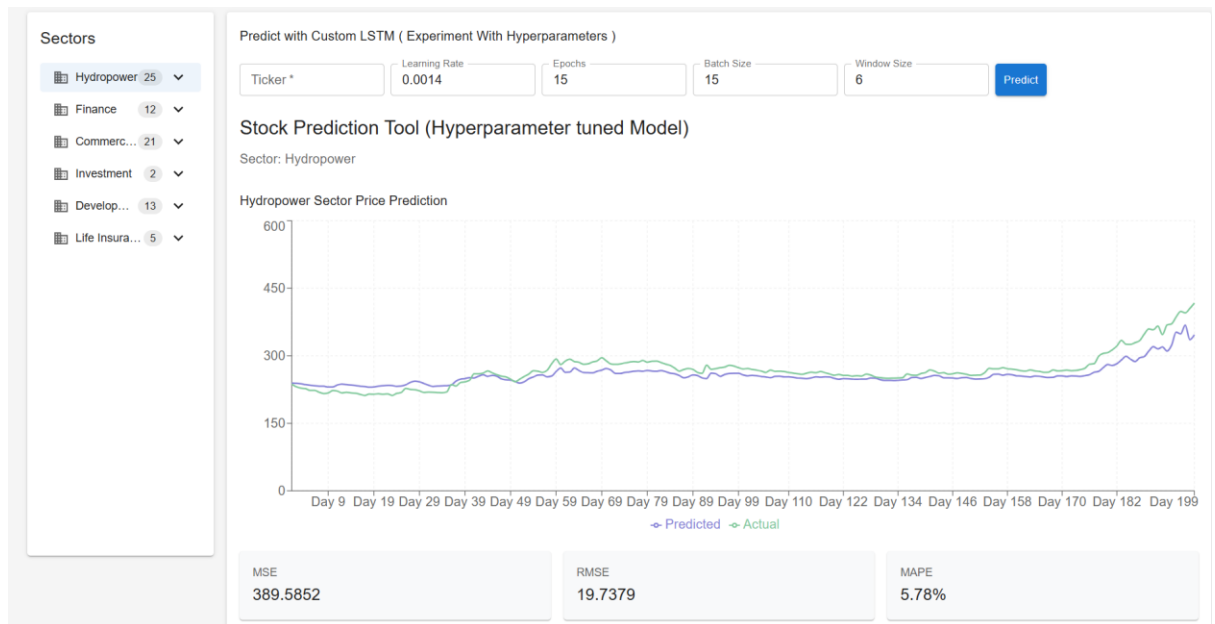
The screenshot shows the 'Create Account' form on the StockSense website. At the top, the StockSense logo is displayed with the tagline 'Predict market trends with confidence'. The form is centered and contains three input fields: 'Username', 'Password', and 'Confirm Password'. Below these fields is a blue 'Sign Up' button. At the bottom of the form, there is a link that says 'Already have an account? Login'.

Appendix A: Signup Page

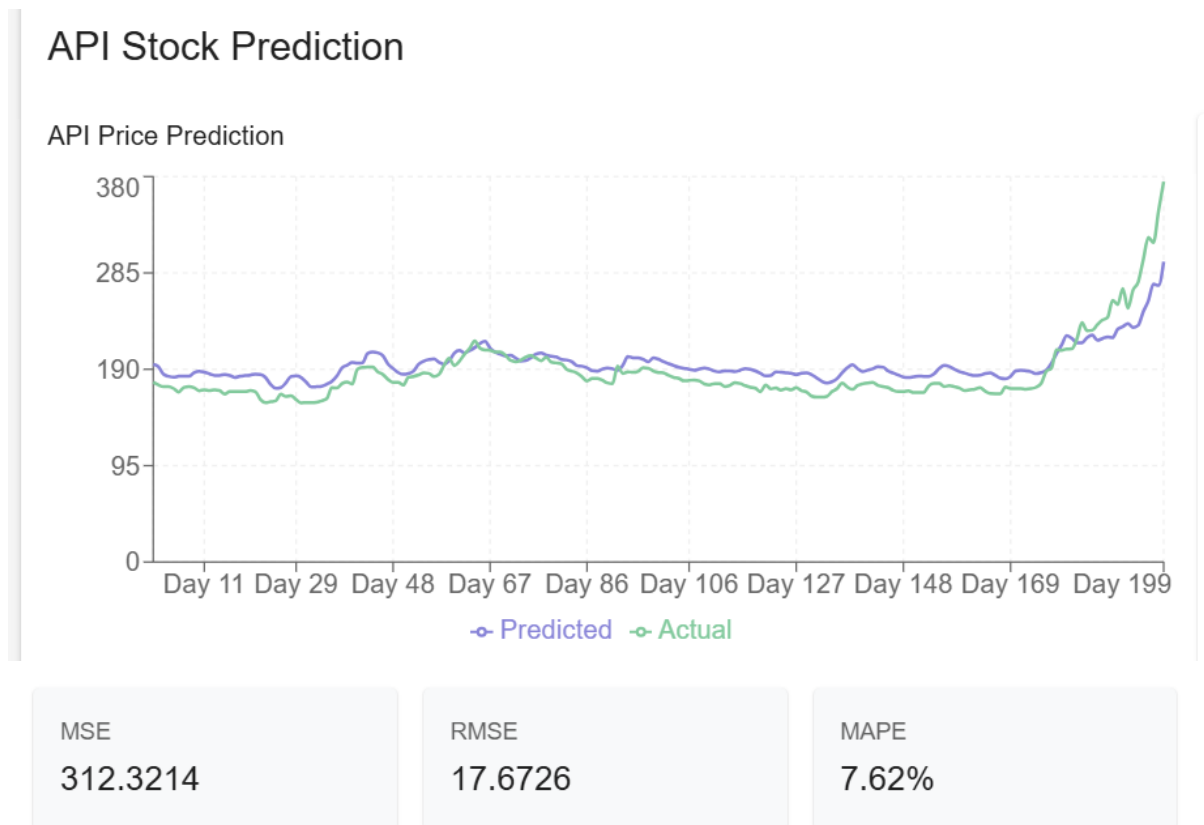


The screenshot shows the 'Welcome Back' form on the StockSense website. At the top, the StockSense logo is displayed with the tagline 'Predict market trends with confidence'. The form is centered and contains two input fields: 'Username' and 'Password'. Below these fields is a blue 'Login' button. At the bottom of the form, there is a link that says 'Dont have an account? Sign Up'.

Appendix B: Login Page



Appendix C: Sector Prediction Page



Appendix D: Hyperparameter Tuned Stock Prediction Page

Predict with Custom LSTM (Experiment With Hyperparameters)

Ticker *

API

Learning Rate

0.002

Epochs

20

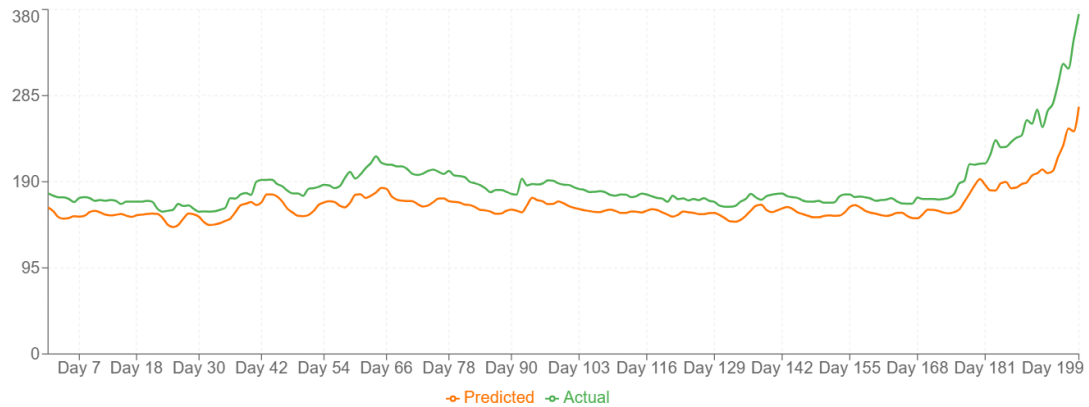
Batch Size

32

Window Size

6

Predict



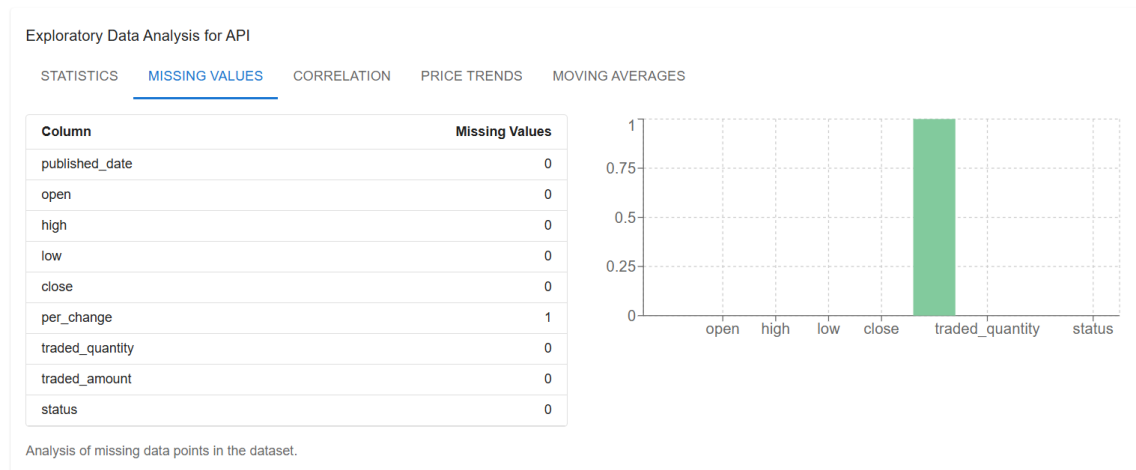
MSE	RMSE	MAPE
795.6399	28.2071	11.84%

Appendix E: Custom LSTM Prediction Page

[Home](#) / [Stock Analysis](#) / [API](#)

API Exploratory Data Analysis

Sector: Hydropower



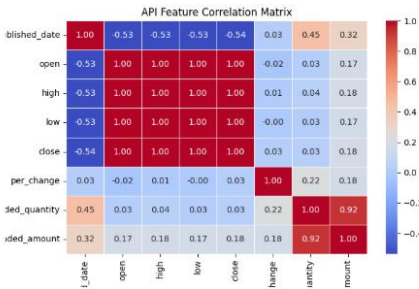
Appendix F: EDA Missing Values Page

API Exploratory Data Analysis

Sector: Hydropower

Exploratory Data Analysis for API

STATISTICS MISSING VALUES CORRELATION PRICE TRENDS MOVING AVERAGES



Correlation between different metrics affecting the stock.

Appendix G: EDA Correlation Page

API Exploratory Data Analysis

Sector: Hydropower

Exploratory Data Analysis for API

STATISTICS MISSING VALUES CORRELATION PRICE TRENDS MOVING AVERAGES



Historical closing price trends for the stock.

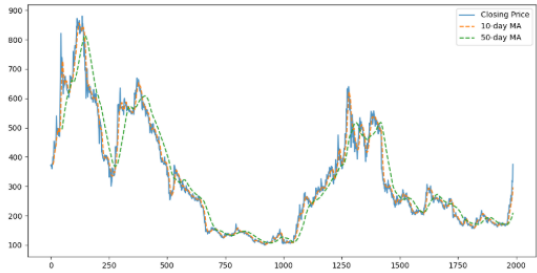
Appendix H: EDA Price Trends Page

API Exploratory Data Analysis

Sector: Hydropower

Exploratory Data Analysis for API

STATISTICS MISSING VALUES CORRELATION PRICE TRENDS MOVING AVERAGES



Moving averages indicate price trends over different time periods.

Appendix I: EDA Moving Averages Page

Stock Price Prediction

HomeExplore DataAboutU

↗ StockSense

Logout

R

Hello rajan

Predictions Made: 2

🕒 Prediction History

Predicted At	Ticker	Batch Size	Learning Rate	Epochs	Window Size	MSE	RMSE	MAPE
2025-03-15 19:10	AHPC	15	0.0014	15	6	298.45	17.28	5.45%
2025-03-18 10:54	API	32	0.002	20	6	795.64	28.21	11.84%

Your recent prediction history is displayed above.

Appendix J: User Dashboard Page