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.

……..…….

Er. Dhiraj Kumar Jha Project Coordinator,

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.

…… **Er. Dhiraj Kumar Jha** Supervisor, Orchid International College Bijayachowk, Gaushala	…… **Er. Dhiraj Kumar Jha** Head of Deprtment, Orchid International College Bijayachowk, Gaushala
…… **Internal Examiner** Orchid International College Bijayachowk, Gaushala	…… **External Examiner** Central Department of Computer Science and IT, Tribhuvan University, Kirtipur, Nepal

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 Er. Dhiraj Kumar Jha, 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

LIST OF FIGURES

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:

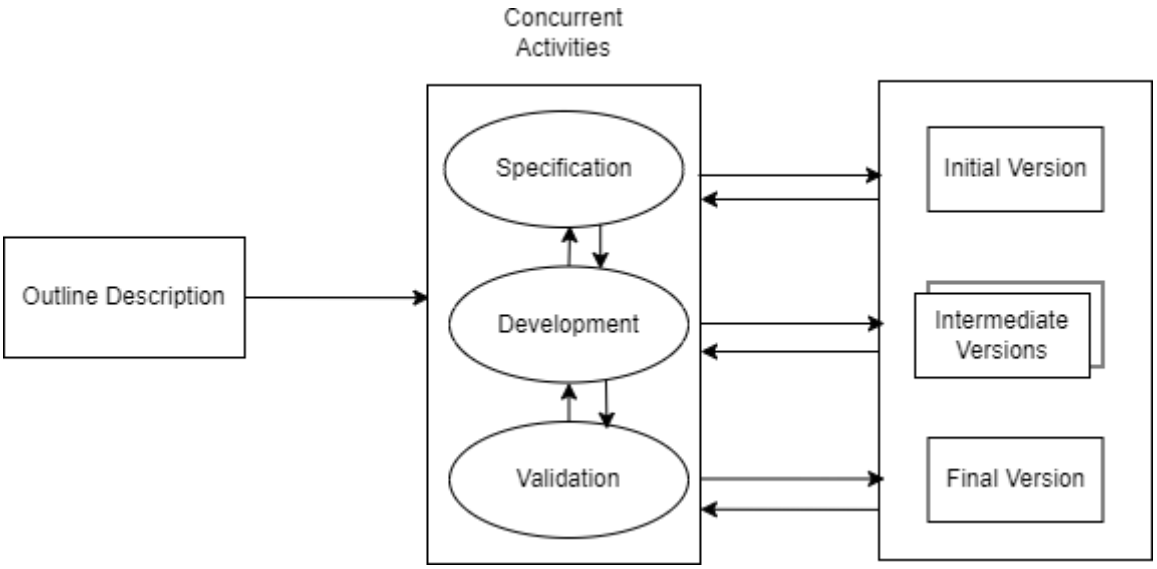


Figure 1.1: Incremental Model

1. Initial Setup

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 and Matplotlib for data visualization, was essential. 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.

2. Data Collection

In this phase, we integrated APIs to collect stock data, allowing the system to fetch historical prices and relevant metrics. The primary challenge was dealing with inconsistent data formats from different sources, which required additional data cleaning and transformation steps. Additionally, rate limits imposed by APIs sometimes hindered the ability to fetch large datasets quickly. We also had to ensure that the data collected was reliable and up-to-date, which involved implementing error handling to manage potential API failures or data retrieval issues.

3. Model Development

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.

4. User Interface

The front-end was built incrementally, creating components like forms and dashboards to display predictions. A challenge during this phase was ensuring a responsive and user-friendly design that worked across different devices and screen sizes. 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.

5. Testing

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, but managing multiple branches and merges was occasionally complex. Additionally, ensuring that the application performed well under various conditions required extensive testing and optimization.

6. Final Integration

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 extensive testing and adjustments based on internal evaluations 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 fetch real-time and historical stock market data from reliable sources.
- To use Long Short-Term Memory (LSTM) networks for stock price prediction.
- To allow users to compare actual and predicted stock prices.
- To display evaluation metrics such as MSE, RMSE and MAPE to assess model accuracy.

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

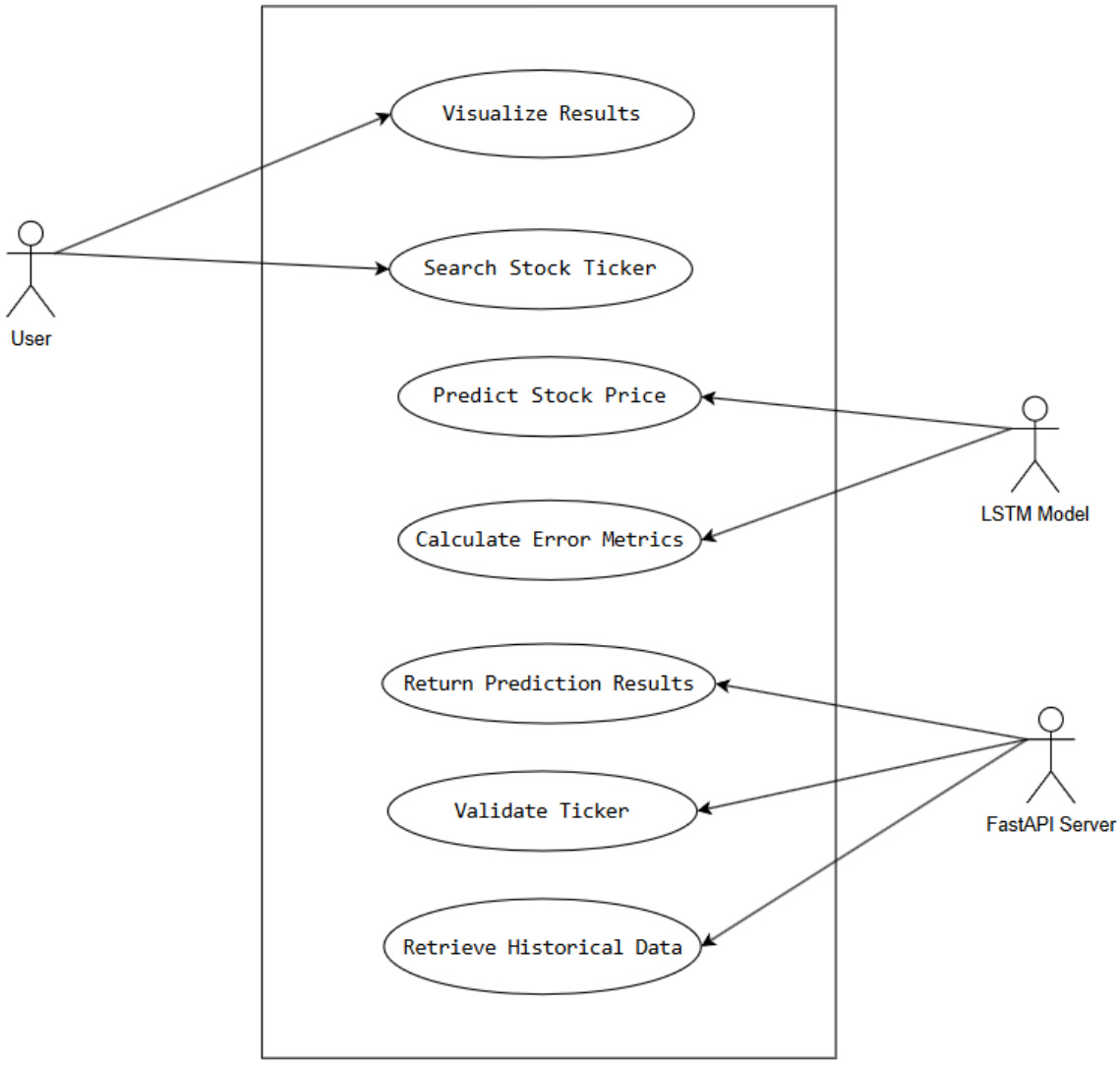


Figure 3.1: Use Case Diagram

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:	- Application is running - User has internet connection - User knows the stock ticker symbol

Postconditions:	- Ticker symbol is validated - Historical stock data is retrieved - Prediction process is initiated

Table 3.1.1: Search Stock Ticker

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:	- Historical stock price data is preprocessed - LSTM model is trained and loaded
Postconditions:	- Price predictions are generated - Prediction confidence is calculated - Error metrics are computed

Table 3.1.2: Predict Stock Price

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:	- Predictions and error metrics are generated - Server connection is

	established
Postconditions:	- Prediction data is packaged into JSON - Response is sent to frontend - Frontend is ready to visualize results

Table 3.1.3: Return Prediction 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:	- Prediction data received from server - Visualization components are loaded
Postconditions:	- Price prediction chart is rendered - Error metrics are displayed - User can interact with visualization

Table 3.1.4: Visualize Results

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 2 seconds 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 such as Python, Numpy, Pandas. 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) to login with ease, 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 10/28/2**	**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

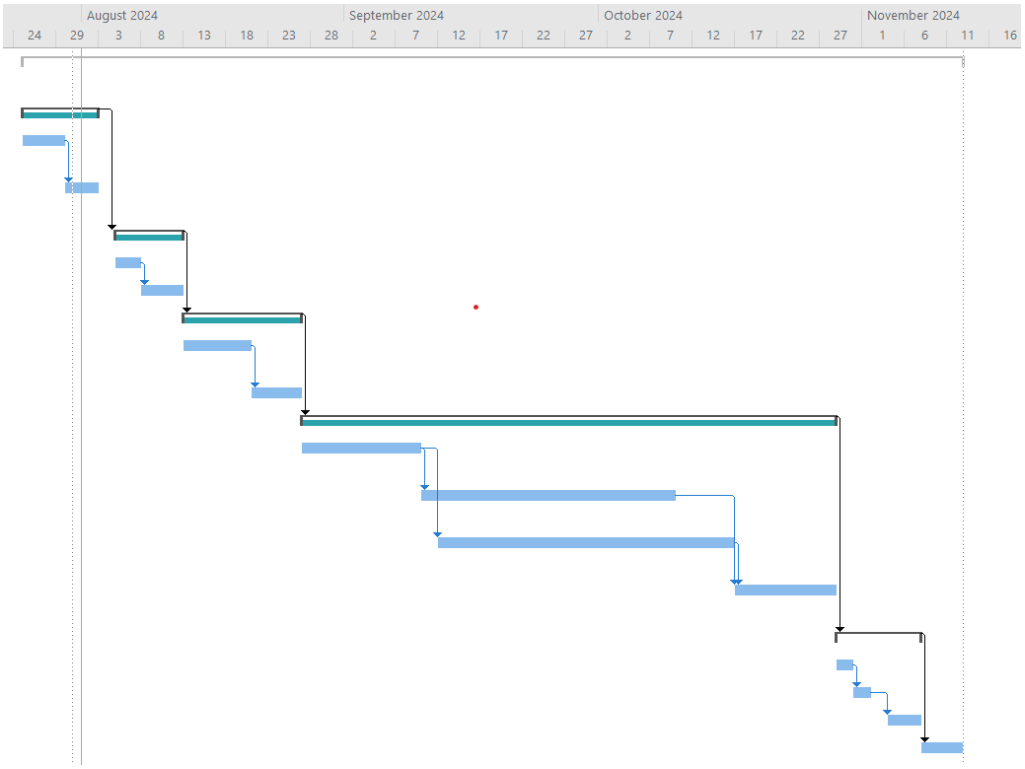


Figure 3.3: Gantt Chart

3.1.3 Analysis

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

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. In this project, various diagrams are used to illustrate the system's entire workflow and functionality

4.1.1 Class Diagram

A class diagram represents the structure of the Stock Price Prediction System by illustrating different classes (components) and their relationships. It visually maps how key elements, such as data handling, user interactions, and stock price predictions, are connected. This diagram helps in understanding how the system organizes data, processes requests, and generates predictions efficiently.

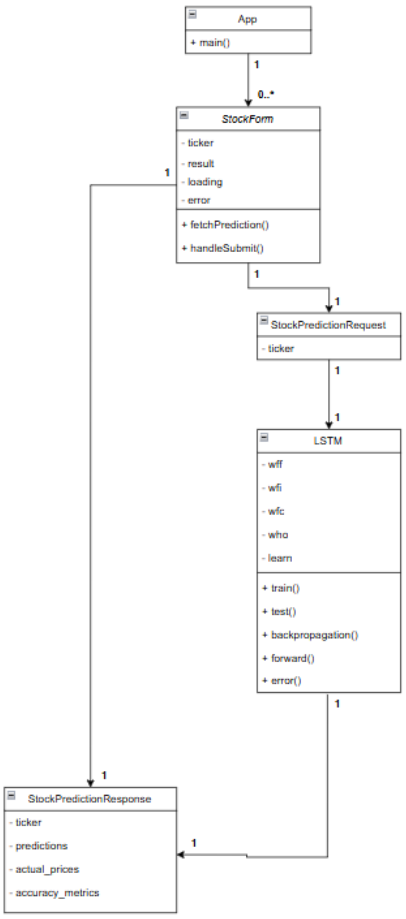


Figure 4.1 Class Diagram

4.1.2 Activity Diagram

An activity diagram depicts the flow of activities or tasks in a system. It helps to depict the steps that users take, such as logging in, entering stock symbols, obtaining predictions, and viewing results. It resembles a flowchart for actions.

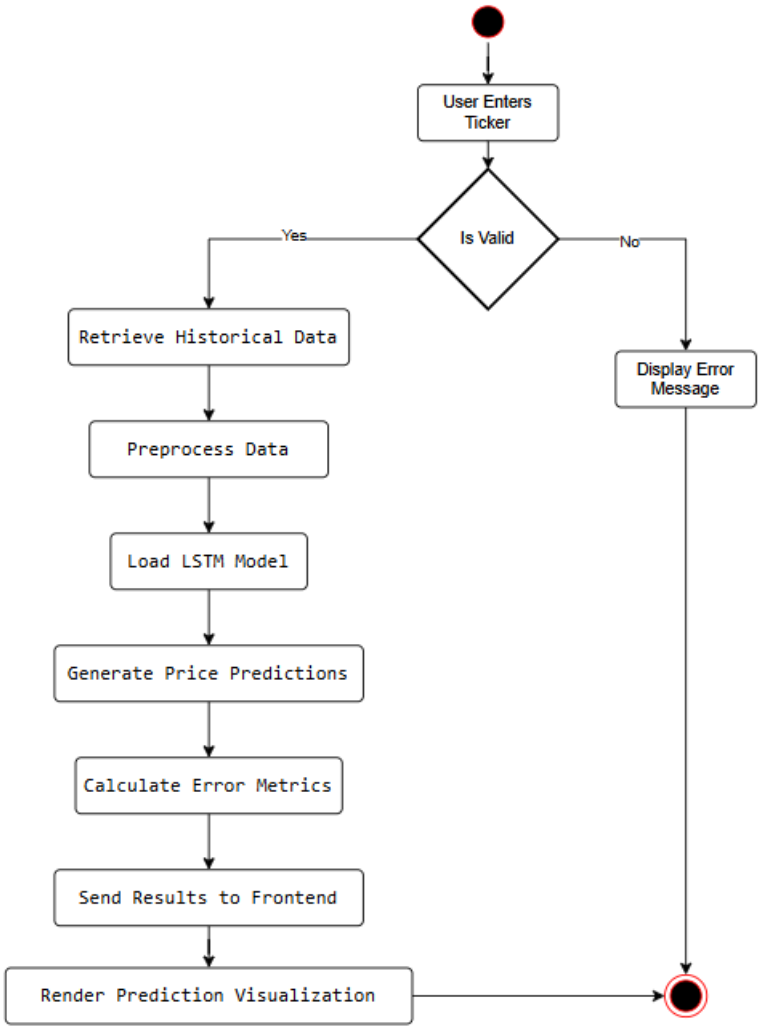


Figure 4.2 Activity Diagram

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:

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

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:

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

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:

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

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:

$$h_t = O_t * \tan h(ct)$$

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.

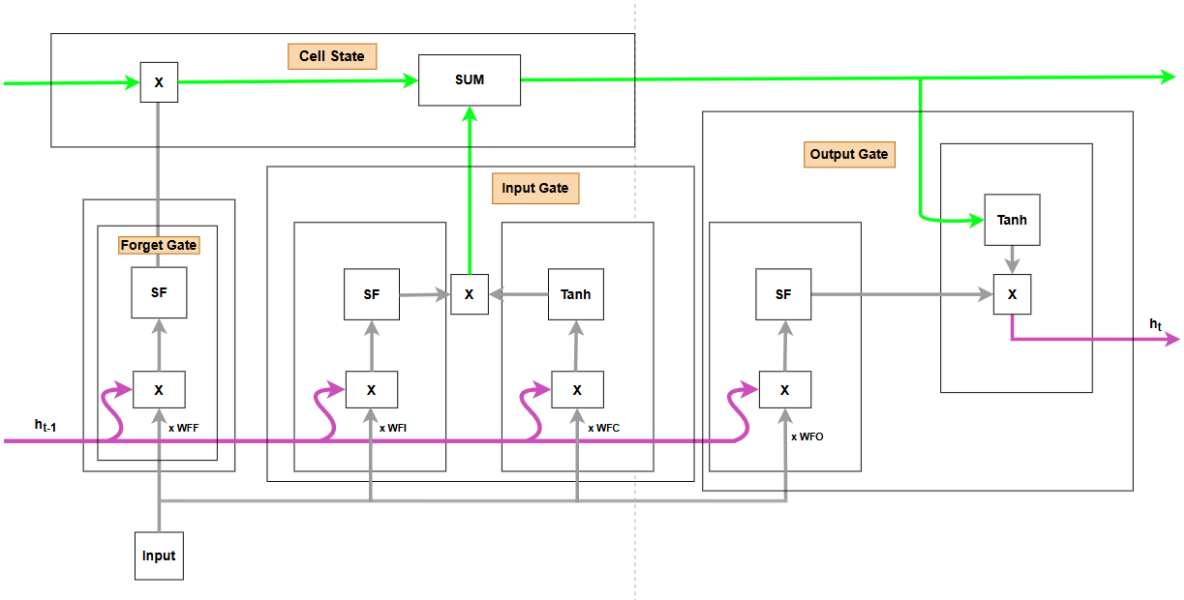


Figure 4.3: Architecture of LSTM

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:

Front-end Tools

An application's user interface is created using front-end tools, which enable direct user interaction with the system. Between the user and the back-end, the front-end serves as a mediator. The following front-end tools were utilized in this project:

React

React is a popular JavaScript library used for building dynamic and interactive user interfaces. The front-end of the Stock Price Prediction System was constructed in this project using React, which allowed for smooth interactions, dynamic updates, and fluid transitions between various views and functionality.

Back-end tools

Back-end tools are used to ensure that the system operates effectively when users interact with the web application. The resources and data access required to support the front-end and react to user requests are provided by the back-end. The following back-end tools were used in the project's development:

Python

Python is a popular and adaptable programming language that is renowned for being easy to understand. Python is utilized in this project to manage data processing, incorporate machine learning models (such the LSTM for stock price prediction), and make it easier for the front-end and back-end components to communicate with one another. In order to run the prediction model, retrieve and process data, and guarantee seamless connection between the client interface and the server, Python scripts are necessary.

FastAPI

FastAPI is a cutting-edge, fast web framework for Python API development. The RESTful API, which handles HTTP requests including retrieving stock data, receiving prediction queries, and sending results to the front-end, is created and managed in this project using FastAPI.

Visual Studio Code

An open-source, cross-platform, and lightweight IDE for developing a variety of applications is Visual Studio Code. It provides a vast array of features for project development.

Algorithm Development Tools

The tools utilized for algorithm development and system deployment are as follows:

Jupyter Notebook

An interactive, open-source web platform called Jupyter Notebook is used to analyze and create a variety of applications. It speeds up project development by facilitating team collaboration and data visualization.

CASE Tools

Project analysis and development are done with CASE tools. They are employed to create the necessary diagrams and offer a range of project artifacts. The project made use of the following CASE tools:

Draw.io

A free and open-source CASE tool that works with various platforms is Draw.io. Object, class, use case, activity, and sequence diagrams are among the many wireframes and diagrams that may be made with it.

Microsoft Project

Microsoft Project is a tool for project management. Microsoft Project is used for this project's scheduling, risk management, resource allocation, and job breakdown.

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:

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.

Table 5.1 External Modules

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.

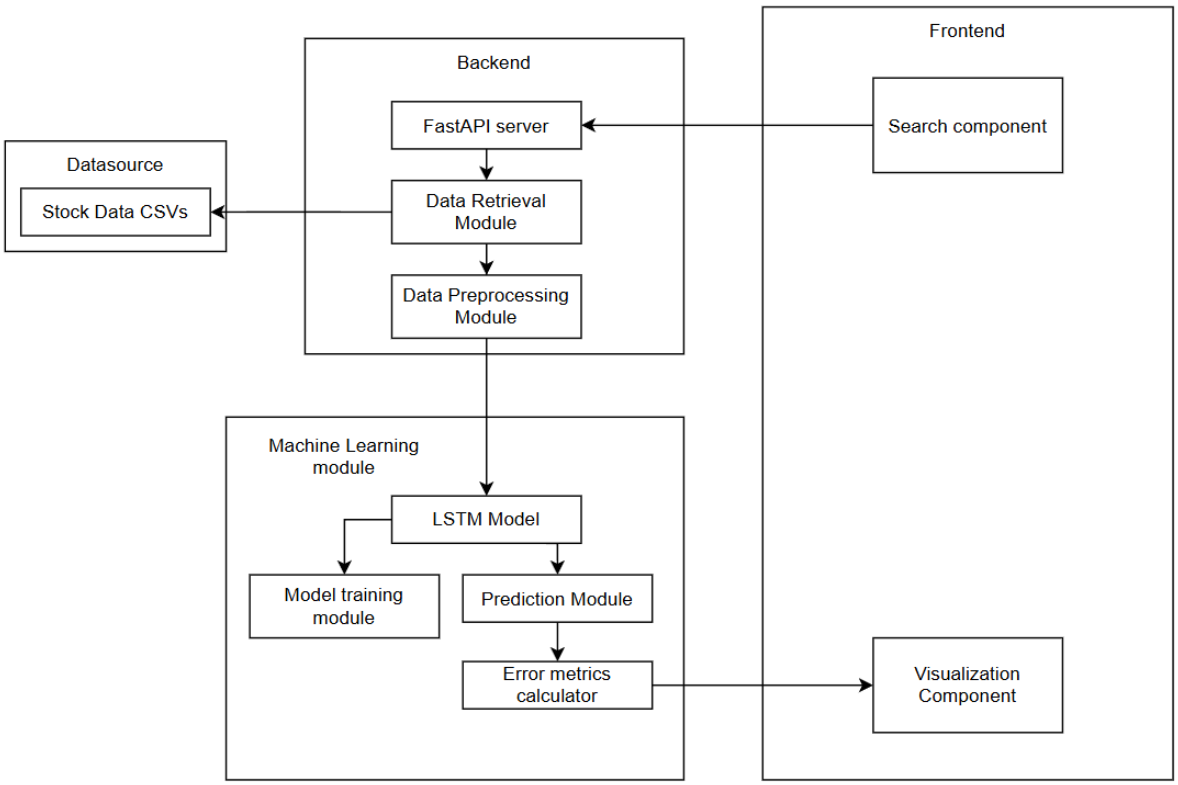


Figure 5.1 Implementation Process

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 (e.g., dividing by 1000) 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 or tensors 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 Absolute Error (MAE) 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.

	MSE	RMSE	MAPE
Before	958.0248	30.9520	4.92%
After	75.5887	8.6942	1.50%

Table 5.1.2.3: Performance mertics

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

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 5.2.1.1: Test get_stock_data function

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

Table 5.2.1.2: Test mae 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 5.2.1.3: Test rmse 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

Table 5.2.1.4: Test mape function

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

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 5.2.2.1: Test for valid ticker

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 5.2.2.2: Test /get_available_tickers

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

Table 5.2.2.3: Test integration of StockForm component with API

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

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 5.2.3.1: End-to-End Test for Stock Prediction

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 5.2.3.2: Test Application Health Check

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

Table 5.2.3.3: Test User Interface for Error Handling

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:

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

RMSE = Root mean squared error

y_i = observed value

$\tilde{y}_i$ = predicted value

n = number of data pointsRoot 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:

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

RMSE = Root mean squared error

y_i = observed value

$\tilde{y}_i$ = predicted value

5.3.2 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:

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

MAPE = mean absolute percentage error

n = number of times the summation iteration happens

A_t = actual value

F_t = forecast value

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.

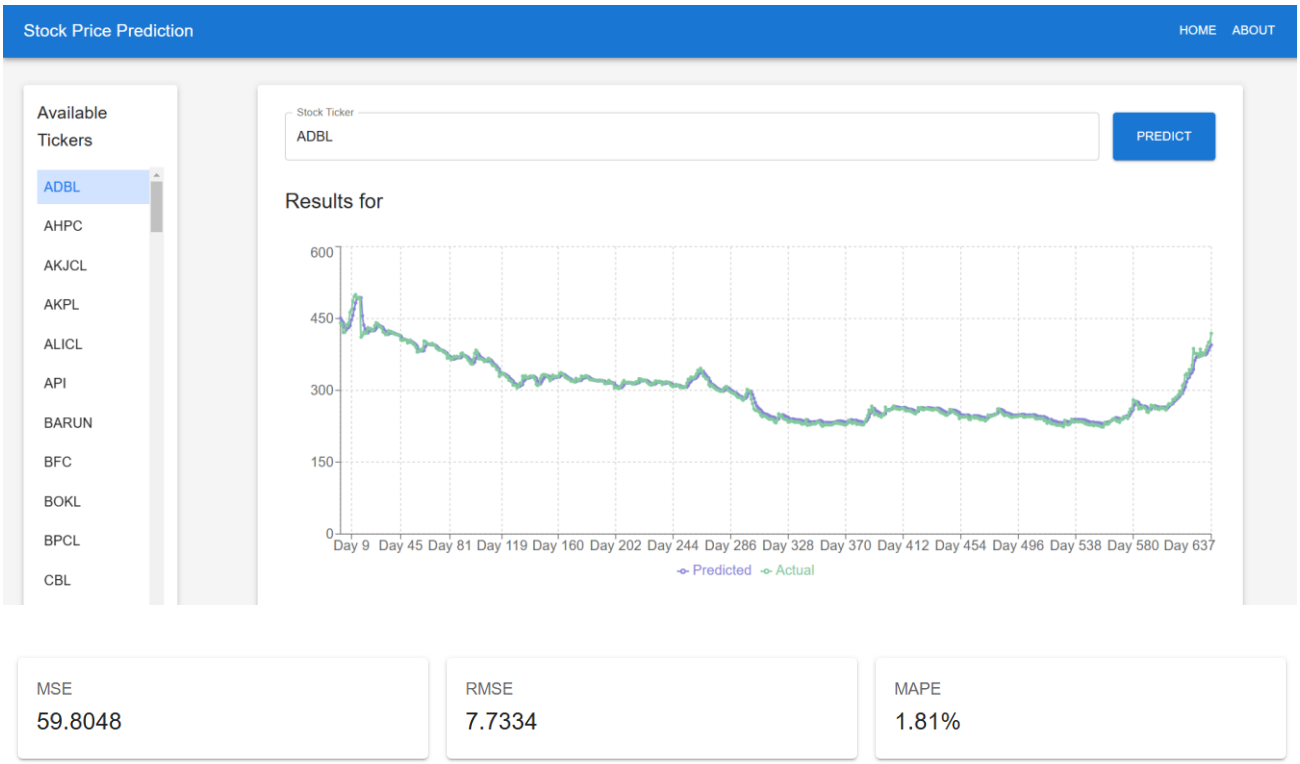
[3] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with LSTM," Neural Comput., vol. 12, no. 10, pp. 2451–2471, Oct. 2000.

[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]. Available: https://www.researchgate.net/publication/380494066_STOCK_PRICE_PREDICTION_USING_LSTM.

APPENDIX

UI Screenshots:



Appendix A: Prediction Page

Pseudocode:

```
import numpy as np

class LSTM:
    def __init__(self, input_size=1, hidden_size=50, output_size=1, learning_rate=0.001):
        # Parameters
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.lr = learning_rate
```

```
    # Gates weights (input, forget, output, cell)
    self.Wi = np.random.randn(hidden_size, input_size + hidden_size) * 0.01
    self.Wf = np.random.randn(hidden_size, input_size + hidden_size) * 0.01
    self.Wo = np.random.randn(hidden_size, input_size + hidden_size) * 0.01
    self.Wc = np.random.randn(hidden_size, input_size + hidden_size) * 0.01

    # Output weights
    self.Wy = np.random.randn(output_size, hidden_size) * 0.01

    # Biases
    self.bi = np.zeros((hidden_size, 1))
    self.bf = np.zeros((hidden_size, 1))
    self.bo = np.zeros((hidden_size, 1))
    self.bc = np.zeros((hidden_size, 1))
    self.by = np.zeros((output_size, 1))

def sigmoid(self, x):
    return 1 / (1 + np.exp(-x))

def tanh(self, x):
    return np.tanh(x)

def forward(self, x_sequence):
    h_prev = np.zeros((self.hidden_size, 1))
    c_prev = np.zeros((self.hidden_size, 1))
    self.cache = []

    for x in x_sequence:
        x = x.reshape(-1, 1)
        combined = np.vstack((x, h_prev))

        # Input gate
        i = self.sigmoid(np.dot(self.Wi, combined) + self.bi)
```

```
            # Forget gate
            f = self.sigmoid(np.dot(self.Wf, combined) + self.bf)
            # Output gate
            o = self.sigmoid(np.dot(self.Wo, combined) + self.bo)
            # Cell state
            c_candidate = self.tanh(np.dot(self.Wc, combined) + self.bc)
            c_prev = f * c_prev + i * c_candidate
            # Hidden state
            h_prev = o * self.tanh(c_prev)

            self.cache.append((x, combined, i, f, o, c_candidate, h_prev, c_prev))

        # Output layer
        y = np.dot(self.Wy, h_prev) + self.by
        return y, h_prev

    def backward(self, dy):
        dWi, dWf, dWo, dWc = np.zeros_like(self.Wi), np.zeros_like(self.Wf),
np.zeros_like(self.Wo), np.zeros_like(self.Wc)
        dbi, dbf, dbo, dbc = np.zeros_like(self.bi), np.zeros_like(self.bf),
np.zeros_like(self.bo), np.zeros_like(self.bc)
        dWhy = np.dot(dy, self.cache[-1][6].T)
        dby = dy

        dh_next = np.zeros_like(self.cache[0][6])
        dc_next = np.zeros_like(self.cache[0][7])

        for t in reversed(range(len(self.cache))):
            x, combined, i, f, o, c_candidate, h_prev, c_prev = self.cache[t]

            dh = np.dot(self.Wy.T, dy) + dh_next
            do = dh * self.tanh(c_prev)
            do_raw = do * o * (1 - o)
```

```
dc = dc_next + (dh * o * (1 - self.tanh(c_prev)**2))
dc_candidate = dc * i
dc_candidate_raw = dc_candidate * (1 - c_candidate**2)

di = dc * c_candidate
di_raw = di * i * (1 - i)

df = dc * c_prev
df_raw = df * f * (1 - f)

dWc_t = np.dot(dc_candidate_raw, combined.T)
dWo_t = np.dot(do_raw, combined.T)
dWf_t = np.dot(df_raw, combined.T)
dWi_t = np.dot(di_raw, combined.T)

dcomb = (np.dot(self.Wc.T, dc_candidate_raw) +
        np.dot(self.Wo.T, do_raw) +
        np.dot(self.Wf.T, df_raw) +
        np.dot(self.Wi.T, di_raw))

dh_prev = dcomb[self.input_size:, :]
dc_prev = f * dc

# Accumulate gradients
dWi += dWi_t
dWf += dWf_t
dWo += dWo_t
dWc += dWc_t
dbi += di_raw
dbf += df_raw
dbo += do_raw
dbc += dc_candidate_raw
```

```
            dh_next = dh_prev
            dc_next = dc_prev

        # Update parameters
        self.Wi -= self.lr * dWi
        self.Wf -= self.lr * dWf
        self.Wo -= self.lr * dWo
        self.Wc -= self.lr * dWc
        self.Wy -= self.lr * dWhy
        self.bi -= self.lr * dbi
        self.bf -= self.lr * dbf
        self.bo -= self.lr * dbo
        self.bc -= self.lr * dbc
        self.by -= self.lr * dby

    def train(self, X, y, epochs):
        for epoch in range(epochs):
            total_loss = 0
            for i in range(len(X)):
                x_seq = X[i]
                y_true = y[i]

                # Forward pass
                y_pred, _ = self.forward(x_seq)
                loss = np.mean((y_pred - y_true)**2)
                total_loss += loss

                # Backward pass
                dy = 2 * (y_pred - y_true) / y_pred.shape[0]
                self.backward(dy)

            print(f"Epoch {epoch+1}/{epochs}, Loss: {total_loss/len(X):.4f}")
```