**­­STOCK MARKET PTEDICTION USING MACHINE LEARNING TECHINIQUES**

A Mini Project Report

Submitted in partial fulfilment of the requirements for the award of the degree of

Bachelor of Engineering

in

Information Technology

By

**Kalakuntala Harshitha 245621737097**

**Raayi Shrija 245621737115**

**Kandhakatla Akshaya 245621737313**

Under the guidance of

MS.V.Harika

****

**DEPARTMENT OF INFORMATION AND TECHNOLOGY**

**Gokaraju Lailavathi Women’s Engineering Collage**

**(Affiliated to Osmania University, Approved by AICTE)**

**Bachupally,Hyderabad-500 090.**

**(2023 – 2024)**

**DEPARTMENT OF INFORMATION AND TECHNOLOGY**

**Gokaraju Lailavathi Women’s Engineering Collage**

**(Affiliated to Osmania University, Approved by AICTE)**

**Bachupally, Hyderabad-500 090.**

**(2023 - 2024)**

****

**CERTIFICATE**

This is to Certify that A Mini Project report entitled “Stock Market Prediction Using Machine Learning Techniques” is being submitted by Kalakuntla Harshitha (2456-21-737-097),Raayi Shrija (2456-21-737-115), Kandhakatla Akshaya(2456-21-737-313) in partial fulfilment of the requirement of the award for the degree of Bachelor of Engineering in “Information Technology” O.U., Hyderabad during the year 2023-2024 is a record of bonafide work carried out by them under my guidance. The results presented in this project have been verified and are found to be satisfactory.

**Project Guide HOD**

**Ms.V.Harika Dr.Padmalaya Nayak**

**External Examiner(s)**

****

**DEPARTMENT OF INFORMATION AND TECHNOLOGY**

**Gokaraju Lailavathi Women’s Engineering Collage**

**(Affiliated to Osmania University, Approved by AICTE)**

**Bachupally,Hyderabad-500 090.**

**(2023 - 2024)**

**DECLARATION**

We, Kalakuntla Harshitha bearing Ht.No.2456-21-737-097, Raayi Shrija bearing Ht.No. 2456-21-737-115,Kandhakatla Akshaya bearing Ht.No.2456-21-737-313 hereby certify that the minor project entitled “**Stock Market Prediction Using Machine Learning Techniques**” is submitted in the partial fulfilment of the required for the award of the degree of Bachelor of Engineering in Information Technology . This is a record work carried out by us under the guidance of Ms.V.Harika, Assistant Professor , CSE, Gokaraju Lailavathi Women’s Engineering College, Bachupally. The results embodied in this report have not been reproduced/copied from any source. The results embodied in this report have not been submitted to any other university or institute for the award of any other degree or diploma.

Kalakuntla Harshitha-245621737097

Raayi Shrija-245621737115

Kandhakatla Akshaya-245621737313

**ACKNOWLEDGEMENT**

It is our privilege and pleasure to express my profound sense of respect, gratitude and indebtedness to our guide Ms.V.Harika, (Assistant Professor, Department of Computer Science and Engineering,) Gokaraju Lailavathi women’s Engineering College, for her inspiration, guidance, cogent discussion, constructive criticisms and encouragement throughout this dissertation work.

We express our sincere thanks to Project Coordinator Ms.Ganga Swathi, Assistant Professor, Department of Information Technology, Gokaraju Lailavathi Womens Engineering College, for her valuable suggestions and constant helping in completing the work.

We express our sincere gratitude to Dr.Padmalaya Nayak, Professor & Head of the Department of Information Technology , Gokaraju Lailavathi Womens Engineering College, for her precious suggestions, motivation and co-operation.

We express our sincere thanks to Dr. A. Hanuman Sai, Principal, Gokaraju Lailavathi Womens Engineering College, Bachupally, Hyderabad, for his encouragement and constant help.

We extend our sincere thanks to all the teaching and non-teaching staff of Information Technology Department for their support and encouragement. Last but not least, we wish to acknowledge my friends and family members for giving moral strength moral strength and helping us to complete this dissertation.

**ABSTRACT**

This project presents an end-to-end machine learning application developed in Python for

predicting stock prices, encompassing data preprocessing, model creation, testing, saving, and

Deployment. The data preprocessing phase involves importing historical stock price data,

handling missing values, outliers, and scaling features. For model creation, Tensor Flow library

are utilized to construct a neural network architecture comprising of LSTM layers. The

is compiled with appropriate loss functions and optimizers. Training the model involves fitting

it to the pre-processed data, specifying epochs and batch sizes. Subsequently, the model's

performance is evaluated using testing data, and relevant metrics are calculated. The trained

model is then serialized and saved to disk. Finally, the model is deployed using frameworks

like Stream lit, where it can make predictions on incoming data through an API endpoint. This

project provides a comprehensive guide to building and deploying a machine learning model

for stock price prediction, offering insights into data preprocessing, model creation, evaluation,

and Deployment.

**V**

**CONTENTS**

**Chapter-1 INTRODUCTION……......….…………………………..………….…..1**

1.1MOTIVATION……………………………………………………………………....….…..1

1.2 PROBLEM STATEMENT………………………………………………………..…….….1

1.3 PROJECT OBJECTIVE……………………………… .........…………………...………...2

**Chapter-2 LITERATURE SURVEY…..…....…………………..…………….........3**

2.1 EXISTING SYSTEM………………………………………...……………………..……..3

2.2 LIMITATIONS OF EXISITNG SYSTEM………………………………………………...4

**Chapter-3 SOFTWARE REQUIREMENT SPECIFICATION……....................6**

3.1 SOFTWARE REQUIREMENTS………………………………………………………....6

3.2 HARDWARE REQUIREMENTS…………………………………………………….......6

3.3 FUNCTIONAL REQUIREMENTS ……………………………...................................…6

3.4 NON-FUNCTIONAL REQUIREMENTS……………………......………………………7

**Chapter-4 SYSTEM DESIGN……………………………….............…………….8**

4.1 SYSTEM DESIGN……………………………………………….………………………8

4.2 SYSTEM ARCHITECTURE…………………………….….......……………………….9

4.3 UML DESIGN…………………………………………..……………………………….9

4.3.1 CLASS DIAGRAM……………………………….…….....…….……………………12

**VI**

4.3.2 USECASE DIAGRAM…………………………………….…………........…………13

4.3.3 COMPONENT DIAGRAM…………………………..………………………....…….14

4.3.4 SEQUENCE DIAGRAM……………………………………….………….……….….15

4.3.5 ACTIVITY DIAGRAM………………………………………..……………....….…...16 4.4 TECHNOLOGY DESCRIPTION……….......……………….………………………….17

**Chapter-5 IMPLEMENTATION……………..…………………..…....................18**

5.1 JAVA SCRIPT……………………………....….………….........…….…….…………...18

5.2 MODULES……………………………………………….……………………………..19

5.3 DATASET ………………………………………………….………….………..............20

5.4 EXECUTABLE CODE………………………………..……….………………….…….21

**Chapter-6 TESTING....……………………….…………………….....………..…..29**

6.1 TESTING DEFINATION………………………………………….…....…….………..29

6.2 UNIT TESTING……………………….……………….………………....…………….29

6.3 TEST CASES……………………………………….…………………………..………30

**Chapter-7 RESULTS……………………..................................................................31**

**Chapter-8 CONCLUSION……………………….………….…………...................34**

8.1 CONCLUSION…………………………………………………………..……………34

8.2 FUTURE SCOPE…………………………………………………………………..…..35

**VII**

**REFERENCES………………………………………………………………………35**

|  |  |  |
| --- | --- | --- |
| **SNO.** | **LIST OF FIGURES** | **PAGE NO.** |
| 4.2 | System Architecture | 8 |
| 4.3.1 | UML Hierarchy Diagrams | 10 |
| 4.3.1.1 | Class Diagram | 11 |
| 4.3.2.1 | Use Case Diagram | 12 |
| 4.3.3.1 | Component diagram | 13 |
| 4.3.4.1 | Sequence Diagram | 14 |
| 4.3.5.1 | Activity Diagram | 15 |

**VIII**

**LIST OF TABLES**

|  |  |  |
| --- | --- | --- |
| **Table. No** | **List of Tables** | **Page. No** |
| 2.1 | Literature Survey | 5 |
| 6.3.1 | Login | 25 |
| 6.3.2 | Upload image | 25 |
| 6.3.3 | Delete image | 26 |
| 6.3.4 | Register | 26 |

**IX**

**CHAPTER-1**

**INTRODUCTION**

**1.1 MOTIVATION**

The stock market is a complex system influenced by numerous factors, making it challenging to predict stock prices accurately. The motivation behind this project is to leverage machine learning and deep learning techniques to create a reliable stock price prediction model. By providing users with accurate predictions, they can make informed investment decisions.

**1.2 PROBLEM STATEMENT**

Developing an accurate stock price prediction system requires analyzing vast amounts of historical data and identifying patterns. The problem is to create a user-friendly application that can download stock data, calculate moving averages, and predict future stock prices using a trained deep learning model.

**1.3 PROJECT OBJECTIVE**

The objective of this project is to build a web application using Streamlit that:

* Allows users to select multiple stocks.
* Downloads historical stock data.
* Calculates and visualizes moving averages.
* Predicts future stock prices using a pre-trained deep learning model.
* Compares predicted prices with actual prices to evaluate the model's performance

**1**

**CHAPTER 2 LITERATURE SURVEY**

**2.1 EXISTING SYSTEM**

Stock price prediction has been a subject of interest for many researchers and financial analysts due to its potential for profit and its complexity. Traditional approaches often utilize statistical models such as Moving Averages (MA), Autoregressive Integrated Moving Average (ARIMA), and Generalized Autoregressive Conditional Heteroskedasticity (GARCH). These models rely on historical price data to make predictions.

**Moving Averages (MA):** The MA method smoothens out price data by creating a constantly updated average price. This helps in identifying the trend direction, but it may not be effective in predicting future prices due to lagging indicators.

**Autoregressive Integrated Moving Average (ARIMA):** ARIMA models are used for analyzing and forecasting time series data. They are powerful in terms of capturing different aspects of time series data, such as trends and seasonality. However, they require stationary data and may not handle non-linear patterns effectively.

**Generalized Autoregressive Conditional Heteroskedasticity (GARCH):** GARCH models are designed to estimate the volatility of stock prices, which can then be used to predict future prices. While effective in certain scenarios, GARCH models also struggle with non-linear patterns and require complex parameter tuning.

**Machine Learning Approaches:** More recent systems employ machine learning techniques such as Support Vector Machines (SVM), Random Forests, and Gradient Boosting. These methods have shown

improved performance over traditional statistical methods. However, they still fall short in capturing the complex, non-linear relationships in stock market data.

**2**

**2.2 LIMITATIONS OF EXISTING SYSTEM**

Despite the advancements in statistical and machine learning methods, existing stock price prediction systems exhibit several limitations:

1. **Limited Accuracy:** Traditional methods like MA, ARIMA, and GARCH often fail to capture the non-linear dependencies in stock market data, resulting in less accurate predictions. Machine learning models, while better, still face challenges in accurately predicting prices due to the volatile nature of the market.
2. **Lack of Real-Time Analysis:** Many existing systems are not capable of providing real-time predictions and analysis. This limitation reduces their practical applicability for day traders and other market participants who need up-to-date information.
3. **Complexity and Usability:** Systems utilizing advanced machine learning models can be complex to set up and use. They often require significant expertise in data science and machine learning, making them inaccessible to non-expert users.
4. **Data Handling and Processing:** Efficiently handling and processing large datasets is crucial for accurate stock price prediction. Many existing systems struggle with the volume, velocity, and variety of stock market data, leading to inefficiencies and inaccuracies.
5. **Overfitting:** Machine learning models, especially deep learning models, are prone to overfitting, where they perform well on training data but poorly on unseen data. This is a significant challenge in stock price prediction due to the ever-changing market dynamics.

**Deep Learning Approaches:** In recent years, deep learning models such as Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM) networks have gained popularity due to their ability to

**3**

capture complex temporal dependencies. These models have shown promise in improving prediction

accuracy by learning from large amounts of historical data. However, they require substantial computational resources and are prone to overfitting if not properly regularized.

**Streamlit Applications:** Streamlit has emerged as a popular framework for building web applications, particularly for data science and machine learning projects. Its simplicity and ease of use make it an attractive option for developing interactive stock prediction applications. However, there is limited literature on integrating deep learning models with Streamlit for real-time stock price prediction, indicating a gap that this project aims to address.

**Integrating Deep Learning with Streamlit:** This project leverages the power of deep learning models, particularly LSTM, to improve stock price prediction accuracy. By integrating these models with Streamlit, the project aims to provide a user-friendly, real-time prediction system accessible to both experts and non-experts.

In summary, while there are numerous existing systems for stock price prediction, they often fall short in terms of accuracy, real-time analysis, usability, and handling complex data patterns. This project addresses these limitations by utilizing deep learning techniques and providing an interactive, user-friendly interface through Streamlit.

**4**

cherished moments can be easily located and shared with loved ones. Additionally, the collaborative capabilities of cloud platforms foster meaningful connections by enabling effortless photo sharing and collaboration

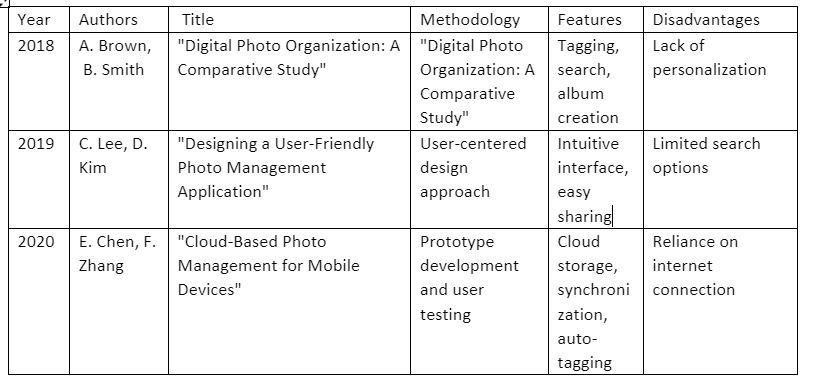


Table- 2.1 Literature Survey

**5**

**CHAPTER-3**

SOFTWARE REQUIREMENT SPECIFICATION

This chapter gives an overview of the software and hardware components required for our project.

# SOFTWARE REQUIREMENTS

Operating System : Windows 8

Coding Language : Python 3.10

# 3.2 HARDWARE REQUIREMENTS

System : intel i5 or above

Storage : Sufficient storage

# 3.3 FUNCTIONAL REQUIREMENTS

These are the requirements that the end user specifically demands as basic facilities that the system should offer. All these functionalities need to be necessarily incorporated into the system as a part of the contract. These are represented or stated in the form of input to be given to the system, the operation performed and the output expected. They are basically the requirements stated by the user which one can see directly in the final product, unlike the non-functional requirements

* **Interactive Widgets**: The application must include interactive widgets for user inputs, such as selecting stock IDs.
* **Real-Time Data Display**: Display the downloaded stock data and calculated moving averages in real-time.
* **Visualization:** Provide visual representations of stock trends, moving averages, and prediction results.
*  **Data Download**: Ability to download historical stock data from Yahoo Finance using yfinance.
*  **Data Preprocessing**: Implement data preprocessing steps, including scaling and preparing data for model input.
*  **Moving Averages Calculation**: Calculate and display moving averages for different time windows (100-day, 200-day, 250-day).
*  **Load Pre-Trained Model**: Load a pre-trained Keras model for predicting stock prices.
*  **Prediction**: Use the model to predict future stock prices based on historical data.
*  **Display Predictions**: Show the original and predicted stock prices in a comparative plot.

# 3.4 NON-FUNCTIONAL REQUIREMENTS

These are basically the quality constraints that the system must satisfy according to the project contract. The priority or extent to which these factors are implemented varies from one project to other. They are also called non-behavioral requirements.

They basically deal with issues like:

* Usability
* Security
* Reliability
* Scalability
* Performance
* Reusability
* Flexibility

**6**

**CHAPTER-4 SYSTEM DESIGN**

**4.1 System Design**

The system design for the stock price prediction application involves creating a structured and modular architecture that ensures efficiency, scalability, and maintainability. This chapter outlines the architectural components, design principles, and detailed diagrams that describe how different parts of the system interact with each.

**4.2 SYSTEM ARCHITECTURE:**

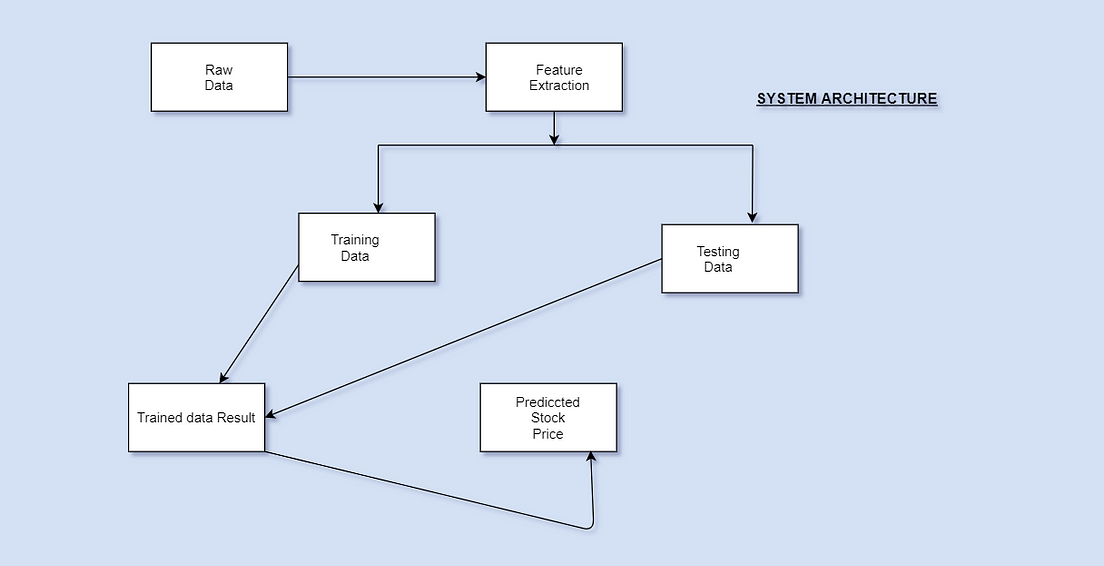


Figure-4.2.1 System Architecture

8

## 4.3 UML Design:

Unified Modeling Language (UML) is a general purpose modeling language. The main aim of UML is to define a standard way to visualize the way a system has been designed. It is quite similar to blueprints used in other fields of engineering.

UML is not a programming language; it is rather a visual language. We use UML diagrams to portray the behaviour and structure of a system, UML helps software engineers, businessmen and system architects with modelling, design and analysis. The Object Management Group (OMG) adopted Unified Modelling Language as a standard in 1997. It’s been managed by OMG ever since. International Organization for Standardization (ISO) published UML as an approved standard in 2005. UML has been revised over the years and is reviewed periodically.

**Do we really need UML?**

* Complex applications need collaboration and planning from multiple teams and hence require a clear and concise way to communicate amongst them.
* Businessmen do not understand code. So UML becomes essential to communicate with non programmer’s essential requirements, functionalities and processes of the system.
* A lot of time is saved down the line when teams are able to visualize processes, user interactions and static structure of the system.
* UML is linked with object oriented design and analysis. UML makes the use of elements and forms associations between them to form diagrams. Diagrams in UML can be broadly classified as

**9**

**The Primary goals in the design of the UML are as follows:**

* Provide users a ready-to-use, expressive visual modelling Language so that they can develop and exchange meaningful models.
* Provide extendibility and specialization mechanisms to extend the core concepts.
* Be independent of particular programming languages and development process.
* Provide a formal basis for understanding the modelling language.
* Encourage the growth of OO tools market.
* Support higher level development concepts such as collaborations, frameworks, patterns and components.
* Integrate best practices.

**Types of UML Diagrams:**

**Structural Diagrams:**

Capture static aspects or structure of a system. Structural Diagrams include: Component Diagrams, Object Diagrams, Class Diagrams and Deployment Diagrams.

**Behaviour Diagrams:**

Capture dynamic aspects or behavior of the system. Behavior diagrams include: Use Case Diagrams, State Diagrams, Activity Diagrams and Interaction Diagrams.

**10**

The image below shows the hierarchy of diagrams according to UML

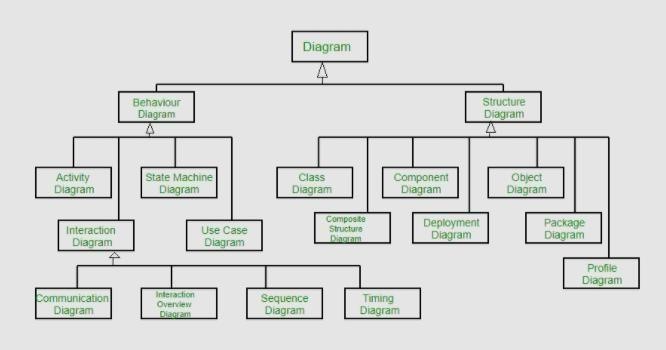


Figure-4.2.1 UML Hierarchy diagrams

**11**

### 4.3.1 CLASS DIAGRAM:

In software engineering, a class diagram in the Unified Modeling Language (UML) is a

type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. It

explains which class contains information.

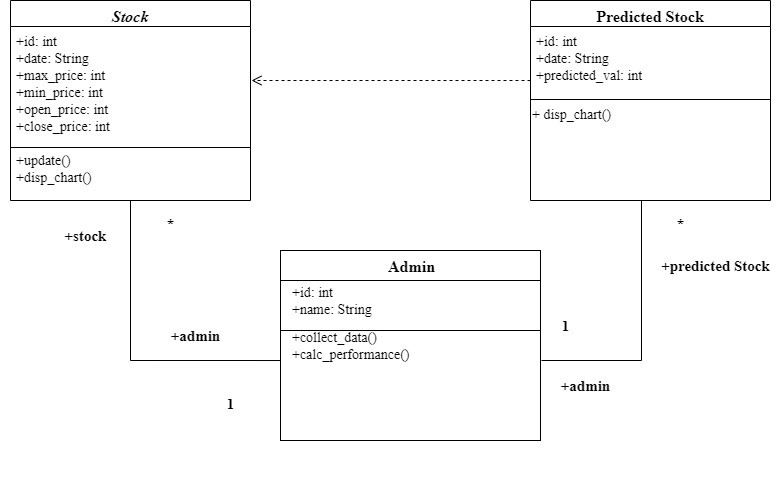


Figure-4.3.1.1 Class Diagram

**4.3.2 USE CASE DIAGRAM:**

A use case diagram in the Unified Modelling Language (UML) is a type of behavioural

diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted

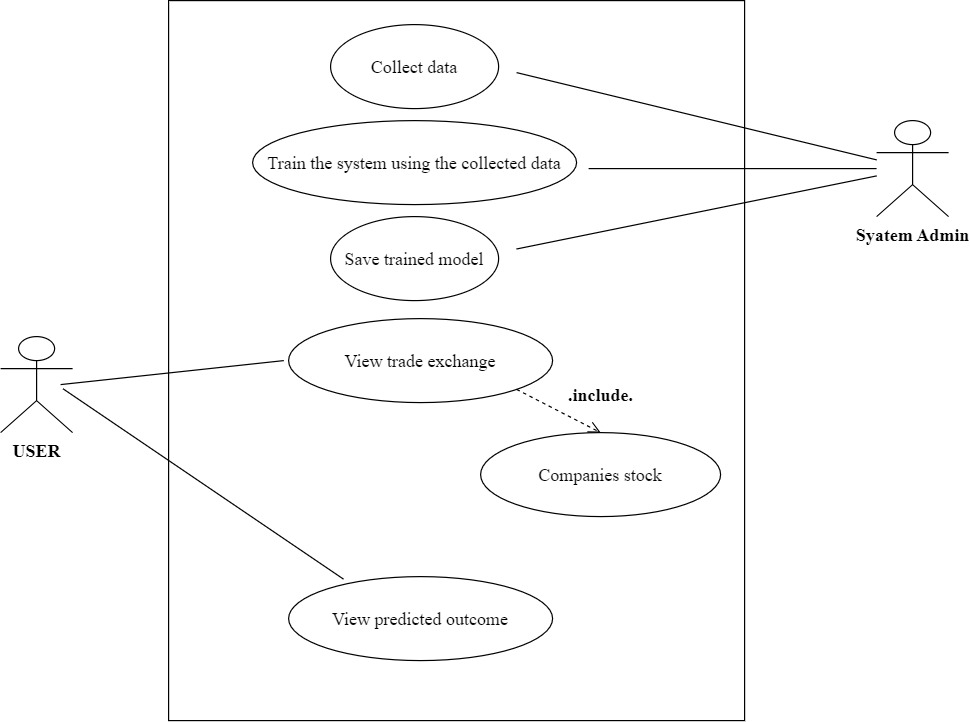


Figure-4.3.2.1 Use Case Diagram

##### 4.3.3 COMPONENT DIAGRAM:

A component diagram, also known as a UML component diagram, describes the

organization and wiring of the physical components in a system. Component diagrams areoften drawn to help model implementation details and double-check that every aspect of the system's required functions is covered by planned development.

.

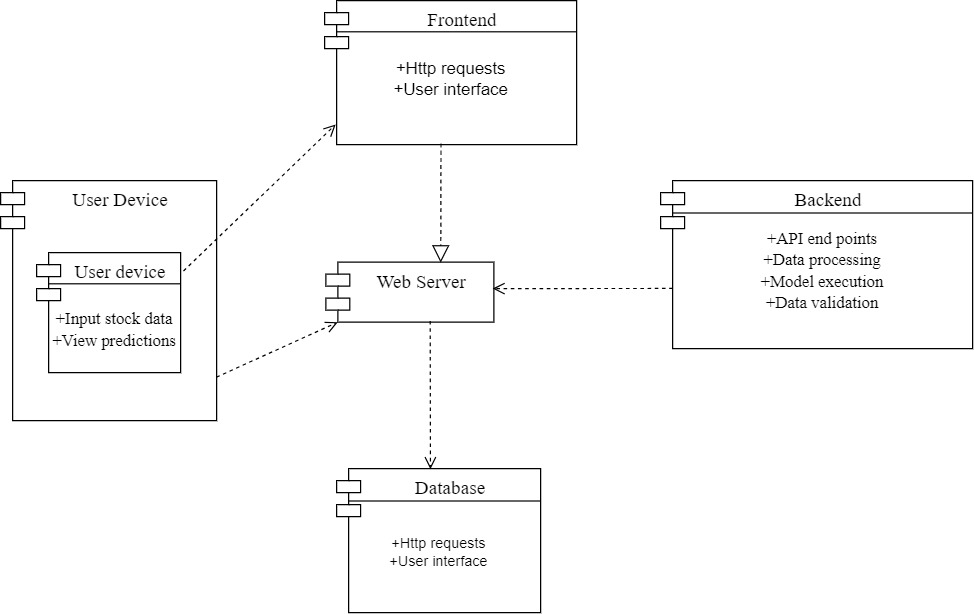


Figure-4.3.3.1 Component Diagram

##### 4.3.4 SEQUENCE DIAGRAM:

A sequence diagram in Unified Modelling Language (UML) is a kind of interaction

diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.

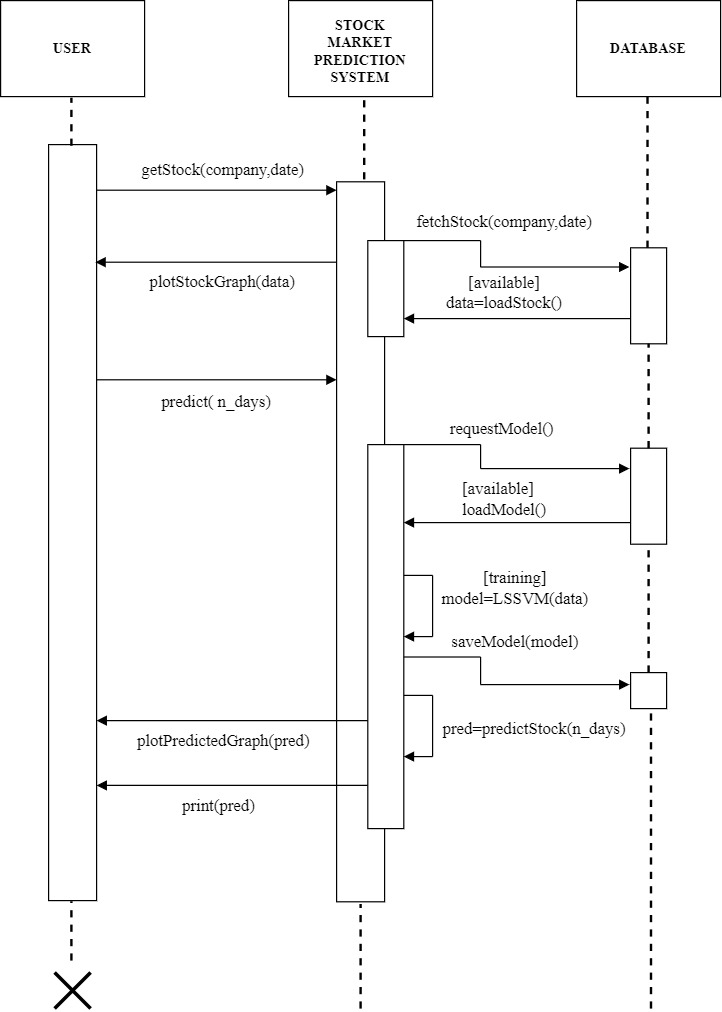


Figure-4.3.4.1 Sequence Diagram

##### 4.3.5 ACTIVITY DIAGRAM:

In UML, an activity diagram is used to display the sequence of activities. Activity

diagrams show the workflow from a start point to the finish point detailing the many decision paths that exist in the progression of events contained in the activity.

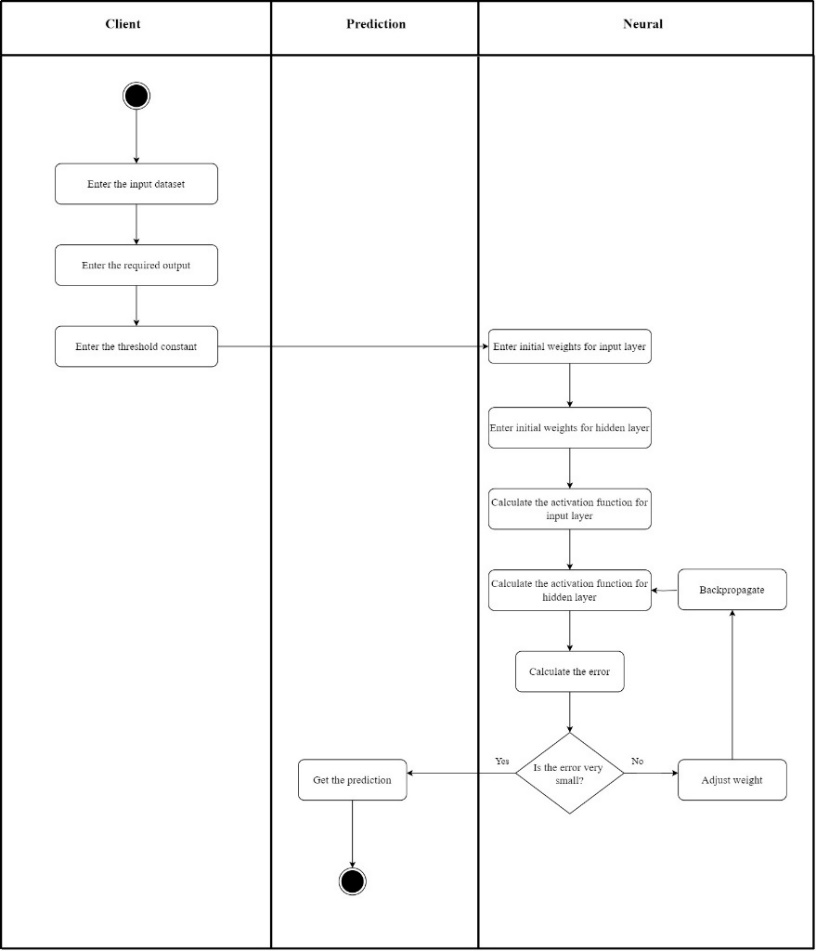


Figure-4.3.5.1 Activity Diagram

#### 4.4 TECHNOLOGY DESCRIPTION

The stock price prediction application leverages a suite of modern technologies to ensure efficient data processing, robust model performance, and an interactive user experience. This section provides an in-depth look at the tools and frameworks utilized in the project.

**Streamlit**

**Streamlit** is an open-source Python library that simplifies the creation of interactive web applications. It allows data scientists and developers to build and deploy powerful data applications with minimal effort.

**Key Features:**

* **Ease of Use**: Streamlit allows rapid prototyping with a simple API, enabling the creation of interactive web apps with just a few lines of code.
* **Widgets**: Streamlit provides a variety of widgets such as sliders, buttons, and multiselect options to create dynamic and responsive user interfaces.
* **Real-time Updates**: Supports real-time updates and displays changes as soon as the code is modified, facilitating an iterative development process.
* **Visualization Integration**: Seamlessly integrates with popular Python visualization libraries such as Matplotlib, allowing for rich, interactive data visualizations.

In this project, Streamlit is used to create the user interface that allows users to select stock IDs, view moving averages, and see stock price predictions.

**Pandas and NumPy**

**Pandas** is a powerful data manipulation library for Python, providing data structures like Data Frames that make data analysis and manipulation straightforward.

**Key Features:**

* **Data Structures**: Provides flexible and efficient data structures like Series and DataFrames.
* **Data Manipulation**: Offers a comprehensive set of tools for data manipulation, including merging, reshaping, selecting, and data cleaning.
* **Integration**: Works seamlessly with other Python libraries such as NumPy, SciPy, and Matplotlib.

**NumPy** is a fundamental package for scientific computing in Python, providing support for arrays, matrices, and a host of mathematical functions.

**Key Features:**

* **Performance**: NumPy arrays are more efficient and perform faster computations than traditional Python lists.
* **Mathematical Functions**: Provides a wide range of mathematical operations and functions for array-based calculations.
* **Integration**: Forms the base for many other scientific libraries in Python, ensuring broad compatibility and integration.

In the stock prediction application, Pandas is used for handling stock data, including downloading, preprocessing, and calculating moving averages. NumPy is used for numerical operations and handling array data for model input.

### Keras with TensorFlow Backend

**Keras** is a high-level neural networks API, written in Python and capable of running on top of TensorFlow. It is designed for easy and fast experimentation with deep learning models.

**Key Features:**

* **User-Friendly**: Keras offers a user-friendly interface, making it accessible for beginners and researchers to build and train neural networks.
* **Modular**: It is highly modular, allowing users to easily add or remove layers, optimizers, and loss functions.
* **Integration with TensorFlow**: Keras is tightly integrated with TensorFlow, leveraging its powerful computational engine for efficient model training and inference.

**TensorFlow**

is an open-source machine learning framework developed by Google. It provides a comprehensive ecosystem for building and deploying machine learning models.

**Key Features:**

* **Flexibility**: TensorFlow supports various machine learning and deep learning algorithms.
* **Scalability**: Can be deployed on various platforms, including CPUs, GPUs, and TPUs, ensuring scalability for large-scale training and inference tasks.
* **Ecosystem**: Offers a rich ecosystem with tools and libraries for data preprocessing, model building, and deployment.

In the project, a pre-trained model is loaded using Keras, and TensorFlow handles the computational aspects of generating stock price predictions.

### Matplotlib

**Matplotlib** is a widely-used Python 2D plotting library that produces publication-quality figures in various formats.

**Key Features:**

* **Versatility**: Supports a wide range of plot types, including line plots, scatter plots, bar charts, histograms, and more.
* **Customization**: Provides extensive options for customizing plots, including titles, labels, colors, and styles.
* **Integration**: Works well with other Python libraries like NumPy, Pandas, and SciPy, facilitating data visualization.

In the stock prediction application, Matplotlib is used to visualize stock trends, moving averages, and prediction results, providing users with a clear and informative view of the data.

### Yfinance

**yfinance** is a Python library that provides easy access to historical market data from Yahoo Finance.

**Key Features:**

* **Ease of Use**: Simplifies the process of downloading historical stock data with a simple API.
* **Comprehensive Data**: Offers a wide range of financial data, including stock prices, dividends, and splits.
* **Integration**: Integrates seamlessly with Pandas, making it easy to manipulate and analyze the downloaded data.

In this project, yfinance is used to download historical stock data for the selected stock IDs, providing the raw data needed for analysis and prediction.

### scikit-learn

**scikit-learn** is a machine learning library in Python that provides simple and efficient tools for data mining and data analysis.

**Key Features:**

* **Preprocessing**: Offers various preprocessing tools, including scaling, normalization, and encoding.
* **Algorithms**: Provides a wide array of machine learning algorithms for classification, regression, clustering, and more.
* **Evaluation**: Includes tools for model evaluation and validation, ensuring robust and reliable machine learning models.

In the stock prediction application, scikit-learn's MinMaxScaler is used to scale the stock price data before feeding it into the neural network model, ensuring that the input data is normalized for better model performance.

# CHAPTER 5

**5.1 IMPLEMENTATION**

The implementation of the stock price prediction application involves several stages, including setting up the environment, data retrieval, data preprocessing, model loading and prediction, and visualization. This chapter outlines the implementation details and steps taken to build the application.

###### 5.2 MODULES:

1. **Data Retrieval Module:**

The data retrieval module is responsible for fetching historical stock price data from Kaggle/GitHub . For each stock selected by the user, the module downloads data spanning the last 20 years. This data includes daily prices, volumes, and other relevant metrics, which are then stored in a dictionary for further processing and analysis. This streamlined approach ensures that comprehensive and accurate historical data is readily available for calculating moving averages and making stock price predictions.

1. **Data Preprocessing Module:**

The data preprocessing module prepares the downloaded stock data for analysis and prediction. It calculates moving averages for different periods (100, 200, and 250 days) and scales the data using MinMaxScaler to ensure that all features are normalized. This step is crucial for enhancing the performance of the neural network model, providing it with well-processed input data for accurate predictions.

1. **Model Handling Module:**

The model handling module loads a pre-trained neural network model using Keras and compiles it with the Adam optimizer. This allows the application to generate accurate stock predictions based on the processed input data.

**20**

1. **Visualization Module:**

The visualization module utilizes Matplotlib to create detailed charts and graphs that display stock trends, moving averages, and prediction results. It plots the moving averages alongside the actual closing prices, providing clear insights into stock performance. The module also compares the predicted stock prices with the original data, offering users an intuitive way to assess the accuracy of the model. These visualizations are seamlessly integrated into the Streamlit interface for an interactive user experience.

###### 5.4 EXECUTABLE CODE:

import streamlit as st

import pandas as pd

import numpy as np

from keras.models import load\_model

from keras.optimizers import Adam

import matplotlib.pyplot as plt

import yfinance as yf

from datetime import datetime

from sklearn.preprocessing import MinMaxScaler

# Set page configuration

st.set\_page\_config(layout="wide")

# Custom CSS for red background

st.markdown(

"""

<style>

.stApp {

background-color: red;

}

</style>

""",

unsafe\_allow\_html=True

)

st.title("📈 SNIST App for Stock Price Predictor ")

# Multiselect widget for choosing multiple stocks

stocks = st.multiselect("Select Stock IDs", ["GOOG", "AAPL", "MSFT", "AMZN"], default=["GOOG"])

end = datetime.now()

start = datetime(end.year - 20, end.month, end.day)

# Dictionary to store stock data

stock\_data = {}

# Download stock data for each selected stock

for stock in stocks:

stock\_data[stock] = yf.download(stock, start, end)

# Calculate moving averages for each selected stock

for stock in stocks:

stock\_data[stock]['MA\_for\_250\_days'] = stock\_data[stock]['Close'].rolling(window=250).mean()

stock\_data[stock]['MA\_for\_200\_days'] = stock\_data[stock]['Close'].rolling(window=200).mean()

stock\_data[stock]['MA\_for\_100\_days'] = stock\_data[stock]['Close'].rolling(window=100).mean()

# Load the model without the optimizer

model = load\_model(r"E:/Mr JP/2025/mini/stock market/stock\_price\_prediction-main/stock\_price\_prediction-main/Latest\_stock\_price\_model.h5", compile=False)

# Compile the model with a compatible optimizer

model.compile(optimizer=Adam(), loss='mean\_squared\_error')

# Display data and charts for each selected stock

for stock in stocks:

st.subheader(f"{stock} Stock Data Overview")

st.dataframe(stock\_data[stock].describe())

# Visualization function

def plot\_graph(figsize, values, full\_data, extra\_data=0, extra\_dataset=None, title=""):

fig = plt.figure(figsize=figsize)

plt.plot(values, 'orange', label='Moving Average')

plt.plot(full\_data.Close, 'b', label='Close Price')

if extra\_data:

plt.plot(extra\_dataset, 'g', label='Additional Moving Average')

plt.title(title)

plt.legend()

return fig

st.subheader(f"{stock} Moving Averages")

col1, col2 = st.columns(2)

with col1:

st.pyplot(plot\_graph((10, 6), stock\_data[stock]['MA\_for\_250\_days'], stock\_data[stock], 0, title='MA for 250 days'))

st.pyplot(plot\_graph((10, 6), stock\_data[stock]['MA\_for\_200\_days'], stock\_data[stock], 0, title='MA for 200 days'))

with col2:

st.pyplot(plot\_graph((10, 6), stock\_data[stock]['MA\_for\_100\_days'], stock\_data[stock], 0, title='MA for 100 days'))

st.pyplot(plot\_graph((10, 6), stock\_data[stock]['MA\_for\_100\_days'], stock\_data[stock], 1, stock\_data[stock]['MA\_for\_250\_days'], title='MA for 100 and 250 days'))

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_data = scaler.fit\_transform(stock\_data[stock][['Close']])

x\_data = []

y\_data = []

for i in range(100, len(scaled\_data)):

x\_data.append(scaled\_data[i-100:i])

y\_data.append(scaled\_data[i])

x\_data, y\_data = np.array(x\_data), np.array(y\_data)

predictions = model.predict(x\_data)

inv\_pre = scaler.inverse\_transform(predictions)

inv\_y\_test = scaler.inverse\_transform(y\_data)

ploting\_data = pd.DataFrame(

{

'original\_test\_data': inv\_y\_test.reshape(-1),

'predictions': inv\_pre.reshape(-1)

},

index=stock\_data[stock].index[len(stock\_data[stock]) - len(inv\_pre):]

)

st.subheader(f"{stock} Original values vs Predicted values")

st.line\_chart(ploting\_data)

st.subheader(f"{stock} Close Price vs Predicted Close Price")

fig = plt.figure(figsize=(15, 6))

plt.plot(stock\_data[stock].Close[:len(stock\_data[stock]) - len(ploting\_data)], 'b', label='Data - not used')

plt.plot(ploting\_data.index, ploting\_data['original\_test\_data'], 'orange', label='Original Test Data')

plt.plot(ploting\_data.index, ploting\_data['predictions'], 'green', label='Predicted Test Data')

plt.legend()

plt.title(f"{stock} Original Close Price vs Predicted Close Price")

st.pyplot(fig)

# CHAPTER - 6

# TESTING

**6.1 TESTING DEFINATION:**

Testing is a crucial phase in the development of the stock price prediction application to ensure the reliability, accuracy, and performance of the system. This chapter details the testing strategies employed, types of tests conducted, and the results obtained during the testing phase.

Testing involves evaluating the application to identify any discrepancies, bugs, or areas for improvement. The primary objectives are to verify that the application meets the specified requirements, functions correctly under various conditions, and provides accurate stock price predictions.

###### 6.2 Unit Testing

Unit testing focuses on verifying the correctness of individual components of the application. Each module, function, and method is tested independently to ensure that it performs as expected.

**Integration Testing**

Integration testing focuses on verifying that different modules of the application work together seamlessly. It tests the interaction between components such as data retrieval, preprocessing, model handling, and visualization.

**Key Aspects Of Integration Testing:**

Data Flow

Interface Capability

End to End Scenarios

**Test Results:** All the test cases mentioned above passed successfully. No defects encountered.

**26**

**Acceptance Testing**

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

**Test Results:** All the test cases mentioned above passed successfully. No defects encountered.

Since, the grey box testing includes access to internal coding for designing test cases. Grey box testing is performed by a person who knows coding as well as testing.

**Outcomes Possible:**

**Pass:** The test case successfully validates the expected behavior of the application, indicating that specific functionality works as intended.

**Fail:** The test case fails to validate the expected behavior, indicating a defect or issue in the application. This outcome requires further investigation and fixing the identified problem.

**Error:** An error occurs during the test execution due to unexpected system behavior or exceptions. This could indicate a bug or potential issue that needs to be addressed.

**Blocked:** The test case cannot be executed due to external dependencies or environmental constraints. This outcome indicates that the test case is blocked and cannot be validated at the moment.

27

**Skipped:** The test case is intentionally skipped from execution, typically due to low priority or specific conditions that prevent it from being applicable at the current stage of testing.

**Test case**

**User requirements:**

**Login:**

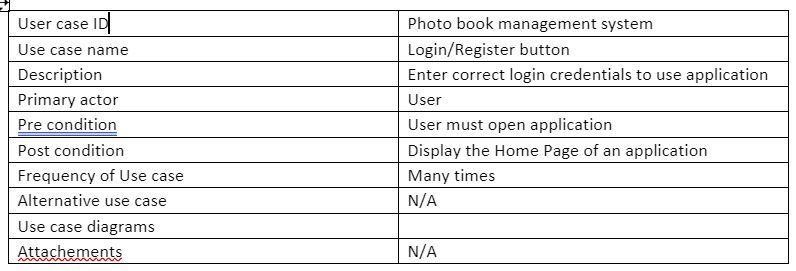
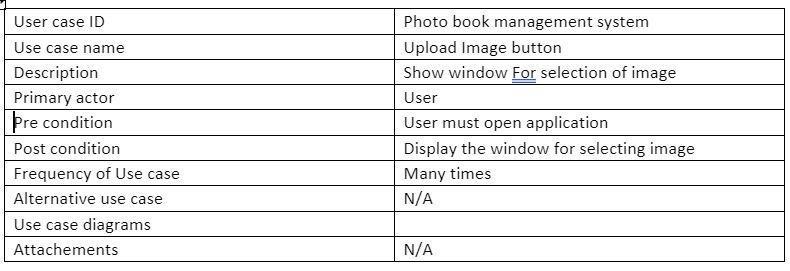


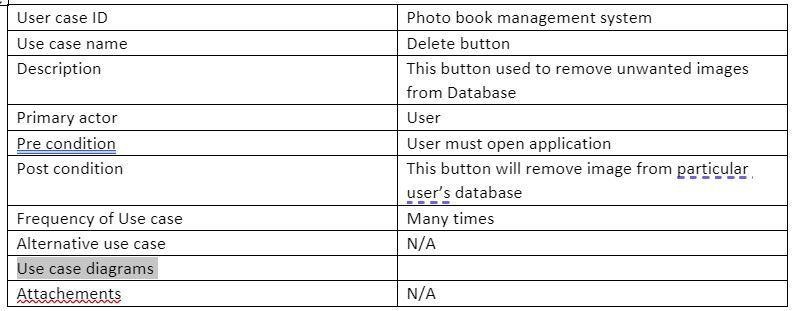
Table-6.3.1

**Upload Image :**



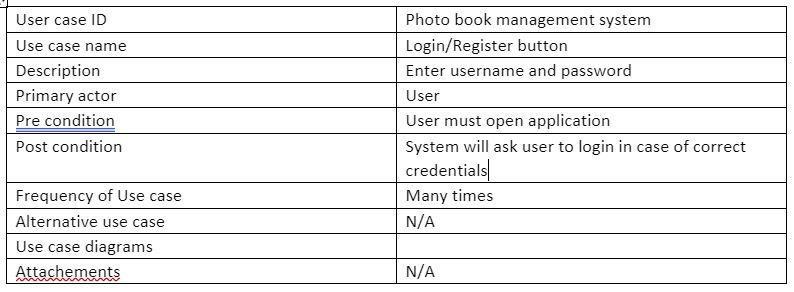
**Table-6.3. Table-6.3.2**

**Delete image :**



**Table-6.3.3**

**Register :**



**Table-6.3.4**

# CHAPTER - 7 RESULTS

* After running the program we can see the following pages

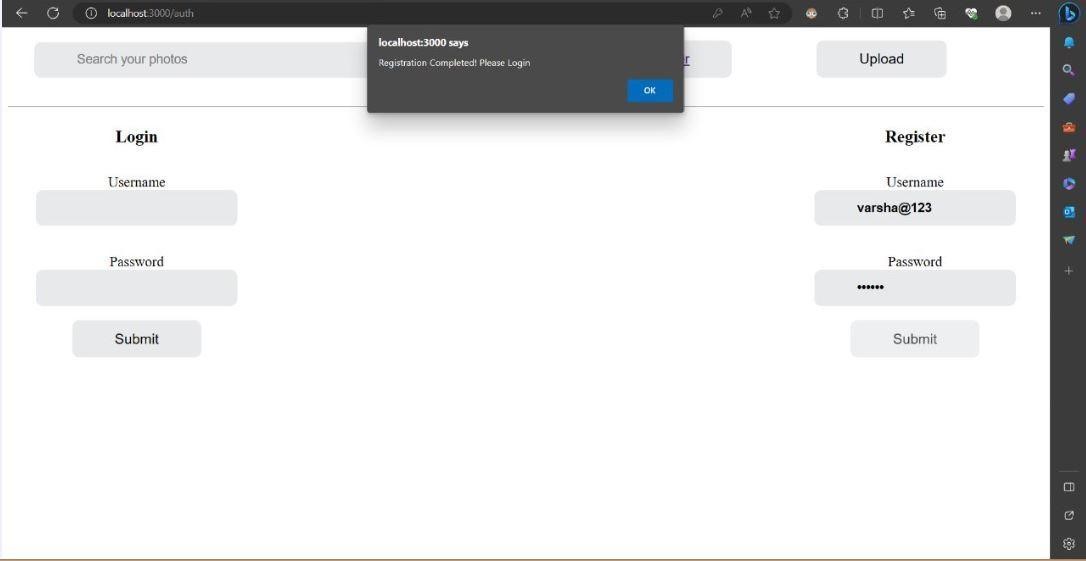


Figure-7.1 Registration and login page

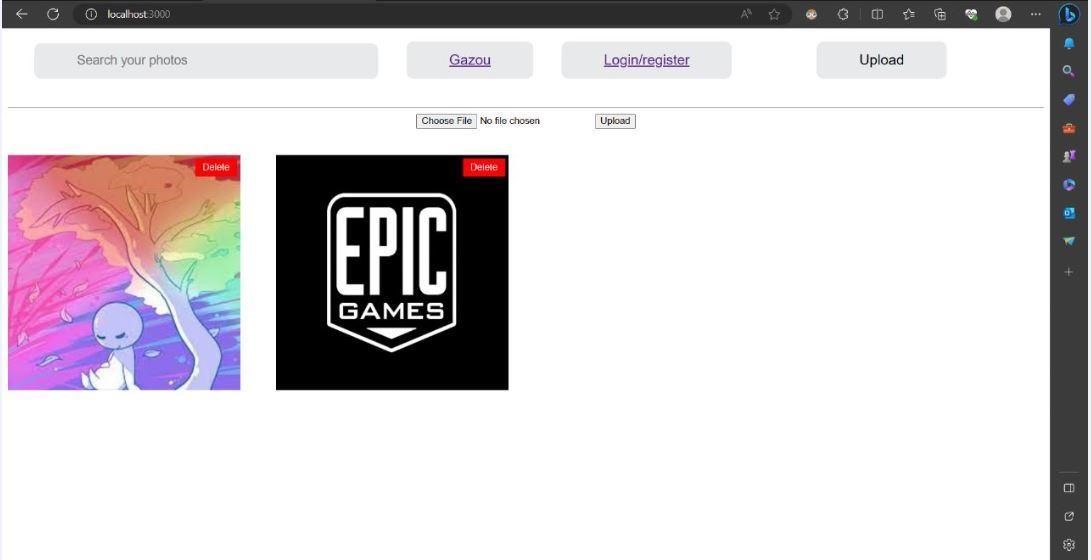


Figure-7.2 Uploading of multiple photos

**CHAPTER 8**

* 1. **CONCLUSION**

In conclusion, we have designed a system, "PhotoPal" is a user-friendly platform for organizing, storing, and sharing photos. Leveraging the MERN stack, it ensures scalability and performance. Robust testing guarantees reliability, and privacy settings secure user data. With intuitive photo editing and seamless sharing features, PhotoPal enhances the photo management experience. Through continuous improvement and user feedback, it aims to be an indispensable tool for photo enthusiasts, delivering a delightful user experience

* 1. **FUTURE SCOPE**
* The future study will be done by applying different filters.
* Incorporating artificial intelligence (AI) and machine learning algorithms can enable features like automatic photo tagging, object recognition, and intelligent photo organization

**REFERENCES**

* "The Best Photo Management Apps" by Michael Muchmore (published on PCMag)
* "Photo Organizing Made Easy: Going from Overwhelmed to Overjoyed" by Cathi Nelson
* "The Best Photo Storage and Sharing Sites" by Eric Griffith (published on PCMag)
* "The DAM Book: Digital Asset Management for Photographers" by Peter Krogh
* "The Best Photo Organizing Software" by David Beren (published on Lifewire)