ABSTRACT

The **Stock Portfolio Tracking Tool** is a web-based application designed to help individual investors manage and monitor their stock investments in real time. The tool enables users to **add, remove, and track the performance** of their stock holdings through an intuitive and interactive interface. It leverages **real-time financial data** from external APIs such as Yahoo Finance to fetch up-to-date stock prices and calculate key metrics including **current value, profit/loss**, and **percentage change.**

Built with **Python (Flask)** on the backend and a lightweight **HTML/JavaScript** frontend, the system offers persistent data storage using **SQLite**, allowing users to maintain a personalized and continually updated portfolio. The application emphasizes simplicity, usability, and performance insight, making it a practical tool for both novice and experienced investors seeking to better understand their portfolio dynamics.

By integrating financial APIs and providing actionable insights, the Stock Portfolio Tracker bridges the gap between manual investment tracking and fully automated financial platforms, offering users a flexible and transparent way to evaluate their investment decisions.

1. INTRODUCTION

In today’s dynamic financial markets, individual investors are increasingly taking charge of their own investment decisions. As portfolios grow in complexity, tracking and evaluating the performance of various stock holdings becomes a critical task. Traditional methods, such as manual spreadsheets, can be error-prone and time-consuming, while full-featured brokerage platforms may overwhelm users with excess functionality or obscure their financial insights behind complex interfaces.

The **Stock Portfolio Tracker** is developed to provide a **simple, transparent, and customizable** solution for monitoring stock investments. This tool allows users to effortlessly **add and remove stocks,** automatically fetch **real-time price data,** and analyze **key performance metrics** such as profit/loss and current portfolio value. By integrating with public financial APIs, the application delivers accurate market information without relying on outdated or static data sources.

Designed using **Python and Flask**, the backend manages user data and handles communication with the stock data API. A clean **web interface** built with HTML/CSS enables intuitive interaction, while a lightweight **SQLite database** ensures data persistence without requiring complex setup.

This project aims to empower users with better financial visibility and decision-making capabilities through a streamlined and user-friendly portfolio tracking experience.

**1.1.SYSTEM REQUIREMENTS**

**1.1.1 Software requirements**

**Operating System**:

Windows, macOS, or Linux (Ubuntu recommended)

**Python**:

Python 3.6+ (Ensure Python is installed on the system. The application is developed and tested with Python 3.7 and later versions.)

**Web Framework**:

Flask (Python Web Framework, lightweight and easy to use for backend development)

**Financial Data API**:

yfinance (for fetching real-time stock data via Yahoo Finance)

**Database**:

SQLite for local database storage; no setup required for SQLite as it’s integrated with Python, but can also be switched to PostgreSQL or MySQL if needed)

**Package Management**:

**pip** (for installing Python libraries)

**Virtual Environment** (recommended for isolating dependencies)

**Frontend Requirements** (Optional):

**Web Browser** (for accessing the tool through the browser; Chrome, Firefox, Safari, etc.)

**1.1.2 Hardware Requirements**

* **Processor**: Any modern processor (Intel i3 or higher)
* **RAM**: 4GB or more (8GB recommended for optimal performance)
* **Storage**: 50MB+ free disk space (for database and libraries)
* **Internet Connection**: Required for fetching live stock data from the financial API

## ****2.SYSTEM STUDY****

### 2.1 ****Existing System****

The **existing systems** for tracking stock portfolios typically fall into two categories:

* **Manual Tracking Systems (Spreadsheets)**:
  + Many individual investors still rely on spreadsheets (e.g., Microsoft Excel, Google Sheets) to manually track their investments.
  + Users input stock symbols, purchase prices, and shares, while calculating portfolio performance manually or using pre-built formulas.
* **Online Brokerage Platforms**:
  + Popular platforms like Robinhood, E\*TRADE, and Fidelity offer built-in portfolio tracking tools.
  + These platforms display real-time stock prices, performance metrics, and account summaries.
  + However, they often have **restricted features** that focus only on the investments within that specific platform (i.e., they do not allow tracking investments across multiple platforms or external investments).
* **Standalone Portfolio Management Software**:
  + Software like **Morningstar, Personal Capital,** or **Stockpile** provides a more advanced level of tracking and analysis, but they often come with high subscription fees or offer limited free functionality.
  + These tools can aggregate data from multiple accounts but may require manual data entry and have limited customization.

### 2.2 ****Drawbacks****

While existing systems serve their purpose, they come with several **drawbacks:**

1. **Manual Tracking (Spreadsheets)**:
   * **Time-consuming and error-prone**: Users must manually enter and update stock prices, shares, and calculations. This leads to potential mistakes and inefficiencies.
   * **Limited Automation**: Real-time price fetching and portfolio performance calculations require constant manual updates.
   * **Limited Insight**: Spreadsheets lack in-depth analytics and performance metrics, such as real-time profit/loss, percentage change, and current portfolio valuation.
   * **Scalability Issues**: As the portfolio grows, the spreadsheet becomes difficult to manage and cumbersome to update.
2. **Online Brokerage Platforms**:
   * **Limited Data Access**: Most brokerage platforms show only data for investments held within that specific platform and do not integrate with external investment accounts or portfolios.
   * **Lack of Flexibility**: These platforms often have rigid interfaces with limited customization options for portfolio tracking.
   * **Hidden Fees**: Some platforms charge fees for advanced tracking features, limiting the full potential of the tool.
3. **Standalone Portfolio Management Software**:
   * **Costly**: High-quality portfolio management services typically require subscriptions, which might be expensive for casual or smaller investors.
   * **Overcomplicated Features**: Many tools come with more advanced features than what an average investor needs, leading to complexity and inefficiency.
   * **Limited Customization**: While some offer external investment tracking, they may lack customizable views or insights that an individual user might need.

### 2.3 ****Proposed System****

1. **Real-Time Data Fetching**:
   * The tool integrates with a financial API to fetch live stock prices, removing the need for users to manually update prices.
2. **Automation of Key Metrics**:
   * The system automatically calculates portfolio metrics such as **current value, profit/loss, percentage change,** and **overall performance.**
3. **Add/Remove Stocks Easily**:
   * Users can **add and remove stocks** from their portfolio with a simple user interface, enabling flexible management of their investments.
4. **Customizable Portfolio Views**:
   * The system allows for easy categorization and filtering of investments, making it easy for users to focus on specific stocks, sectors, or types of investments.
5. **Data Persistence**:
   * Portfolio data is stored locally in a **SQLite database**, ensuring that user inputs are saved between sessions, without the need to manually re-enter data.
6. **User-Friendly Web Interface**:
   * Built using **Flask** (Python web framework), the system provides a clean and simple interface accessible through any web browser. Users can quickly visualize their portfolio’s performance.
7. **Lightweight and Cost-Effective**:
   * Unlike other portfolio management tools, the system is **free** and **lightweight**, providing all the necessary features without the complexities or costs of other solutions.

### 2.4 ****Features****

The **Stock Portfolio Tracker** offers the following features:

1. **Add Stocks**:
   * Users can add stock entries by providing the stock symbol, number of shares, and buy price. This information is saved in the database.
2. **Remove Stocks**:
   * Users can remove stocks from the portfolio with a simple click, allowing for easy management and updates.
3. **Real-Time Stock Data**:
   * The system fetches the latest stock price using the **Yahoo Finance API (yfinance)**. This allows users to track the market value of each stock in real time.
4. **Performance Metrics**:
   * Key performance metrics, including **current value**, **profit/loss**, and **percentage change**, are calculated automatically and displayed.
5. **Portfolio Overview**:
   * The tool provides an **overview of the entire portfolio**, showing the total current value, total profit/loss, and portfolio allocation by stock.
6. **Search Functionality**:
   * Users can search for specific stocks by symbol, making it easy to find and view individual stock details.
7. **Responsive Web Interface**:
   * The interface is designed to be intuitive and responsive, adapting to different screen sizes for a seamless experience on both desktop and mobile devices.
8. **Data Persistence**:
   * Portfolio data is saved between sessions, ensuring that the user’s stock information is always up to date when they return.
9. **Customizable Metrics**:
   * Users can adjust the view to display specific data points, such as profit/loss, total investment, or portfolio percentage distribution.

## ****3.SYSTEM DESIGN AND DEVELOPMENT****

### 3.1 ****File Design****

* + **app.py** – The main Python script containing the Flask app logic for handling routes and interactions with the database.
  + **templates/** – Directory containing HTML files for rendering the user interface.
  + **index.html** – Displays the portfolio and allows users to add/remove stocks.
  + **static/** – Directory for static files like CSS, JavaScript, and images.
  + **style.css** – Custom CSS for styling the web interface.
  + **portfolio.db** – SQLite database file where user portfolio data is stored.
  + **requirements.txt** – List of required Python libraries (Flask, yfinance, etc.), which can be installed via pip install -r requirements.txt.

### 3.2 ****Input Design****

1. **Add Stock**:
   * **Symbol**: Stock symbol (e.g., AAPL, TSLA)
   * **Shares**: Number of shares to add to the portfolio
   * **Buy Price**: The price at which the stock was purchased

These inputs are provided through an HTML form in the index.html file and submitted to the server for processing.

**Form fields**:

html

CopyEdit

<form action="/add" method="post">

<input type="text" name="symbol" placeholder="Stock Symbol" required>

<input type="number" name="shares" placeholder="Shares" required>

<input type="number" name="buy\_price" placeholder="Buy Price" required step="0.01">

<button type="submit">Add Stock</button>

</form>

1. **Remove Stock**:
   * **Stock Symbol**: The stock symbol is used to identify which stock to remove from the portfolio. This is a clickable action tied to each stock row in the portfolio list.

**HTML for removal**:

html

CopyEdit

<a href="/remove/{{ stock['symbol'] }}">Remove</a>

**Validation**

* **Symbol**: Should be a valid stock symbol (e.g., AAPL, TSLA).
* **Shares**: Must be a positive integer.
* **Buy Price**: Should be a positive float number.

### 3.3 ****Output Design****

1. **Portfolio Overview**:
   * Displays the entire portfolio with stock symbols, the number of shares owned, buy price, current price, total value, profit/loss, and percentage change.
2. profi**t/Loss**:
   * The system calculates and displays the overall profit or loss based on the difference between the total value of the portfolio and the total cost of investment.

### 3.4 ****CODE****

python

CopyEdit

from flask import Flask, render\_template, request, redirect

import sqlite3

import yfinance as yf

app = Flask(\_\_name\_\_)

DB = 'portfolio.db'

# Initialize the database and create the portfolio table if it doesn't exist

def init\_db():

with sqlite3.connect(DB) as conn:

conn.execute('''CREATE TABLE IF NOT EXISTS portfolio (

id INTEGER PRIMARY KEY,

symbol TEXT NOT NULL,

shares INTEGER,

buy\_price REAL

)''')

# Get the current stock price from Yahoo Finance

def get\_current\_price(symbol):

stock = yf.Ticker(symbol)

return stock.history(period="1d")["Close"].iloc[-1]

@app.route('/')

def index():

with sqlite3.connect(DB) as conn:

cur = conn.cursor()

cur.execute("SELECT \* FROM portfolio")

rows = cur.fetchall()

portfolio = []

total\_value = 0

for row in rows:

symbol, shares, buy\_price = row[1], row[2], row[3]

current\_price = get\_current\_price(symbol)

current\_value = current\_price \* shares

profit\_loss = current\_value - (buy\_price \* shares)

percent\_gain = (profit\_loss / (buy\_price \* shares)) \* 100

portfolio.append({

'symbol': symbol,

'shares': shares,

'buy\_price': buy\_price,

'current\_price': current\_price,

'value': current\_value,

'profit\_loss': profit\_loss,

'percent\_gain': percent\_gain

})

total\_value += current\_value

return render\_template('index.html', portfolio=portfolio, total\_value=total\_value)

@app.route('/add', methods=['POST'])

def add\_stock():

symbol = request.form['symbol'].upper()

shares = int(request.form['shares'])

buy\_price = float(request.form['buy\_price'])

with sqlite3.connect(DB) as conn:

conn.execute("INSERT INTO portfolio (symbol, shares, buy\_price) VALUES (?, ?, ?)",

(symbol, shares, buy\_price))

return redirect('/')

@app.route('/remove/<symbol>')

def remove\_stock(symbol):

with sqlite3.connect(DB) as conn:

conn.execute("DELETE FROM portfolio WHERE symbol = ?", (symbol,))

return redirect('/')

if \_\_name\_\_ == '\_\_main\_\_':

init\_db()

app.run(debug=True)

### 3.5 ****Database Design****

.Table: portfolio

| **Column Name** | **Data Type** | **Description** |
| --- | --- | --- |
| Id | INTEGER | Unique identifier (Primary Key) |
| symbol | TEXT | Stock symbol (e.g., AAPL, TSLA) |
| shares | INTEGER | Number of shares owned |
| buy\_price | REAL | Price at which the stock was bought |

SQL Schema:

sql

CopyEdit

CREATE TABLE IF NOT EXISTS portfolio (

id INTEGER PRIMARY KEY,

symbol TEXT NOT NULL,

shares INTEGER,

buy\_price REAL

);

### 3.6 ****System Development****

1. **Environment Setup**:
   * Set up the Python environment, install required libraries (Flask, yfinance, SQLite).
   * Create a virtual environment to isolate dependencies.
   * Initialize a Git repository to track code changes.
2. **Backend Development (Flask)**:
   * Set up Flask routes for managing the portfolio (/, /add, /remove).
   * Implement database interaction for storing and retrieving stock data.
   * Integrate the Yahoo Finance API (via yfinance) to fetch real-time stock prices.
3. **Frontend Development (HTML/CSS)**:
   * Design the user interface with HTML to display the portfolio, stock details, and add/remove forms.
   * Apply styling using CSS to make the interface user-friendly and responsive.
4. **Testing**:
   * Perform unit tests for individual functions (e.g., adding stocks, removing stocks, fetching data).
   * Test the application for usability and bugs in various browsers.
5. **Deployment**:
   * Deploy the application to a local environment for testing.
   * Consider deploying to a platform like Heroku or DigitalOcean for broader access.

**4.TESTING AND IMPLEMENTATION**

### 4.1 ****Testing****

Testing is a crucial part of the software development lifecycle to ensure the system functions as expected and handles errors gracefully. Below is an outline of the **testing strategies** and types of tests performed on the Stock Portfolio Tracker.

#### 4.1.1 ****Unit Testing****

Unit tests focus on testing individual components or functions of the system to ensure they perform as expected.

* **Test Functions**:
  + **get\_current\_price(symbol)**: Verifies that the function returns a valid stock price from the Yahoo Finance API.
  + **add\_stock(symbol, shares, buy\_price)**: Tests if the stock is properly added to the database.
  + **remove\_stock(symbol)**: Ensures that the stock is correctly removed from the database.

#### 4.1.2 ****Functional Testing****

Functional tests check if the system works as expected from an end-user perspective. These tests cover scenarios such as:

* **Adding Stocks**:
  + Test adding a new stock by submitting the form with stock symbol, shares, and buy price. Verify that the stock appears in the portfolio with the correct data.
* **Removing Stocks**:
  + Test the functionality to remove a stock by clicking the "Remove" link next to a stock. Ensure that the stock is no longer present in the portfolio after the operation.
* **Real-Time Price Fetching**:
  + Simulate the fetching of real-time prices from Yahoo Finance to ensure the system updates the portfolio with the correct data.
* **Profit/Loss Calculation**:
  + Test whether the system correctly calculates profit/loss and percentage gain/loss based on the real-time price and the number of shares.

#### 4.1.3 ****Integration Testing****

Integration tests focus on verifying that the interaction between various components (backend, database, financial APIs) works seamlessly.

* **Database Integration**:
  + Test that when a stock is added, it is stored in the database. Similarly, ensure that removing a stock updates the database correctly.
* **API Integration**:
  + Test the connection between the system and the **Yahoo Finance API** (or any other financial data source) to fetch real-time stock prices.

#### 4.1.4 ****User Acceptance Testing (UAT)****

User Acceptance Testing is performed by actual users to ensure the system meets their requirements and expectations.

* **Test Case 1**: Add a stock to the portfolio with valid data. Verify that the stock shows up in the portfolio with the correct price, number of shares, and total value.
* **Test Case 2**: Remove a stock and ensure that it no longer appears in the portfolio list.
* **Test Case 3**: Simulate a drop in stock prices and check if the system correctly updates the profit/loss and portfolio value.

#### 4.1.5 ****Error Handling Testing****

Testing error handling ensures that the system behaves gracefully under unexpected conditions:

* **Invalid Stock Symbol**:
  + Test what happens when an invalid stock symbol (e.g., "INVALID123") is entered. The system should show an appropriate error message or fallback behavior.
* **Database Error**:
  + Simulate a database error (e.g., database file is not accessible) to ensure that the system handles it and provides meaningful feedback to the user.
* **Empty Portfolio**:
  + Test the system’s behavior when there are no stocks in the portfolio. Ensure it displays a message like "Your portfolio is empty" in the portfolio table.

#### 4.2 IMPLEMENTATION

#### 4..2.1 ****Setup the Development Environment****

1. **Install Python**:

* Ensure **Python 3.6+** is installed on the system.
* Install **Flask**, **yfinance**, and **SQLite** (or any other database of choice) via pip:

1. **Create a Virtual Environment** (recommended):

bash

CopyEdit

python -m venv venv

source venv/bin/activate # On Windows: venv\Scripts\activate

1. **Clone the Project Repository** (if applicable):

bash

CopyEdit

git clone https://github.com/your-username/stock-portfolio-tracker.git

cd stock-portfolio-tracker

1. **Database Initialization**:
   * On first run, the **SQLite database** is initialized by the init\_db() function in app.py that creates a table for storing stock data.

#### 4.2.2 ****Development Stages****

**Backend Implementation**:

The core Flask application is developed in app.py to handle routes such as:

Displays the portfolio overview.

**/add**: Handles the addition of new stocks to the portfolio.

**/remove/<symbol>**: Handles the removal of stocks from the portfolio.

**Frontend Implementation**:

**HTML templates** (index.html) are created to display the portfolio data, stock details, and forms to add/remove stocks.

Basic styling is done using **CSS** in style.css.

**Financial API Integration**:

The **Yahoo Finance API** (via yfinance) is integrated to fetch real-time stock data based on stock symbols.

The API returns stock prices which are then used to calculate portfolio metrics like **current value, profit/loss,**  and **percentage change.**

**Testing**:

**Unit tests** and **functional tests** are written to ensure the individual components work correctly.

**Error handling** is implemented to deal with potential issues such as invalid symbols or API failures.

**Debugging and Refining**:

Debugging tools like **Flask's debug mode** and browser developer tools are used to identify and fix issues.

User feedback and testing results are used to refine the functionality, improve the user interface, and fix bugs.

**5.CONCLUSION**

The development of the Stock Portfolio Tracker using Python demonstrates the powerful capabilities of the language in financial data analysis and visualization. By integrating libraries such as pandas for data manipulation for fetching real-time stock data, and matplotlib or plotly for visualization, the tracker provides users with an intuitive and dynamic way to monitor their investment performance.

The project successfully automates the tracking of stock prices, calculates portfolio returns, and provides insightful metrics like daily change, overall gains/losses, and sector allocation. Python’s flexibility also allows for future enhancements such as alert systems, predictive analytics using machine learning, and database integration for long-term performance tracking.

Overall, this project showcases how Python can be leveraged to build practical tools that empower individuals to manage their finances more effectively and make data-driven investment decisions.

Bottom of Form

**BIBLIOGRAPHY**

1. **Python Software Foundation.** (2024). Python Programming Language. <https://www.python.org>
2. Python **development team.** (2024). pandas: Powerful Python data analysis toolkit. <https://pandas.pydata.org>

**APPENDICES**

**DATA FLOW DIAGRAM**

#### ****External Entity:****

* **User** – Adds/removes stocks, requests portfolio view

#### ****Processes:****

1. **Add Stock to Portfolio**
2. **Remove Stock from Portfolio**
3. **View Portfolio Performance**
4. **Fetch Real-Time Stock Data**

#### ****Data Flows:****

* User → [1] Add Stock to Portfolio → Portfolio Data
* User → [2] Remove Stock from Portfolio → Portfolio Data
* User → [3] View Portfolio Performance → [4] Fetch Stock Data → Yahoo Finance API → Real-Time Prices
* [3] View Portfolio Performance → Portfolio Data
* [3] View Portfolio Performance → Output to User (Table with values)

### Level 1

sql

CopyEdit

+-------+ +------------------------+

| | Add/Remove | |

| User | -------------> | 1. Add/Remove Stock |

| | | |

+-------+ +------------------------+

| ▲

v |

+--------------------------+

| Portfolio Data Store | <----------------+

+--------------------------+ |

| |

View Request v |

+-------+ +------------------------+ |

| | <------------- | 3. View Portfolio | ---------------+

| User | +------------------------+

+-------+ |

v

+--------------------------+

| 4. Fetch Stock Prices |

| (Yahoo Finance API) |

+--------------------------+

|

v

+------------------------+

| Real-Time Stock Prices |

+------------------------+

**SAMPLE INPUT**

python

CopyEdit

from flask import Flask, render\_template, request, redirect

import sqlite3

import yfinance as yf

app = Flask(\_\_name\_\_)

DB = 'portfolio.db'

# Initialize the database and create the portfolio table if it doesn't exist

def init\_db():

with sqlite3.connect(DB) as conn:

conn.execute('''CREATE TABLE IF NOT EXISTS portfolio (

id INTEGER PRIMARY KEY,

symbol TEXT NOT NULL,

shares INTEGER,

buy\_price REAL

)''')

# Get the current stock price from Yahoo Finance

def get\_current\_price(symbol):

stock = yf.Ticker(symbol)

return stock.history(period="1d")["Close"].iloc[-1]

@app.route('/')

def index():

with sqlite3.connect(DB) as conn:

cur = conn.cursor()

cur.execute("SELECT \* FROM portfolio")

rows = cur.fetchall()

portfolio = []

total\_value = 0

for row in rows:

symbol, shares, buy\_price = row[1], row[2], row[3]

current\_price = get\_current\_price(symbol)

current\_value = current\_price \* shares

profit\_loss = current\_value - (buy\_price \* shares)

percent\_gain = (profit\_loss / (buy\_price \* shares)) \* 100

portfolio.append({

'symbol': symbol,

'shares': shares,

'buy\_price': buy\_price,

'current\_price': current\_price,

'value': current\_value,

'profit\_loss': profit\_loss,

'percent\_gain': percent\_gain

})

total\_value += current\_value

return render\_template('index.html', portfolio=portfolio, total\_value=total\_value)

@app.route('/add', methods=['POST'])

def add\_stock():

symbol = request.form['symbol'].upper()

shares = int(request.form['shares'])

buy\_price = float(request.form['buy\_price'])

with sqlite3.connect(DB) as conn:

conn.execute("INSERT INTO portfolio (symbol, shares, buy\_price) VALUES (?, ?, ?)",

(symbol, shares, buy\_price))

return redirect('/')

@app.route('/remove/<symbol>')

def remove\_stock(symbol):

with sqlite3.connect(DB) as conn:

conn.execute("DELETE FROM portfolio WHERE symbol = ?", (symbol,))

return redirect('/')

if \_\_name\_\_ == '\_\_main\_\_':

init\_db()

app.run(debug=True)

### 

### SAMPLE OUTPUT

### 

### 

### 