CHAPTER 1

INTRODUCTION

1.1 OBJECTIVE

A trading journal is a record of all your trades. Its main goal is to help you to find patterns in your trades that can improve your performance and avoid mistakes you have made in the past. It gives you more insights into your trading and allows you to make better decisions.

1.2 PROBLEM DEFINITION

A trading journal is a log that you can use to record your trades. Traders use a trading journal to reflect upon previous trades so that they may evaluate themselves, and you should too! In existing system, you can only access the foreign stock exchange market values and trades and if you want to explore more about the thing it asks for premium subscription**.**

1.3 FEATURES OF TRADING JOURNAL AND ANALYSIS WEBSITE

In the world of trading, it is essential to maintain a trading journal to track trades and analyze their success or failure. A trading journal serves as a record of trades, including entry and exit points, position sizing, and reasoning behind the decision to trade.

The analysis of trading data in a journal helps traders identify patterns and understand their strengths and weaknesses. By reviewing past trades, traders can identify what worked well and what did not, enabling them to adjust their strategies accordingly.

In addition to tracking trades, a trading journal can also be used to analyze market data and trends. This can help traders make more informed decisions about when to enter and exit the market. Overall, a trading journal is an essential tool for any trader looking to improve their performance and achieve success in the markets. It enables traders to track their progress, learn from their mistakes, and continuously improve their trading strategies.

1. Help You Develop Your Strategies

Paper trading, experimenting with different strategies, and keeping a trading journal of your progress can all assist you in determining which strategies work best for you. That is how you can work to improve over time.

2. Help You Develop Your Discipline in Trading

Keeping a journal forces you to confront the truth about what you’re doing right and wrong. When you’re honest about where you could improve, it can help you focus your study time. It can also help to improve discipline by instilling a sense of responsibility.

3. Help You Master Your Emotions

It would be fantastic if you could trade without feeling. Unfortunately, this is not an option. That much is obvious. Consider all the obstinate short-sellers and their poor decisions when caught in one of the recent insane short squeezes. Tracking your trades can assist you in analyzing your behavior patterns and determining what factors influence your personal trading psychology.

1.4 NEEDS OF TRADING JOURNAL AND ANALYSIS WEBSITE

Trading journal and analysis are tools used in trading to track and evaluate the performance of trades made by traders. Some of the features of trading journal and analysis include:

Trade History: Trading journal and analysis keep a record of all trades made by the trader, including the entry and exit points, the size of the position, and the market conditions at the time of the trade.

Performance Metrics: Trading journal and analysis provide metrics to measure the performance of trades, such as the profit or loss, win rate, and risk-reward ratio.

Analytics: Trading journal and analysis provide analytics tools to measure the effectiveness of trading strategies and identify patterns and trends.

Real-time data: Real-time data provided by trading journal and analysis helps traders make informed decisions based on market conditions.

Trade Planning: Trading journal and analysis assist traders in planning trades by providing a historical context for market movements, identifying recurring patterns, and assessing risk.

Portfolio management: Trading journal and analysis help traders manage their portfolios by providing information on diversification, risk management, and performance tracking.

CHAPTER 2

LITRATURE SURVEY

2.1 TITLE :AN INTERMARKET APPROACH TO BETA ROTATION: THE STRATEGY SIGNAL, AND POWER OF UTILITIES

AUTHOR : BILELLO CHARLES V AND MICHAEL A GAYED

YEAR : 2014

DESCRIPTION:

It is often said by proponents of the Efficient Market Hypothesis that no strategy can consistently outperform a simple buy and hold investment in broad stock averages over time. However, using a strategy based on the principles of intermarket analysis, we find that this assertion is not entirely accurate. The Utilities sector has many unique characteristics relative to other sectors of the broader stock market, including its higher yield, lower beta, and relative insensitivity to cyclical behavior. Our analysis suggests that rolling outperformance in the sector is not only exploitable, but also provides important signals about market volatility, seasonality, and extreme market movement. We explore historical price behavior and create a simple buy and rotate strategy that is continuously exposed to equities, positioning into either the broad market or the Utilities sector based on lead-lag dynamics. Absolute performance and risk-adjusted returns for this beta rotation approach significantly outperform a buy and hold strategy of the market and of the Utilities sector throughout multiple market cycles.

Buy and hold is often touted as the ultimate investment strategy when it comes to stock market investing. The reasoning for this relates to the belief in the Efficient Market Hypothesis, which states that because all known information is factored into price, there is largely no edge to active and dynamic trading. Indeed, numerous studies have documented the inability of investment managers benchmarked to a market average to consistently outperform passive strategies through stock selection.1However, academic studies have also noted persistent anomalies and phenomena in the marketplace which are consistent and exploitable.

2.2 TITLE : DOW'S THEORY OF CONFIRMATION MODERNIZED

AUTHOR : DAHLBERG.C

YEAR : 2016

DESCRIPTION:

One of the greatest advantages of technical analysis is that you can apply TA on any asset class as long as the asset type has historical time series data. Time series data in technical analysis is the price information, namely – open high, low, close, volume, etc.

Here is an analogy that may help. Think about learning how to drive a car. Once you learn how to drive a car, you can drive any car, whether a Mahindra XUV or a Maruti Swift. Likewise, you only need to learn technical analysis once. Once you do so, you can apply TA on any asset class – equities, commodities, foreign exchange, fixed income, etc.

The fact that TA can be applied to multiple assets is probably one of the biggest advantages of TA compared to the other stock market research techniques. For example, one has to study the profit and loss, balance sheet, and cash flow statements when it comes to the fundamental analysis of equity. However, the fundamental analysis of commodities is completely different.

When dealing with an agricultural commodity like Coffee or Pepper, the fundamental analysis includes analyzing rainfall, harvest, demand, supply, inventory etc. However, the fundamentals of metal commodities are different, so it is for energy commodities. so every time you choose a commodity, the fundamentals change.

On the other hand, the concept of technical analysis will remain the same irrespective of the asset you are studying. For example, an indicator such as 'Moving Average Convergence Divergence' (MACD) or 'Relative Strength Index' (RSI) is used the same way on equity, commodity, or currency.

2.3 TITLE : LEVERAGE FOR THE LONG RUN - A SYSTEMATIC APPROUCH TO MANAGING RISK AND MAGNIFYING RETURNS IN STOCKS

AUTHOR : MICHAEL A GAYED, AND CHARLES V. BILELLO

YEAR : 2016

DESCRIPTION:

Using leverage to magnify performance is an idea that has enticed investors and traders throughout history. The critical question of when to employ leverage and when to reduce risk, though, is not often addressed. We establish that volatility is the enemy of leverage and that streaks in performance tend to be beneficial to using margin. The conditions under which higher returns would be achieved from using leverage, then, are low volatility environments that are more likely to experience consecutive positive returns. We find that Moving Averages are an effective way to identify such environments in a systematic fashion. When the broad U.S. equity market is above its Moving Average, stocks tend to exhibit lower than average volatility going forward, higher average daily performance, and longer streaks of positive returns. When below its Moving Average, the opposite tends to be true, as volatility often rises, average daily returns are lower, and streaks in positive returns become less frequent. Armed with this finding, we developed a strategy that employs leverage when the market is above its Moving Average and deleverages (moving to Treasury bills) when the market is below its Moving Average. This strategy shows better absolute and risk-adjusted returns than a comparable buy and hold unleveraged strategy as well as a constant leverage strategy. The results are robust to various leverage amounts, Moving Average time periods, and across multiple economic and financial market cycles.

2.4 TITLE : EVALUATION OF SYSTEMATIC TRADING PROGRAMS

AUTHOR : MUNENZON, MIKHAIL

YEAR : 2015

DESCRIPTION:

This paper is intended as a non-technical overview of the issues I found valuable in evaluation of systematic trading programs both as a systematic trader and as a large, institutional investor, having looked at numerous, diverse managers in this space on a global basis over the years. Not having been able to find similar material in one source elsewhere when I began research of systematic trading managers, I hope that other investors will become more informed about opportunities and risks of the space that is too often poorly understood. While technical education can be helpful, it is not in my view required to make a high-quality allocation decision. Rather, a process for learning about and evaluating issues relevant to systematic trading, particularly those that are unique to this investment universe, are key. Additional references are listed as necessary to allow the reader to engage in deeper research. Some of the topics discussed below apply to discretionary traders and all types of investment organizations though the focus will always remain on systematic trading. Throughout this paper, I assume that systematic trading refers to an investment program for exchange listed instruments or spot FX that generates signals, manages positions, and executes applicable orders via an automated, previously programmed process with little or no human interference. The list of topics is not by any means exhaustive, but it should be sufficient to allow an investor to start on the path towards successful allocations with systematic trading managers. As I hope will become clear over the course of this paper, systematic trading is a highly unique field with its own set of advantages over other investment approaches that investors should not overlook. The paper is structured as follows. In Part I, I discuss the value proposition of systematic trading that ensures it a unique niche among available investment options. In Part 2, I then proceed to discuss key areas of evaluation of systematic trading programs: intellectual framework, signal generation, risk management, back testing, evaluation of live performance, technology, and operational structure. Part 3 includes my concluding remarks and references.

CHAPTER 3

SYSTEM ANALYSIS

3.1 INTRODUCTION

A trading journal records your trades and their outcomes and gives a summary of your trading experience. However, it is not a brokerage account statement as one can find the reasons behind opting for or avoiding a trading strategy. All successively executed trades are methodically planned, and a trading journal can be a record of the performance of each trading strategy. Regardless of how the market performs, you can adequately assess the potential of a particular trade using a trading journal. Moreover, you don't need to spend much to create a trading journal. Spreadsheets or Excel would suffice, and it would help you to become disciplined and follow consistent trading strategies. You should record trading entries in your journal if you can't always stick to your trading strategy. You can figure out how to avoid responding the same way to comparable situations in future trades by noting when things go wrong and why they did so.

3.2 EXISTING SYSTEM

A trading journal is a log that you can use to record your trades. Traders use a trading journal to reflect upon previous trades so that they may evaluate themselves, and you should too! In existing system, you can only access the foreign stock exchange market values and trades and if you want to explore more about the thing it asks for premium subscription.

3.3 DISADVANTAGES OF EXISTING SYSTEM

- In this existing system we cannot access and visualize the Indian Market data.
- It doesn't have trading journal which keeps track of records of trades which will be used to realizing their mistakes

3.4 PROPOSED SYSTEM

A trading journal is a log that you can use to record your trades. Traders use a trading journal to reflect upon previous trades so that they may evaluate themselves. We can track the data of the live data in NSE which will show the top gainers and top losers. For Options traders it will show the PE data & CE data Having a trading journal should be one of the first steps traders implement when learning to trade. A journal is of utmost important to testing different strategies and finding which trading plans work for individual traders.

A trading journal is essential in testing whether a current trading strategy is working. To summarize:

- Trading journals are there to log your trading activity.
- They help traders test different trading plans and strategies.
- Trading journals can also help traders pinpoint strengths and weaknesses in a trading style.

To add to your knowledge, see the Number One Mistake Traders Make where we analyzed thousands of live trades and came to a striking conclusion.

A forex trading journal is a log of your trades that can help you refine your strategies based on learning from previous experiences. Just as a business owner tracks inventory, a trader should also keep up with their closed positions.

While keeping a trading journal may be difficult at first, recording your trades can help answer some critical questions about your trading techniques. It can increase the consistency of your trading, keep you accountable, and improve your technique overall. In this piece we will explore what you need to know about journaling,

3.5 ADVANTAGES OF PROPOSED SYSTEM

A trading journal helps you keep track of your trades and trading progress and helps you become a more efficient trader by learning where you do well and where you do less well. This way you can focus on what you're best.

1. Improved data visualization: By visualizing stock data in a graphical format, traders can more easily identify patterns and trends in the data. This can help traders make more informed decisions about when to buy, sell or hold a particular stock.

2. Enhanced analysis: A trading journal analysis stock visualization project can provide traders with a more detailed and comprehensive analysis of their trading performance. This can help traders identify strengths and weaknesses in their trading strategies, and make adjustments accordingly.

3. Better decision-making: By providing traders with a clear and concise visual representation of their trading data, a stock visualization project can help traders make more informed decisions. This can lead to improved performance and more successful trades over time.

4. Increased efficiency: By automating the process of analyzing and visualizing trading data, a trading journal analysis stock visualization project can save traders time and increase efficiency. This can free up more time for traders to focus on other aspects of their trading strategies.

CHAPTER 4
SYSTEM REQUIREMENTS

4.1 HARDWARE REQUIREMENTS

- Processor : Dual core processor 2.6.0 GHz
- RAM : 1GB
- Hard disk : 160 GB
- Compact Disk : 650 MB
- Keyboard : Standard Keyboard

4.2 SOFTWARE REQUIREMENTS

- Processor :1.6gigahertz (GHz) or faster, 2 cores
- Operating system : Windows 11, Windows 10, Windows Server 2019
- Memory : 4 GB RAM
- Hard disk space : 4.0 GB of available disk space

CHAPTER 5
MODULE DESCRIPTION

5.1 INTRODUCTION

The ASP.NET Core MVC framework is a lightweight, open source, highly testable presentation framework optimized for use with ASP.NET Core. ASP.NET Core MVC provides a patterns-based way to build dynamic websites that enables a clean separation of concerns. It gives you full control over markup, supports TDD-friendly development and uses the latest web standards. It allows developers to build lightweight and high-performance web services that can be used by a wide range of client applications, including web applications, mobile apps, and desktop applications.

MVC Pattern:

The Model-View-Controller (MVC) architectural pattern separates an application into three main groups of components: Models, Views, and Controllers. This pattern helps to achieve separation of concerns. Using this pattern, user requests are routed to a Controller which is responsible for working with the Model to perform user actions and/or retrieve results of queries. The Controller chooses the View to display to the user, and provides it with any Model data it requires. The following diagram shows the three main components and which ones reference the others:

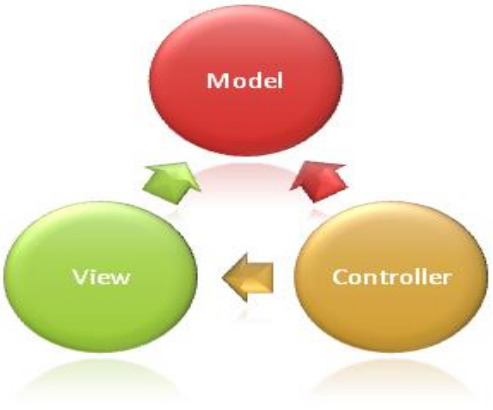


Fig 5.1.1 MVC Architecture

Model Responsibilities:

The Model in an MVC application represents the state of the application and any business logic or operations that should be performed by it. Business logic should be encapsulated in the model, along with any implementation logic for persisting the state of the application. Strongly-typed views typically use ViewModel types designed to contain the data to display on that view. The controller creates and populates these ViewModel instances from the model.

View Responsibilities:

Views are responsible for presenting content through the user interface. They use the Razor view engine to embed .NET code in HTML markup. There should be minimal logic within views, and any logic in them should relate to presenting content. If you find the need to perform a great deal of logic in view files in order to display data from a complex model, consider using a View Component, ViewModel, or view template to simplify the view.

Controller Responsibilities:

Controllers are the components that handle user interaction, work with the model, and ultimately select a view to render. In an MVC application, the view only displays information; the controller handles and responds to user input and interaction. In the MVC pattern, the controller is the initial entry point, and is responsible for selecting which model types to work with and which view to render (hence its name - it controls how the app responds to a given request).

5.2 FRONT-END DESIGN: .NET CORE WEB MVC

Razor Page:

.NET Core Web API with Razor Pages is a combination of two frameworks developed by Microsoft: .NET Core Web MVC and Razor Pages. Razor Pages is a web application framework that allows developers to create server-side web pages using C# and HTML. When using .NET Core Web MVC with Razor Pages, developers can create web applications that include both server-side pages and RESTful MVCs.

HTML:

HTML (Hypertext Markup Language) is a markup language used to create web pages and web applications. It is the standard markup language used to structure content on the World Wide Web and is essential for creating websites. It is a high-level, dynamic, and interpreted language that is used to create interactive and dynamic web pages. Here are some of the key features and functionalities of JavaScript.

CSS:

CSS (Cascading Style Sheets) is a style sheet language used to describe the presentation of web pages and web applications. It is used in combination with HTML and JavaScript to create visually appealing and interactive websites.CSS provides a way to separate the content and structure of a web page from its presentation.

Bootstrap:

Bootstrap is an open-source front-end development framework that was originally developed by Twitter. It is a popular choice for creating responsive and mobile-first websites and web applications. Bootstrap provides a set of CSS.

JQuery:

JQuery is a popular and widely used JavaScript library that simplifies client-side scripting of HTML. It was introduced in 2006 by John Resign and has since become one of the most popular libraries for web development.

JavaScript:

JavaScript is a popular and widely used programming language that is used primarily for developing web applications. It is a high-level, dynamic, and interpreted language that is used to create interactive and dynamic web pages. Here are some of the key features and functionalities of JavaScript:

5.3 BACK-END: .NET CORE WEB MVC

.NET Core Web MVC is a framework for building HTTP-based web services using the .NET Core platform. It allows developers to create RESTful MVC that can be consumed by web and mobile applications, as well as other services.

Browser Developer Tools:

Browser developer tools are built-in tools in web browsers that allow developers to inspect and debug HTML, CSS, and JavaScript code in real-time. Some popular browser developer tools include Chrome DevTools, Firefox Developer Tools, and Safari Web Inspector. These are just some of the many tools available for web development.

Client-Side Validation:

JavaScript is a dynamic, high-level programming language that is primarily used to add interactivity to web pages. Unlike traditional programming languages like C++ or Java, JavaScript does not need to be compiled into machine code before it can be executed. Instead, it is interpreted at runtime by the web browser.

One of the key features of JavaScript is its lightweight nature. It is a relatively simple language, with a syntax that is easy to learn and understand. This makes it an ideal choice for beginners who are just starting out with programming.

JavaScript is also known for its client-side scripting capabilities, which allow it to interact with users and make web pages more dynamic. For example, JavaScript can be used to create interactive forms, add animations, and update content dynamically without requiring the user to reload the page.

Another important aspect of JavaScript is its object-oriented capabilities. Although JavaScript is not strictly an object-oriented language like Java or C++, it does support object-oriented programming concepts such as encapsulation, inheritance, and polymorphism.

5.4 DATA FLOW DIAGRAM

In this diagram, the process starts with raw data collection, which may involve gathering data from sensors, inspection tools, or other sources. The collected data is then passed to a data processing and analysis component, which may involve various algorithms and statistical methods to he main interface for users to interact with the system.

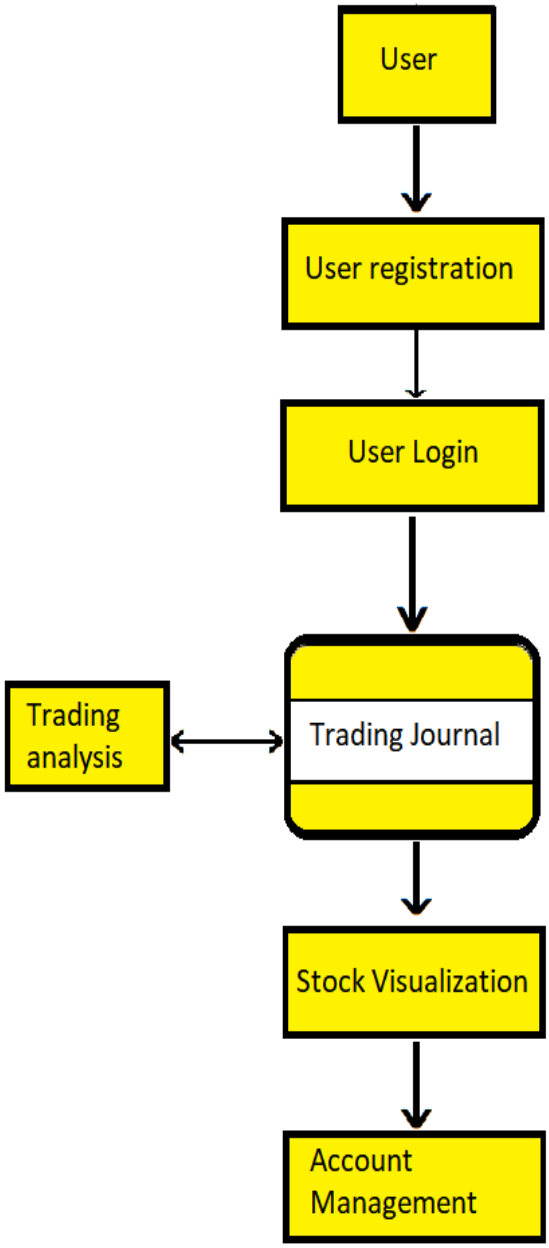


Fig.5.4.1 Data Flow Diagram

5.5 SYSTEM ARCHITECTURE

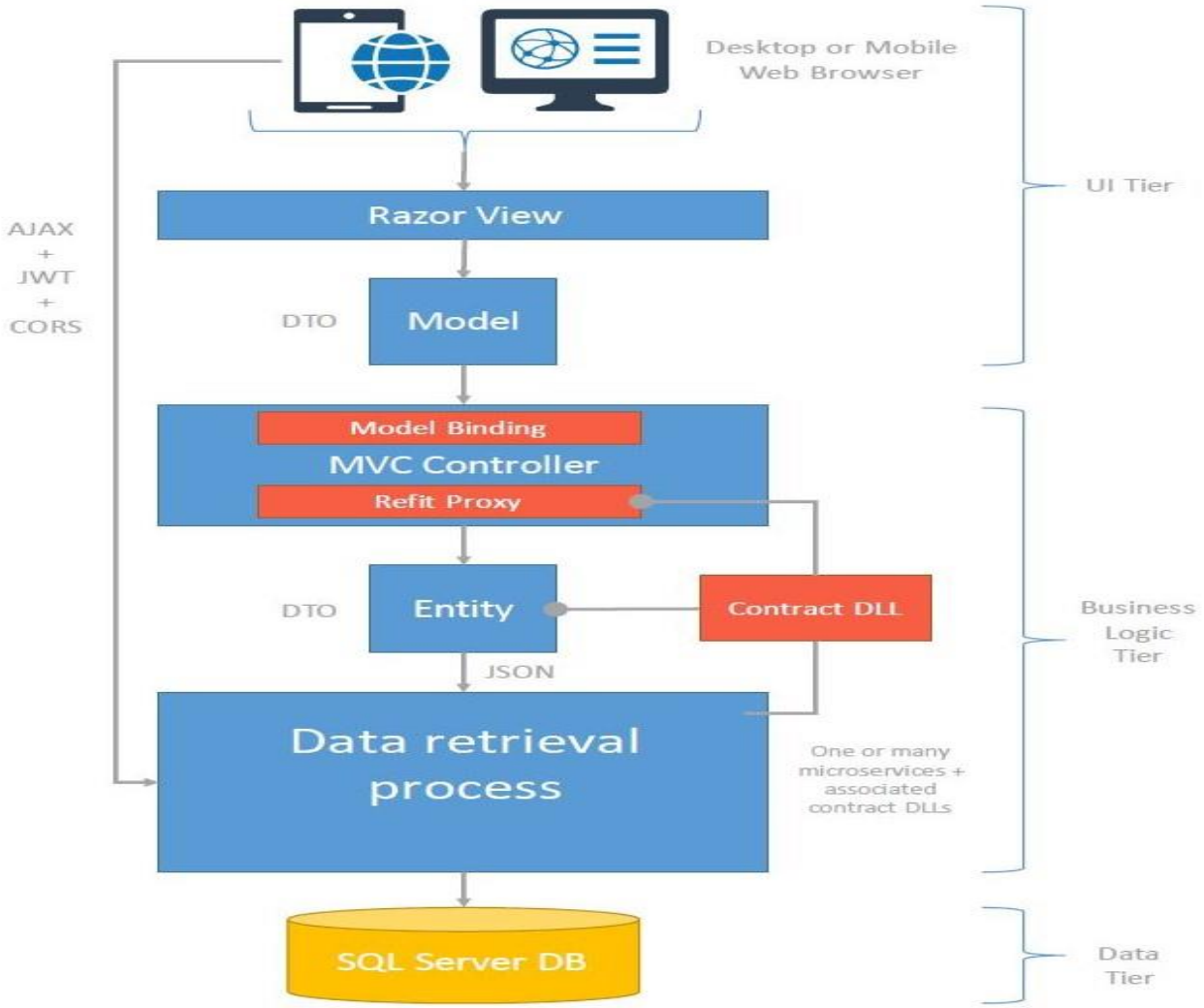


Fig.5.5.1 System Architecture

How it is Done?

Bitbucket is a web-based hosting service for Git and Mercurial version control systems. It provides a range of features and functions that enable teams to collaborate on code and manage software projects more efficiently.

Repository Management: Bitbucket provides a platform for hosting repositories and managing code. Teams can create repositories for their projects and manage access and permissions to ensure that only authorized.

Version Control: Bitbucket uses Git and Mercurial version control systems to track changes to code over time. This allows teams to collaborate on code and keep track of changes made by each contributor.

Issue Tracking: Bitbucket provides a built-in issue tracking system that allows teams to track and manage tasks, bugs, and feature requests. Issues can be assigned to team members, tagged with labels, and tracked through various stages of completion.

5.6 SYSTEM FLOW DIAGRAM

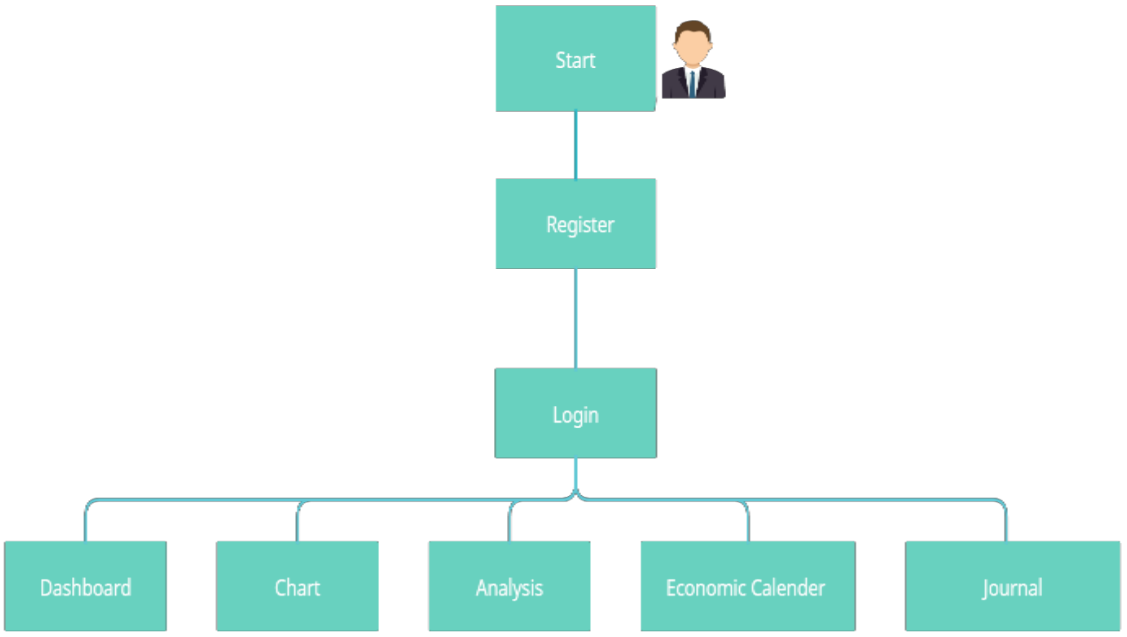


Fig.5.6.1 System flow diagram

5.7 DESCRIPTION ABOUT PROPOSED SYSTEM

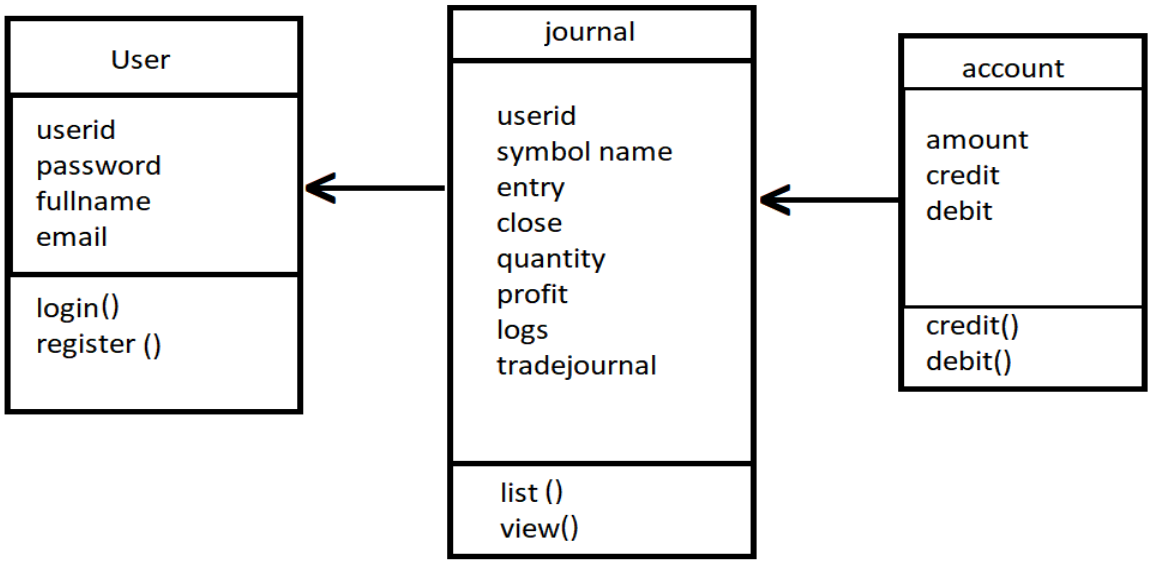


Fig.5.7.1 Database table

5.8 MODULE-1: TRADING JOURNAL

What is a Trading Journal?

A trading journal is a log that you can use to record your trades. Traders use a trading journal to reflect upon previous trades so that they may evaluate themselves, and you should too! You can use journals to evaluate where you can improve your trading. They are a useful form of record keeping.

Why Trading Journals are Useful?

Main reasons to keep a trading journal include:

- They help you identify weak points and strong points in your style.
- Journals could increase trading consistency.
- The journal could keep you accountable.
- The journal can help you choose your best trading strategy.

Keeping a journal is a simple yet extremely effective way to improve a trading plan. A trading plan is a set of rules and guidelines you will follow that includes strategy, risk management, and trader psychology.

A trading journal is a written log or record of your trades that helps you keep track of your trading performance and progress over time. It's an essential tool for any trader who wants to improve their skills and profitability.

Here are some tips on how to create an effective trading journal:

1. Include the basics: Your trading journal should contain essential information about each trade, such as the date, time, instrument, position size, entry and exit price, and any fees or commissions.

2. Analyze your trades: Take the time to reflect on each trade and analyze what went right and what went wrong. Consider factors such as your entry and exit strategy, market conditions, and your emotions during the trade.

3. Use charts and graphs: Visual aids such as charts and graphs can help you spot patterns in your trading and identify areas for improvement. You can use tools like Excel or Google Sheets to create these visuals.

4. Keep it organized: Use a consistent format for each trade and keep your trading journal organized so that you can quickly reference past trades and track your progress over time.

5. Review regularly: Set aside time each week or month to review your trading journal and look for patterns or trends in your performance. Use this information to adjust your trading strategy and improve your profitability over time.

Remember, a trading journal is only as useful as the information you put into it. Be honest with yourself and take the time to analyze your trades in-depth. By doing so, you can develop a better understanding of your strengths and weaknesses as a trader and improve your performance over time.

5.9 MODULE-2: STOCKS VISUALIZING

- Stock analysis is a method for investors and traders to make buying and selling decisions. By studying and evaluating past and current data, investors and traders attempt to gain an edge in the markets by making informed decisions.
- The notion of stock analysis relies on the assumption that available market information can be used to determine the intrinsic value of a stock. In the primary methods discussed below, investors use financial statements, stock price movement, market indicators, or industry trends to make investment decisions.
- Fundamental analysis concentrates on data from sources, including financial records, economic reports, company assets, and market share. To conduct fundamental analysis on a public company or sector, investors and analysts.

5.10 MODULE-3: TRADING CHARTS

- The trading chart displays information that can help you decide when to enter and exit a position. There are many kinds of trading charts: bar charts, line charts, point and figure, market profile and candlesticks. For this example, we'll focus on candlesticks, one commonly used chart type.
- A candlestick chart is a combination of a line chart and a bar graph. You can change chart types depending on your preferred view, but most traders prefer candlesticks because of the depth of information each stick can convey. Each candlestick gives you four key pieces of information within your selected time interval.

5.11 MODULE-4: ACCOUNT MANAGEMENT

- To check the balance of your trading account, log in to your account using the trading platform or mobile app provided by your broker. Once you're logged in, you should be able to see your account balance displayed prominently on the screen.
- To add funds to your trading account, go to the "Deposit" or "Add Funds" section of your trading platform or mobile app. Choose your preferred deposit method, such as bank transfer or credit card, and enter the amount you wish to add. Follow the prompts to complete the transaction.
- If you want to withdraw funds from your trading account, go to the "Withdraw" or "Request Withdrawal" section of your trading platform or mobile app. Choose your preferred withdrawal method, such as bank transfer or credit card, and enter the amount you wish to withdraw. Follow the prompts to complete the transaction.
- To view your trading account history, go to the "Account History" section of your trading platform or mobile app. This will show you a record of all your transactions, including deposits, withdrawals, trades, and any fees or commissions charged by your broker.
- To place trades in your trading account, use the trading platform or mobile app provided by your broker. You will need to select the asset you wish to trade, such as stocks, forex, or commodities, and choose whether to buy or sell. Enter the amount you wish to trade, and follow the prompts to complete the transaction.

CHAPTER 6
TESTING

6.1 INTRODUCTION

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of tests. Each test type addresses a specific testing requirement.

6.2 TYPES OF TESTING

6.2.1 Unit Testing

The first test in the development process is the unit test. The source code is normally divided into modules, which in turn are divided into smaller units called units. These units have specific behavior. The test done on these units of code is called unit test. Unit test depends upon the language on which the project is developed. Unit tests ensure that each unique path of the project performs accurately to the documented specifications and contains clearly defined inputs and expected results. A Unit corresponds to a screen /form in the package. Unit testing focuses on verification of the corresponding class or Screen. This testing includes testing of control paths, interfaces, local data structures, logical decisions, boundary conditions, and error handling. Unit testing may use Test Drivers, which are control programs to coordinate test case inputs and outputs and Test stubs, which replace low-level modules. A stub is a dummy subprogram.

6.2.2 Integration Testing

Testing in which modules are combined and tested as a group is known as integration testing. Modules are typically code modules, individual applications, source and destination applications on a network, etc. Integration Testing follows unit testing and precedes system testing. Testing after the product is code complete. Betas are often widely distributed or even distributed to the public at large in hopes that they will buy the final product when it is release. Integration testing is used to verify the combining of the software modules. Integration testing addresses.

6.2.3 System Testing

System testing is a critical aspect of Software Quality Assurance and represents the ultimate review of specification, design and coding. Testing is a process of executing a program with the intent of finding an error. A good test is one that has a probability of finding an as yet undiscovered error. The purpose of testing is to identify and correct bugs in the developed system.

Nothing is complete without testing. Testing is the vital to the success of the system. In the code testing the logic of the developed system is tested. For this every module of the program is executed to find an error. To perform specification test, the examination of the specifications stating what the program should do and how it should perform under various condition

6.2.4 Functional Testing

It is a type of software testing whereby the system is tested against the functional requirements/specifications. Functions (or features) are tested by feeding them input and examining the output. Functional testing ensures that the requirements are properly satisfied by the application. This type of testing is not concerned with how processing occurs, but rather, with the results of processing. It simulates actual system usage but does not make any system structure assumptions. Apart from these tests, there are some special tests conducted which are given below:

6.2.5 Performance Time Testing

This test determines the length of the time used by the system to process transaction data. In this phase the software developed Testing is exercising the software to uncover errors and ensure the system meets defined requirements. Testing may be done at 4 levels.

- Module Level Testing
- Regression Testing

6.2.5.1 Module Level Testing

Module Testing is done using the test cases prepared earlier. Module is defined during the time of design

6.2.5.2 Regression Testing

Each modification in software impacts unmodified areas, which results serious injuries to that software. So, the process of re-testing for rectification of errors due to modification is known as regression testing. Delivery Installation and Delivery are the process of delivering the developed and tested software to the customer. Refer the support procedures Acceptance and Project Closure Acceptance is the part of the project by which the customer accepts the product. This will be done as per the Project Closure, once the customer accepts the product closure of the project is started. This includes metrics collection, PCD etc.

CHAPTER 7

SOFTWARE IMPLEMENTATION

7.1 CREATE A WEB PROJECT

Visual Studio

- Visual Studio Code
- Visual Studio for Mac
- From the File menu, select New > Project.
- Enter MVC Web-Controller in the search box.
- Select the ASP.NET Core MVC template and select Next.

In the Configure your new project dialog, name the project TodoApi and select Next.In the Additional information dialog:

- Confirm the Framework is .NET 6.0 (or later).
- Select Create.

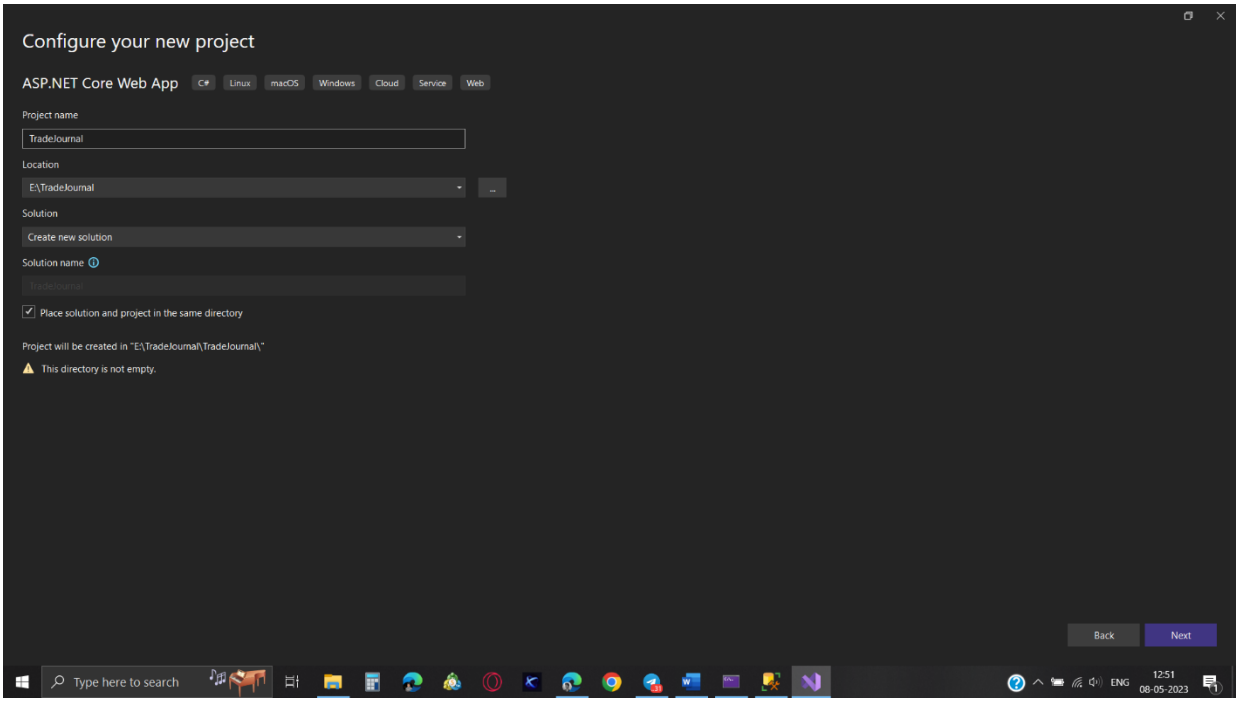


Fig.7.1.1 Create Web Project

7.2. ADD A MODEL CLASS

A model is a set of classes that represent the data that the app manages. The model for this app is the TodoItem class.

Visual Studio

Visual Studio Code

Visual Studio for Mac

In Solution Explorer, right-click the project. Select Add >New Folder.

Name the folder Models.

Right-click the Models folder and select Add > Class.

Name the class TodoItem and select Add.

Replace the template code with the following:

C#

```
using System;
using System.Collections.Generic;
namespace TradingJournal.Models
{
    public partial class UserReg
    {
        public UserReg()
        {
            Journals = new HashSet<Journal>();
        }
        public string UserId { get; set; } = null!;
        public string Password { get; set; } = null!;
        public string FullName { get; set; } = null!;
        public string Email { get; set; } = null!;
        public DateTime Joined { get; set; }
```

```
    public bool Active { get; set; }
    public virtual ICollection<Journal> Journals { get; set; }}
```

The Id property functions as the unique key in a relational database. This code defines a class named UserReg with some properties and a constructor. The constructor is a special method that is called when an object of the class is created. It can be used to initialize the properties or perform other actions. In this case, the constructor initializes the Journals property to a new list of Journal objects. The other properties are initialized to null values using the null-forgiving operator (!). This operator tells the compiler to ignore the possibility of null values for these properties. The properties also have getters and setters, which allow accessing and modifying their values from outside the class.

This model refers to the journal model class

```
using System;
using System.Collections.Generic;
namespace TradingJournal.Models
{
  public partial class Journal
  {
    public int JournalId { get; set; }
    public string UserId { get; set; } = null!;
    public string SymbolName { get; set; } = null!;
    public DateTime Created { get; set; }
    public int EntryPrice { get; set; }
    public int ClosePrice { get; set; }
    public int Quantity { get; set; }
    public int? Profit { get; set; }
    public int? Loss { get; set; }
    public string TradeJournal { get; set; } = null!;
    public virtual UserReg User { get; set; } = null!;
  }}
```

7.3 ADD A DATABASE CONTEXT

The database context is the main class that coordinates Entity Framework functionality for a data model. This class is created by deriving from the Microsoft.EntityFrameworkCore.DbContext class.

scaffold-dbcontext

"Server=DESKTOP-4O1D65I\SQLEXPRESS;

database=Project DB;

trusted_connection=true" Microsoft.EntityFrameworkCore .SqlServer -OutPutDir Entities

7.4 ADD NUGET PACKAGES

From the Tools menu, select NuGet Package Manager > Manage NuGet Packages for Solution.Select the Browse tab, and then enter Microsoft.EntityFrameworkCore.InMemory in the search box. Select Microsoft.EntityFrameworkCore. InMemory in the left pane.

Select the Project checkbox in the right pane and then select Install.

7.5 ADD THE DBCONTEXT DATABASE CONTEXT

Right-click the Models folder and select Add > Class. Name the class TodoContext and click Add.

C#

Copy

```
using System;
using System.Collections.Generic;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata;
namespace TradingJournal.Models
{
   public partial class ProjectDBContext : DbContext
   {
```

```
public ProjectDBContext()
{ }
public ProjectDBContext(DBContextOptions<ProjectDBcontext> options)
    : base(options)
{}
public virtual DBSet<Journal> Journals { get; set; } = null!;
public virtual DBSet<UserReg> UserRegs { get; set; } = null!;
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    if (!optionsBuilder.IsConfigured)
    {
optionsBuilder.UseSqlServer("Server=MRWHITE\\SQLEXPRESS;Database=ProjectDB;Trusted_Connection=True;");
    } }
    OnModelCreatingPartial(modelBuilder);
}
 partial void OnModelCreatingPartial(ModelBuilder modelBuilder);
}}
```

7.6 REGISTER THE DATABASE CONTEXT

In ASP.NET Core, services such as the DB context must be registered with the dependency injection (DI) container. The container provides the service to controllers.

Update Program.cs with the following highlighted code:

```
builder.Services.AddDbContext<IdentityContext>( options =>
        {
options.UseSqlServer(builder.Configuration["ConnectionString:DbConnection"]);
        });
```

builder.Services.AddIdentity<IdentityUser,
IdentityRole>().AddEntityFrameworkStores<IdentityContext>().AddDefaultTokenProvid
ers();

7.7 SCAFFOLD A CONTROLLER

Right-click the Controllers folder.

Select Add > New Scaffolded Item.

Select API Controller with actions, using Entity Framework, and then select Add.

In the Add Controller with actions, using Entity Framework dialog:

Select TodoItem (User.Models) in the Model class.

Select TodoContext (journal.Models) in the Data context class.

Select Add.

1.If the scaffolding operation fails, select Add to try scaffolding a second time

2. Right-click on the "Controllers" folder in your project, and select "Add" -> "Controller" from the context menu.

3. In the "Add Scaffold" dialog that appears, select the "MVC Controller with views, using Entity Framework" option and click "Add".

4. In the "Add Controller" dialog, select the Model class that you want to create a controller for and theDbcontextthat you want to use, and click "Add". This will generate a new controller class that has basic CRUD (Create, Read, Update, Delete) actions for the selected Model.

5. You can customize the generated code if needed, such as adding additional actions or modifying the existing ones.

6. Run the application to test the controller and make sure it's functioning as expected.

That's it! By following these steps, you should have a basic scaffolded controller with CRUD actions for your selected Model class. From here, you can add more functionality to your controller as needed for your application.

7.8 USER REGISTRATION CREATE METHOD

```
using Microsoft.AspNetCore.Identity.UI.V4.Pages.Account.Internal;
using Microsoft.AspNetCore.Mvc;

using TradingJournal.Models;

namespace TradingJournal.Controllers

{

  public class UserController : Controller

    {

        private readonly ProjectDBContext _dbContext;

        public UserController(ProjectDBContext dbContext)

        {

            this._dbContext = dbContext;

        }

        [HttpGet]

        public IActionResult Signup()

        {

            return View();

        }

        public Task <IActionResult> Signup(UserModel userModel)

        {

          return View(userModel,_dbContext);

        }

        private Task<IActionResult> View(UserModel userModel, ProjectDBContext
dbContext)

        {

            throw new NotImplementedException();

        }

        [HttpPost]
```

```
public async Task<IActionResult> Signup(UserModel userModel,ProjectDBContext projectDBContext)
{
    var User = new UserReg()
    {
        UserId = userModel.UserId,
        Password = userModel.Password,
        Email = userModel.Email,
        FullName = userModel.FullName,
    };
    await _dbContext.UserRegs.AddAsync(User);
    await _dbContext.SaveChangesAsync();
    return RedirectToAction("Login");
}
[HttpGet]
public IActionResult Login()
{
    UserModel loginModel = new UserModel();
    return View(loginModel);
}
[HttpPost]
public IActionResult Login(UserModel loginModel )
{
    ProjectDBContext projectDBContext = new ProjectDBContext();
Var status = projectDBContext.UserRegs.Where(m => m.Email loginModel.Email && m.Password == loginModel.Password).FirstOrDefault();
    if(status != null)
    {
```

```
            ViewBag.Message = "Success";
            return RedirectToAction("Index");
        }
        else
        {
            return RedirectToAction("Signup");
        }
        return View();
    }}}
```

The `UserController` class extends the `Controller` base class and has a constructor that takes a `ProjectDBContext` object as a parameter. `ProjectDBContext` is a custom class that represents the database context used by the application.

The controller has four action methods:
`Signup()` has two overloads, one for GET requests and another for POST requests. The GET overload returns a view for the user registration page, while the POST overload accepts a `UserModel` object (which represents user input from the registration form) and adds a new user record to the database.
`Login()` also has GET and POST overloads. The GET overload returns a view for the login page, while the POST overload accepts a `UserModel` object (which represents user input from the login form) and checks if the entered email and password match a record in the database. If there's a match, the user is redirected to the homepage; otherwise, they're redirected to the signup page.
Overall, this controller provides basic functionality for user authentication and registration, but it's worth noting that it could be improved in several ways, such as by adding input validation, error handling, and password hashing.

7.9 ADDING JOURNAL DATA TO THE DATABASE

Model class for Journal Table

The below given code represents the data adding to the database in SQL Server using Entity Framework Core

```
using TradingJournal.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
namespace TradingJournal.Controllers
{
    public class JournalController : Controller
    {
  private readonly ProjectDBContext projectDBContext;
  public JournalController(ProjectDBContext projectDBContext)
  {
    this.projectDBContext = projectDBContext;
  }
  [HttpGet]
        public IActionResult Add()
        {
            return View();
        }
  public Task<IActionResult> Add(AddJournalModel addJournal)
  {
    return Add(addJournal, projectDBContext);
  }
  [HttpPost]
        public async Task<IActionResult> Add(AddJournalModel addJournal, ProjectDBContext projectDBContext)
```

```
        {
            var Journals = new Journal()
            {
                JournalId = addJournal.JournalId,
                SymbolName = addJournal.SymbolName,
                Created = addJournal.Created,
                EntryPrice = addJournal.EntryPrice,
                ClosePrice = addJournal.ClosePrice,
                Quantity = addJournal.Quantity,
                Profit = addJournal.Profit,
                Loss = addJournal.Loss,
                TradeJournal = addJournal.TradeJournal
            };
            await projectDBContext.Journals.AddAsync(Journals);
            await projectDBContext.SaveChangesAsync();
            return RedirectToAction("Add");
        }
        //public async Task<IActionResult> view()
        //{
        //    var journals = await projectDBContext.Journals.ToListAsync();
        //    return View(journals);
        //}
        public IActionResult view()
        {
            return View();        }}}
```

This is a `JournalController` class in the TradingJournal application that is responsible for handling requests related to trading journal entries.

The class has a constructor that takes an instance of the `ProjectDBContext` class, which is used to access the database.

The `Add()` method is an HTTP GET action method that returns the `Add.cshtml` view, which is a form for adding a new trading journal entry.

The overloaded `Add()` method with an HTTP POST attribute is called when the form is submitted. This method takes an instance of the AddJournalModel` class as a parameter, which contains the details of the new journal entry to be added. The method creates a new instance of the `Journal` class and populates its properties with the data from the `AddJournalModel`. The new `Journal` object is then added to the `Journals` table in the database using the `AddAsync()` method, and the changes are saved to the database using the `SaveChangesAsync()` method. Finally, the method redirects the user to the `Add` view.

The `view()` method is an HTTP GET action method that returns the `view.cshtml` view, which will display all the journal entries in the database. However, this method is currently commented out, and the `view.cshtml` view is empty.

7.10 REPOSITORIES FOR CONTROLLER FUNCTION

In the context of ASP.NET MVC, a repository is an abstraction layer between the application's business logic and the data access layer. It provides a way to separate concerns and decouples the application from the underlying data store, allowing for easier testing and maintainability.

A repository typically defines a set of operations for CRUD (Create, Read, Update, Delete) operations on entities in the data store. These operations are typically implemented using an Object-Relational Mapping (ORM) framework such as Entity Framework, or a low-level database access API such as ADO.NET.

To use a repository in an ASP.NET MVC application, you would typically create a separate class library project to contain the repository implementation. The repository would be responsible for defining the data access operations for each entity in the application, and would provide methods for the controllers to interact with the data store.

For example, suppose you have an `Order` entity in your application. The repository for this entity might define methods such as `GetOrders()`, `AddOrder()`, `UpdateOrder()`, and

`DeleteOrder()`, which would use Entity Framework or another data access technology to perform the corresponding CRUD operations on the `Order` table in the database.

In the controller, you would create an instance of the `OrderRepository` class and use its methods to perform the required data access operations. This allows the controller to be decoupled from the specific data access technology being used, and makes it easier to swap out different data access technologies in the future if needed.

Overall, using repositories in an ASP.NET MVC application can help to improve maintainability, scalability, and testability, by providing a clear separation of concerns between the application's business logic and the data access layer.

This code defines a repository class for the "UserReg" model in the TradingJournal project. The repository is responsible for handling data access logic for the UserReg model, specifically for adding and updating UserReg entities to the database using the ProjectDBContext. The constructor initializes the repository with a new instance of the ProjectDBContext, which is used in the Add and Update methods to perform the corresponding database operations. The methods catch any exceptions that occur during the database operations and re-throw them to the caller

Journal Repository:

This code defines a JournalRepository class that contains methods for adding, removing, and retrieving Journal objects from a database. The repository relies on an instance of the ProjectDBContext class, which is an Entity FrameworkDbcontextthat represents the database context for the application.

The Add method adds a Journal object to the Journals DbSet of the context and saves changes to the database using the SaveChanges method. The Remove method removes a Journal object from the Journals DbSet of the context and saves changes to the database using the SaveChanges method.

The GetUserById method retrieves a Journal object from the Journals DbSet of the context based on the Email parameter passed to the method. It uses the Find method of the DbSet to search for the Journal object with the matching Email and returns the result. If no matching Journal object is found, the method returns null.

7.11 VIEWS FOR LISTING THE TABLE FROM DATABASE

The `UserController` class extends the `Controller` base class and has a constructor that takes a `ProjectDBContext` object as a parameter. `ProjectDBContext` is a custom class that represents the database context used by the application.

The controller has four action methods:

- `Signup()` has two overloads, one for GET requests and another for POST requests. The GET overload returns a view for the user registration page, while the POST overload accepts a `UserModel` object (which represents user input from the registration form) and adds a new user record to the database.
- `Login()` also has GET and POST overloads. The GET overload returns a view for the login page, while the POST overload accepts a `UserModel` object (which represents user input from the login form) and checks if the entered email and password match a record in the database. If there's a match, the user is redirected to the homepage; otherwise, they're redirected to the signup page.

Overall, this controller provides basic functionality for user authentication and registration, but it's worth noting that it could be improved in several ways, such as by adding input validation, error handling, and password hashing.

CHAPTER 8
CONCLUSION

The trading journal analysis project is an effective way to improve trading skills and profitability over time. By keeping a record of every trade, analyzing the results, and making adjustments based on the insights gained, traders can identify patterns and trends in their performance and make more informed decisions in the future.

Throughout the project, traders should focus on key aspects of their trading, such as entry and exit strategies, market conditions, risk management, and emotions. By reflecting on each trade and analyzing what went right and what went wrong, traders can gain a better understanding of their strengths and weaknesses and develop a more effective trading plan.

Using charts and graphs to visualize data is an effective way to identify patterns and trends that may not be immediately obvious from a written record. This can help traders to quickly spot areas for improvement and adjust their strategies accordingly.

It is also important to review the trading journal regularly and use the insights gained to make adjustments to the trading plan. By continually refining the plan based on actual trading results, traders can improve their profitability over time.

In conclusion, the trading journal analysis project is a valuable tool for any trader who wants to improve their trading skills and profitability. By keeping a detailed record of every trade, analyzing the results, and making adjustments based on the insights gained, traders can develop a more effective trading plan and achieve greater success in the markets..

CHAPTER 9
FUTURE IMPLEMENTATION

There are several future enhancements that can be made to the trading journal analysis project to further improve its effectiveness:

1. Automating the data collection: Manually recording every trade can be time-consuming and prone to errors. By automating the data collection process, traders can save time and ensure that their trading journal is accurate and up-to-date.

2. Adding more detailed analysis: While basic analysis can provide useful insights into trading performance, adding more detailed analysis can uncover more nuanced patterns and trends. For example, traders could use statistical analysis to identify correlations between different trading variables, or use machine learning algorithms to make predictions about future market movements.

3. Integrating with trading platforms: Many trading platforms now offer APIs that allow traders to access their trading data programmatically. By integrating the trading journal analysis project with a trading platform, traders can automatically import their trading data and perform more detailed analysis.

4. Incorporating social trading data: Social trading platforms allow traders to follow and copy the trades of other traders. By incorporating social trading data into the trading journal analysis project, traders can gain insights into the performance of other traders and use that information to inform their own trading decisions.

5. Adding visualization tools: While charts and graphs are a useful way to visualize trading data, there are many other visualization tools that can be used to gain insights into trading performance. For example, traders could use heat maps to identify the most profitable trading times or use network graphs to visualize.

APPENDIX 1

SAMPLE SOURCE CODE

```
@model TradingJournal.Models.UserModel

@{
    Layout = null;
}

<!DOCTYPE html>

<html lang="en">

<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <title>Login</title>
    <link rel="stylesheet" href="~/vendors/mdi/css/materialdesignicons.min.css">
    <link rel="stylesheet" href="~/vendors/css/vendor.bundle.base.css">
    <link rel="stylesheet" href="~/css/style.css">
    <link rel="shortcut icon" href="~/\images/logo-mini.png" />
</head>

<body>
    <div class="container-scroller">
        <div class="container-fluid page-body-wrapper full-page-wrapper">
            <div class="content-wrapper d-flex align-items-center auth">
                <div class="row flex-grow">
                    <div class="col-lg-4 mx-auto">
                        <div class="auth-form-light text-left p-5">
                            <div class="brand-logo">
                                <img src="~/images/logo.png">
```

```
</div>
 <h4>Hello! let's get started</h4>
<h6 class="font-weight-light">Sign in to continue.</h6>
@using (Html.BeginForm("Index", "Home")){
<form method="post" class="pt-3">
  <div class="form-group">
@Html.TextBoxFor(m=>m.Email,new{@class="form-control form-control-
      lg" })
   <!--  <input asp-type="email" class="form-control form-control-lg"
id="exampleInputEmail1" placeholder="Username"> -->
</div>
<div class="form-group">
@Html.PasswordFor(m => m.Password, new { @class = "form-control form-control-
      lg" })
  <!-- <input type="password" class="form-control form-control-lg"
id="exampleInputPassword1" placeholder="Password"> -->
</div>
<div class="mt-3">
   <a class="btn btn-block btn-gradient-primary btn-lg font-weight-
medium auth-form-btn" asp-controller="Home" asp-action="index" value="Submit"
type="Submit">SIGN IN</a>
</div>
<div class="my-2 d-flex justify-content-between align-items-center">
  <div class="form-check">
    <label class="form-check-label text-muted">
   <input type="checkbox" class="form-check-input"> Keep me
      signed in
```

```
                    </label>
                </div>
                <a href="#" class="auth-link text-black">Forgot password?</a>
            </div>
            <div class="mb-2">
                <button type="button" class="btn btn-block btn-facebook auth-form-
                    btn">
                    <i class="mdi mdi-facebook me-2"></i>Connect using facebook
                </button>
            </div>
            <div class="text-center mt-4 font-weight-light">
                Don't have an account? <a asp-controller="User" asp-action="Signup"
class="text-primary">
                    Create</a>
                </div>
            </form>
        </div>
    </div>
  </div>
</div>
<script src="~/vendors/js/vendor.bundle.base.js"></script>
<script src="~/js/off-canvas.js"></script>
<script src="~/js/hoverable-collapse.js"></script>
<script src="~/js/misc.js"></script>
```

```
</body>
</html>

@{
    Layout = "~/views/Shared/_Layout.cshtml";
}
<!-- TradingView Widget BEGIN -->
<div class="tradingview-widget-container">
    <div class="tradingview-widget-container__widget"></div>
    <script type="text/javascript" src="https://s3.tradingview.com/external-embedding/embed-widget-events.js" async>
      {
          "colorTheme": "light",
          "isTransparent": false,
          "width": "1170",
          "height": "600",
          "locale": "en",
          "importanceFilter": "-1,0,1",
          "currencyFilter":
"USD,EUR,ITL,NZD,CHF,AUD,FRF,JPY,ZAR,TRL,CAD,DEM,MXN,ESP,GBP"
  }
    </script>
</div>
@{
    Layout = "~/views/Shared/_Layout.cshtml";
}
```

```
<!-- TradingView Widget BEGIN -->
<div class="tradingview-widget-container">
  <div id="tradingview_e2952"></div>
  <script type="text/javascript" src="https://s3.tradingview.com/tv.js"></script>
  <script type="text/javascript">
    new TradingView.widget(
      {
        "width": 1150,
        "height": 600,
        "symbol": "BSE:ADANIENT",
        "interval": "D",
        "timezone": "Etc/UTC",
        "theme": "light",
        "style": "1",
        "locale": "en",
        "toolbar_bg": "#f1f3f6",
        "enable_publishing": false,
        "withdateranges": true,
        "hide_side_toolbar": false,
        "allow_symbol_change": true,
        "watchlist": [
          "OANDA:EURUSD",
          "OANDA:GBPUSD",
          "OANDA:GBPJPY",
          "OANDA:AUDUSD",
          "OANDA:USDJPY",
```

"OANDA:AUDUSD",

"OANDA:NZDUSD",

"BSE:ADANIENT",

"BSE:ADANIPORTS",

"BSE:APOLLOHOSP",

"BSE:ASIANPAINT",

"BSE:AXISBANK",

"BSE:BAJAJFINSV",

"BSE:BAJFINANCE",

"BSE:BHARTIARTL",

"BSE:BPCL",

"BSE:BRITANNIA",

"BSE:CIPLA",

"BSE:COALINDIA",

"BSE:DIVISLAB",

"BSE:DRREDDY",

"BSE:EICHERMOT",

"BSE:GRASIM",

"BSE:HCLTECH",

"BSE:HDFC",

"BSE:HDFCBANK",

"BSE:HDFCLIFE",

"BSE:HEROMOTOCO",

"BSE:HINDALCO",

"BSE:HINDUNILVR",

"BSE:ICICIBANK",

```
    "BSE:INDUSINDBK",
    "BSE:INFY",
    "BSE:ITC",
    "BSE:JSWSTEEL",
    "BSE:KOTAKBANK",
    "BSE:LT",
    "BSE:MARUTI",
    "BSE:NESTLEIND",
    "BSE:NTPC",
    "BSE:ONGC",
    "BSE:POWERGRID",
    "BSE:RELIANCE",
    "BSE:SBILIFE",
    "BSE:SBIN",
    "BSE:SUNPHARMA",
    "BSE:TATACONSUM",
    "BSE:TATAMOTORS",
    "BSE:TATASTEEL",
    "BSE:TCS",
    "BSE:TECHM",
    "BSE:TITAN",
    "BSE:ULTRACEMCO",
    "BSE:UPL",
    "BSE:WIPRO"
],
"details": true,
```

```
            "container_id": "tradingview_e2952"
        }   );
    </script>
</div>
namespace TradingJournal.Models
{
    public class UserModel
    {
        public string UserId { get; set; } = null!;
        public string Password { get; set; } = null!;
        public string FullName { get; set; } = null!;
        public string Email { get; set; } = null!;
        public DateTime Joined { get; set; }
        public bool Active { get; set; }
    }}


using System;
using System.Collections.Generic;
namespace TradingJournal.Models
{
    public partial class Journal
    {
        public int JournalId { get; set; }
        public string UserId { get; set; } = null!;
        public string SymbolName { get; set; } = null!;
        public DateTime Created { get; set; }
```

```
        public int EntryPrice { get; set; }
        public int ClosePrice { get; set; }
        public int Quantity { get; set; }
        public int? Profit { get; set; }
        public int? Loss { get; set; }
        public string TradeJournal { get; set; } = null!;
        public virtual UserReg User { get; set; } = null!;
    }}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using TradingJournal.Models;
namespace TradingJournal.Controllers
{
    public class UserRegsController : Controller
    {
        private readonly ProjectDBContext _context;
        public UserRegsController(ProjectDBContext context)
        {
            _context = context;
        }
```

```
// GET: UserRegs
public async Task<IActionResult> Index()
{
     return _context.UserRegs != null ?
                View(await _context.UserRegs.ToListAsync()) :
                Problem("Entity set 'ProjectDBContext.UserRegs'  is null.");
}
// GET: UserRegs/Details/5
public async Task<IActionResult> Details(string id)
{
   if (id == null || _context.UserRegs == null)
   {
      return NotFound();
   }
   var userReg = await _context.UserRegs
      .FirstOrDefaultAsync(m => m.UserId == id);
   if (userReg == null)
   {
      return NotFound();
   }
   return View(userReg);
}
// GET: UserRegs/Create
public IActionResult Create()
{
   return View();
```

```
    }
    // POST: UserRegs/Create
    // To protect from overposting attacks, enable the specific properties you want to bind to.
    // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult>
Create([Bind("UserId,Password,FullName,Email,Joined,Active")] UserReg userReg)
    {
        if (ModelState.IsValid)
        {
            _context.Add(userReg);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }
        return View(userReg);
    }
    // GET: UserRegs/Edit/5
    public async Task<IActionResult> Edit(string id)
    {
        if (id == null || _context.UserRegs == null)
        {
            return NotFound();
        }
        var userReg = await _context.UserRegs.FindAsync(id);
        if (userReg == null)
```

```
        {
            return NotFound();
        }
        return View(userReg);
    }
    // POST: UserRegs/Edit/5
    // To protect from overposting attacks, enable the specific properties you want to bind to.
    // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Edit(string id,
[Bind("UserId,Password,FullName,Email,Joined,Active")] UserReg userReg)
    {
        if (id != userReg.UserId)
        {
            return NotFound();
        }
        if (ModelState.IsValid)
        {
            try
            {
                _context.Update(userReg);
                await _context.SaveChangesAsync();
            }
            catch (DbUpdateConcurrencyException)
            {
```

```
            if (!UserRegExists(userReg.UserId))
            {
                return NotFound();
            }
            else
            {
                throw;
            } }
        return RedirectToAction(nameof(Index));
    }
    return View(userReg);
}
// GET: UserRegs/Delete/5
public async Task<IActionResult> Delete(string id)
{
    if (id == null || _context.UserRegs == null)
    {
        return NotFound();
    }
    var userReg = await _context.UserRegs
        .FirstOrDefaultAsync(m => m.UserId == id);
    if (userReg == null)
    {
        return NotFound();
    }
    return View(userReg);
```

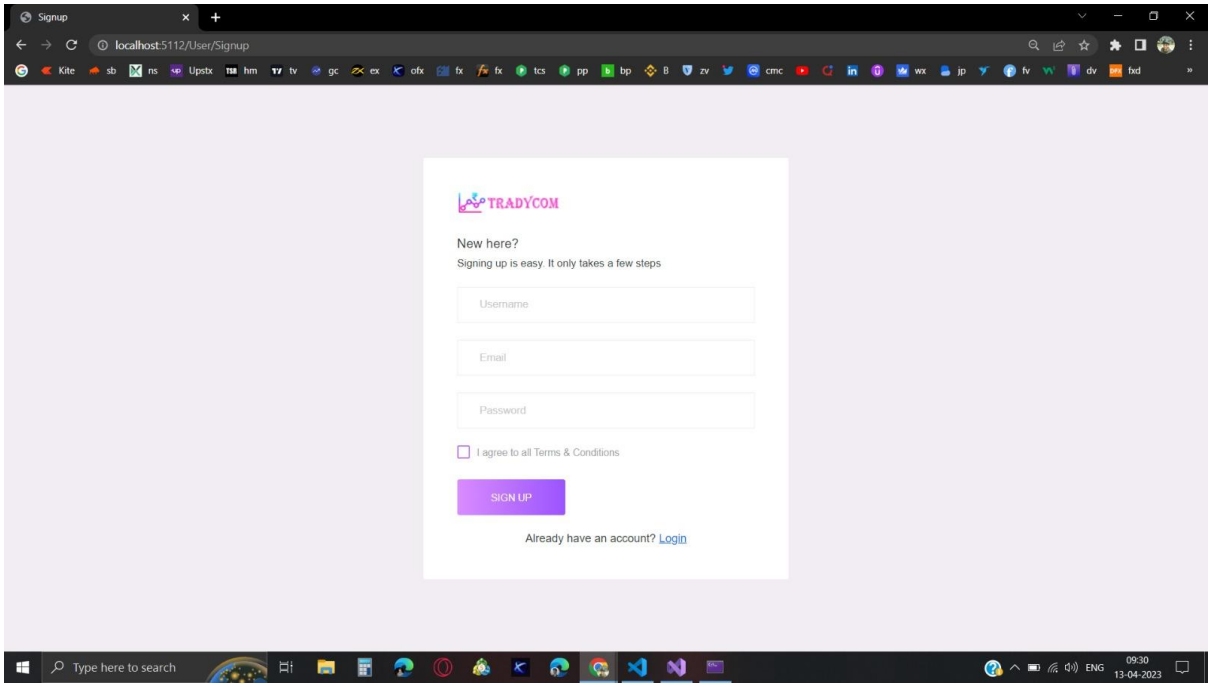
```
}
// POST: UserRegs/Delete/5
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> DeleteConfirmed(string id)
{
   if (_context.UserRegs == null)
   {
      return Problem("Entity set 'ProjectDBContext.UserRegs'  is null.");
   }
   var userReg = await _context.UserRegs.FindAsync(id);
   if (userReg != null)
   {
      _context.UserRegs.Remove(userReg);
   }
            await _context.SaveChangesAsync();
   return RedirectToAction(nameof(Index));
}
private bool UserRegExists(string id)
{
  return (_context.UserRegs?.Any(e => e.UserId == id)).GetValueOrDefault();
}  }}
```

```
using TradingJournal.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
namespace TradingJournal.Controllers
{
    public class JournalController : Controller
    {
    private readonly ProjectDBContext projectDBContext;
    public JournalController(ProjectDBContext projectDBContext)
    {
    this.projectDBContext = projectDBContext;
    }
    [HttpGet]
            public IActionResult Add()
            {
                return View();
            }
    public Task<IActionResult> Add(AddJournalModel addJournal)
    {
    return Add(addJournal, projectDBContext);
    }
    [HttpPost]
            public async Task<IActionResult> Add(AddJournalModel addJournal,
ProjectDBContext projectDBContext)
```

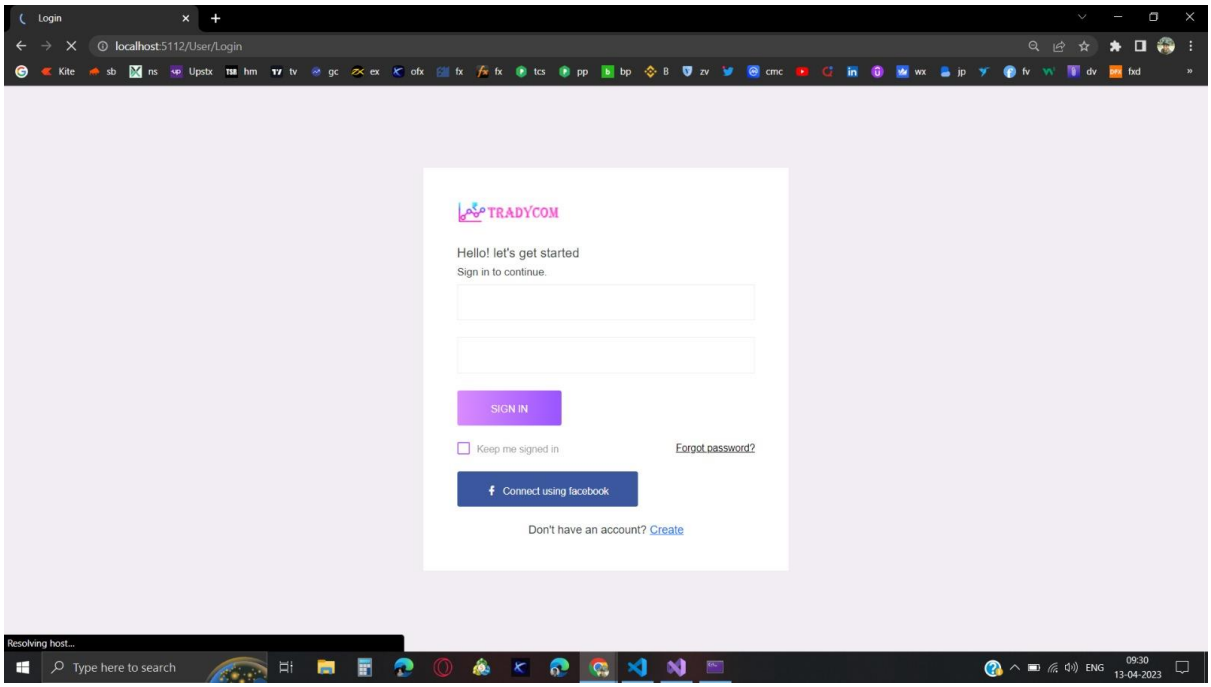
```
{
        var Journals = new Journal()
        {
                JournalId = addJournal.JournalId,
                SymbolName = addJournal.SymbolName,
                Created = addJournal.Created,
                EntryPrice = addJournal.EntryPrice,
                ClosePrice = addJournal.ClosePrice,
                Quantity = addJournal.Quantity,
                Profit = addJournal.Profit,
                Loss = addJournal.Loss,
                TradeJournal = addJournal.TradeJournal
        };
        await projectDBContext.Journals.AddAsync(Journals);
        await projectDBContext.SaveChangesAsync();
        return RedirectToAction("Add");
}
//public async Task<IActionResult> view()
//{
//        var journals = await projectDBContext.Journals.ToListAsync();
//        return View(journals);}
public IActionResult view()
{ }
```

APPENDIX 2

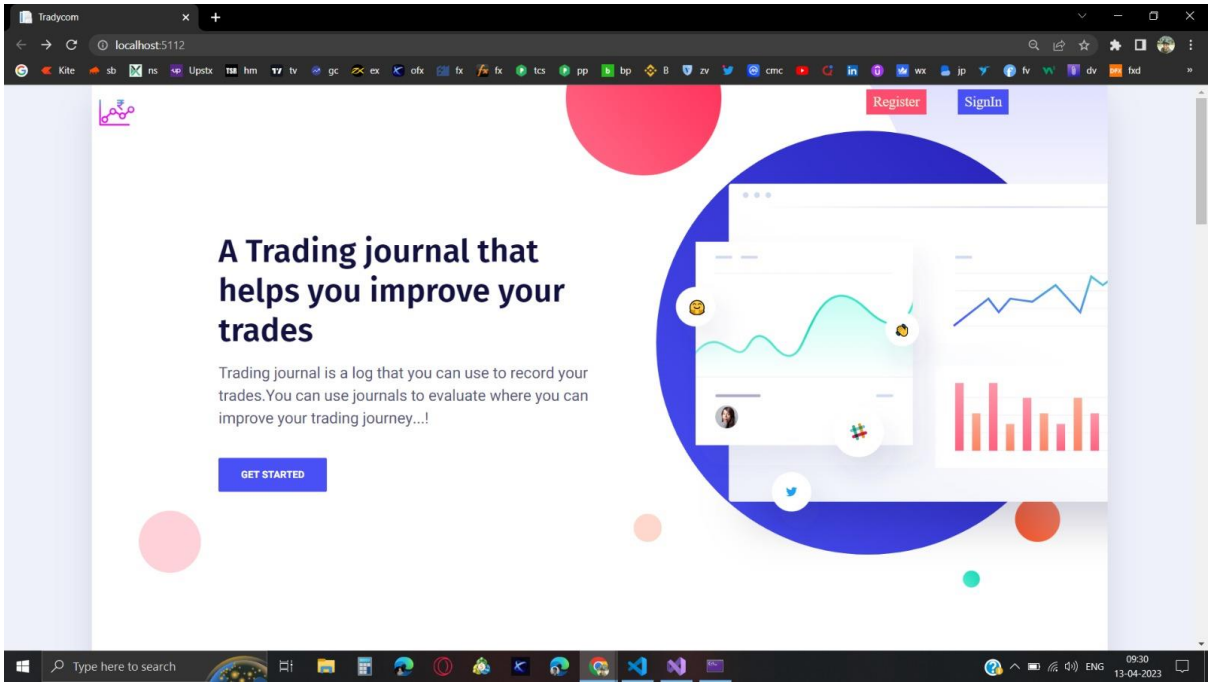
SCREENSHOTS



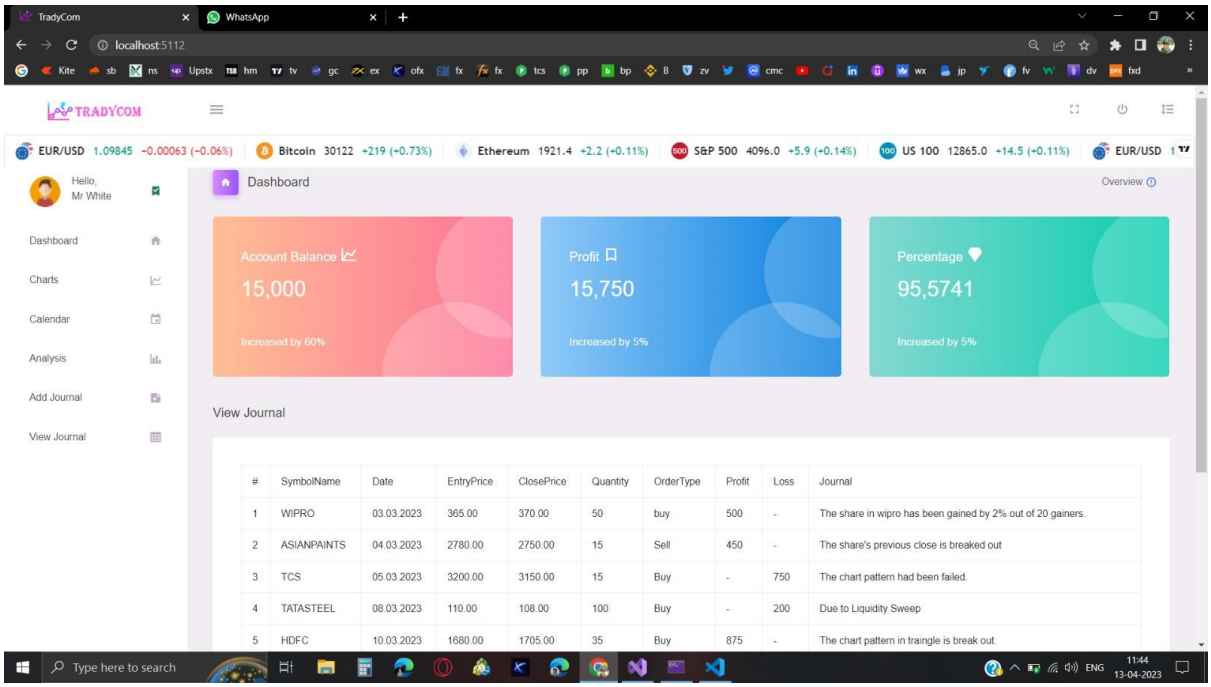
User Registration



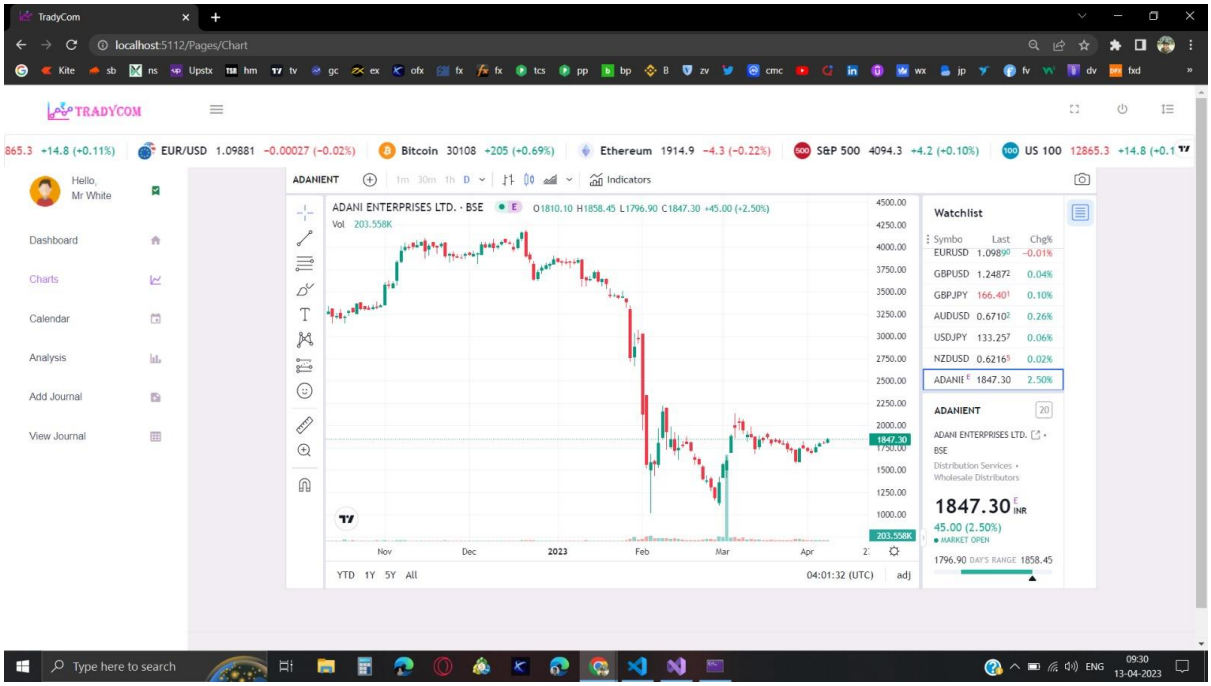
User Login



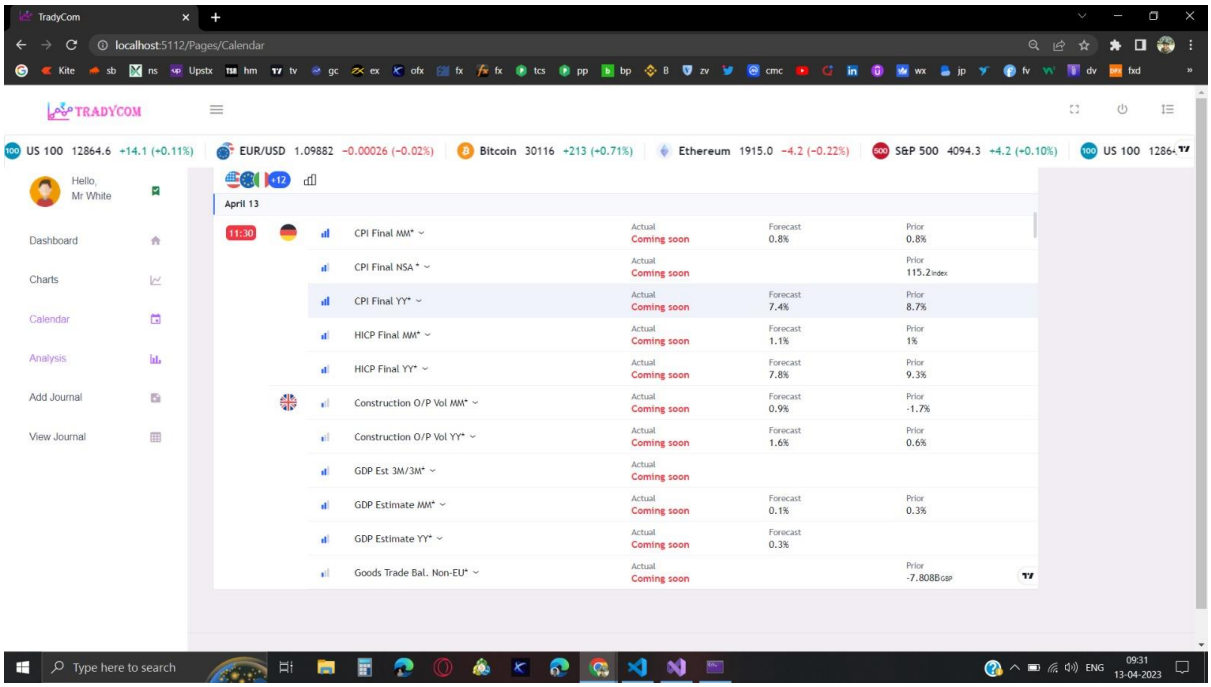
Landing Page



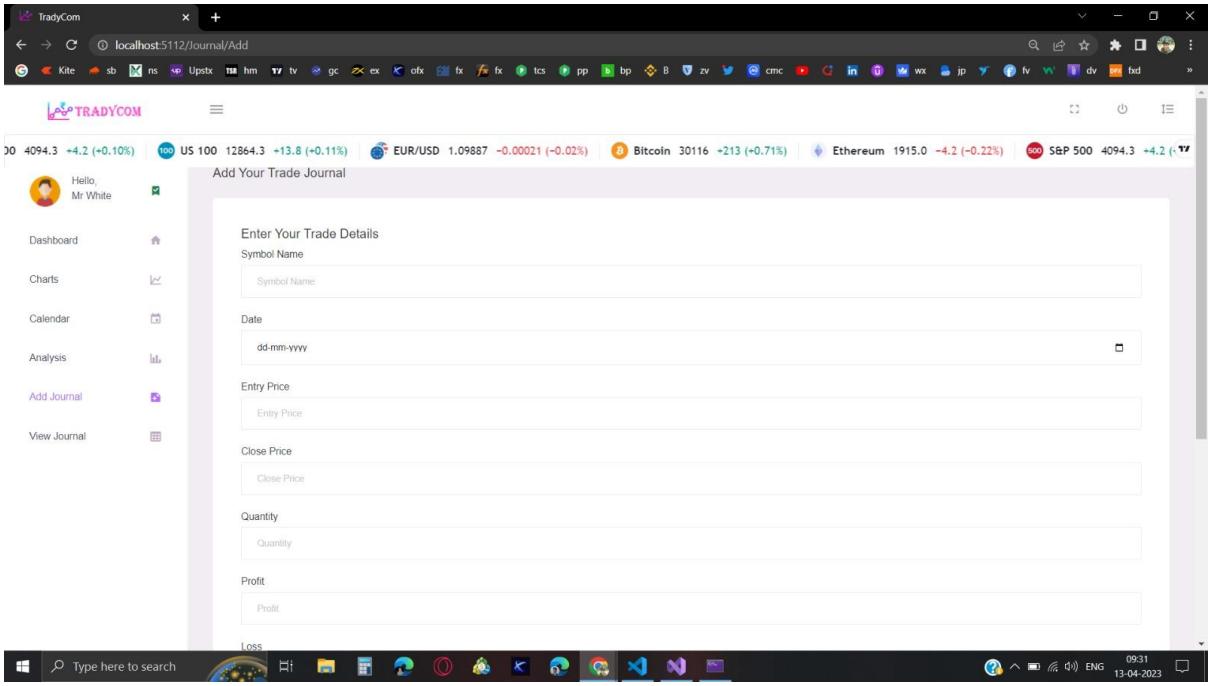
Dashboard



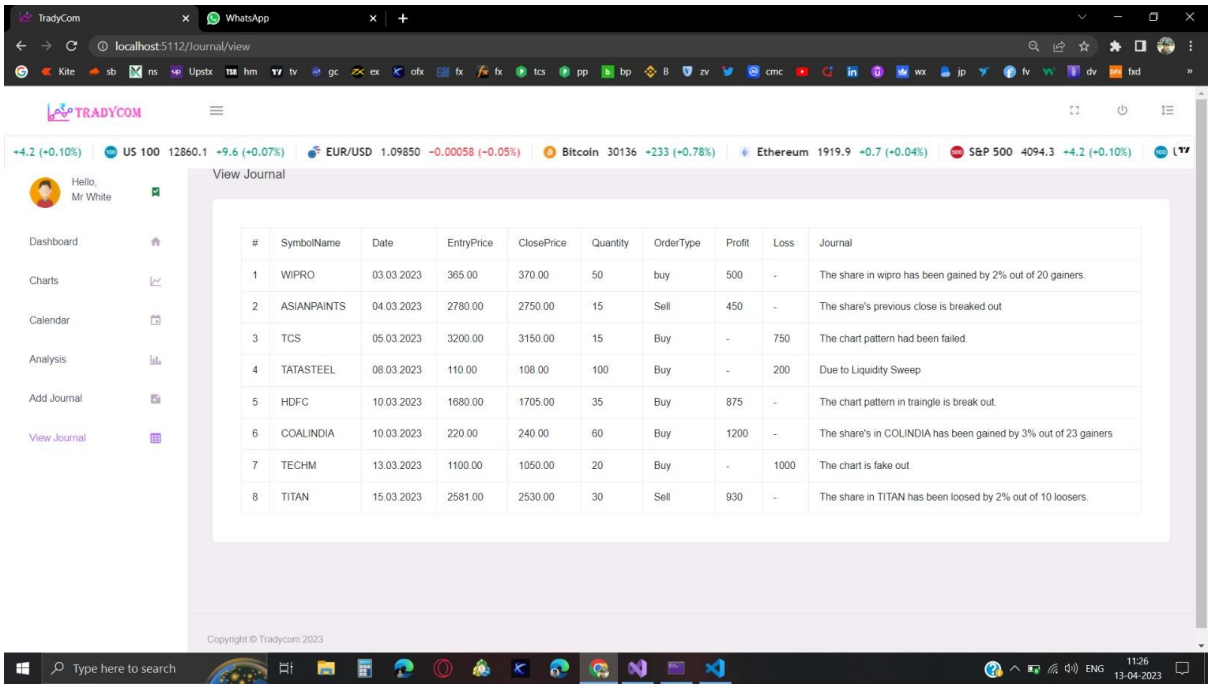
Trading Chart



Economic Calendar



Journal Form



View journal

APPENDIX 3

REFERENCES

1. https://ieeexplore.ieee.org/document/9310973
2. Atip Keeratipish, “Techniques for Finding Good Stocks and Building Stock Portfolio forSalary Man,” Available from:https://en.wikipedia.org/wiki/Fundamental_analysis.
3. "SET," The Stock Exchange of Thailand, Available from: https://www.set.or.th/set/mainpage.do?language=en&country=US.
4. "Settrade.com," Settrade.com Co., Ltd., Available from: https://www.settrade.com/.
5. "Data visualization," Wikipedia, Available from: https://en.wikipedia.org/wiki/Data_visualization.
6. "Chart Type Guide," SAP Analytics Cloud, Available from: https://www.sapanalytics.cloud/resources-chart-type-guide/.
7. "SET," The Stock Exchange of Thailand, Available from: https://www.set.or.th/set/mainpage.do?language=en&country=US.
8. "Settrade.com," Settrade.com Co., Ltd., Available from: https://www.settrade.com/.
9. "JITTA," Jitta Dot Com Co., Ltd., Available from: https://www.jitta.com/en.
10. https://in.investing.com/
11. https://www.moneycontrol.com/stocksmarketsindia/
12. https://in.tradingview.com/chart/?symbol=NSE%3ANIFTY
13. https://www.myfxbook.com/
14. https://zerodha.com/varsity/chapter/single-candlestick-patterns-part-1/
15. https://coinmarketcap.com/new/
16. https://www.forexfactory.com/
17. https://web.sensibull.com/home