**1.INTRODUCTION**

* 1. **Problem Definition:**

The "Stocks and Investment Management System" project seeks to develop an intuitive software solution for facilitating stock trading and financial management. Leveraging VB.NET and MS SQL Server, it provides users with features for buying and selling stocks, managing budgets, tracking expenses, and accessing financial news. The system aims to empower users with enhanced financial awareness, budget control, and data-driven decision-making capabilities. Key objectives include intuitive interface design, accurate transaction tracking, customizable categorization, budget monitoring, and insightful analysis. Ultimately, the project aims to streamline stock trading processes and promote financial responsibility among users.

**1.2 Scope of the project:**

* Stock Trading: Enable users to buy and sell stocks, providing access to a variety of stocks categorized by sector, allowing for informed investment decisions.
* Financial News and Analysis: Integrate financial news and analysis features to keep users informed about market trends, stock performance, and investment opportunities.
* User Authentication and Security: Implement user authentication mechanisms to ensure secure access to personal financial data, protecting user privacy and confidentiality.
* Reporting and Analytics: Include reporting and analytics tools to analyze stock performance, investment returns, and portfolio diversification, enabling users to make data-driven investment decisions.

**1.3 Modules of the Project:**

* User Interface Development: Design and develop a user-friendly interface using VB.NET, focusing on ease of use and intuitive navigation.
* Budget Allocation: Develop modules to assist users in allocating their income into expenses, investments, and savings, providing visual representations for better financial planning.
* Expense Tracking: Implement features for users to track expenses related to stock trading, including transaction fees, brokerage charges, and taxes.
* Admin Panel: Create an admin panel for administrators to manage stocks, users, transactions, and system configurations, ensuring smooth operation and maintenance.
* User Authentication and Registration: Develop modules for user authentication and registration, ensuring secure access to the system and user data.

**2. System Configuration**

**2.1 Hardware Requirements:**

Desktop/Laptop: Laptop

Processor: Intel I7 core 1200H

Ram: 16GB

System Type: 63 BIT Operating System

SSD: 1TB

**2.2 Software Requirements:**

Operating System: Windows 11

Programming Language: VB.NET

Frontend: Visual Studio 2022

Data Base: Microsoft SQL Server 2022

Documentation: MS Word

**3. System Study**

**3.1 Existing System:**

Online Trading Platforms: Existing online trading platforms like E\*TRADE, TD Ameritrade, and Robinhood allow users to buy and sell stocks, ETFs, and other securities. They provide features for portfolio management, real-time market data, and trading tools to facilitate investment decisions.

Financial News Websites: Websites like Bloomberg, CNBC, and Yahoo Finance offer financial news, market analysis, and investment insights to help users stay informed about market trends, stock performance, and economic indicators.

Portfolio Management Tools: Portfolio management tools like Morningstar and Personal Capital allow users to track and analyze their investment portfolios. They provide features for asset allocation, performance tracking, and investment goal setting.

**Problems in Existing System:**

* Complexity: Existing stock trading and investment management platforms may be complex for novice users, requiring a steep learning curve to understand the features and functionalities. This complexity can deter new investors and hinder their ability to navigate the platform effectively.
* Limited Customization: Some stock trading platforms offer limited customization options, restricting users from personalizing their trading experience according to their preferences and trading strategies. This lack of flexibility may lead to suboptimal trading experiences and hinder users from fully utilizing the platform's capabilities.
* Dependency on Manual Input: Many existing investment management tools rely on manual input of data, requiring users to manually enter trades, portfolio updates, and other investment-related information. This manual process can be time-consuming and prone to errors, leading to discrepancies in portfolio tracking and performance analysis.
* Security Concerns: Investment management platforms that store sensitive financial data may raise concerns about security and data privacy. Users may be apprehensive about sharing their personal and financial information on these platforms, especially in light of increasing cybersecurity threats and data breaches.
* Lack of Integrated Portfolio Management: Some investment platforms lack integrated portfolio management features, requiring users to use multiple tools and platforms to track and manage their investments effectively. This fragmented approach can lead to inefficiency

**3.2 Feasibility Study:**

The feasibility study of the Stocks and Investment Management System project involves evaluating its workability, impact on users, and efficient use of resources. The system enables users to register, log in, and access various features such as budget allocation, expense tracking, reports, financial goals, and customization options. Users can manage their budgets, expenses, categories, and financial goals, allowing for flexibility and control over their financial data.

* Technical Feasibility: The project utilizes VB.NET and MS SQL Server, commonly used technologies for software development. These technologies offer robust capabilities for building scalable and secure applications, ensuring technical feasibility.
* Economic Feasibility: The project's economic feasibility involves assessing the cost-effectiveness of development and implementation compared to potential benefits. By leveraging existing technologies and resources, the project aims to achieve cost efficiency and provide value to users.

Overall, the feasibility study concludes that the Stocks and Investment Management System project is technically, economically, operationally, and legally feasible. It has the potential to provide users with effective tools for managing their investments, tracking expenses, and achieving financial goals, thereby adding value to the organization and its stakeholders.

**Schedule:**

Time duration of this project requires 40 days covering 5 days for initiation phase, 10 days for definition phase, 3 days for Design phase, 10 days for implementation phase, 2 days for Testing phase and 10 days for documentation.

**Operational:**

* User Registration: The project simplifies the user registration process by providing clear fields for username and password entry. This streamlines the onboarding process and ensures that users can quickly access the system.
* Expense Tracking: The system facilitates expense tracking by allowing users to easily input, categorize, and view their expenses. Users can record expenses with essential details such as date, amount, category, and description, enabling comprehensive expense management.
* Budget Management: The project enhances budget management by providing a straightforward interface for setting budget amounts for different expense categories. Users can allocate budget amounts according to their financial priorities, promoting effective budget planning and control.
* Financial Goal Tracking: The system improves financial goal tracking by enabling users to define goals, such as saving targets, and monitor their progress within the system. Users can set specific financial objectives and track their achievements over time, fostering financial awareness and accountability.
* Expense-to-Goal Mapping: The project improves efficiency in expense-to-goal mapping, allowing users to link specific expenses to their financial goals for better financial planning. By associating expenses with relevant goals, users can prioritize their spending and allocate resources more effectively.

**2.3 Proposed System:**

The Proposed System for the Stocks and Investment Management System project aims to provide a comprehensive solution for managing investments, expenses, and financial goals.

**Advantages**

**Simplified Investment Management:**

Users can easily buy and sell stocks, access real-time market data, and track their investment portfolio performance within the platform, simplifying the process of managing investments.

**Comprehensive Financial Tracking:**

The system offers comprehensive tools for tracking expenses, investments, and financial goals in one centralized platform, providing users with a holistic view of their financial health.

**Enhanced Financial Planning:**

Users can set budgets for different expense categories, monitor their spending, and receive alerts when approaching budget limits, empowering them to make informed financial decisions and improve budget control.

**Goal-oriented Investing:**

The project allows users to set and track financial goals such as saving targets or retirement planning, enabling them to align their investment strategies with their long-term financial objectives.

**4. System Design**

In the design phase the architecture is established. This phase starts with the requirement document delivered by the requirement phase and maps the requirements into an architecture.

The architecture defines the components, their interfaces, and behaviors. The deliverable design document is the architecture. The design document describes a plan to implement the requirements. This phase represents the "how" phase. Details on computer programming languages and environments, machines, packages, and application architecture. distributed architecture layering, memory size, platform, algorithms, data structures, global type definitions, interfaces, and other engineering details are established. The design may include the usage of existing components.

System design defines the architecture components, modules, interfaces, and data for a system to satisfy specified requirements. Systems design is the application of systems theory to product development. There is some overlap with the disciplines of system analysis, systems architecture, and systems engineering

If the broader topic of product development "blends the perspective of marketing. design, and manufacturing into a single approach to product development," then design is the act of taking the marketing information and creating the design of the product to be manufactured. Systems design is therefore the process of defining and developing systems to satisfy specified requirements of the user.

Object-oriented analysis and design methods are becoming the most widely used methods for computer systems design. The UML. has become the standard language in object-oriented analysis and design. It is widely used for modeling software systems and is increasingly used for designing non-software systems and organizations.

**Architectural design:**

The architectural design of a system emphasizes the design of the system architecture that describes the structure, behavior, and more views of that system and analysis.

**Logical design:**

The logical design of a system pertains to an abstract representation of the data flows, inputs, and outputs of the system. This is often conducted via modeling, using an over-abstract (and sometimes graphical) model of the actual system. In the context of systems, designs are included. Logical design includes entity-relationship diagrams (ER diagrams).

**Physical design:**

The physical design relates to the actual input and output processes of the system. This is explained in terms of how data is input into a system, how it is verified/authenticated, how it is processed, and how it is displayed. In physical design, the following requirements about the system are decided.

* Input requirements,
* Output requirements,
* Storage requirements,
* Processing requirements,
* System control, and backup or recovery.

Put another way, the physical portion of system design can generally be broken down into three sub-tasks:

* User Interface Design.
* Data Design
* Process Design.

User Interface Design is concerned with how users add information to the system and with how the system presents information back to them. Data Design is concerned with how the data is represented and stored within the system. Finally, Process Design is concerned with how data moves through the system, and with how and where it is validated. secured and/or transformed as it flows into, through, and out of the system. At the end of the system design phase, documentation describing the three sub-tasks is produced and made available for use in the next phase.

Physical design, in this context, does not refer to the tangible physical design of an information system. To use an analogy, a personal computer's physical design involves input via a keyboard, processing within the CPU, and output via a monitor, printer, etc. It would not concern the actual layout of the tangible hardware, which for a PC would be a monitor, CPU, motherboard, hard drive, modems, video/graphics cards, USB slots, etc. It involves a detailed design of a user and a product database structure processor and a control processor. The H/S personal specification is developed for the proposed system.

**4.1 ER Diagram:**

An entity relationship diagram (ERD) shows the relationships of entity sets stored in a database. An entity in this context is a component of data. In other words, ER diagrams illustrate the logical structure of databases.

At first glance, an entity relationship diagram looks very much like a flowchart. It is the specialized symbols, and the meanings of those symbols, that make it unique. Because this ER tutorial focuses on beginners below are some tips that will help you build effective ER diagrams:

Identify all the relevant entities in a given system and determine the relationships among these entities.

 An entity should appear only once in a particular diagram.

Provide a precise and appropriate name for each entity, attribute, and relationship in the diagram. Terms that are simple and familiar always beat vague, technical-sounding words. In naming entities, remember to use singular nouns. However, adjectives may be used to distinguish entities belonging to the same class (part-time employee and full-time employee, for example). Meanwhile, attribute names must be meaningful, unique, system-independent, and easily understandable man and James Martin have added some slight refinements to the basic ERD principles.

**Structure of an Entity Relationship Diagram with Common ERD Notations:**

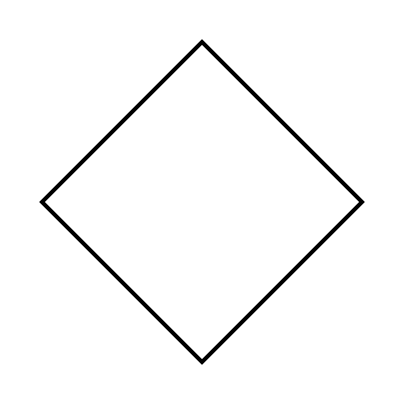
An entity relationship diagram is a means of visualizing how the information a system produces is related. There are five main components of an ERD:

Entities:

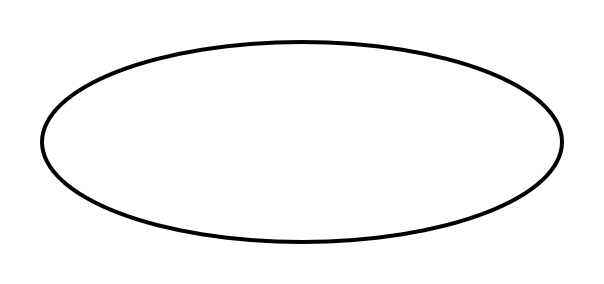


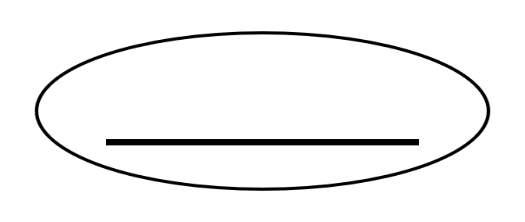
Entities, which are represented by rectangles. An entity is an object or concept you want to store information about. A weak entity is an entity that must defined by a foreign key relationship with another entity as it cannot be uniquely identified by its attributes alone.

Actions:



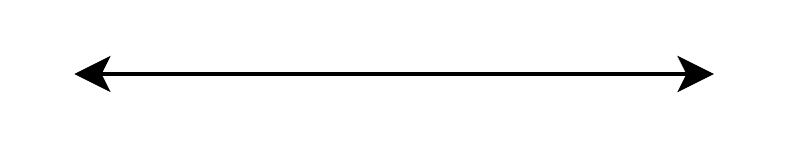
Actions, which are represented by diamond shapes, show how two entities share information in the database. In some cases, entities can be self-linked. For example, employees can supervise other employees.

Attributes:  


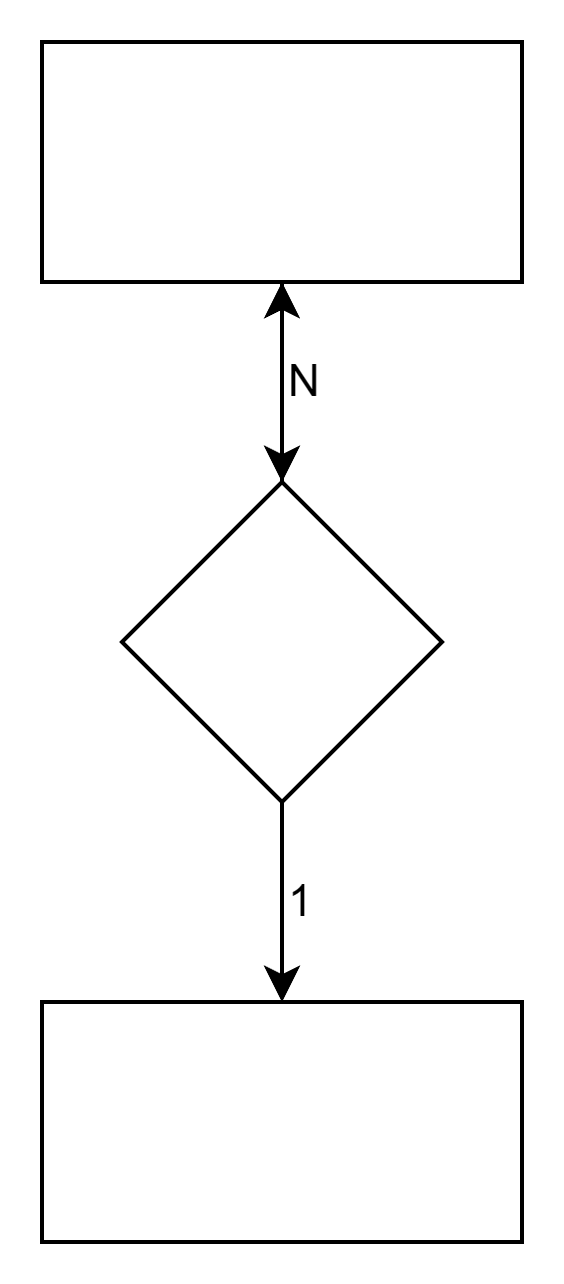


Attributes are represented by ovals. A key attribute is the unique. they were distinguishing characteristics of the entity. The first diagram shows only the attributes of an entity whereas the second diagram shows the attribute with the primary key. For example. an employee's social security number might be the employee's key attribute.

Connecting lines:

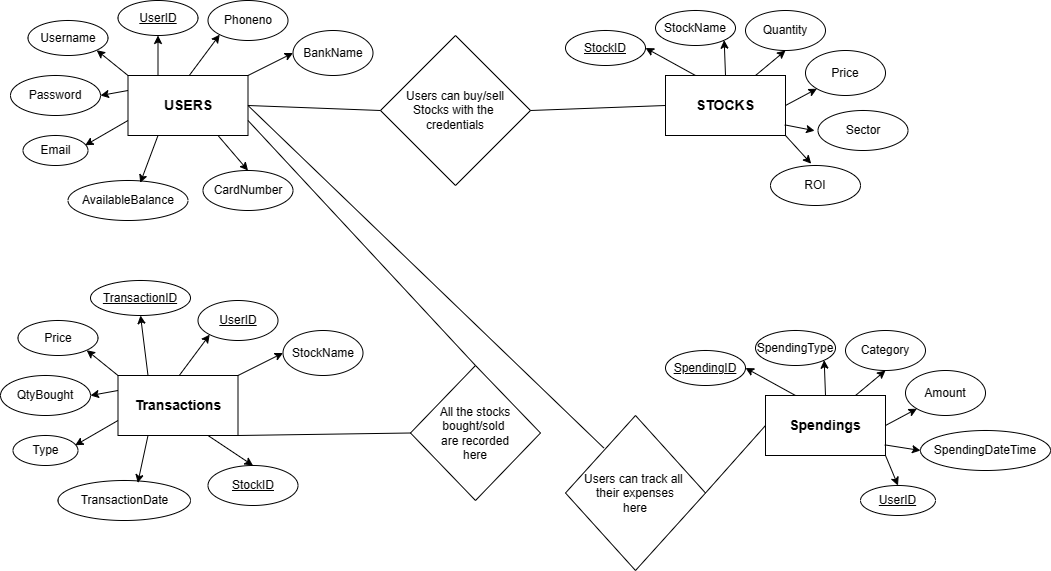


Connecting lines, are solid lines that connect attributes to show the relationships of entities in the ER Diagram.

Cardinality: 

Cardinality specifies how many instances of an entity relate to one instance of another entity. Cardinality is also closely linked to cardinality. While cardinality specifies the occurrences of a relationship, ordinality describes the relationship as either mandatory or optional. In other words, cardinality specifies the maximum number of relationships and ordinality specifies the absolute minimum number of relationships.

**ER diagram for Stocks & Investments Management System :**



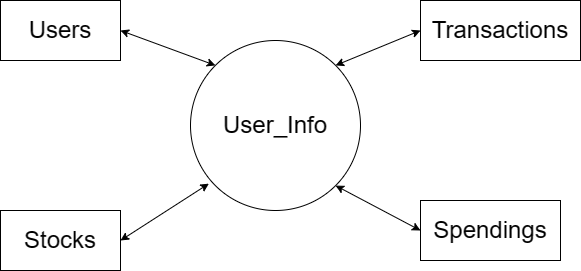
**4.2 Data Flow Diagram (DFD):**

**Introduction:**

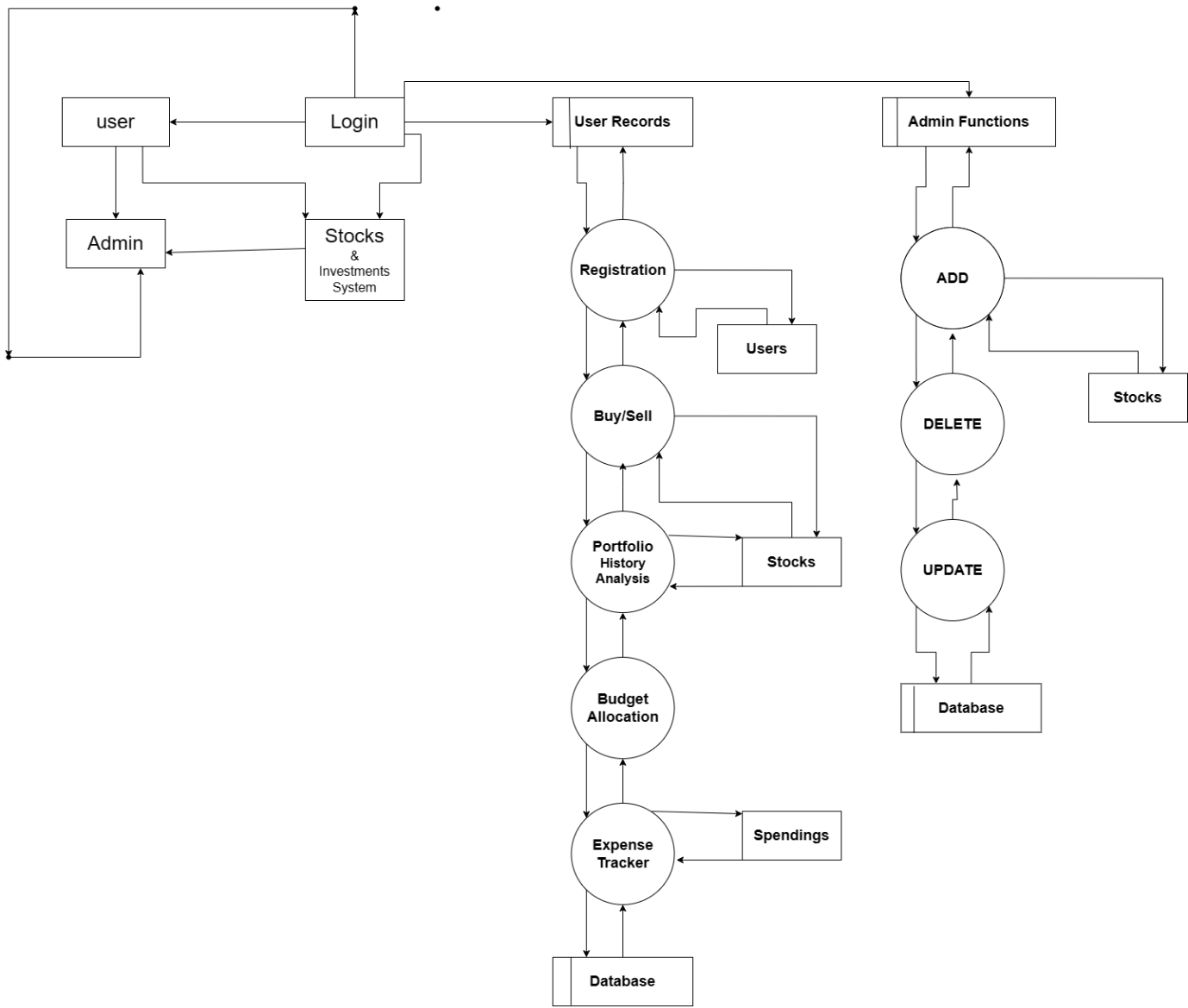
The Data Flow Diagrams (DFDs) are used for structure analysis and design. DFDs show the flow of data from external entities into the system. DFDs also show how the data moves and is transformed from one process to another, as well as its logical storage.

A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system, modeling its process aspects. A DFD is often used as a preliminary step to create an overview of the system, which can later be elaborated. DFDS can also be used for the visualization of data processing (structured design). A DFD shows what kind of information will be input to and output from the system, where the data will come from and go to, and where the data will be stored. It does not show information about the timing of the process or information about whether processes will operate in sequence or in parallel (which is shown on a flowchart).

**Level 0 DFD :**

****

**Level 1 DFD :**

****

**4.3 GANTT CHART:**

A Gantt chart is a type of bar chart, devised by Henry Gantt in the 1910s, that illustrates a project schedule. Gantt charts illustrate the start and finish dates of the terminal elements and summary elements of a project Terminal elements and summary elements comprise the work breakdown structure of the project, Modern Gantt charts also show the dependency (i.e., precedence network) relationships between activities. Gantt charts can be used to show current schedule status using percent-complete shadings and a vertical "TODAY" line as shown here.

**GANTT CHART BENEFITS:**

**Visual Representation:**

Gantt charts provide a visual representation of project tasks, timelines, and dependencies, allowing stakeholders to quickly understand the project's progress and status.

**Task Management:**

Gantt charts break down complex projects into individual tasks, making it easier to manage and track each task's duration, start date, and completion date.

**Resource Allocation:**

By depicting task dependencies and resource availability, Gantt charts help project managers allocate resources effectively, ensuring that tasks are completed on time and within budget.

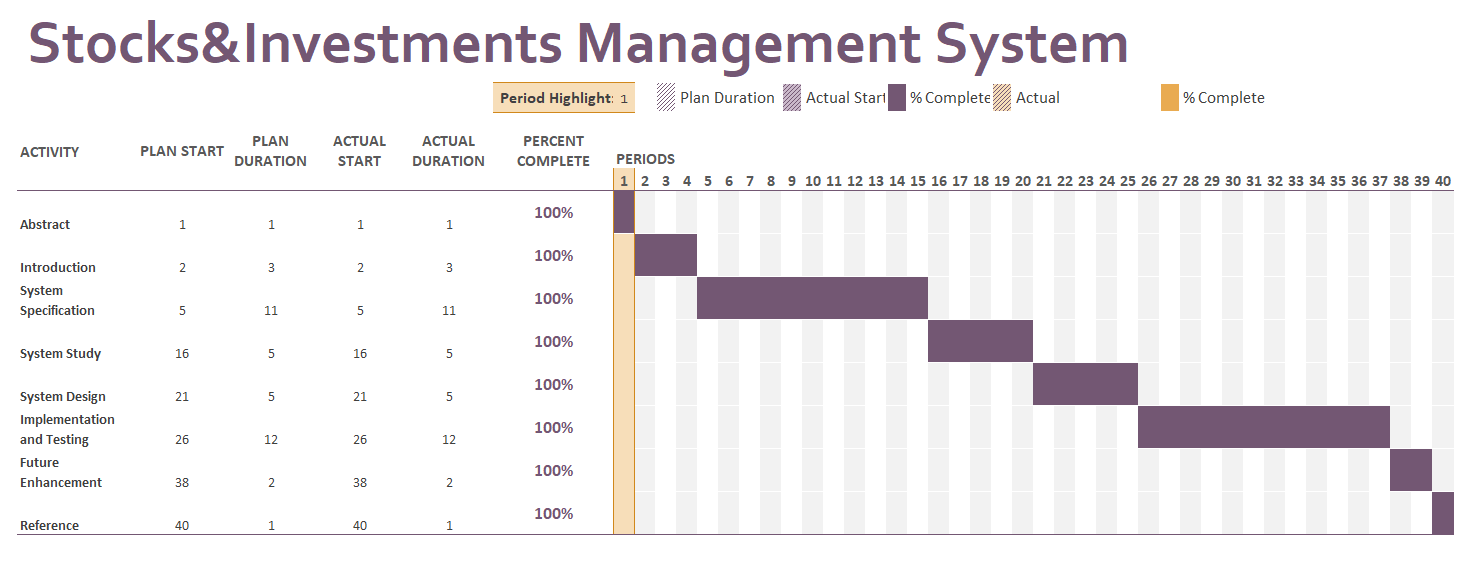
**Timeline Management:**

Gantt charts display project timelines in a clear and organized manner, enabling project managers to identify critical paths, potential delays, and overlapping tasks.

**Communication Tool:**

Gantt charts serve as a communication tool between project team members, stakeholders, and clients, facilitating discussions about project progress, deadlines, and priorities.

**Gantt Chart of Stocks & Investments Management System :**



**5. Implementation and Testing**

**5.1 OVERVIEW OF FRONT END:**

**MICROSOFT VISUAL STUDIO 2022:**

Microsoft Visual Studio 2022 is an integrated development environment (IDE) from Microsoft. It is used to develop computer programs for Microsoft Windows, as well as web sites, web apps, web services and mobile apps. Visual Studio uses Microsoft software development platforms such as Windows API, Windows Forms, Windows Presentation Foundation, Windows Store and Microsoft Silverlight. It can produce both native code and managed code.

Visual Studio supports different programming languages and allows the code editor and debugger to support nearly any programming language, provided a language- specific service exists. Built-in languages include C, C++, Visual C++ and VB.NET. Support for other languages such as Python, Ruby, Node.js, and M among others is available via language services installed separately. It also supports XML/XSLT. HTML/XHTML, JavaScript and CSS, Java (and J#) were supported in the past.

Microsoft provides a free version of Visual Studio called the Community edition that supports plugins and is available at no cost for all users. Support for programming languages is added by using a specific VS Package called a Language Service. A language service defines various interfaces which the VS Package implementation can implement to add support for various functionalities. Functionalities that can be added this way include syntax coloring, statement completion, brace matching, parameter information tooltips, member lists and error markers for background compilation.

If the interface is implemented, the functionality will be available for the language. Language services are implemented on a per-language basis. The implementations can reuse code from the parser or the compiler for the language. Language services can be implemented either in native code or managed code. For native code, either the native COM interfaces or the Babel Framework can be used. For managed code, the MPF includes wrappers for writing managed language services.

**FEATURES:**

* Boolean Conditions
* Automatic Garbage Collection
* Standard Library
* Assembly Versioning
* Properties and Events
* Delegates and Events Management
* Easy-to-use Generics
* Indexers
* Conditional Compilation
* Simple Multithreading

**5.2 OVERVIEW OF BACK-END:**

**MICROSOFT SQL SERVER 2022:**

Microsoft SQL Server is a relational database management system developed by Microsoft. As a database server, it is a software product with the primary function of storing and retrieving data as requested by other software applications which may run either on the same computer or on another computer across a network (including the Internet).

Microsoft markets at least a dozen different editions of Microsoft SQL Server, aimed at different audiences and for workloads ranging from small single-machine applications to large Internet-facing applications with many concurrent users. The protocol layer implements the external interface to SQL Server. All operations that can be invoked on SQL Server are communicated to it via a Microsoft-defined format, called Tabular Data Stream (TDS). TDS is an application layer protocol, used to transfer data between a database server and a client.

Initially designed and developed by Sybase Inc. for their Sybase SQL Server relational database engine in 1984, and later by Microsoft in Microsoft SQL Server, TDS packets can be encased in other physical transport dependent protocols, including TCP/IP, named pipes, and shared memory. Consequently, access to SQL Server is available over these protocols. In addition, the SQL Server API is also exposed over web services.

A Relational Data base Management System (RDBMS) is software that:

* Enables you to implement a database with tables, columns and indexes.
* Guarantees the Referential Integrity between rows of various tables.
* Updates the indexes automatically.
* Interprets an SQL query and combines information from various tables

**Advantages of Microsoft SQL Server:**

Robust Performance: SQL Server is known for its high performance, scalability, and reliability, making it suitable for handling large volumes of data and demanding workloads.

Integration with Microsoft Products: SQL Server seamlessly integrates with other Microsoft products and technologies such as Windows Server, Visual Studio, and .NET framework, providing a comprehensive development and deployment environment.

Advanced Security Features: SQL Server offers advanced security features such as encryption, authentication mechanisms, role-based access control, and auditing capabilities to protect sensitive data from unauthorized access and breaches.

Comprehensive Business Intelligence (BI) Tools: SQL Server includes powerful BI tools such as Reporting Services (SSRS), Analysis Services (SSAS), and Integration Services (SSIS) for data analysis, reporting, and ETL (Extract, Transform, Load) processes.

Easy Management and Administration: SQL Server Management Studio (SSMS) provides a user-friendly interface for managing databases, configuring settings, monitoring performance, and troubleshooting issues, simplifying database administration tasks.

**5.3 ABOUT THE PLATFORM:**

Windows is a series of Operating Systems developed by Microsoft. Each version of Windows includes a Graphical User Interface, with a desktop that allows users to view files and folders in Windows. For the past two decades, Windows has been the most widely used operating system for personal computers PCs. Microsoft Windows is designed for both home computing and professional purposes. Past versions of Windows home editions include Windows 3.0 (1990), Windows 3.1 (1992), Windows 95 (1995), Windows 98 (1998), Windows Me (2000), Windows XP (2001), and Windows Vista (2006), The current version, Windows 7, was released in 2009. The first business-oriented version of Windows, called Windows NT 3.1, was in 1993. This was followed by Windows 3.5, 4.0, and Windows 2000. When Microsoft released Windows XP in 2001, the company simply created different editions of the operating system for personal and business purposes. Windows Vista and Windows 7 have followed the same release strategy. Windows is designed to run on standard x86 hardware, such as Intel and AMD processors.

Therefore, it can be installed on multiple brands of hardware, such as Dell, HP, and Sony computers, as well as home-built PCs Windows 7 also includes several touchscreen features, that allow the operating system to run on certain tablets and computers with touchscreen displays, Microsoft's mobile operating system, Windows Phone, is designed specifically for smartphones and runs on several brands of phones, including HTC, Nokia, and Samsung.

**.NET FrameWork:**

NET Framework (pronounced as "dot net") is a software framework developed by Microsoft that runs primarily on Microsoft Windows. It includes a large class library named Framework Class Library (FCL) and provides language interoperability (each language can use code written in other languages) across several programming languages. Programs written for .NET Framework execute in a software environment (in contrast to a hardware environment) named Common Language Runtime (CLR), an application virtual machine that provides services such as security, memory management, and exception handling. As such, computer code written using NET Framework is called "managed code".

The framework is intended to be used by newest applications created for the Windows platform. Microsoft also produces an integrated development environment largely for .NET software called Visual Studio. .NET Framework began as proprietary software, although the firm worked to standardize the software stack almost immediately. even before its first release. Despite the standardization efforts, developers, mainly those in the free and open-source software communities, expressed their unease with the selected terms and the prospects of any free and open-source implementation, especially regarding software patents. Since then, Microsoft has changed NET development to more closely follow a contemporary model of a community developed software project, including issuing an update to its patent promising to address the concerns.

**5.4 Testing:**

Testing is a vital part of software development, and it is important to start it as carly as possible, and to make testing a part of the process of deciding requirements. To get the most useful perspective on your development project, it is worthwhile devoting some thought to the entire lifecycle including how feedback from users will influence the future of the application. The tools and techniques we've discussed in this book should help your team to be more responsive to changes without extra cost, despite the necessarily wide variety of different development processes. Nevertheless, new tools and process improvements should be adopted gradually, assessing the results after each step.

Testing is part of a lifecycle. The software development lifecycle is one in which you hear of a need, you write some code to fulfil it, and then you check to see whether you have pleased the stakeholders the users, owners, and other people who have an interest in what the software does. Hopefully they like it, but would also like some additions or changes, so you update or augment your code, and so the cycle continues Software development life cycle Testing is a proxy for the customer. You could conceivably do your testing by releasing it into the wild and waiting for the complaints and compliments to come back.

Tests represent requirements. Whether you write user stories on sticky notes on the wall, or use cases in a big thick document, your tests should be derived from and linked to those requirements. And as we've said, devising tests is a good vehicle for discussing the requirements.

We're not done till the tests pass. The only useful measure of completion is when tests have been performed successfully.

Those principles apply no matter how you develop your software.

**Software Testing Types:**

**Black box Testing** - Internal system design is not considered in this type of testing. Tests are based on requirements and functionality.

**White box Testing** - This testing is based on knowledge of the internal logic of an application's code. Also known as Glass box Testing. Internal software and code working should be known for this type of testing. Tests are based on coverage of code statements, branches, paths, conditions.

**Unit Testing** - Testing of individual software components or modules. Typically done by the programmer and not by testers, as it requires detailed knowledge of the internal program design and code, may require developing test driver modules or test harnesses.

**Incremental Integration Testing** - Bottom up approach for testing ie continuous testing of an application as new functionality is added: Application functionality and modules should be independent enough to test separately, done by programmers or by testers.

**Integration Testing** -Testing of integrated modules to verify combined functionality after integration. Modules are typically code modules, individual applications, client and server applications on a network, etc. This type of testing is especially relevant to client/server

and distributed systems.

**Functional Testing** - This type of testing ignores the internal parts and focus on the output is as per requirement or not. Black-box type testing geared to functional requirements of an application system testing.

**Application System Testing** - Entire system is tested as per the requirements, Black-box type testing that is based on overall requirements specifications, covers all combined parts of a system.

**Validation Testing:**

Validation Testing In software project management, software testing, and software engineering, verification and validation (V&V) is the process of checking that a software system meets specifications and that it fulfils its intended purpose. It may also be referred to as software quality control. It is normally the responsibility of software testers as part of the software development lifecycle. In simple terms, software verification is: "Assuming we should build X, does our software achieve its goals without any bugs or gaps?" On the other hand, software validation is: "Was X what we should have built? Does X meet the high level requirements?" Validation checks that the product design satisfies or fits the intended use (high-level checking), i.e., the software meets the user requirements. This is done through dynamic testing and other forms of review. Verification and validation are not the same thing, although they are often confused. Boehm succinctly expressed the difference between

• **Validation:** Are we building the right product?

• **Verification:** Are we building the product right? According to the Capability Maturity Model (CMMI-SW v1.1),

• **Software Verification:** The process of evaluating software to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase. [IEEE-STD-610]

• **Software Validation:** The process of evaluating software during or at the end of the development process to determine whether it satisfies specified requirements.

**Verification and Validation:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **SL**  **No.** | **Test CaseID** | **Test**  **Description** | **Steps**  **To**  **Execute** | **Test**  **Data** | **Expected**  **Result** | **Actual**  **Result** | **Status** |
| 01 | Test  CaseID 1 | Correct Username  And Password | Start-Enter the Username,  Password login data | Correct Username And Password | Login is successful | Login is successful | Pass |
| 02 | Test  CaseID 2 | Wrong Username  And Password | Start-Enter  The  Username,  Password  Login data | Wrong  Username  And  Password | Invalid  Username  And  Password | Invalid  Username  And  Password | Pass |
| 03 | Test  CaseID 3 | Buy Functionality | Navigate to stocks page and click the buy button enter the required credentials to buy stocks | Entering wrong StockID as input | Stock With (StockID) does not Found | Stock With (StockID) does not exist | Pass |
| 04 | Test  CaseID 4 | Sell Functionality | Navigate to stocks page and click the Sell button enter the required credentials to Sell stocks | Entering wrong StockID as input | Stock With (StockID) does not Found | Stock With (StockID) does not Found | Pass |
| 05 | Test  CaseID 5 | ADD , DELETE & UPDATE Functionality | Log-In to admin page and perform anyone operation | Entering wrong StockID as input to perform an operation | Stock With (StockID) does not Exist | Stock With (StockID) does not Exist | Pass |

**6. Conclusion And Future Enhancements**

**6.1 Conclusion:**

The "Stocks and Investments Management System" project presents a versatile platform for users to engage in stock trading and financial management. Through intuitive interfaces and robust functionalities, including stock listing, buying, and selling capabilities, budget allocation, expense tracking, and user portfolio management, the system fosters informed decision-making and financial responsibility. Utilizing VB.NET for frontend development and Microsoft SQL Server for backend support ensures a secure and efficient management of financial data. By facilitating seamless interaction between users and the stock market, the project contributes to enhancing financial literacy and promoting prudent investment practices.

**6.2 Future Enhancements:**

**Automated Stock Recommendations:** Integrate AI algorithms to provide personalized stock recommendations based on user preferences, risk tolerance, and market trends, enhancing the user's investment decisions.

**Portfolio Diversification Tools:** Develop tools to help users diversify their investment portfolios by analyzing asset allocation, sector exposure, and risk distribution, ensuring a balanced and resilient investment strategy.

**Real-time Market Data Integration:** Incorporate real-time stock market data feeds and news updates into the system to keep users informed about market fluctuations, company announcements, and industry developments, enabling timely decision-making.

**7. Reference**

www.youtube.com

[Stack Overflow - Where Developers Learn, Share, & Build Careers](https://stackoverflow.com/)

[www.wikipedia.com](http://www.wikipedia.com)

**8. Appendices A-Structure**

**Users Table:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Data Type** | **Allow Null** |
| UserID | Int | Not Null |
| Username | varchar(50) | Not Null |
| Password | varchar(50) | Not Null |
| Email | varchar(50) | Not Null |
| Phoneno | varchar(50) | Not Null |
| BankName | varchar(50) | Not Null |
| CardNumber | varchar(50) | Not Null |

**Stocks Table:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Data Type** | **Allow Null** |
| StockID | Int | Not Null |
| StockName | varchar(50) | Not Null |
| Quantity | Int | Not Null |
| Price | decimal(10,2) | Not Null |
| Sector | varchar(50) | Not Null |
| ROI | decimal(10,2) | Not Null |

**Transactions Table:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Data Type** | **Allow Null** |
| TransactionID | Int | Not Null |
| UserID | Int | Not Null |
| StockName | varchar(50) | Not Null |
| Price | decimal(10,2) | Not Null |
| QtyBought | Int | Not Null |
| Type | varchar(5) | Not Null |

**Spendings Table:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Data Type** | **Allow Null** |
| SpendingID | Int | Not Null |
| SpendingType | varchar(5) | Not Null |
| Category | varchar(20) | Not Null |
| Amount | decimal(10,2) | Not Null |
| SpendingDateTime | date | Not Null |
| UserID | Int | Not Null |

**9. Appendices B-Sample Source Code**

**Login:**

Imports System.Data.SqlClient

Imports System.Data

Imports System.Globalization

Public Class login\_page

    Public con As New SqlConnection("Data Source=LAPTOP-N6QJ5U6O\SQLEXPRESS01;Initial Catalog=user\_info;Integrated Security=True;Encrypt = False")

    Private Sub login\_page\_Load(sender As Object, e As EventArgs) Handles MyBase.Load

    End Sub

    Private Sub Label1\_Click(sender As Object, e As EventArgs) Handles Label1.Click

    End Sub

    Private Sub LinkLabel1\_LinkClicked(sender As Object, e As LinkLabelLinkClickedEventArgs) Handles LinkLabel1.LinkClicked

        Me.Hide()

        register\_page.Show()

    End Sub

    Private Sub Login\_btn\_Click(sender As Object, e As EventArgs) Handles Login\_btn.Click

        PerformLogin()

    End Sub

    Private Sub PerformLogin()

        Try

            If String.IsNullOrWhiteSpace(Username\_txt.Text) OrElse String.IsNullOrWhiteSpace(Password\_txt.Text) Then

                MessageBox.Show("Please enter username and password.")

                Return

            End If

            con.Open()

            Dim sql As String = "SELECT \* FROM Users WHERE Username = @Username AND Password = @Password"

            Using cmd As New SqlCommand(sql, con)

                cmd.Parameters.AddWithValue("@Username", Username\_txt.Text)

                cmd.Parameters.AddWithValue("@Password", Password\_txt.Text)

                Dim reader As SqlDataReader = cmd.ExecuteReader()

                If reader.HasRows Then

                    ' User is authenticated

                    MessageBox.Show("Login successful")

                    Me.Hide()

                    home\_page.Show()

                    home\_page.usrnme\_label.Text = Username\_txt.Text

                    stock\_page.usrnme2\_label.Text = Username\_txt.Text

                    portfolio\_page.usrnme\_label.Text = Username\_txt.Text

                    Username\_txt.Text = ""

                    Password\_txt.Text = ""

                Else

                    MessageBox.Show("Invalid username or password")

                End If

                reader.Close()

            End Using

        Catch ex As Exception

            MessageBox.Show("Error: " & ex.Message)

        Finally

            con.Close()

        End Try

    End Sub

    Private Sub CheckBox1\_CheckedChanged(sender As Object, e As EventArgs) Handles CheckBox1.CheckedChanged

        If CheckBox1.Checked Then

            Password\_txt.PasswordChar = ControlChars.NullChar

        Else

            Password\_txt.PasswordChar = "\*"

        End If

    End Sub

    Private Sub Password\_txt\_TextChanged(sender As Object, e As EventArgs) Handles Password\_txt.TextChanged

        Password\_txt.PasswordChar = "\*"

    End Sub

    Private Sub LinkLabel2\_LinkClicked(sender As Object, e As LinkLabelLinkClickedEventArgs) Handles LinkLabel2.LinkClicked

        admin\_login.Show()

        Me.Hide()

    End Sub

**Inventory Page(admin)**:

Imports System.Data.SqlClient

Public Class Inventory\_page

    Private Sub LoadStocks()

        Try

            Using connection As New SqlConnection("Data Source=LAPTOP-N6QJ5U6O\SQLEXPRESS01;Initial Catalog=user\_info;Integrated Security=True;Encrypt = False")

                connection.Open()

                Dim adapter As New SqlDataAdapter("SELECT \* FROM Stocks", connection)

                Dim table As New DataTable()

                adapter.Fill(table)

                Guna2DataGridView1.DataSource = table

            End Using

        Catch ex As Exception

            MessageBox.Show($"An error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

        End Try

    End Sub

    Private Sub Inventory\_page\_Load(sender As Object, e As EventArgs) Handles MyBase.Load

        Me.StocksTableAdapter.Fill(Me.User\_infoDataSet3.Stocks)

    End Sub

    Private Sub Guna2Panel1\_Paint(sender As Object, e As PaintEventArgs)

    End Sub

    Private Sub Guna2DataGridView1\_CellContentClick(sender As Object, e As DataGridViewCellEventArgs) Handles Guna2DataGridView1.CellContentClick

    End Sub

    Private Sub Panel2\_Paint(sender As Object, e As PaintEventArgs) Handles Panel2.Paint

    End Sub

    Private Sub addbtn\_Click(sender As Object, e As EventArgs) Handles addbtn.Click

        add\_stock.Show()

    End Sub

    Private Sub Guna2CircleButton1\_Click(sender As Object, e As EventArgs) Handles Guna2CircleButton1.Click

        LoadStocks()

    End Sub

    Private Sub deletebtn\_Click(sender As Object, e As EventArgs) Handles deletebtn.Click

        Dim stockID As String = InputBox("Enter the stock ID you want to delete:", "Delete Stock")

        If Not String.IsNullOrEmpty(stockID) Then

            Try

                Using connection As New SqlConnection("Data Source=LAPTOP-N6QJ5U6O\SQLEXPRESS01;Initial Catalog=user\_info;Integrated Security=True;Encrypt = False")

                    connection.Open()

                    Dim deleteCommand As New SqlCommand("DELETE FROM Stocks WHERE StockID = @StockID", connection)

                    deleteCommand.Parameters.AddWithValue("@StockID", stockID)

                    Dim rowsAffected As Integer = deleteCommand.ExecuteNonQuery()

                    If rowsAffected > 0 Then

                        MessageBox.Show($"Stock with ID {stockID} has been deleted successfully.", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information)

                        LoadStocks()

                    Else

                        MessageBox.Show($"No stock found with ID {stockID}.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

                    End If

                End Using

            Catch ex As Exception

                MessageBox.Show($"An error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

            End Try

        End If

    End Sub

    Private Sub updatebtn\_Click(sender As Object, e As EventArgs) Handles updatebtn.Click

        Dim stockID As String = InputBox("Enter the stock ID you want to update:", "Update Stock")

        If Not String.IsNullOrEmpty(stockID) Then

            Dim newStockName As String = InputBox("Enter the new Stock Name:", "Update Stock")

            Dim newQuantity As String = InputBox("Enter the new Quantity:", "Update Stock")

            Dim newPrice As String = InputBox("Enter the new Price:", "Update Stock")

            Dim newSector As String = InputBox("Enter the new Sector:", "Update Stock")

            Dim newROI As String = InputBox("Enter the new ROI:", "Update Stock")

            Try

                Using connection As New SqlConnection("Data Source=LAPTOP-N6QJ5U6O\SQLEXPRESS01;Initial Catalog=user\_info;Integrated Security=True;Encrypt = False")

                    connection.Open()

                    Dim updateCommand As New SqlCommand("UPDATE Stocks SET StockName = @StockName, Quantity = @Quantity, Price = @Price, Sector = @Sector, ROI = @ROI WHERE StockID = @StockID", connection)

                    updateCommand.Parameters.AddWithValue("@StockID", stockID)

                    updateCommand.Parameters.AddWithValue("@StockName", newStockName)

                    updateCommand.Parameters.AddWithValue("@Quantity", newQuantity)

                    updateCommand.Parameters.AddWithValue("@Price", newPrice)

                    updateCommand.Parameters.AddWithValue("@Sector", newSector)

                    updateCommand.Parameters.AddWithValue("@ROI", newROI)

                    Dim rowsAffected As Integer = updateCommand.ExecuteNonQuery()

                    If rowsAffected > 0 Then

                        MessageBox.Show($"Stock with ID {stockID} has been updated successfully.", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information)

                        LoadStocks()

                    Else

                        MessageBox.Show($"No stock found with ID {stockID}.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

                    End If

                End Using

            Catch ex As Exception

                MessageBox.Show($"An error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

            End Try

        End If

    End Sub

    Private Sub PictureBox4\_Click(sender As Object, e As EventArgs) Handles PictureBox4.Click

        admin\_login.Show()

        Me.Close()

    End Sub

End Class

**Stock Forms(buy\_btn)**:

Imports System.Data.SqlClient

Imports System.Security.Cryptography.X509Certificates

Imports System.Windows.Forms.VisualStyles.VisualStyleElement.StartPanel

Imports Guna.UI2.WinForms

Imports Microsoft.VisualBasic.ApplicationServices

Public Class buy\_stock

    Dim ConnectionString As String = "Data Source=LAPTOP-N6QJ5U6O\SQLEXPRESS01;Initial Catalog=user\_info;Integrated Security=True"

    Dim transactionDate As Date = Date.Today

    Dim stockID As Integer

    Dim stockprice As Decimal

    Dim Quantity As Integer

    Dim username As String

    Dim stockName As String

    Dim userID As Integer

    Private Sub buystockbtn\_Click(sender As Object, e As EventArgs) Handles buystockbtn.Click

        Dim transactionDate As Date = Date.Today

        Dim stockID As Integer = Convert.ToInt32(StockID\_txt.Text)

        Dim stockprice As Decimal = Convert.ToDecimal(price\_label.Text)

        Dim Quantity As Integer = If(Integer.TryParse(Quantity\_txt.Text, Quantity), Quantity, 0)

        Dim username As String = home\_page.usrnme\_label.Text

        Dim stockName As String = GetStockNameByStockID(ConnectionString, stockID)

        Dim userID As Integer = GetUserIDByUsername(ConnectionString, username)

        Try

            If stockprice <= 0 Or Quantity <= 0 Then

                MessageBox.Show("Invalid Stock Price or Quantity.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

                Return

            End If

            Using connection As New SqlConnection(ConnectionString)

                connection.Open()

                Dim selectBalanceQuery As String = "SELECT Available\_balance FROM Users WHERE UserID = @UserID"

                Using selectBalanceCommand As New SqlCommand(selectBalanceQuery, connection)

                    selectBalanceCommand.Parameters.AddWithValue("@UserID", userID)

                    Dim availableBalance As Decimal = Convert.ToDecimal(selectBalanceCommand.ExecuteScalar())

                    If availableBalance >= stockprice Then

                        Dim updateBalanceQuery As String = "UPDATE Users SET Available\_balance = Available\_balance - @StockPrice WHERE UserID = @UserID"

                        Using updateBalanceCommand As New SqlCommand(updateBalanceQuery, connection)

                            updateBalanceCommand.Parameters.AddWithValue("@UserID", userID)

                            updateBalanceCommand.Parameters.AddWithValue("@StockPrice", stockprice)

                            updateBalanceCommand.ExecuteNonQuery()

                            Dim insertTransactionQuery As String = "INSERT INTO Transactions (UserID, StockName, Price, QtyBought, Type, TransactionDate, StockID) VALUES (@UserID, @StockName, @Price, @QtyBought, @Type, @TransactionDate, @StockID)"

                            Using insertTransactionCommand As New SqlCommand(insertTransactionQuery, connection)

                                insertTransactionCommand.Parameters.AddWithValue("@UserID", userID)

                                insertTransactionCommand.Parameters.AddWithValue("@StockName", stockName) ' Replace with the actual stock name

                                insertTransactionCommand.Parameters.AddWithValue("@Price", stockprice)

                                insertTransactionCommand.Parameters.AddWithValue("@QtyBought", Quantity)

                                insertTransactionCommand.Parameters.AddWithValue("@Type", "Buy")

                                insertTransactionCommand.Parameters.AddWithValue("@TransactionDate", transactionDate)

                                insertTransactionCommand.Parameters.AddWithValue("@StockID", stockID)

                                insertTransactionCommand.ExecuteNonQuery()

                                MessageBox.Show($"Stocks purchased successfully! Amount deducted: {stockprice}", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information)

                            End Using

                        End Using

                    Else

                        MessageBox.Show("Insufficient available balance.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

                    End If

                End Using

            End Using

        Catch ex As Exception

            MessageBox.Show($"An error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

        End Try

    End Sub

    Private Sub PictureBox1\_Click(sender As Object, e As EventArgs) Handles PictureBox1.Click

        Dim Price As Decimal

        Try

            If String.IsNullOrEmpty(StockID\_txt.Text) Then

                MessageBox.Show("Stock ID is empty.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

                Return

            End If

            Using connection As New SqlConnection("Data Source=LAPTOP-N6QJ5U6O\SQLEXPRESS01;Initial Catalog=user\_info;Integrated Security=True")

                connection.Open()

                Dim selectPriceQuery As String = "SELECT Price FROM Stocks WHERE StockID = @StockID"

                Using selectPriceCommand As New SqlCommand(selectPriceQuery, connection)

                    selectPriceCommand.Parameters.AddWithValue("@StockID", StockID\_txt.Text)

                    Dim stockPrice As Decimal = Convert.ToDecimal(selectPriceCommand.ExecuteScalar())

                    Price = stockPrice

                End Using

            End Using

            If Quantity\_txt.Text = "" Or "0" Then

                price\_label.Text = ""

            Else

                price\_label.Text = Price \* Quantity\_txt.Text

            End If

        Catch ex As Exception

            MessageBox.Show($"An error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

        End Try

    End Sub

    Private Sub Quantity\_txt\_KeyPress(sender As Object, e As KeyPressEventArgs) Handles Quantity\_txt.KeyPress

        If Not Char.IsDigit(e.KeyChar) AndAlso Not Char.IsControl(e.KeyChar) Then

            e.Handled = True

        End If

    End Sub

    Private Function GetUserIDByUsername(connectionString As String, username As String) As Integer

        Dim userID As Integer = 0

        Try

            Using con As New SqlConnection(connectionString)

                If con.State = ConnectionState.Closed Then

                    con.Open()

                End If

                Dim query As String = "SELECT UserID FROM Users WHERE Username = @Username"

                Using cmd As New SqlCommand(query, con)

                    cmd.Parameters.AddWithValue("@Username", username)

                    Dim result As Object = cmd.ExecuteScalar()

                    If result IsNot Nothing AndAlso Not DBNull.Value.Equals(result) Then

                        userID = Convert.ToInt32(result)

                    End If

                End Using

            End Using

        Catch ex As Exception

            MessageBox.Show($"Error: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

        End Try

        Return userID

    End Function

    Private Function GetStockNameByStockID(connectionString As String, stockID As Integer) As String

        Dim stockName As String = ""

        Try

            Using con As New SqlConnection(connectionString)

                If con.State = ConnectionState.Closed Then

                    con.Open()

                End If

                Dim query As String = "SELECT StockName FROM Stocks WHERE StockID = @StockID"

                Using cmd As New SqlCommand(query, con)

                    cmd.Parameters.AddWithValue("@StockID", stockID)

                    Dim result As Object = cmd.ExecuteScalar()

                    If result IsNot Nothing AndAlso Not DBNull.Value.Equals(result) Then

                        stockName = result.ToString()

                    End If

                End Using

            End Using

        Catch ex As Exception

            MessageBox.Show($"Error: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

        End Try

        Return stockName

    End Function

    Private Sub buy\_stock\_Load(sender As Object, e As EventArgs) Handles MyBase.Load

    End Sub

End Class

**Stock Form(sell-btn)**:

Imports System.Data.SqlClient

Imports System.Security.Cryptography.X509Certificates

Imports System.Windows.Forms.VisualStyles.VisualStyleElement.StartPanel

Imports Guna.UI2.WinForms

Imports Microsoft.VisualBasic.ApplicationServices

Public Class sell\_stock

    Dim ConnectionString As String = "Data Source=LAPTOP-N6QJ5U6O\SQLEXPRESS01;Initial Catalog=user\_info;Integrated Security=True"

    Dim transactionDate As Date = Date.Today

    Dim stockID As Integer

    Dim stockprice As Decimal

    Dim Quantity As Integer

    Dim username As String

    Dim stockName As String

    Dim userID As Integer

    Private Sub sellstockbtn\_Click(sender As Object, e As EventArgs) Handles sellstockbtn.Click

        Dim transactionDate As Date = Date.Today

        Dim stockID As Integer = Convert.ToInt32(StockID\_txt.Text)

        Dim stockprice As Decimal = Convert.ToDecimal(price\_label.Text)

        Dim quantity As Integer = If(Integer.TryParse(Quantity\_txt.Text, quantity), quantity, 0)

        Dim username As String = home\_page.usrnme\_label.Text

        Dim stockName As String = GetStockNameByStockID(ConnectionString, stockID)

        Dim userID As Integer = GetUserIDByUsername(ConnectionString, username)

        Try

            If stockprice <= 0 OrElse quantity <= 0 Then

                MessageBox.Show("Invalid Stock Price or Quantity.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

                Return

            End If

            Using connection As New SqlConnection(ConnectionString)

                connection.Open()

                Dim selectStockQuery As String = "SELECT SUM(QtyBought) FROM Transactions WHERE UserID = @UserID AND StockID = @StockID AND Type = 'Buy'"

                Using selectStockCommand As New SqlCommand(selectStockQuery, connection)

                    selectStockCommand.Parameters.AddWithValue("@UserID", userID)

                    selectStockCommand.Parameters.AddWithValue("@StockID", stockID)

                    Dim boughtQuantity As Integer = Convert.ToInt32(selectStockCommand.ExecuteScalar())

                    If boughtQuantity < quantity Then

                        MessageBox.Show("You don't have enough stocks to sell.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

                        Return

                    End If

                End Using

                Dim selectBalanceQuery As String = "SELECT Available\_balance FROM Users WHERE UserID = @UserID"

                Using selectBalanceCommand As New SqlCommand(selectBalanceQuery, connection)

                    selectBalanceCommand.Parameters.AddWithValue("@UserID", userID)

                    Dim availableBalance As Decimal = Convert.ToDecimal(selectBalanceCommand.ExecuteScalar())

                    If availableBalance >= stockprice Then

                        Dim totalAmount = Label5.Text

                        Dim updateBalanceQuery As String = "UPDATE Users SET Available\_balance = Available\_balance + @StockPrice WHERE UserID = @UserID"

                        Using updateBalanceCommand As New SqlCommand(updateBalanceQuery, connection)

                            updateBalanceCommand.Parameters.AddWithValue("@UserID", userID)

                            updateBalanceCommand.Parameters.AddWithValue("@StockPrice", totalAmount)

                            updateBalanceCommand.ExecuteNonQuery()

                            Dim insertTransactionQuery As String = "INSERT INTO Transactions (UserID, StockName, Price, QtyBought, Type, TransactionDate, StockID) VALUES (@UserID, @StockName, @Price, @QtyBought, @Type, @TransactionDate, @StockID)"

                            Using insertTransactionCommand As New SqlCommand(insertTransactionQuery, connection)

                                insertTransactionCommand.Parameters.AddWithValue("@UserID", userID)

                                insertTransactionCommand.Parameters.AddWithValue("@StockName", stockName)

                                insertTransactionCommand.Parameters.AddWithValue("@Price", totalAmount)

                                insertTransactionCommand.Parameters.AddWithValue("@QtyBought", quantity)

                                insertTransactionCommand.Parameters.AddWithValue("@Type", "Sell")

                                insertTransactionCommand.Parameters.AddWithValue("@TransactionDate", transactionDate)

                                insertTransactionCommand.Parameters.AddWithValue("@StockID", stockID)

                                insertTransactionCommand.ExecuteNonQuery()

                                MessageBox.Show($"Stocks Sold successfully! Amount: {totalAmount}", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information)

                                Me.Close()

                            End Using

                        End Using

                    Else

                        MessageBox.Show("Insufficient available balance.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

                    End If

                End Using

            End Using

        Catch ex As Exception

            MessageBox.Show($"An error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

        End Try

    End Sub

    Private Sub PictureBox1\_Click(sender As Object, e As EventArgs) Handles PictureBox1.Click

        Try

            Dim stockID As Integer = Convert.ToInt32(StockID\_txt.Text

            Using connection As New SqlConnection(ConnectionString)

                connection.Open()

                Dim selectPriceAndROIQuery As String = "SELECT Price, ROI FROM Stocks WHERE StockID = @StockID"

                Using selectPriceAndROICommand As New SqlCommand(selectPriceAndROIQuery, connection)

                    selectPriceAndROICommand.Parameters.AddWithValue("@StockID", stockID)

                    Using reader As SqlDataReader = selectPriceAndROICommand.ExecuteReader()

                        If reader.Read() Then

                            Dim stockPrice As Decimal = Convert.ToDecimal(reader("Price"))

                            price\_label.Text = stockPrice.ToString()

                            Dim quantity As Integer = If(Integer.TryParse(Quantity\_txt.Text, quantity), quantity, 0)

                            Dim totalAmount As Decimal = stockPrice \* quantity \* (1 + (Convert.ToDecimal(reader("ROI")) / 100))

                            Label5.Text = totalAmount.ToString()

                            Dim priceIncrease As Decimal = totalAmount - (stockPrice \* quantity)

                            Label8.Text = priceIncrease.ToString()

                        Else

                            MessageBox.Show("Stock ID not found.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

                        End If

                    End Using

                End Using

            End Using

        Catch ex As Exception

            MessageBox.Show($"An error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

        End Try

    End Sub

    Private Function getroi() As Decimal

        Dim roi As Decimal = 0

        Try

            Using connection As New SqlConnection("Data Source=LAPTOP-N6QJ5U6O\SQLEXPRESS01;Initial Catalog=user\_info;Integrated Security=True")

                connection.Open()

                Dim selectROIQuery As String = "SELECT ROI FROM Stocks WHERE StockID = @StockID"

                Using selectROICommand As New SqlCommand(selectROIQuery, connection)

                    selectROICommand.Parameters.AddWithValue("@StockID", StockID\_txt.Text)

                    Dim roiObject As Object = selectROICommand.ExecuteScalar()

                    If roiObject IsNot Nothing AndAlso Not DBNull.Value.Equals(roiObject) Then

                        roi = Convert.ToDecimal(roiObject)

                    Else

                        MessageBox.Show("ROI not found for the given StockID.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

                    End If

                End Using

            End Using

        Catch ex As Exception

            MessageBox.Show($"An error occurred while fetching ROI: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

        End Try

        Return roi

    End Function

    Private Sub Quantity\_txt\_KeyPress(sender As Object, e As KeyPressEventArgs) Handles Quantity\_txt.KeyPress

        If Not Char.IsDigit(e.KeyChar) AndAlso Not Char.IsControl(e.KeyChar) Then

            e.Handled = True

        End If

    End Sub

    Private Function GetUserIDByUsername(connectionString As String, username As String) As Integer

        Dim userID As Integer = 0

        Try

            Using con As New SqlConnection(connectionString)

                If con.State = ConnectionState.Closed Then

                    con.Open()

                End If

                Dim query As String = "SELECT UserID FROM Users WHERE Username = @Username"

                Using cmd As New SqlCommand(query, con)

                    cmd.Parameters.AddWithValue("@Username", username)

                    Dim result As Object = cmd.ExecuteScalar()

                    If result IsNot Nothing AndAlso Not DBNull.Value.Equals(result) Then

                        userID = Convert.ToInt32(result)

                    End If

                End Using

            End Using

        Catch ex As Exception

            MessageBox.Show($"Error: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

        End Try

        Return userID

    End Function

    Private Function GetStockNameByStockID(connectionString As String, stockID As Integer) As String

        Dim stockName As String = ""

        Try

            Using con As New SqlConnection(connectionString)

                If con.State = ConnectionState.Closed Then

                    con.Open()

                End If

                Dim query As String = "SELECT StockName FROM Stocks WHERE StockID = @StockID"

                Using cmd As New SqlCommand(query, con)

                    cmd.Parameters.AddWithValue("@StockID", stockID)

                    Dim result As Object = cmd.ExecuteScalar()

                    If result IsNot Nothing AndAlso Not DBNull.Value.Equals(result) Then

                        stockName = result.ToString()

                    End If

                End Using

            End Using

        Catch ex As Exception

            MessageBox.Show($"Error: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

        End Try

        Return stockName

    End Function

    Private Sub buy\_stock\_Load(sender As Object, e As EventArgs) Handles MyBase.Load

    End Sub

End Class

**Expense Tracker Form:**

Imports System.Data.SqlClient

Imports System.Security.Cryptography.X509Certificates

Imports System.Windows.Forms.DataVisualization.Charting

Imports System.Windows.Forms.VisualStyles.VisualStyleElement.StartPanel

Imports Guna.UI2.WinForms

Imports Microsoft.VisualBasic.ApplicationServices

Public Class Expense\_tracker

    Private Sub Guna2Button1\_Click(sender As Object, e As EventArgs) Handles Guna2Button1.Click

        Try

            Dim category As String = Guna2TextBox3.Text

            Dim amount As Decimal = Decimal.Parse(Guna2TextBox4.Text)

            Dim spendingDateTime As DateTime = DateTime.Now

            Dim userID As Integer = GetUserIDByUsername("Data Source=LAPTOP-N6QJ5U6O\SQLEXPRESS01;Initial Catalog=user\_info;Integrated Security=True", home\_page.usrnme\_label.Text)

            Using connection As New SqlConnection("Data Source=LAPTOP-N6QJ5U6O\SQLEXPRESS01;Initial Catalog=user\_info;Integrated Security=True")

                connection.Open()

                Dim insertQuery As String = "INSERT INTO Spendings (SpendingType, Category, Amount, SpendingDateTime, UserID) VALUES (@SpendingType, @Category, @Amount, @SpendingDateTime, @UserID)"

                Using command As New SqlCommand(insertQuery, connection)

                    command.Parameters.AddWithValue("@SpendingType", "expense")

                    command.Parameters.AddWithValue("@Category", category)

                    command.Parameters.AddWithValue("@Amount", amount)

                    command.Parameters.AddWithValue("@SpendingDateTime", spendingDateTime)

                    command.Parameters.AddWithValue("@UserID", userID)

                    command.ExecuteNonQuery()

                End Using

            End Using

            LoadSpendings()

        Catch ex As Exception

            MessageBox.Show($"An error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

        End Try

    End Sub

    Private Sub Guna2Button2\_Click(sender As Object, e As EventArgs) Handles Guna2Button2.Click

        Try

            Dim category As String = Guna2TextBox6.Text

            Dim amount As Decimal = Decimal.Parse(Guna2TextBox5.Text)

            Dim spendingDateTime As DateTime = DateTime.Now

            Dim userID As Integer = GetUserIDByUsername("Data Source=LAPTOP-N6QJ5U6O\SQLEXPRESS01;Initial Catalog=user\_info;Integrated Security=True", home\_page.usrnme\_label.Text)

            Using connection As New SqlConnection("Data Source=LAPTOP-N6QJ5U6O\SQLEXPRESS01;Initial Catalog=user\_info;Integrated Security=True")

                connection.Open()

                Dim insertQuery As String = "INSERT INTO Spendings (SpendingType, Category, Amount, SpendingDateTime, UserID) VALUES (@SpendingType, @Category, @Amount, @SpendingDateTime, @UserID)"

                Using command As New SqlCommand(insertQuery, connection)

                    command.Parameters.AddWithValue("@SpendingType", "income")

                    command.Parameters.AddWithValue("@Category", category)

                    command.Parameters.AddWithValue("@Amount", amount)

                    command.Parameters.AddWithValue("@SpendingDateTime", spendingDateTime)

                    command.Parameters.AddWithValue("@UserID", userID)

                    command.ExecuteNonQuery()

                End Using

            End Using

            LoadSpendings()

        Catch ex As Exception

            MessageBox.Show($"An error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

        End Try

    End Sub

    Private Sub LoadSpendings()

        Try

            Dim userID As Integer = GetUserIDByUsername("Data Source=LAPTOP-N6QJ5U6O\SQLEXPRESS01;Initial Catalog=user\_info;Integrated Security=True", home\_page.usrnme\_label.Text)

            Using connection As New SqlConnection("Data Source=LAPTOP-N6QJ5U6O\SQLEXPRESS01;Initial Catalog=user\_info;Integrated Security=True")

                connection.Open()

                Dim selectQuery As String = "SELECT Category, Amount, SpendingDateTime FROM Spendings WHERE UserID = @UserID"

                Dim adapter As New SqlDataAdapter(selectQuery, connection)

                adapter.SelectCommand.Parameters.AddWithValue("@UserID", userID)

                Dim dataTable As New DataTable()

                adapter.Fill(dataTable)

                Guna2DataGridView1.DataSource = dataTable

            End Using

        Catch ex As Exception

            MessageBox.Show($"An error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

        End Try

    End Sub

    Private Function GetUserIDByUsername(connectionString As String, username As String) As Integer

        Dim userID As Integer = 0

        Try

            Using con As New SqlConnection(connectionString)

                If con.State = ConnectionState.Closed Then

                    con.Open()

                End If

                Dim query As String = "SELECT UserID FROM Users WHERE Username = @Username"

                Using cmd As New SqlCommand(query, con)

                    cmd.Parameters.AddWithValue("@Username", username)

                    Dim result As Object = cmd.ExecuteScalar()

                    If result IsNot Nothing AndAlso Not DBNull.Value.Equals(result) Then

                        userID = Convert.ToInt32(result)

                    End If

                End Using

            End Using

        Catch ex As Exception

            MessageBox.Show($"Error: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

        End Try

        Return userID

    End Function

    Private Sub PictureBox2\_Click(sender As Object, e As EventArgs) Handles PictureBox2.Click

        Try

            Dim userID As Integer = GetUserIDByUsername("Data Source=LAPTOP-N6QJ5U6O\SQLEXPRESS01;Initial Catalog=user\_info;Integrated Security=True", home\_page.usrnme\_label.Text)

            Using connection As New SqlConnection("Data Source=LAPTOP-N6QJ5U6O\SQLEXPRESS01;Initial Catalog=user\_info;Integrated Security=True")

                connection.Open()

                Dim incomeQuery As String = "SELECT SUM(Amount) FROM Spendings WHERE UserID = @UserID AND SpendingType = 'income'"

                Using incomeCommand As New SqlCommand(incomeQuery, connection)

                    incomeCommand.Parameters.AddWithValue("@UserID", userID)

                    Dim incomeSum As Object = incomeCommand.ExecuteScalar()

                    If incomeSum IsNot DBNull.Value Then

                        Label12.Text = Convert.ToDecimal(incomeSum).ToString("0.00")

                    Else

                        Label12.Text = "0.00"

                    End If

                End Using

                Dim expenseQuery As String = "SELECT SUM(Amount) FROM Spendings WHERE UserID = @UserID AND SpendingType = 'expense'"

                Using expenseCommand As New SqlCommand(expenseQuery, connection)

                    expenseCommand.Parameters.AddWithValue("@UserID", userID)

                    Dim expenseSum As Object = expenseCommand.ExecuteScalar()

                    If expenseSum IsNot DBNull.Value Then

                        Label13.Text = Convert.ToDecimal(expenseSum).ToString("0.00")

                    Else

                        Label13.Text = "0.00"

                    End If

                End Using

            End Using

        Catch ex As Exception

            MessageBox.Show($"An error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

        End Try

    End Sub

    Private Sub Expense\_tracker\_Load(sender As Object, e As EventArgs) Handles MyBase.Load

        Me.SpendingsTableAdapter.Fill(Me.User\_infoDataSet5.Spendings)

        Try

            Using connection As New SqlConnection("Data Source=LAPTOP-N6QJ5U6O\SQLEXPRESS01;Initial Catalog=user\_info;Integrated Security=True")

                connection.Open()

                Dim deleteCommandText As String = "DELETE FROM Spendings;"

                Using deleteCommand As New SqlCommand(deleteCommandText, connection)

                    deleteCommand.ExecuteNonQuery()

                End Using

            End Using

        Catch ex As Exception

            MessageBox.Show($"An error occurred while deleting records from Spendings table: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

        End Try

    End Sub

    Private Sub Guna2DataGridView1\_CellContentClick(sender As Object, e As DataGridViewCellEventArgs) Handles Guna2DataGridView1.CellContentClick

    End Sub

    Private Sub PictureBox3\_Click(sender As Object, e As EventArgs) Handles PictureBox3.Click

        Try

            Dim totalExpense As Decimal = Decimal.Parse(Label13.Text)

            Dim expenseData As New Dictionary(Of String, Decimal)()

            Using connection As New SqlConnection("Data Source=LAPTOP-N6QJ5U6O\SQLEXPRESS01;Initial Catalog=user\_info;Integrated Security=True")

                connection.Open()

                Dim query As String = "SELECT Category, Amount FROM Spendings WHERE SpendingType = 'expense'"

                Using command As New SqlCommand(query, connection)

                    Using reader As SqlDataReader = command.ExecuteReader()

                        While reader.Read()

                            Dim category As String = reader.GetString(0)

                            Dim amount As Decimal = reader.GetDecimal(1)

                            expenseData.Add(category, amount / totalExpense)

                        End While

                    End Using

                End Using

            End Using

            UpdateExpenseChart(expenseData)

        Catch ex As Exception

            MessageBox.Show($"An error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

        End Try

    End Sub

    Private Sub UpdateExpenseChart(expenseData As Dictionary(Of String, Decimal))

        Chart1.Series.Clear()

        Dim series As New Series("Expenses")

        series.ChartType = SeriesChartType.Pie

        For Each kvp As KeyValuePair(Of String, Decimal) In expenseData

            Dim expenseAmount As Decimal = kvp.Value \* Label13.Text

            Dim percentage As Decimal = kvp.Value \* 100

            series.Points.AddXY($"{kvp.Key}: {expenseAmount.ToString("C")} ({percentage.ToString("F2")}%)", kvp.Value)

        Next

        Chart1.Series.Add(series)

    End Sub

    Private Sub PictureBox4\_Click(sender As Object, e As EventArgs) Handles PictureBox4.Click

        Me.Hide()

        home\_page.Show()

    End Sub

End Class

**Categories Form:**

Imports System.Data.SqlClient

Imports System.Text

Public Class Form\_categories

    Public con As New SqlConnection("Data Source=LAPTOP-QDECFD8R\SQLEXPRESS01;Initial Catalog=expense;Integrated Security=True;Encrypt=False")

    Private Sub Save2\_btn\_Click(sender As Object, e As EventArgs) Handles Save2\_btn.Click

        Dim categoryName As String = Enter\_categories\_txt.Text.Trim()

        Dim connectionString As String = "Data Source=LAPTOP-QDECFD8R\SQLEXPRESS01;Initial Catalog=expense;Integrated Security=True;Encrypt=False"

        If categoryName <> "" Then

            If IsCategoryExists(connectionString, categoryName) Then

                MessageBox.Show("Category already exists. Please enter a different category name.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

                Return

            End If

            Dim username As String = Mainform.Login\_info1.Text.Substring("User: ".Length).Trim()

            Dim userID As Integer = GetUserIDByUsername(connectionString, username)

            Try

                Using con As New SqlConnection(connectionString)

                    con.Open()

                    Dim query As String = "INSERT INTO Category (CategoryName, IsPredefined, CreatedByUserID) VALUES (@CategoryName, @IsPredefined, @CreatedByUserID); SELECT @@IDENTITY;"

                    Using cmd As New SqlCommand(query, con)

                        cmd.Parameters.AddWithValue("@CategoryName", categoryName)

                        cmd.Parameters.AddWithValue("@IsPredefined", 0)

                        cmd.Parameters.AddWithValue("@CreatedByUserID", userID)

                        Dim newCategoryID As Integer = Convert.ToInt32(cmd.ExecuteScalar())

                        MessageBox.Show($"Category '{categoryName}' added successfully.", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information)

                    End Using

                End Using

            Catch ex As Exception

                MessageBox.Show($"Error: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

            End Try

        Else

            MessageBox.Show("Please enter a category.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

        End If

    End Sub

    Private Sub Button2\_Click(sender As Object, e As EventArgs) Handles Button2.Click

        Me.Close()

        Mainform.Show()

    End Sub

    Private Sub Remove\_btn\_Click(sender As Object, e As EventArgs) Handles Remove\_btn.Click

        Dim categoryToRemove As String = InputBox("Enter the category to remove:", "Remove Category")

        Dim connectionString As String = "Data Source=LAPTOP-QDECFD8R\SQLEXPRESS01;Initial Catalog=expense;Integrated Security=True;Encrypt=False"

        If categoryToRemove <> "" Then

            Try

                Using con As New SqlConnection(connectionString)

                    con.Open()

                    Dim query As String = "DELETE FROM Category WHERE CategoryName = @CategoryName AND IsPredefined = 0 AND CreatedByUserID = @CreatedByUserID;"

                    Using cmd As New SqlCommand(query, con)

                        Dim username As String = Mainform.Login\_info1.Text.Substring("User: ".Length).Trim()

                        Dim userID As Integer = GetUserIDByUsername(connectionString, username)

                        cmd.Parameters.AddWithValue("@CategoryName", categoryToRemove)

                        cmd.Parameters.AddWithValue("@CreatedByUserID", userID)

                        Dim rowsAffected As Integer = cmd.ExecuteNonQuery()

                        If rowsAffected > 0 Then

                            MessageBox.Show($"Category '{categoryToRemove}' removed successfully.", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information)

                       Else

                            MessageBox.Show($"Category '{categoryToRemove}' not found or could not be removed.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

                        End If

                    End Using

                End Using

            Catch ex As Exception

                MessageBox.Show($"Error: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

            End Try

        Else

            MessageBox.Show("Please enter a category to remove.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

        End If

    End Sub

    Private Function GetUserIDByUsername(connectionString As String, username As String) As Integer

        Dim userID As Integer = 0

        Try

            Using con As New SqlConnection(connectionString)

                If con.State = ConnectionState.Closed Then

                    con.Open()

                End If

                Dim query As String = "SELECT UserID FROM users WHERE Username = @Username"

                Using cmd As New SqlCommand(query, con)

                    cmd.Parameters.AddWithValue("@Username", username)

                    Dim result As Object = cmd.ExecuteScalar()

                    If result IsNot Nothing AndAlso Not DBNull.Value.Equals(result) Then

                        userID = Convert.ToInt32(result)

                    End If

                End Using

            End Using

        Catch ex As Exception

            MessageBox.Show($"Error: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

        End Try

        Return userID

    End Function

    Private Function IsCategoryExists(ByVal connectionString As String, ByVal categoryName As String) As Boolean

        Dim isExists As Boolean = False

        Try

            Using con As New SqlConnection(connectionString)

                If con.State = ConnectionState.Closed Then

                    con.Open()

                End If

                Dim query As String = "SELECT COUNT(\*) FROM Category WHERE CategoryName = @CategoryName"

                Using cmd As New SqlCommand(query, con)

                    cmd.Parameters.AddWithValue("@CategoryName", categoryName)

                    Dim count As Integer = Convert.ToInt32(cmd.ExecuteScalar())

                    isExists = (count > 0)

                End Using

            End Using

        Catch ex As Exception

            MessageBox.Show($"Error checking category existence: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

        End Try

        Return isExists

    End Function

    Private Sub View\_categories\_btn\_Click(sender As Object, e As EventArgs) Handles View\_categories\_btn.Click

        Dim connectionString As String = "Data Source=LAPTOP-QDECFD8R\SQLEXPRESS01;Initial Catalog=expense;Integrated Security=True;Encrypt=False"

        Try

            Dim customCategories As String = GetCustomCategories(connectionString)

            If customCategories <> "" Then

                MessageBox.Show("Custom Categories:" & vbCrLf & customCategories, "Custom Categories", MessageBoxButtons.OK, MessageBoxIcon.Information)

            Else

                MessageBox.Show("No custom categories found.", "Custom Categories", MessageBoxButtons.OK, MessageBoxIcon.Information)

            End If

        Catch ex As Exception

            MessageBox.Show($"Error: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

        End Try

    End Sub

    Private Function GetCustomCategories(connectionString As String) As String

        Dim customCategories As New StringBuilder()

        Try

            Using con As New SqlConnection(connectionString)

                con.Open()

                Dim query As String = "SELECT CategoryName FROM Category WHERE IsPredefined = 0"

                Using cmd As New SqlCommand(query, con)

                    Dim reader As SqlDataReader = cmd.ExecuteReader()

                    While reader.Read()

                        Dim categoryName As String = Convert.ToString(reader("CategoryName"))

                        customCategories.AppendLine(categoryName)

                    End While

                End Using

            End Using

        Catch ex As Exception

            MessageBox.Show($"Error: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

        End Try

        Return customCategories.ToString()

    End Function

End Class

**Budget Allocation Form:**

Public Class budget\_allocation

    Private Sub Guna2TextBox4\_TextChanged(sender As Object, e As EventArgs) Handles Guna2TextBox4.TextChanged

    End Sub

    Private Sub PictureBox4\_Click(sender As Object, e As EventArgs) Handles PictureBox4.Click

        Me.Hide()

        home\_page.Show()

    End Sub

    Private Sub Guna2Button1\_Click(sender As Object, e As EventArgs) Handles Guna2Button1.Click

        Try

            Dim income As Decimal = Decimal.Parse(Guna2TextBox1.Text)

            Dim expense As Decimal = Decimal.Parse(Guna2TextBox3.Text)

            Dim investment As Decimal = Decimal.Parse(Guna2TextBox4.Text)

            Dim total As Decimal = income

            Dim expensePercentage As Decimal = (expense / total) \* 100

            Dim investmentPercentage As Decimal = (investment / total) \* 100

            Dim savingPercentage As Decimal = 100 - (expensePercentage + investmentPercentage)

            Dim expenseAmount As Decimal = expense

            Dim investmentAmount As Decimal = investment

            Dim savingAmount As Decimal = total - (expenseAmount + investmentAmount)

            Chart1.Series.Clear()

            Chart1.Series.Add("Budget")

            Chart1.Series("Budget").ChartType = DataVisualization.Charting.SeriesChartType.Pie

            Chart1.Series("Budget").Points.AddXY("Expense", expensePercentage)

            Chart1.Series("Budget").Points.AddXY("Investment", investmentPercentage)

            Chart1.Series("Budget").Points.AddXY("Saving", savingPercentage)

            Chart2.Series.Clear()

            Chart2.Series.Add("Amount")

            Chart2.Series("Amount").Points.AddXY("Expense", expenseAmount)

            Chart2.Series("Amount").Points.AddXY("Investment", investmentAmount)

            Chart2.Series("Amount").Points.AddXY("Saving", savingAmount)

        Catch ex As Exception

            MessageBox.Show($"An error occurred: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

        End Try

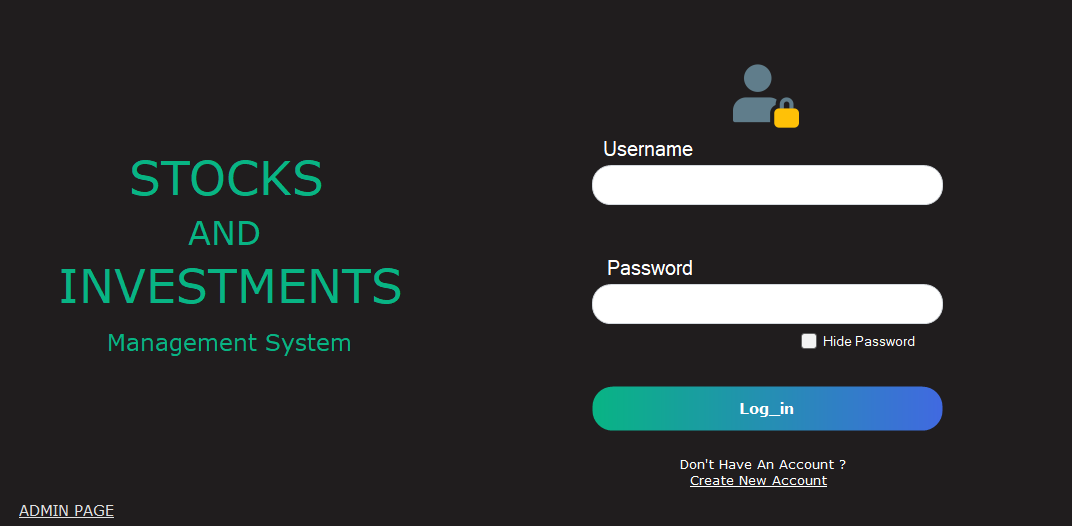
    End Sub

End Class

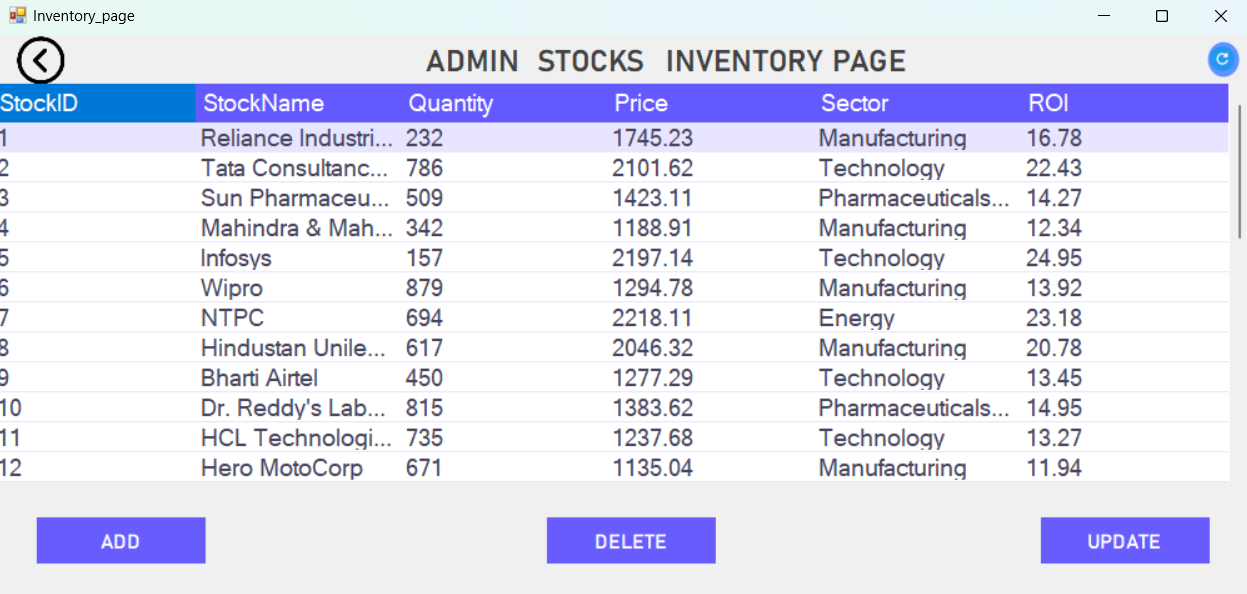
**10. Appendices C-Screenshots**

**Form Designs:**

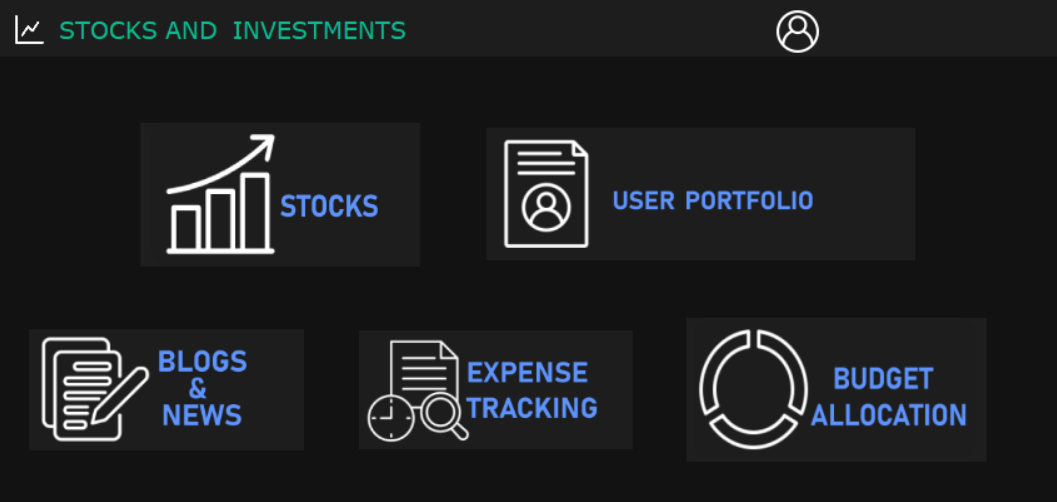
**Login Form Design:**

****

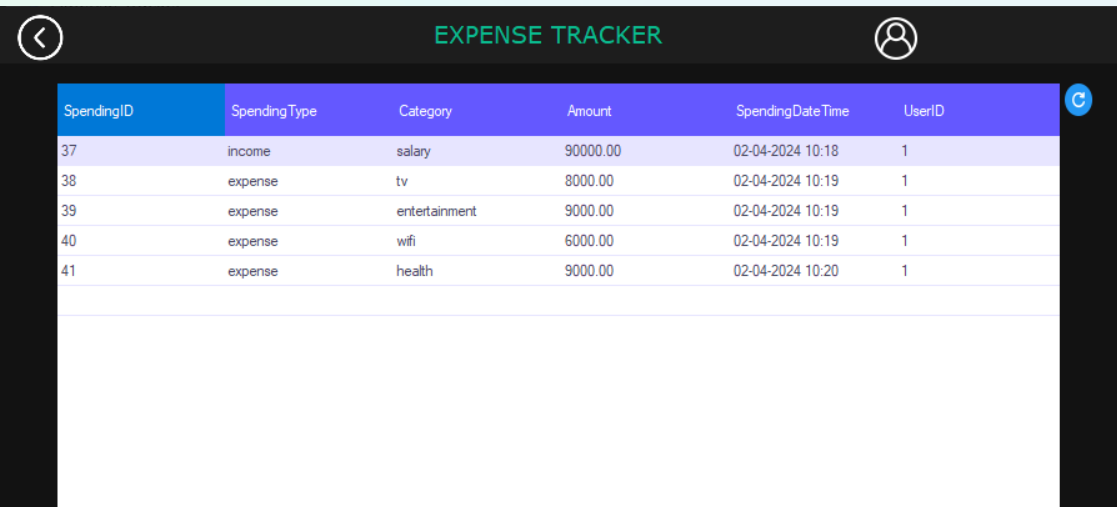
**Admin Inventory Page Form Design:**

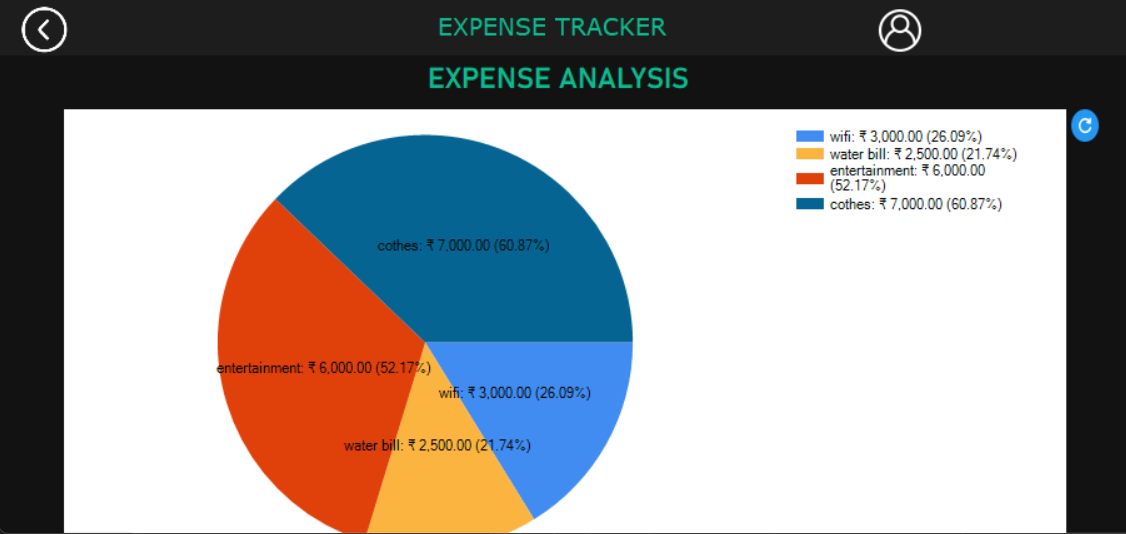
****

**Home Page Form Design:**

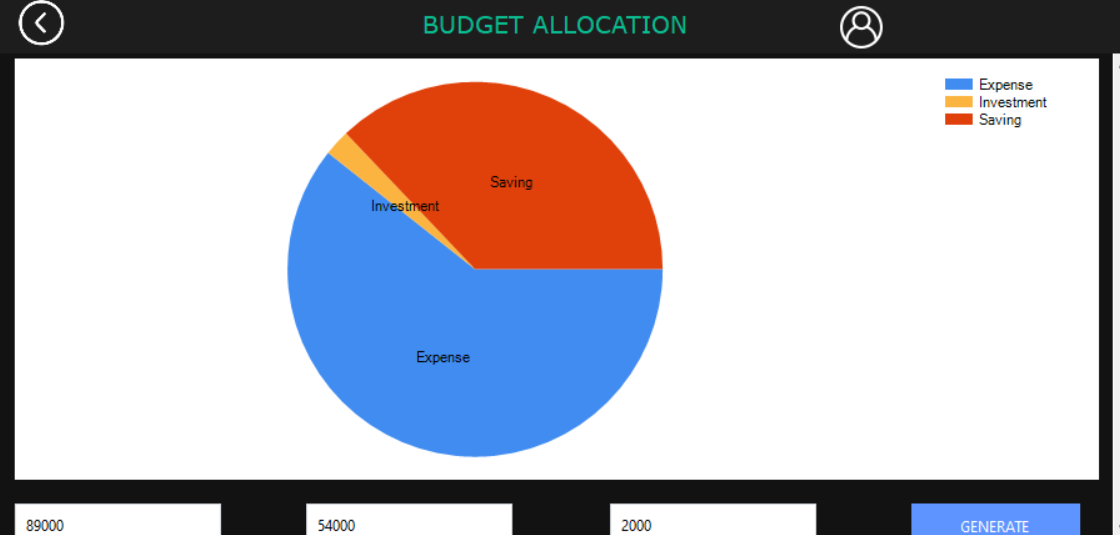
****

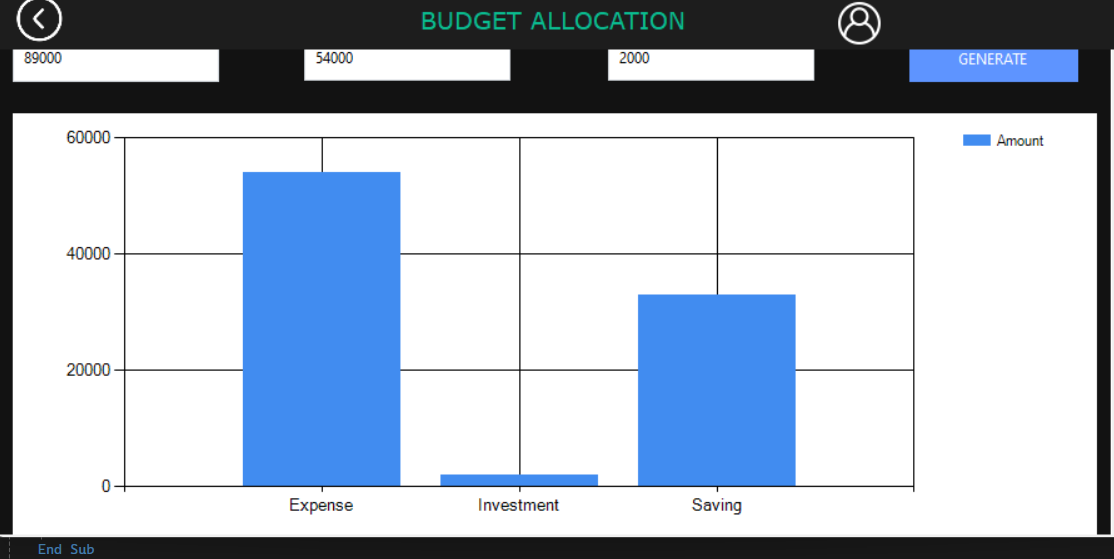
**Expense Tracker Form Design:**

****

****

**Budget Allocation Form Design:**

****

****