Faculty of Computers and Artificial Intelligence

ASSIGNMENT - 2

CS251 - Introduction to Software Engineering

TEAM MEMBERS

1. Hassan Walid Hassan

2. Ahmed Sherif Sayed

3. Mohammed Sherif

PERSONAL FINANCE MANAGEMENT SYSTEM

Introduction:

Managing personal finances is a critical part of everyday life, but it can be overwhelming. This project aims to simplify and centralize the process by providing users with a Personal Finance Management System. The application enables users to track their assets, set and monitor financial goals, calculate zakat obligations, and import bank transactions from XML files. The goal is to offer a clear, user-friendly, and effective platform to empower individuals in handling their financial responsibilities and ambitions.

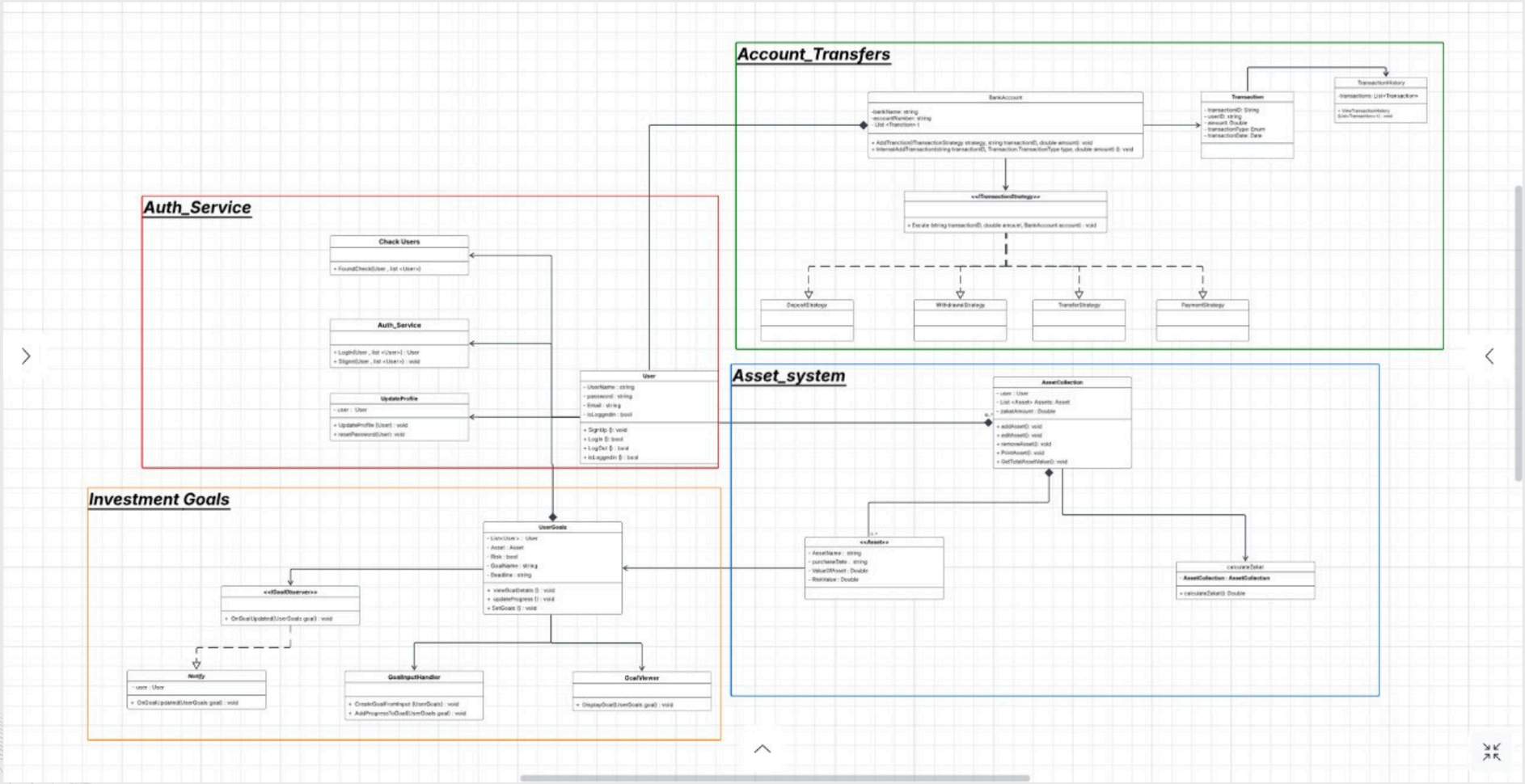
DESIGN OVERVIEW

This Financial Manager application is designed to help users track their financial assets, calculate zakat obligations, set financial goals, and manage banking transactions. The system follows a modular architecture with clear separation of concerns.

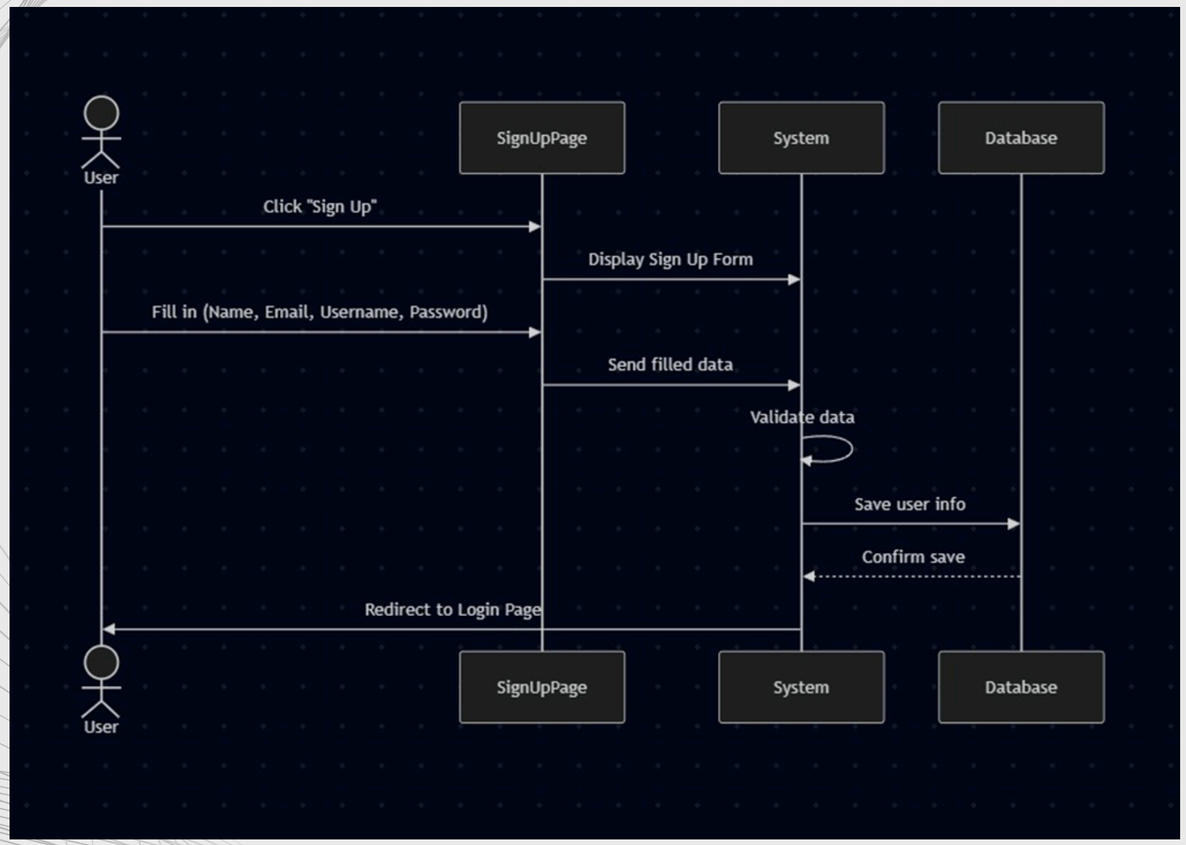
Key Features:

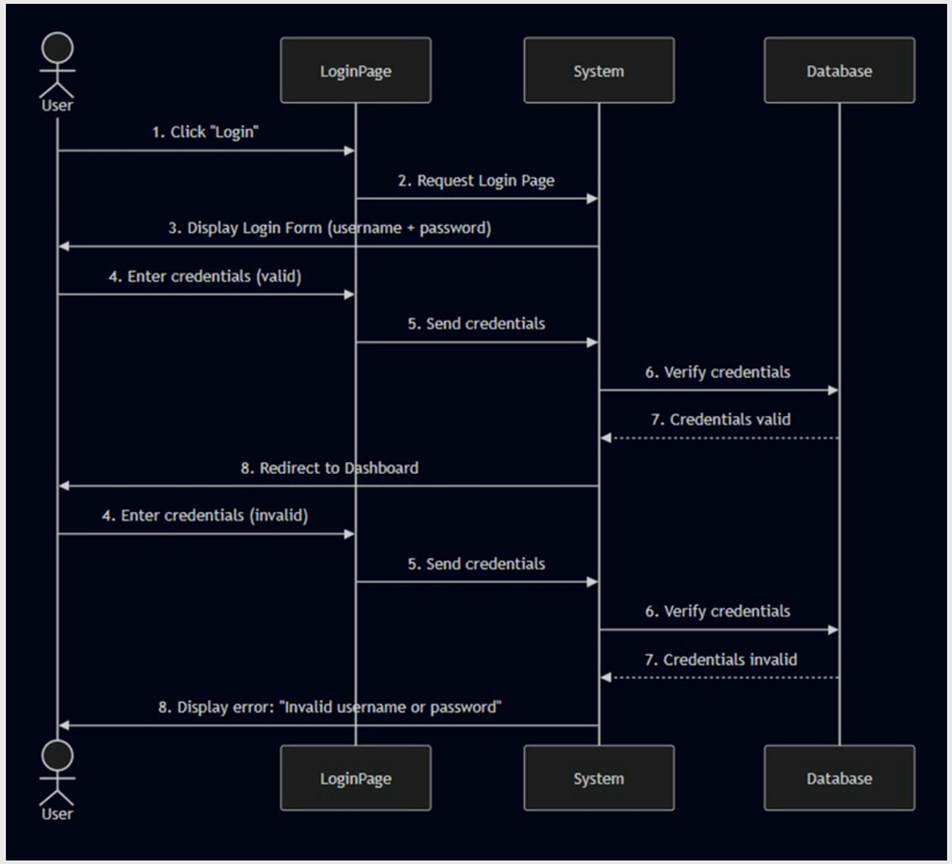
- User authentication (sign up/log in)
- Asset management (add, edit, view, remove)
- Financial goal tracking with progress notifications
- Zakat calculation (2.5% of total assets)
- Bank account management
- Transaction history

CLASS DIAGRAM

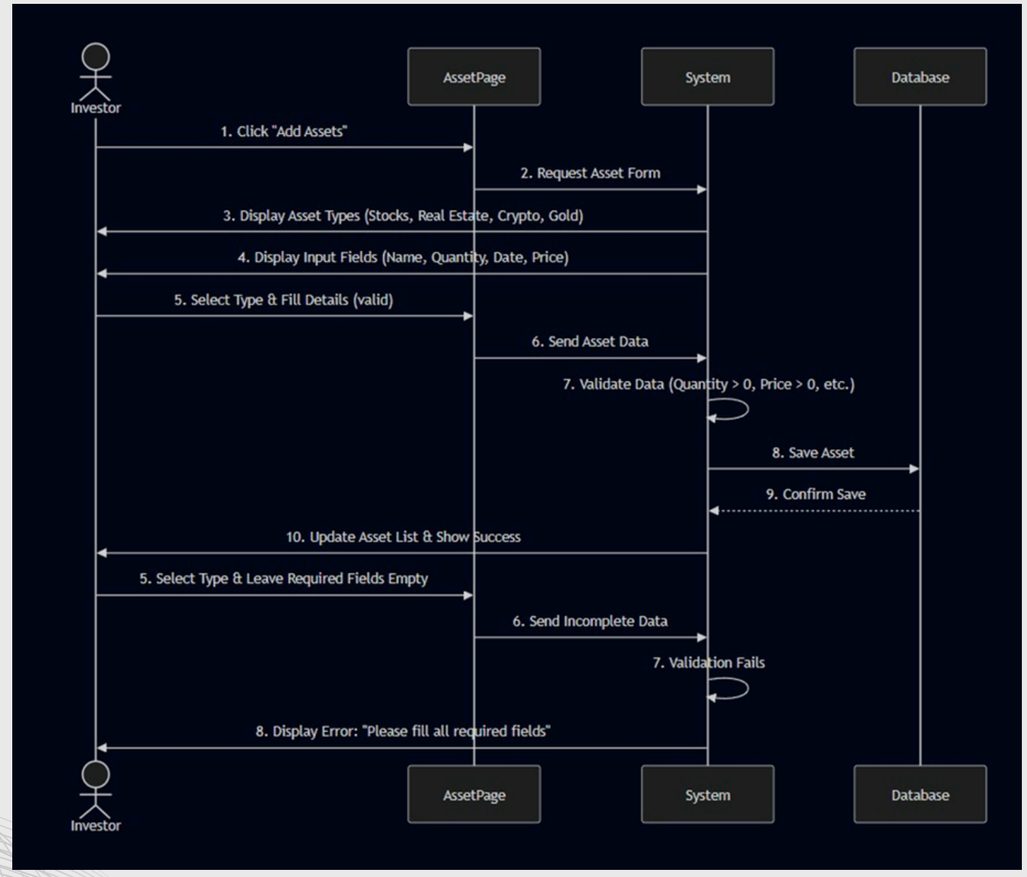


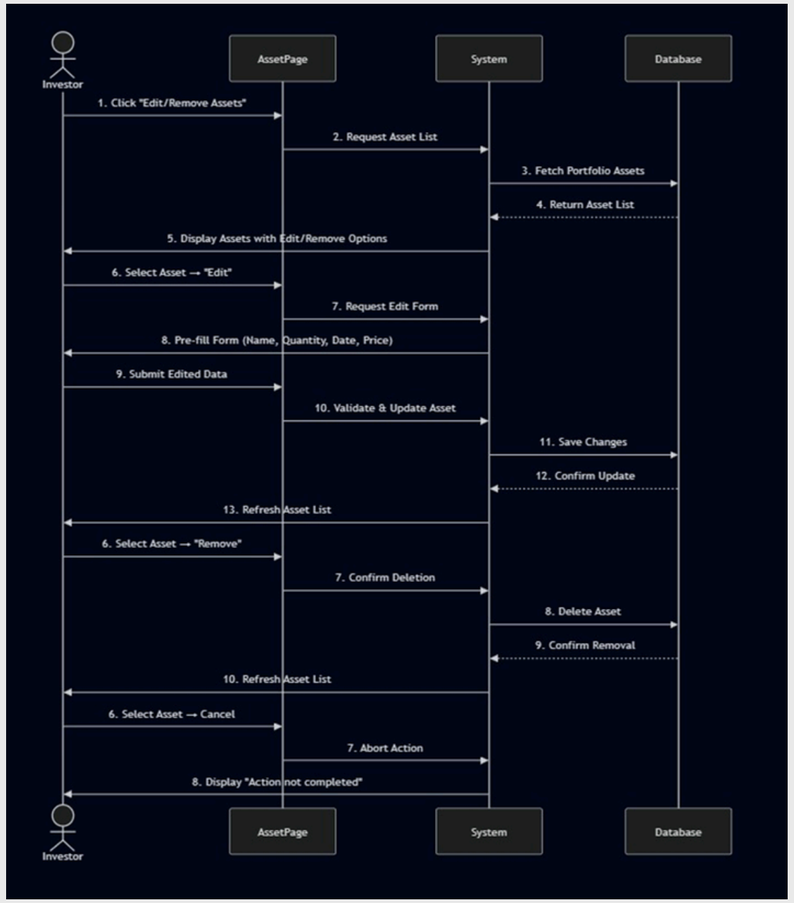
SEQUENCE DIAGRAMS



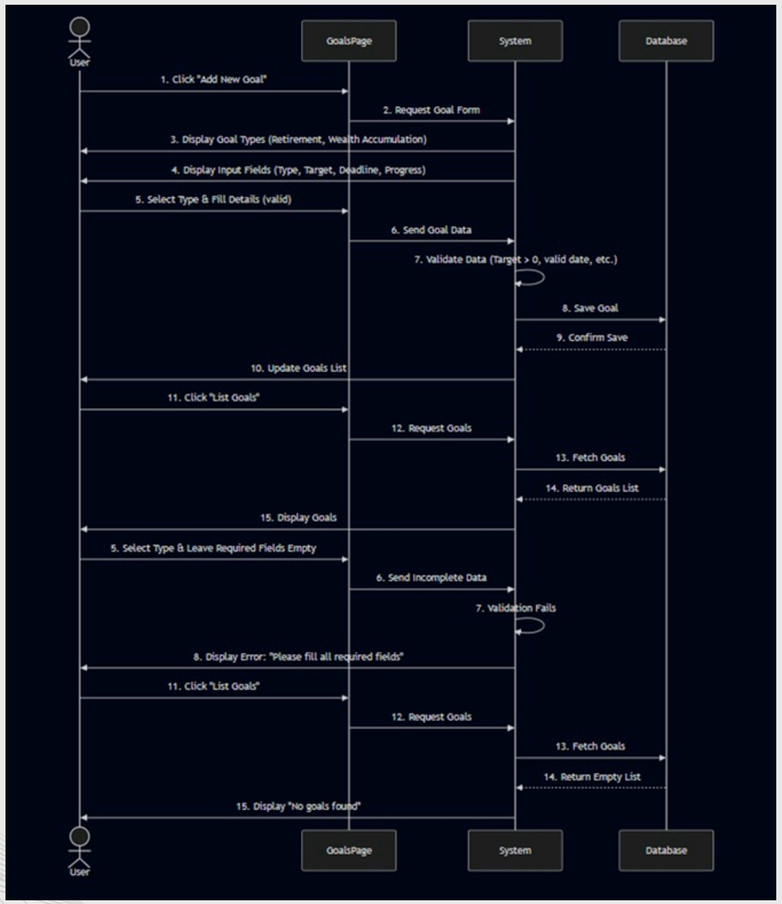


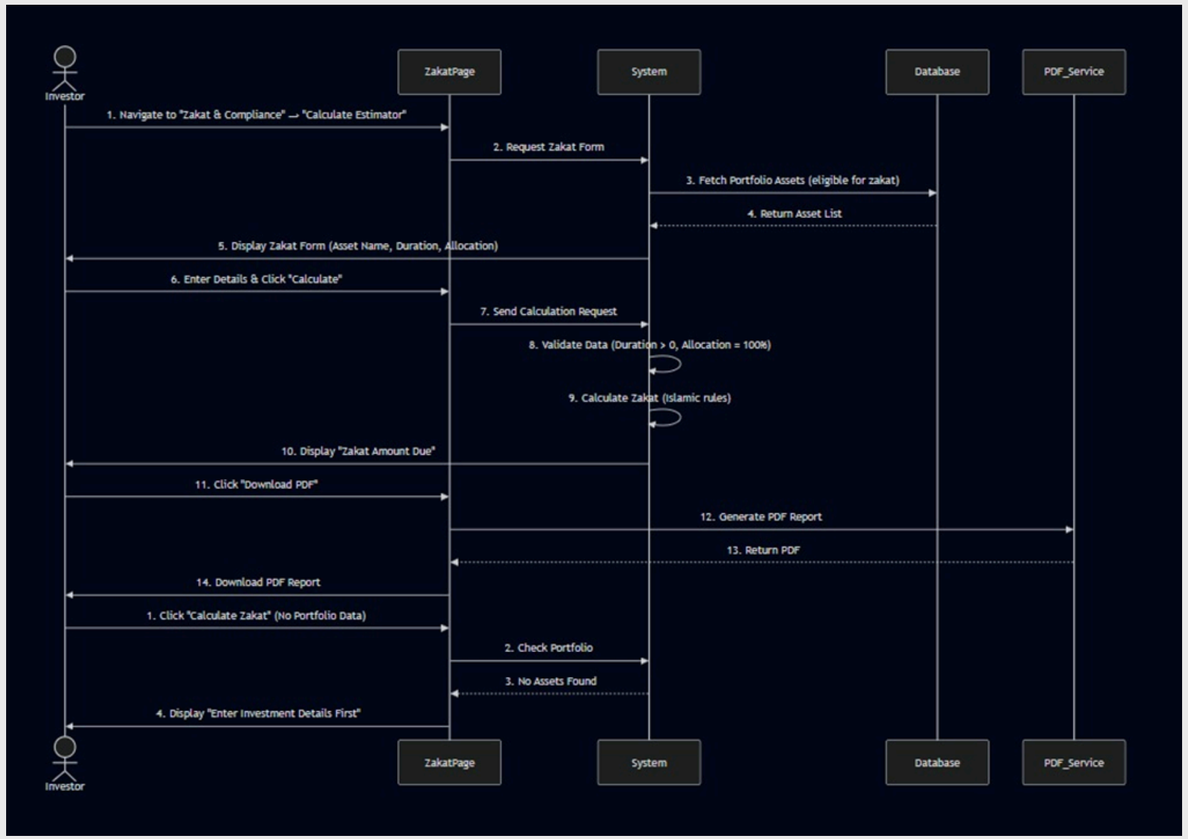
SEQUENCE DIAGRAMS



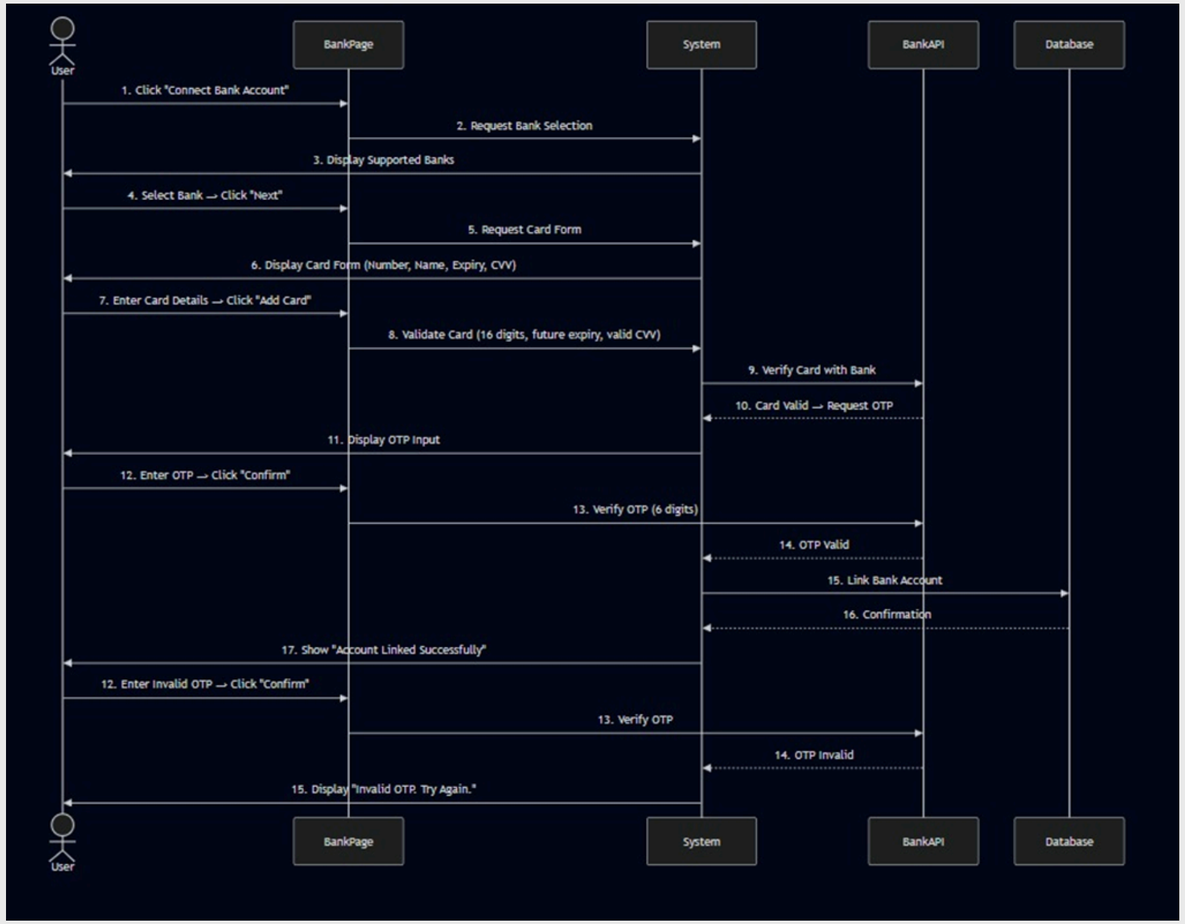


SEQUENCE DIAGRAMS

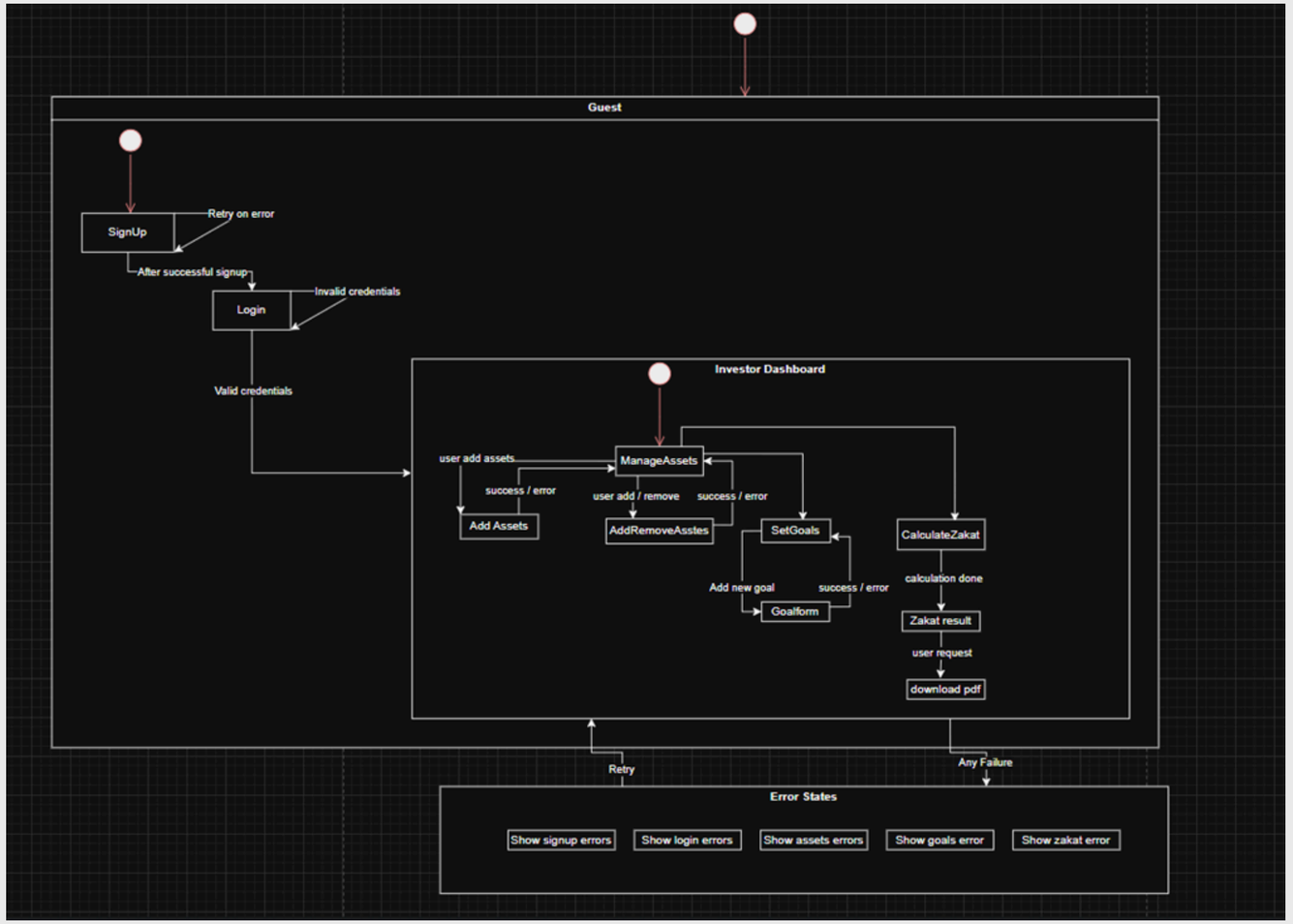




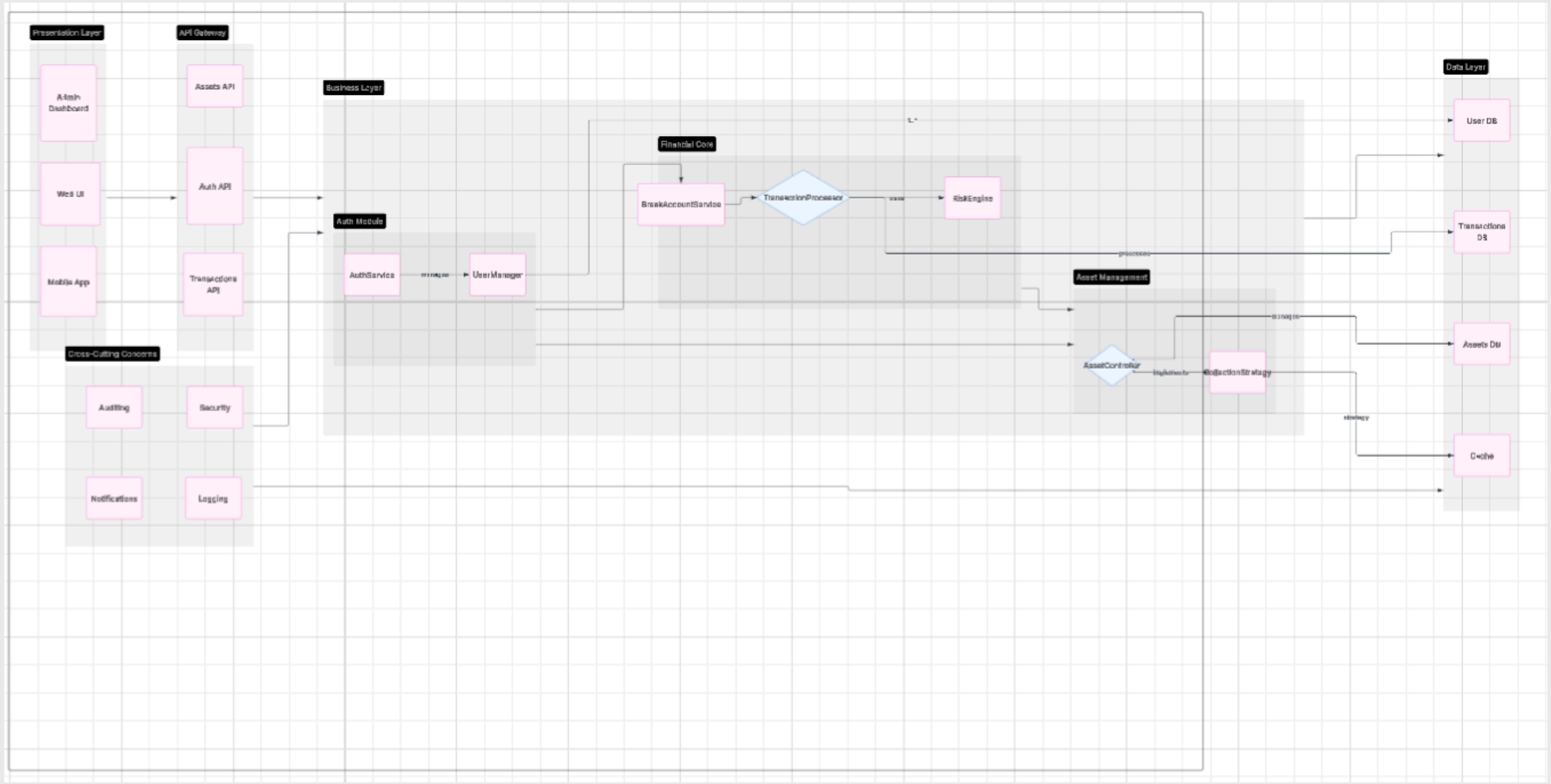
SEQUENCE DIAGRAMS



STATE DIAGRAM



ARCHITECTURE DIAGRAM



SOLID PRINCIPLES

User

- Represents the system's user, storing username, password, email, and login status. Provides functions for signing up, logging in, logging out, and checking if the user is active.

UpdateProfile

- Handles editing user information like username, email, and password. Works closely with the User class to keep account data updated and secure.

UserGoals

- Links users with their investment goals and associated assets. Sends alerts to users if an asset’s risk level changes.

BankAccount

- Contains bank account information for users. Provides secure methods to connect a bank and verify the ownership of the account.

ChaseBankProvider

- Handles secure connection with Chase Bank. Manages authentication credentials and session control for banking operations.

Transaction

- Models financial transactions, tracking amount, type, and date. Allows creating new transactions and viewing transaction records.

SOLID PRINCIPLES

TransactionHistory

- Stores a user's full history of transactions. Offers features to add new transactions, search by date, and count total transactions.

Asset

- Represents an individual asset like stock, real estate, or cryptocurrency. Tracks the asset's purchase date, current value, and risk factor.

AssetCollection

- Manages all assets owned by a user. Supports adding and removing assets and calculates the total zakat amount based on asset value.

calculateZakat

- Calculates zakat from a user's total assets according to Islamic financial principles, based on the current asset values.

Crypto(Subclass of Asset)

- Includes crypto-specific details like wallet address and blockchain network. Calculates value based on market data.

RealEstate(Subclass of Asset)

- Represents property. Includes fields like location and rental income, and supports yield calculation.

SOLID PRINCIPLES

Gold(Subclass of Asset)

- Represents gold holdings. Includes purity and weight, with methods for calculating value by karat.

NotifyUsers

- Observes UserGoals and sends notifications when changes occur, such as high-risk levels or deadlines approaching.

FinancialGoal

- A specific type of investment goal targeting financial milestones like reaching a certain net worth or saving goal.

EmergencyFundGoal(Subclass of UserGoals)

- Focused on building a liquid emergency reserve fund. Tracks saved amount versus target.

DESIGN PATTERNS

Design Patterns Used

1.Observer Pattern

- Used between UserGoals and NotifyUsers.
- Enables automatic user notification when a goal's risk level or deadline changes.
- Promotes loose coupling and event-driven architecture.

▪ **Strategy Pattern**

- Used in the Asset system via subclasses: Crypto, RealEstate, and Gold.
- Each subclass implements its own method for calculating value, zakat, or yield.
- Promotes flexibility and open extension of new asset types.

2. SOLID Principles Applied

▪ **Single Responsibility Principle (SRP)**

- Each class is responsible for one purpose (e.g., User handles login, UpdateProfile handles profile updates).

▪ **Open/Closed Principle (OCP)**

- Asset class is open for extension via subclasses like Crypto and RealEstate, but closed for modification.

▪ **Liskov Substitution Principle (LSP)**

Subclasses of Asset (Crypto, RealEstate, Gold) can be used anywhere an Asset is expected without

CODE ORGANIZATION

1. **User Management Subsystem (User class)**
   - Handles user profiles and authentication
   - Manages user sessions (login/logout)
2. **Asset Management Subsystem**
   - Asset base class with specialized asset types (Crypto, RealState, Gold)
   - AssetCollection for managing multiple assets
   - Zakat calculation functionality
3. **Goal Tracking Subsystem**
   - UserGoals class for setting financial targets
   - Observer pattern for progress notifications
   - Goal progress visualization
4. **Banking Subsystem**
   - BankAccount class for account management
   - Transaction class for financial transactions
   - Transaction history tracking
5. **Main Application Flow**
   - main_menu class handles the primary user interface
   - Program class contains the entry point and main loop

TECHNOLOGY BRIEF

The application is built using:

- C# (.NET): Primary programming language for the backend logic
- Object-Oriented Programming: Utilizes classes, inheritance, and interfaces

- **Design Patterns:**
- strategy Patterns on transaction
- Observer pattern for goal notifications
- Separation of concerns for maintainability
- Console Interface: Simple text-based user interface

STRATEGY PATTERNS ON TRANSACTION

```
438  interface ITransactionStrategy
439  {
         5 references
440      public void Execute(string transactionID, double amount, BankAccount account);
441  }
442
         1 reference
443  class DepositStrategy : ITransactionStrategy
444  {
         2 references
445      public void Execute(string transactionID, double amount, BankAccount account)
446      {
447          account.InternalAddTransaction(transactionID, Transaction.TransactionType.Deposit, amount)
448          Console.WriteLine($"Deposited {amount:C} successfully.");
449      }
450  }
451
         1 reference
452  class WithdrawalStrategy : ITransactionStrategy
453  {
         2 references
454      public void Execute(string transactionID, double amount, BankAccount account)
455      {
456          account.InternalAddTransaction(transactionID, Transaction.TransactionType.Withdrawal, amou
457          Console.WriteLine($"Withdrew {amount:C} successfully.");
458      }
459  }
460
```

DEMO WALKTHROUGH

1. **User Authentication**
   - Sign up new users
   - Log in existing users
   - Update profile information
2. **Asset Management**
   - Add different types of assets (generic, crypto, real estate, gold)
   - View and edit existing assets
   - Remove assets
3. **Goal Tracking**
   - Set financial goals for assets
   - Track progress toward goals
   - Receive notifications when goals are updated or achieved
4. **Zakat Calculation**
   - Automatic calculation of zakat obligation (2.5% of total assets)
   - Display of total asset value
5. **Banking Features**
   - Create bank accounts
   - Record transactions (deposits, withdrawals, transfers, payments)
   - View transaction history

KEY CODE HIGHLIGHTS

Observer Pattern for Goals:

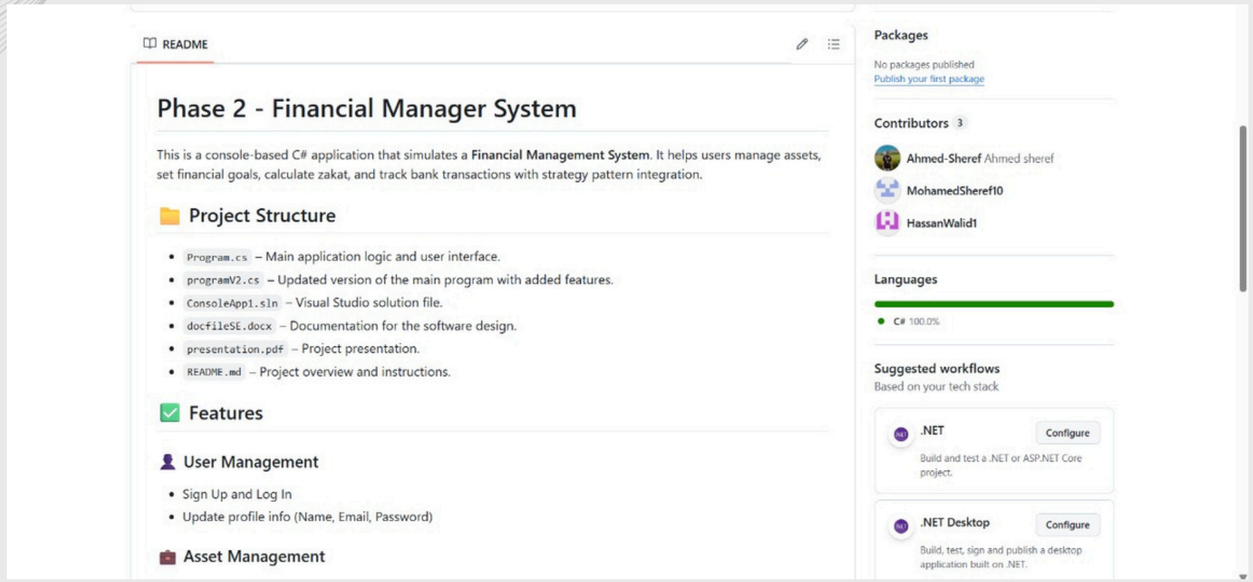
```
interface IGoalObserver {
    void OnGoalUpdated(UserGoals goal);
}

class Notify : IGoalObserver {
    public void OnGoalUpdated(UserGoals goal) {
        Console.WriteLine($"\nHey {user.GetName()}, your goal was updated.");
        if (goal.IsAchieved()) {
            Console.WriteLine("Congratulations! You achieved your goal!");
        }
    }
}
```

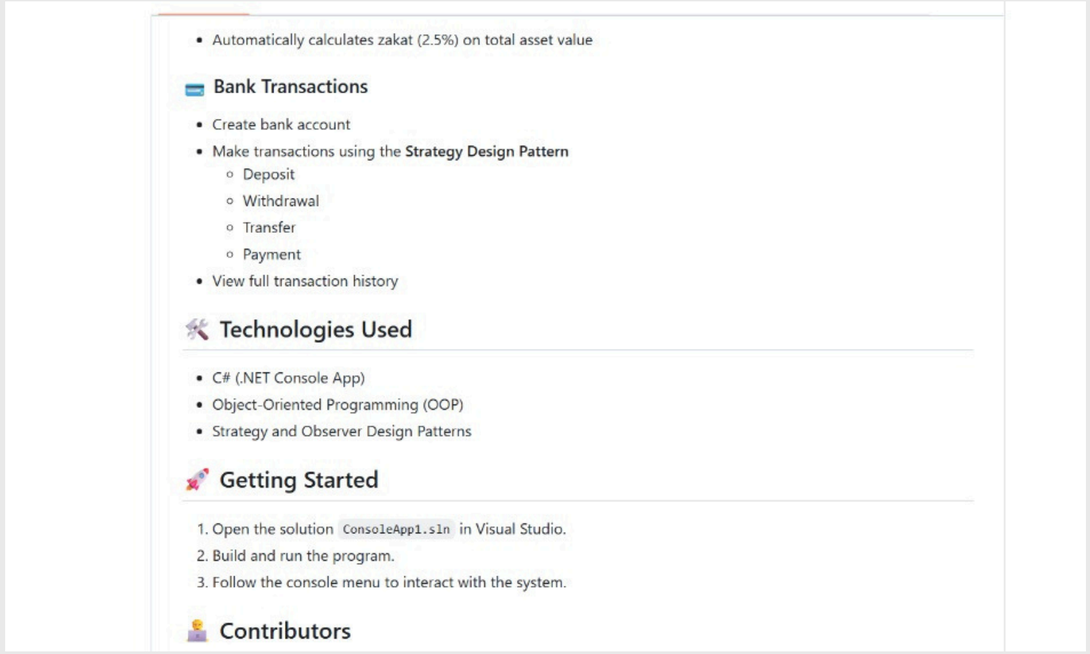
KEY CODE HIGHLIGHTS

Zakat Calculation:

```
class calculateZakat {
    public double CalculateZakat() {
        double total = assetCollection.GetTotalAssetValue();
        return total * 0.025; // 2.5% zakat calculation
    }
}
```

GIT HUP


README
Phase 2 - Financial Manager System
This is a console-based C# application that simulates a Financial Management System. It helps users manage assets, set financial goals, calculate zakat, and track bank transactions with strategy pattern integration.
Project Structure
Program.cs – Main application logic and user interface.
programV2.cs – Updated version of the main program with added features.
ConsoleApp1.sln – Visual Studio solution file.
docfileSE.docx – Documentation for the software design.
presentation.pdf – Project presentation.
README.md – Project overview and instructions.
Features
User Management
Sign Up and Log In
Update profile info (Name, Email, Password)
Asset Management
Packages
No packages published
Publish your first package
Contributors 3
Ahmed-Sheref Ahmed sheref
MohamedSheref10
HassanWalid1
Languages
C# 100.0%
Suggested workflows
Based on your tech stack
.NET
Configure
Build and test a .NET or ASP.NET Core project.
.NET Desktop
Configure
Build, test, sign and publish a desktop application built on .NET.


Automatically calculates zakat (2.5%) on total asset value
Bank Transactions
Create bank account
Make transactions using the Strategy Design Pattern
Deposit
Withdrawal
Transfer
Payment
View full transaction history
Technologies Used
C# (.NET Console App)
Object-Oriented Programming (OOP)
Strategy and Observer Design Patterns
Getting Started
1. Open the solution ConsoleApp1.sln in Visual Studio.
2. Build and run the program.
3. Follow the console menu to interact with the system.
Contributors

CONCLUSION

The Financial Manager application provides a robust and user-friendly solution for managing personal finances with a focus on asset tracking, goal setting, zakat calculation, and transaction management.

THANK YOU