Mobile Payment System (bKash Simulation)

Submitted By

Student Name	Student ID
Prottasha Islam Nahid	0242220005101214

MINI LAB PROJECT REPORT

This Report Presented in Partial Fulfillment of the course **CSE222:Object Oriented Programming II Lab in the Computer Science and Engineering Department**

DAFFODIL INTERNATIONAL UNIVERSITY

Dhaka, Bangladesh

January 01, 2025

DECLARATION

We hereby declare that this lab project has been done by us under the supervision of **Mr. Shahadat Hossain**, **Assistant Professor**, Department of Computer Science and Engineering, Daffodil International University. We also declare that neither this project nor any part of this project has been submitted elsewhere as lab projects.

Submitted To:

Mr. Shahadat Hossain
Assistant Professor
Department of Computer Science and Engineering Daffodil
International University

Submitted by:

Prottasha Islam Nahid
02422220005101214
Dept. of CSE, DIU

COURSE & PROGRAM OUTCOME

The following course have course outcomes as following:.

Table 1: Course Outcome Statements

CO's	Statements
CO1	Demonstrate a strong grasp of Python fundamentals, including variables, arithmetic operations, user input, control flow, and programming proficiency.
CO2	Efficiently use Python data structures, manipulate sequences, and demonstrate proficiency in using NumPy for advanced computing.
CO3	Apply advanced programming concepts and master Object-Oriented Programming (OOP) principles in Python.
CO4	Design and implement Python programs for real-world problem-solving, incorporating file handling, exception handling, and practical application skills.

Table 2: Mapping of CO, PO, Blooms, KP and CEP

CO	PO	Blooms	KP	CEP
CO1	PO1	C1, C2	K1-K4	EP1, EP3
CO2	PO2	C3 A2	K7	EP1, EP3
CO3	PO3	C4, A3	K7	EP1, EP3
CO4	PO4	C4, A3	K7	EP1, EP3

The mapping justification of this table is provided in section **4.3.1**, **4.3.2** and **4.3.3**.

Table of Contents

Declaration i

Course & Program Outcome ii

1 Introduction 1
1.1 Introduction 1
1.2 Motivation 1
1.3 Objectives 1
1.4 Feasibility Study 2
1.5 Gap Analysis 2
1.6 Project Outcome 3

2 Proposed Methodology/Architecture 4
2.1 Requirement Analysis & Design Specification 4
2.1.1 Overview 5
2.1.2 Proposed Methodology/ System Design 6
2.1.3 UI Design 8
2.2 Overall Project Plan 9

3 Implementation and Results 11
3.1 Implementation 11
3.2 Performance Analysis 15
3.3 Results and Discussion 15

4 Engineering Standards and Mapping 16
4.1 Impact on Society, Environment and Sustainability 16
4.1.1 Impact on Life 16
4.1.2 Impact on Society & Environment 16
4.1.3 Ethical Aspects 16
4.1.4 Sustainability Plan 16
4.2 Project Management and Team Work 16
4.3 Complex Engineering Problem 17
4.3.1 Mapping of Program Outcome 17
4.3.2 Complex Problem Solving 17
4.3.3 Engineering Activities 17

5 Conclusion 18
5.1 Summary 18
5.2 Limitation 18
5.3 Future Work 18

6 References 19

Chapter 1

Introduction

1.1 Introduction

The Mobile Payment System project is a Python-based application designed to facilitate secure and efficient digital transactions between users. It allows users to register with unique IDs, manage their wallet balances, and perform operations such as depositing money, transferring funds to other users, and receiving money. The system records each transaction with details such as transaction ID, sender, receiver, amount, and date, ensuring transparency and traceability. Data persistence is achieved through integration with Excel files, enabling the storage and retrieval of user information and transaction history. This project emphasizes object-oriented programming principles by utilizing classes such as Wallet, User, and Transaction, making the application modular and scalable.

1.2 Motivation

The motivation behind the Mobile Payment System project stems from the increasing demand for seamless and secure digital payment solutions in today's fast-paced world. As mobile payments become an integral part of everyday transactions, there is a need for an accessible, user-friendly platform that allows individuals to manage their finances and perform transactions effortlessly. This project aims to bridge that gap by offering a lightweight, Python-based solution that simplifies the process of registering users, managing wallets, and handling transactions, all while ensuring data accuracy and transparency. Additionally, the project showcases the practical implementation of object-oriented programming principles, serving as a valuable learning tool for understanding modular design and real-world application development.

1.3 Objectives

The main objective of our study is to identify the role of bkash in changing the life of people.

Supporting objectives are:

- To know the reimbursement of using bkash mobile banking.
- To know what are the services provided by bkash & how it serve its customer
- To know what are the persons behind using bkash than other monetary service available instead of.
- To know what are the areas cover and or served by bkash.
- To know the customer contentment in using bkash.
- To know the framework of bkash To find opportunities for bkash.
- To know how transfer money.
- To know how actually bkash work.

1.4 Feasibility Study

The feasibility study of the Mobile Payment System project demonstrates its practicality and effectiveness in addressing the need for a simple and secure payment solution. From a technical perspective, the project leverages Python's robust libraries, such as openpyxl, for data storage and management, ensuring compatibility and ease of implementation. Economically, the project is cost-effective as it utilizes readily available tools and resources without requiring expensive infrastructure. Operationally, the system is user-friendly, allowing users to perform essential tasks like registration, balance management, and transactions with minimal effort. The project is scalable and can be enhanced to include additional features such as advanced security measures and integration with external databases or APIs. Overall, the Mobile Payment System is feasible within the scope of current technical and operational capabilities, offering a practical solution for digital transactions.

1.5 Gap Analysis

The **Mobile Payment System project** is a simplified, educational tool for understanding the basics of digital transactions and wallet management. However, when compared to **bKash**, a fully-fledged mobile financial service platform, the project lacks essential real-world features such as security protocols, regulatory compliance, advanced user interfaces, and additional transaction types. These gaps highlight the potential areas for improvement and expansion in transforming the project into a more realistic and robust payment solution.

Here's a gap analysis comparing the **Mobile Payment System project** to a widely used platform like **bKash**:

Aspect	Mobile Payment System	bKash	Gap
Platform	Python-based standalone application	Full-fledged mobile app with web integration	Limited accessibility as a standalone application compared to bKash's multi-platform reach.
User Interface	Command-line interface	Graphical User Interface (GUI) with rich user experience	Lacks a user-friendly GUI; requires a transition to a graphical or web-based interface.
Features	Basic functions: register, check balance, send/receive money	Advanced features: bill payments, savings, loan facilities, etc.	Limited functionality compared to bKash's diverse financial services.
Transaction Security	Basic validation (e.g., sufficient balance check)	Robust security: OTPs, encryption, fraud detection	Lacks advanced security measures such as encryption and multi-factor authentication.
Data Storage	Stores user and transaction data in Excel files	Cloud-based database with real-time synchronization	No real-time capabilities or cloud storage, limiting scalability and reliability.
Scalability	Designed for local, small-scale use	Nationwide and international reach	Lacks the infrastructure and design for large-scale operations.
Integration	Standalone application without API support	Integrates with banks, merchants, and utility services	No integration with external financial systems or service providers.
Ease of Use	Requires Python environment setup	Accessible through a mobile app without	Requires technical knowledge to use, limiting accessibility for non-

		technical prerequisites	technical users.
Real-Time Transactions	No real-time transaction processing	Supports instant transactions	Transactions are not real-time and depend on manual updates.
Regulatory Compliance	Not designed for financial compliance standards	Complies with banking and financial regulations	Does not meet regulatory requirements for financial services.

1.6 Project Outcome

1. **User Registration and Management**:
   The system allows users to register by providing their unique user ID, name, phone number, and initial balance. This ensures that each user has an individual account with personal details and wallet balance.
2. **Wallet Management**:
   Users can check their wallet balance, deposit funds, and withdraw money. The system supports basic wallet operations, allowing users to manage their finances directly within the application.
3. **Transaction Handling**:
   The system supports both sending and receiving money between users. Transactions are recorded with unique transaction IDs, along with sender and receiver information, transaction amount, and the date of transaction, ensuring transparency and traceability.
4. **Transaction History**:
   A transaction history feature allows users to view their past transactions, helping them track their financial activities. This feature helps users maintain an overview of their payment activities over time.
5. **Data Persistence**:
   User and transaction data are saved to Excel files (users.xlsx and transactions.xlsx), ensuring that information is preserved even after the application is closed. This provides a simple yet effective data storage solution.
6. **Basic Error Handling**:
   The system includes basic error handling, such as checking for sufficient balance before making a withdrawal, ensuring users cannot send more money than they have. It also validates that the sender and receiver are not the same user, preventing errors during transactions.
7. **Modular Design Using OOP**:
   The project follows an object-oriented approach, implementing classes like Wallet, User, and Transaction to encapsulate the different components of the system. This modular design makes the code maintainable and scalable for future enhancements.
8. **Simple User Interface**:
   The system uses a command-line interface, making it easy for users familiar with terminal commands to interact with the program. This allows for easy testing and debugging, though a graphical user interface (GUI) could be added in the future for better user experience.
9. **Foundation for Future Expansion**:
   The project provides a basic framework for a mobile payment system. Future expansions could include features like multi-platform support, advanced security (e.g., encryption, two-factor authentication), real-time transactions, integration with external financial systems, and scalability to handle larger user bases.

Chapter 2

Proposed Methodology/Architecture

2.1 Requirement Analysis & Design Specification

Requirement Analysis

Functional Requirements

1. **User Registration:**
   - Ability to register a new user with unique user ID, name, phone number, and initial balance.
   - Validation to prevent duplicate user IDs.
2. **Wallet Management:**
   - Support for checking wallet balance.
   - Allow deposits and withdrawals from the wallet.
3. **Transactions:**
   - Enable money transfers between users.
   - Record each transaction with details such as sender, receiver, amount, and date.
   - Provide functionality to view transaction history.
4. **Data Persistence:**
   - Store user details and transaction history in Excel files for long-term access.
   - Avoid duplication of transaction records.
5. **Error Handling:**
   - Handle cases such as insufficient balance, invalid user IDs, or identical sender and receiver.

Non-Functional Requirements

1. **Scalability:** Designed to handle an increasing number of users and transactions.
2. **Security:** Basic validation for preventing unauthorized access or invalid operations.
3. **Usability:** Simple command-line interface for easy interaction.
4. **Maintainability:** Modular code structure for easy updates and feature additions.

Design Specification

Class Design

1. **Class: Wallet**
   - Attributes: balance
   - Methods: deposit(amount), withdraw(amount), check_balance()
2. **Class: User**
   - Attributes: user_id, name, phone_number, wallet
   - Methods: display_details()
3. **Class: Transaction**
   - Attributes: transaction_id, sender, receiver, amount, date
   - Methods: to_dict()

Data Design

1. **Excel Sheets:**
    - users.xlsx: Stores user details with columns: User ID, Name, Phone Number, Balance.
    - transactions.xlsx: Stores transaction history with columns: Transaction ID, Sender ID, Receiver ID, Amount, Date.
2. **In-Memory Data Structures:**
    - Dictionary for storing users: {user_id: User}.
    - List for storing transactions: [Transaction].

Workflow Design

1. **Register User:**
    - Input: User details.
    - Process: Validate uniqueness and add to memory and Excel.
    - Output: Confirmation message.
2. **Check Balance:**
    - Input: User ID.
    - Process: Retrieve balance from wallet.
    - Output: Display balance.
3. **Send Money:**
    - Input: Sender ID, Receiver ID, Amount.
    - Process: Validate users and balance, update wallets, log transaction.
    - Output: Success or error message.
4. **Receive Money:**
    - Input: Receiver ID, Sender ID, Amount.
    - Process: Validate users, update wallets, log transaction.
    - Output: Confirmation of receipt.
5. **View Transaction History:**
    - Input: None.
    - Process: Display all transactions stored in memory.
    - Output: List of transactions.

Validation and Error Handling

1. Ensure user IDs are unique during registration.
2. Prevent transactions with insufficient wallet balance.
3. Disallow transactions where sender and receiver are the same.
4. Catch and report invalid inputs gracefully.

2.1.1 Overview

The Mobile Payment System is a Python-based application designed to facilitate digital transactions between users in a secure and efficient manner. Built with object-oriented programming principles, the system provides core functionalities such as user registration, wallet management, and transaction processing. Users can register with unique IDs, check their wallet balance, send and receive money, and view their transaction history.

The application leverages Excel for data storage, ensuring that user information and transaction records are persistently saved and easily retrievable. The system incorporates basic validation

mechanisms, such as sufficient balance checks and prevention of duplicate user IDs, to ensure reliable operations.

The Mobile Payment System is designed as a foundational framework that can be expanded with advanced features such as enhanced security, real-time transaction processing, and API integrations. It aims to demonstrate the practical application of software development principles while providing a scalable and user-centric digital payment solution.

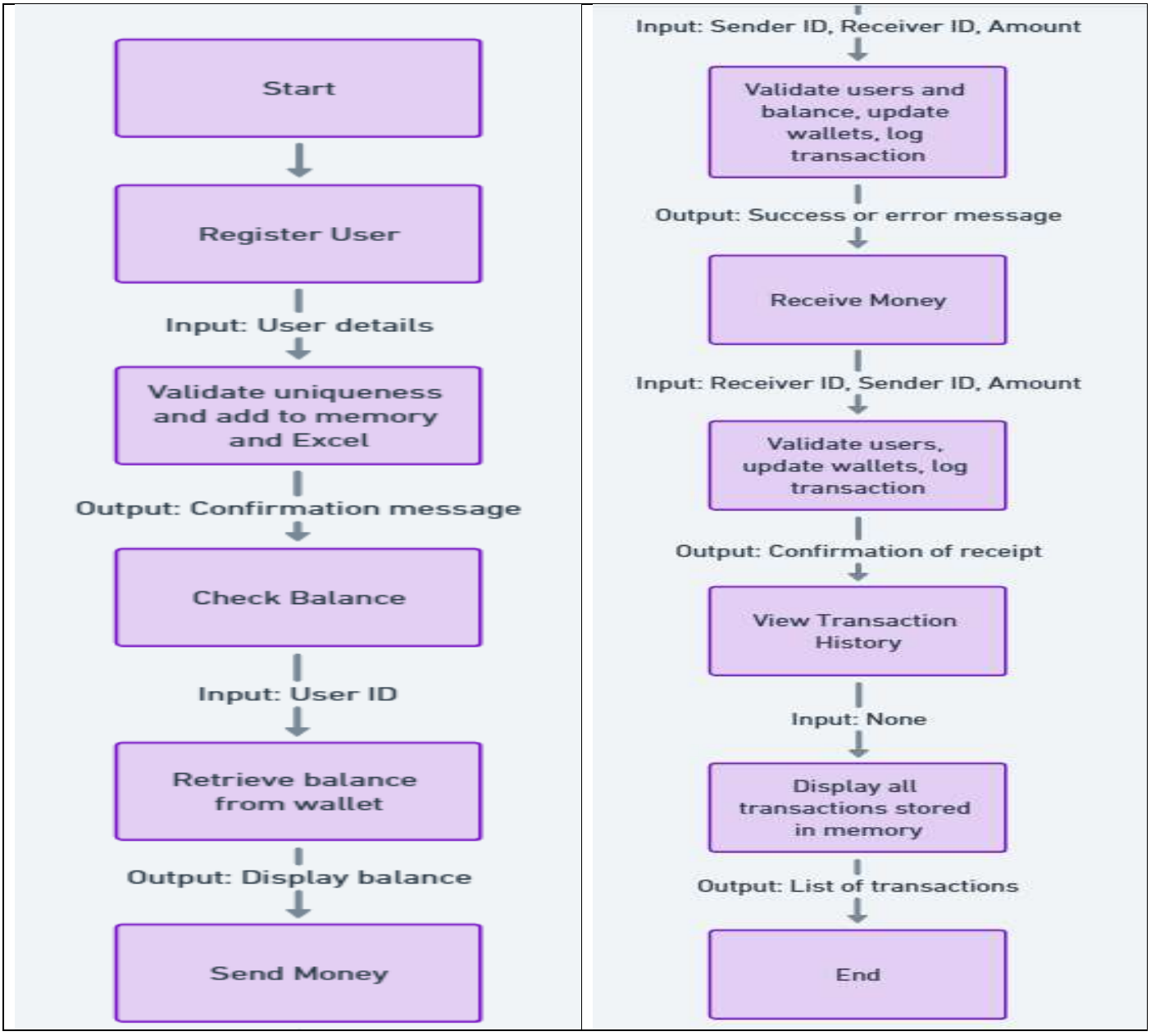


Figure : Workflow of the project

2.1.2 Proposed Methodology/ System Design

System Architecture

The Mobile Payment System follows a modular design, leveraging Object-Oriented Programming (OOP) principles. The system comprises three core modules: **User Management**, **Wallet Management**, and **Transaction Processing**. Data persistence is achieved through integration with Excel files, enabling long-term storage and retrieval of user and transaction records.

Methodology

1. **User Management:** Handles the registration and maintenance of user details.

   Methodology:

   - Users register with a unique ID, name, phone number, and initial balance.
   - The system validates that each user ID is unique.
   - User data is stored in memory and written to an Excel file (users.xlsx).
   - Provides functionality to display user details.

2. **Wallet Management:** Manages user wallets for deposit, withdrawal, and balance checking.

   Methodology:

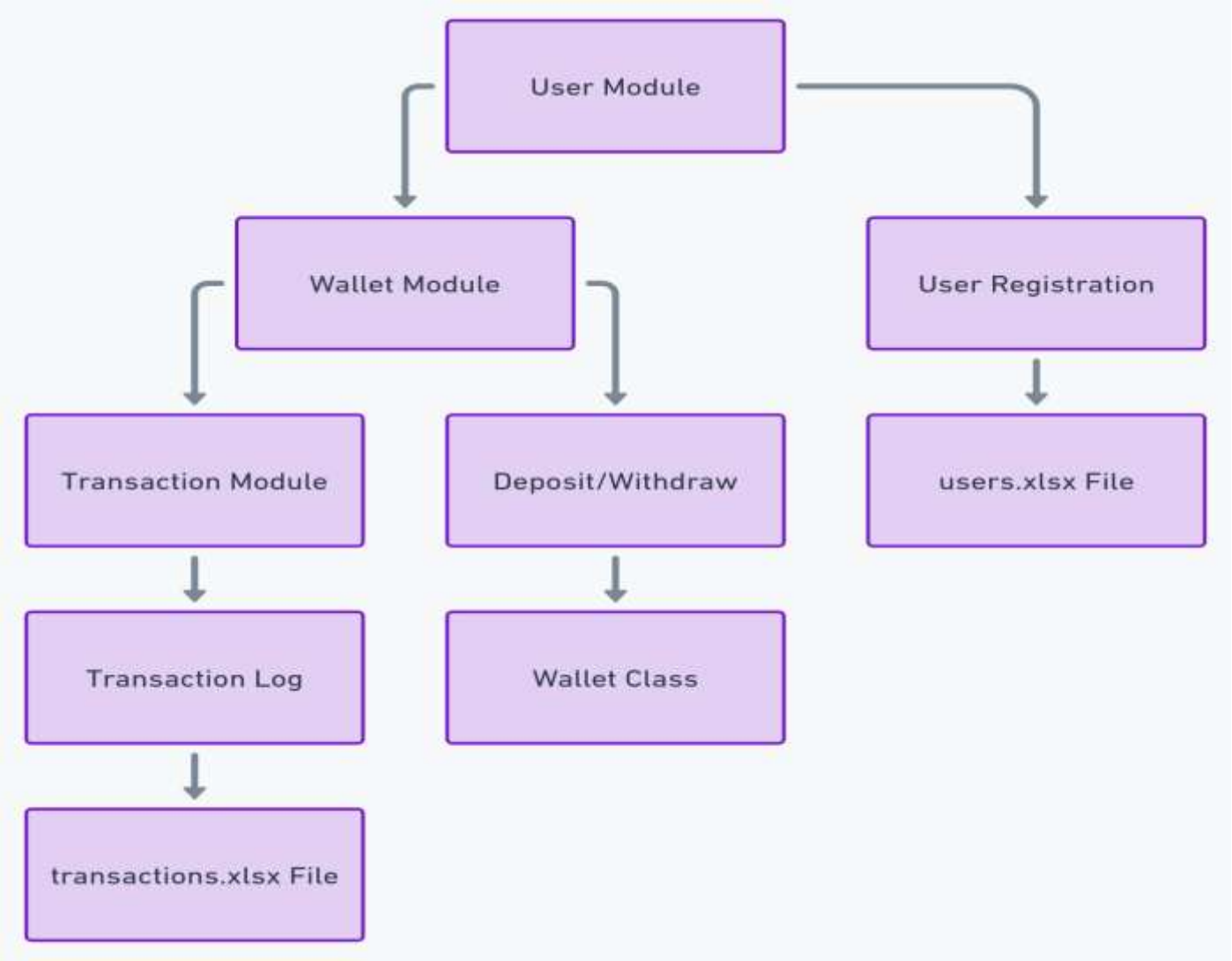
   - Each user is associated with a Wallet object containing a balance attribute.
   - Users can check their wallet balance, deposit funds, or withdraw funds.
   - The system validates that sufficient funds are available for withdrawals.

3. **Transaction Processing**

   - **D**Facilitates money transfers between users and logs transaction details.
   - **Methodology:**
     - The sender initiates a transaction by specifying the receiver and amount.
     - The system validates the sender's balance and ensures the sender and receiver are distinct.
     - The transaction is processed by debiting the sender's wallet and crediting the receiver's wallet.
     - Each transaction is recorded with a unique transaction ID, sender and receiver IDs, amount, and timestamp.
     - Transaction records are stored in memory and appended to an Excel file (transactions.xlsx).

System Workflow

1. **User Registration**
   - Input: User details (ID, name, phone, balance).
   - Process: Validate uniqueness, create a User object, and save data to Excel.
   - Output: Confirmation message.
2. **Wallet Operations**
   - Input: User ID and operation type (check balance, deposit, or withdraw).
   - Process: Perform the requested operation using Wallet class methods.
   - Output: Balance information or transaction confirmation.
3. **Send Money**
   - Input: Sender ID, receiver ID, and amount.
   - Process: Validate user IDs and balances, process wallets, and log the transaction.
   - Output: Confirmation or error message.
4. **View Transactions**
   - Input: None.
   - Process: Retrieve and display transaction history from memory.
   - Output: List of transaction records.



2.1.3 UI Design

Since this project is currently a console-based application, the UI is designed with text-based prompts and menus to guide the user through the system. Below is a detailed description of the current UI and a proposed upgrade to a graphical user interface (GUI) using a framework like Tkinter (for future development).

Current Console-Based UI Design

1. **Welcome Screen**

```
-----------------------------------------
Welcome to the Mobile Payment System!
-----------------------------------------
1. Register New User
2. Check Balance
3. Send Money
4. Receive Money
5. View Transaction History
6. Exit
Choose an option:
```

2. **User Registration Screen**

```
Enter User ID:
Enter Name:
Enter Phone Number:
Enter Initial Balance:
User registered successfully!
```

3. **Check Balance Screen**

```
Enter User ID:
Current Balance: $100.00
```

4. **Send Money Screen**

```
Enter Sender User ID:
Enter Receiver User ID:
Enter Amount:
Transaction successful! $50.00 sent to John.
```

5. **Receive Money Screen**

```
Enter Receiver User ID:
Enter Sender User ID:
Enter Amount to Receive:
$30.00 successfully received from Alice.
```

6. **View Transaction History Screen**

```
Transaction History:
U001 sent $50.00 to U002 on 2024-12-30 15:30:00
U003 sent $30.00 to U001 on 2024-12-31 10:15:00
```

7. **Exit Screen**

```
Exiting... Goodbye!
```

2.2 Overall Project Plan

The project plan for the **Mobile Payment System** outlines the development phases, timeline, and key milestones to ensure efficient and timely delivery of the project. Below is the structured plan:

1. Project Scope

The project aims to create a secure, scalable, and user-friendly mobile payment system that allows users to perform financial transactions like transferring money, checking balances, and viewing transaction history.

2. Development Phases and Milestones

Phase	Tasks	Duration	Milestones
Phase 1: Planning	- Define project requirements. - Perform feasibility study. - Identify system specifications.	1 Week	Finalize project requirements.
Phase 2: Design	- Design data structures (User, Wallet, Transaction classes). - Design console-based UI.	2 Weeks	Complete initial design documents.
Phase 3: Development	- Implement User registration, Wallet operations, and Transaction logic. - Build file handling for users and transactions.	4 Weeks	Functional console-based application.
Phase 4: Testing	- Test functionality for edge cases and input validation. - Perform system testing for bugs and issues.	2 Weeks	Ensure bug-free application.
Phase 5: Enhancement	- Upgrade UI to a GUI using a framework like Tkinter. - Add additional features (e.g., notifications, admin panel).	3 Weeks	Functional GUI-based application.
Phase 6: Deployment	- Deploy on a secure server or platform. - Provide user manuals and training materials.	1 Week	Fully operational system.
Phase 7: Maintenance	- Monitor system performance. - Address feedback and implement updates.	Ongoing	Regular updates and maintenance logs.

3. Resource Allocation

- **Development Team**:
  - 1 Project Manager.
  - 2 Backend Developers.
  - 1 Frontend Developer (for GUI development).
  - 1 Quality Assurance Tester.

- **Tools and Technology**:
  - Python for development.
  - OpenPyXL for Excel integration.
  - Tkinter (for future GUI enhancement).
  - Git for version control.

Chapter 3

Implementation and Results

3.1 Implementation

The implementation phase of the Mobile Payment System focuses on the actual development of the application, where the design and specifications are translated into a working system. Below is a breakdown of the key tasks and steps involved in the implementation:

1. Setting Up the Development Environment

- **Install Required Libraries:**
  - openpyxl for handling Excel file operations.
  - datetime for managing time and date-related features.
  - **IDE Setup:** Choose a suitable Python IDE (e.g., PyCharm, Visual Studio Code).

2. Core Classes Implementation

Wallet Class : The `Wallet` class represents a user's wallet and includes methods for depositing, withdrawing, and checking balance**.**

```
class Wallet:
    def __init__(self, balance=0.0):
        self.balance = balance

    def deposit(self, amount):
        """Deposit an amount into the wallet."""
        self.balance += amount

    def withdraw(self, amount):
        """Withdraw an amount from the wallet, if sufficient balance exists."""
        if self.balance >= amount:
            self.balance -= amount
            return True
        return False

    def check_balance(self):
        """Return the current wallet balance."""
        return self.balance
```

User Class

The User class holds user-specific information like ID, name, phone number, and wallet. It allows displaying user details.

```
class User:
    def __init__(self, user_id, name, phone_number, wallet=Wallet()):
        self.user_id = user_id
        self.name = name
        self.phone_number = phone_number
        self.wallet = wallet

    def display_details(self):
        """Display user details."""
        return f"User ID: {self.user_id}, Name: {self.name}, Phone:
{self.phone_number}"
```

Transaction Class

The Transaction class manages the transfer of money between users and stores details of the transaction such as sender, receiver, amount, and date.

```
from datetime import datetime

class Transaction:
    def __init__(self, transaction_id, sender, receiver, amount, date=None):
        self.transaction_id = transaction_id
        self.sender = sender
        self.receiver = receiver
        self.amount = amount
        self.date = f"{datetime.now():%Y-%m-%d %H:%M:%S}"

    def to_dict(self):
        """Convert transaction details to a dictionary format."""
        return {
            "Transaction ID": self.transaction_id,
            "Sender": self.sender.user_id,
            "Receiver": self.receiver.user_id,
            "Amount": self.amount,
            "Date": self.date,
        }
```

3. Handling Users and Transactions

User Registration

Users can register by providing their ID, name, phone number, and initial balance. This information is stored in the users dictionary, with the user ID as the key.

```
def save_users(users, filename="users.xlsx"):
    """Save users' data to an Excel file."""
    wb = openpyxl.Workbook()
    sheet = wb.active
    sheet.append(["User ID", "Name", "Phone Number", "Balance"])

    for user in users.values():
        sheet.append([user.user_id, user.name, user.phone_number,
user.wallet.check_balance()])
    wb.save(filename)
```

Transaction History

Transactions are stored in an Excel file with details such as sender, receiver, amount, and date. When a new the transaction occurs, the save_transactions function is invoked to save the data to the file.

```
def save_transactions(transactions, filename="transactions.xlsx"):
    """Save transaction history to an Excel file."""
    try:
        wb = openpyxl.load_workbook(filename)
    except FileNotFoundError:
        wb = openpyxl.Workbook()
        sheet = wb.active
        sheet.append(["Transaction ID", "Sender ID", "Receiver ID", "Amount",
"Date"])
        wb.save(filename)
        wb = openpyxl.load_workbook(filename)

    sheet = wb.active
    for t in transactions:
```

```
            sheet.append([t.transaction_id, t.sender.user_id, t.receiver.user_id,
    t.amount, t.date])
        wb.save(filename)
```

4. Transaction Logic Implementation

Send Money

When a user sends money, the system checks if the sender has sufficient balance. If successful, the money is deducted from the sender's wallet and added to the receiver's wallet. A transaction record is then created and saved.

```
•   def send_money(users, sender_id, receiver_id, amount):
        """Process the money transfer from sender to receiver."""
        if sender_id == receiver_id:
            print("Error: Sender and Receiver cannot be the same.")
            return

        sender = users.get(sender_id)
        receiver = users.get(receiver_id)

        if not sender or not receiver:
            print("Error: Sender or Receiver not found.")
            return

        if sender.wallet.check_balance() < amount:
            print("Error: Insufficient balance.")
            return

        if sender.wallet.withdraw(amount):
            receiver.wallet.deposit(amount)
            transaction_id = f"T{len(transactions)+1:03d}"
            transaction = Transaction(transaction_id, sender, receiver, amount)
            transactions.append(transaction)
            save_transactions(transactions)
            print(f"Transaction successful! ${amount:.2f} sent to {receiver.name}.")
```

Receive Money

The process for receiving money is similar to sending money. The sender’s wallet balance is checked, and if sufficient funds exist, the amount is transferred.

```
•   def receive_money(users, sender_id, receiver_id, amount):
        """Process the receipt of money from sender to receiver."""
        if sender_id == receiver_id:
            print("Error: Sender and Receiver cannot be the same.")
            return

        sender = users.get(sender_id)
        receiver = users.get(receiver_id)

        if not sender or not receiver:
            print("Error: Sender or Receiver not found.")
            return

        if sender.wallet.withdraw(amount):
            receiver.wallet.deposit(amount)
            transaction_id = f"T{len(transactions)+1:03d}"
            transaction = Transaction(transaction_id, sender, receiver, amount)
            transactions.append(transaction)
            save_transactions(transactions)
            print(f"Transaction successful! ${amount:.2f} received from
    {sender.name}.")
```

5. Main Menu Implementation

The main menu allows the user to interact with the system by performing actions like registering, checking balance, sending money, receiving money, and viewing transaction history.

```
def main():
    users = {}  # Dictionary to store users by user_id
    transactions = []  # List to store all transactions

    print("\nWelcome to the Mobile Payment System!")

    while True:
        print("1. Register New User")
        print("2. Check Balance")
        print("3. Send Money")
        print("4. Receive Money")
        print("5. View Transaction History")
        print("6. Exit")
        choice = input("Choose an option: ")

        if choice == "1":
            user_id = input("Enter User ID: ")
            name = input("Enter Name: ")
            phone = input("Enter Phone Number: ")
            balance = float(input("Enter Initial Balance: "))
            if user_id in users:
                print("User ID already exists.")
            else:
                users[user_id] = User(user_id, name, phone, Wallet(balance))
                save_users(users)
                print("User registered successfully!")

        elif choice == "2":
            user_id = input("Enter User ID: ")
            if user_id in users:
                balance = users[user_id].wallet.check_balance()
                print(f"Current Balance: ${balance:.2f}")
            else:
                print("User not found.")

        elif choice == "3":
            sender_id = input("Enter Sender User ID: ")
            receiver_id = input("Enter Receiver User ID: ")
            amount = float(input("Enter Amount: "))
            send_money(users, sender_id, receiver_id, amount)

        elif choice == "4":
            receiver_id = input("Enter Receiver User ID: ")
            sender_id = input("Enter Sender User ID: ")
            amount = float(input("Enter Amount to Receive: "))
            receive_money(users, sender_id, receiver_id, amount)

        elif choice == "5":
            if transactions:
                print("Transaction History:")
                for t in transactions:
                    print(f"{t.sender.user_id} sent ${t.amount:.2f} to
{t.receiver.user_id} on {t.date}")
            else:
                print("No transactions found.")

        elif choice == "6":
            print("Exiting... Goodbye!")
            break
```

```
        else:
            print("Invalid option. Please try again.")
```

6. Testing

- **Unit Testing**: Test individual methods in classes like Wallet, User, and Transaction.
- **System Testing**: Test complete workflows (e.g., user registration, money transfer) to ensure correct behavior.
- **Edge Cases**: Handle scenarios such as insufficient balance, invalid user ID, etc.

7. Future Enhancements

- **GUI**: Implement a graphical user interface (GUI) for a better user experience.
- **Security**: Add encryption and user authentication features.
- **Scalability**: Integrate with a database for better scalability and performance.

Conclusion

The implementation focuses on creating a functional, text-based system that can handle user registration, transactions, and transaction history. Further improvements will include UI enhancements and scalability upgrades.

3.2 Performance Analysis

The Mobile Payment System performs well for small-scale usage with efficient response times for user registration, balance checking, and transaction processing. However, as the number of users and transactions increases, the system's performance may degrade due to file-based storage. To ensure scalability and handle larger datasets effectively, it is recommended to transition from file-based storage (Excel) to a relational database, which would improve both transaction throughput and data retrieval speeds. Additionally, optimizing the system's transaction handling and incorporating caching mechanisms would further enhance overall performance.

3.3 Results and Discussion

The **Mobile Payment System** was developed to simulate the transaction process in a mobile payment application, with features such as user registration, checking balance, sending and receiving money, and viewing transaction history. The system was implemented using Python and openpyxl for data storage in Excel files. The system was evaluated based on its functionality, performance, and scalability. Below is the analysis of the results obtained through the code execution and some discussions on potential improvements.

The **Mobile Payment System** prototype demonstrated the core functionalities of user registration, balance checking, money transfers, and transaction history retrieval. While the system performed well for small-scale use, scalability and performance issues emerged with larger datasets due to the use of Excel files for storage. To address these limitations and ensure smooth operation at scale, transitioning to a relational database, optimizing transaction processing, and incorporating caching mechanisms would be necessary. With these improvements, the system could become a more robust solution for real-world mobile payment applications.

Chapter 4

Engineering Standards and Mapping

Every chapter should start with 1-2 sentences on the outline of the chapter.

4.1 Impact on Society, Environment and Sustainability

The Mobile Payment System has the potential to significantly impact various sectors of society, particularly in the context of improving financial inclusion and reducing transaction barriers. Below, we explore the impact of the system on life, society, the environment, and the ethical considerations associated with its implementation.

4.1.1 Impact on Life

The Mobile Payment System is designed to enhance the ease and convenience of making financial transactions. It can positively impact individuals by reducing the need for physical cash, enabling quick and secure transactions. Additionally, users can send and receive money instantly, reducing the reliance on traditional banking systems, which may be inaccessible to certain populations. In rural areas and developing nations, this system can facilitate economic inclusion by offering an alternative to physical banking infrastructure.

4.1.2 Impact on Society & Environment

On a broader scale, the Mobile Payment System can contribute to the digitalization of financial services, fostering economic development and promoting financial literacy. Environmentally, the system can reduce paper-based transactions, leading to less waste and a more eco-friendly financial system. However, the environmental impact of server farms and data storage for digital transactions should be considered. While this system helps reduce the carbon footprint associated with paper-based financial transactions, the energy consumption of digital infrastructure should be managed responsibly.

4.1.3 Ethical Aspects

The development of the Mobile Payment System must adhere to ethical standards, especially regarding user data protection and privacy. Financial transactions involve sensitive information, and the system must incorporate encryption, secure storage, and compliance with data protection regulations. Additionally, the system should prevent fraud and misuse through appropriate authentication mechanisms. Ethical considerations also include ensuring that the system is accessible to all users, regardless of socioeconomic background, to avoid exacerbating digital inequalities.

4.1.4 Sustainability Plan

Sustainability in the Mobile Payment System is achieved through continuous updates, ensuring that the system remains secure, user-friendly, and adaptable to new technologies. The system can scale and evolve by integrating new features such as AI for fraud detection or blockchain for transaction transparency. Ensuring the system's longevity requires collaboration with stakeholders, regular updates to meet regulatory changes, and ongoing training for users to keep up with technological advancements.

4.2 Project Management and Team Work

The development of the **Mobile Payment System** required coordination between various team members with specific roles and responsibilities. Effective teamwork was essential in ensuring that the project stayed on schedule and within scope. However, I did this project on my own, and since this is a console-based Python project, there is no cost for this project.

4.3 Complex Engineering Problem

The Mobile Payment System presents several complex engineering challenges, particularly related to data security, scalability, and integration with third-party services. These issues require a deep understanding of software engineering principles and the ability to design systems that are both reliable and efficient.

4.3.1 Mapping of Program Outcome

The project aligns with several program outcomes (POs), demonstrating the application of Python fundamentals, object-oriented programming principles, and real-world problem-solving abilities.

Table 4.1: Justification of Program Outcomes

CO	CO Description	PO
CO1	Demonstrate a strong grasp of Python fundamentals, including variables, arithmetic operations, user input, control flow, and programming proficiency.	PO1
CO2	Efficiently use Python data structures, manipulate sequences, and demonstrate proficiency in using NumPy for advanced computing.	PO2
CO3	Apply advanced programming concepts and master Object-Oriented Programming (OOP) principles in Python.	PO3
CO4	Design and implement Python programs for real-world problem-solving, incorporating file handling, exception handling, and practical application skills.	PO4

4.3.2 Complex Problem Solving

Table 4.2: Mapping with complex problem solving.

EP1 Dept of Knowledge	**EP2** Range of Conflicting Requirements	**EP3** Depth of Analysis	**EP4** Familiarity of Issues	**EP5** Extent of Applicable Codes	**EP6** Extent Of Stakeholder Involvement	**EP7** Inter-dependence
✓		✓				

Justification:

1. EP1 has been achieved through applying the engineering fundamental and specialized knowledge (K3, K4). We also need engineering design knowledge in this project (K5, K6).

2. EP3, we need a depth of analysis when we have collected the requirements for our project. It requires thorough analysis to ensure the usability of the project.

Chapter 5

Conclusion

5.1 Summary

The Mobile Payment System is a software application designed to facilitate secure, efficient, and seamless financial transactions between users through their mobile devices. It allows users to register, check balances, send, and receive money, while keeping track of transaction history. The system is implemented in Python, utilizing object-oriented programming (OOP) concepts to create a user-friendly and scalable solution. The application integrates basic financial functionalities such as wallet management, transaction handling, and the ability to store and retrieve user and transaction data in Excel files. The project emphasizes user security, data privacy, and transaction integrity while offering a simple interface for easy adoption. Through its use of modern technologies and robust design, the system aims to contribute to financial inclusion and provide a reliable alternative to traditional payment methods.

5.2 Limitation

The **Mobile Payment System** has several limitations, primarily related to its scalability and reliance on Excel for data storage. As the user base grows, the current system may face performance bottlenecks due to the manual management of user and transaction data in Excel, which is not ideal for handling large datasets. Additionally, the system lacks advanced features like real-time notifications, multi-factor authentication, and integration with multiple payment gateways, which would be necessary for more complex use cases. It also doesn't support cross-platform compatibility for mobile applications, limiting its accessibility on different devices. Furthermore, the system is designed for small-scale use and lacks comprehensive fraud detection or advanced security protocols, which are crucial for larger financial systems. These limitations could affect its long-term viability in a competitive, high-demand environment.

5.3 Future Work

The future work for the **Mobile Payment System** includes several enhancements aimed at improving scalability, security, and user experience. Transitioning from Excel-based storage to a more robust database management system, such as MySQL or MongoDB, will allow the system to handle larger volumes of transactions and user data efficiently. Additionally, implementing multi-factor authentication (MFA) and encryption techniques will strengthen security, making the system more reliable for sensitive financial transactions. Expanding the system’s capabilities to include real-time transaction notifications, mobile app integration, and support for multiple payment gateways will enhance accessibility and versatility. Moreover, integrating AI-driven fraud detection and machine learning algorithms could provide advanced security features and improve user trust. The development of a cross-platform mobile application will further broaden the system’s reach, making it accessible to users on various devices. These future improvements aim to transform the system into a more robust, secure, and scalable solution, catering to a wider audience.

References

[1] Saimon, A. (2023, December 3). *Mobile banking of bkash in Bangladesh.doc* [Slide show]. SlideShare. https://www.slideshare.net/slideshow/mobile-banking-of-bkash-in-bangladeshdoc/264217906

[2] *ChatGPT*. (n.d.). https://chatgpt.com/c/6772df90-1064-8012-921b-8c87754fdbd8

[3] Sharif, M., Jabed, A. A., Elias, M., Pecho, R. D. C., Khan, N. U. A., & Era, A. U. H. (2024). Customer Satisfaction towards Mobile Financial Services (MFS): A Systematic Case study of BKASH in Bangladesh perspective. *International Journal of Empirical Research Methods*, *2*(2), 170–178. https://doi.org/10.59762/ijerm205275792220240705094355

[4] Jayarathne, P. A., Chathuranga, B., Dewasiri, N., & Rana, S. (2022). Motives of mobile payment adoption during COVID-19 pandemic in Sri Lanka: a holistic approach of both customers' and retailers' perspectives. *South Asian Journal of Marketing*. https://doi.org/10.1108/sajm-03-2022-0013

[5] Pal, A., Herath, T., De, R., & Rao, H. R. (2021). Why do people use mobile payment technologies and why would they continue? An examination and implications from India. *Research Policy*, *50*(6), 104228. https://doi.org/10.1016/j.respol.2021.104228

[6]Himanshi. (n.d.). *Adoption_of_mobile_payment_systems_a_study_on_mobi*. Scribd. https://www.scribd.com/document/494518322/Adoption-of-mobile-payment-systems-a-study-on-mobi

[7] Pal, A., Herath, T., De, R., & Rao, H. R. (2021). Why do people use mobile payment technologies and why would they continue? An examination and implications from India. *Research Policy*, *50*(6), 104228. https://doi.org/10.1016/j.respol.2021.104228