MANIPAL INSTITUTE OF TECHNOLOGY

MANIPAL

(A constituent unit of MAHE, Manipal)

DISTRIBUTED SYSTEMS (CSE -3261) MINI PROJECT REPORT ON

MoneyFlo
Distributed Banking Management Application

SUBMITTED TO

Department of Computer Science & Engineering

by

Name	**Registration Number**	**Roll Number**	**Sem/Sec**
Akshay Gupta	200905040	04	VI/C
Kaustubh Pandey	200905142	24	VI/C
Rajpreet Lal Das	200905052	08	VI/C
Shubham Srivastava	200905152	26	VI/C

Dr. P.B. Shanthi
(Assistant Professor)

Ms. Anjana S
(Assistant Professor)

Name & Signature of Evaluator 1

Name & Signature of Evaluator 2

(Jan 2023 - May 2023)

TABLE OF CONTENTS

Chapter 1. Introduction
- 1.1 Overview
- 1.2 Motivation

Chapter 2. Background Theory
- 2.1 Abstract
- 2.2 Theory
  - 2.2.1 Message Passing and Twilio API
  - 2.2.2 Scalability
  - 2.2.3 Fault-tolerance
  - 2.2.4 Consistency
  - 2.2.5 Security

Chapter 3. Methodology & Implementation

Chapter 4. Results and Screenshots

Chapter 5. Conclusion and Future Enhancements

Chapter 6. References

1. INTRODUCTION

1.1 Overview

In the world of banking and finance, transaction SMS (Short Message Service) is a crucial tool for both customers and banks. It is a way for banks to provide real-time updates and information to their customers regarding transactions on their accounts, and for customers to stay informed and in control of their finances.

Furthermore, transaction SMS can be used to send important notifications and alerts to customers. For example, customers can be notified when their account balance falls below a certain threshold, or when a payment is due. This helps customers to avoid late payments and overdraft fees, while also promoting responsible financial management.

MoneyFlo is a distributed system Application that enables users to open accounts, deposit and withdraw money, and receive SMS notifications using the Twilio API. The system consists of two main components: the server-side code written in Python, and the MySQL database used to store user data. MoneyFlo is designed to be scalable, fault-tolerant, and consistent.

1.2 Motivation

One of the key motivating factors is scalability. As banks grow and expand their operations, the demand for a more scalable and efficient system increases. A distributed system allows banks to expand their network without compromising on the performance of the system. This ensures that the bank can handle a larger volume of transactions, serve more customers, and provide better services.

Security is also a key motivating factor. A distributed system can be more secure compared to a centralized system. With a distributed system, the data is stored in multiple locations, making it harder for hackers to compromise the system. Additionally, a distributed system can provide better data privacy and security, which is critical in the banking industry.

In conclusion, a bank management distributed system offers several advantages over a centralized system. The scalability and security benefits of a distributed system can help banks to provide more efficient and secure services to their customers. As such, there is a growing motivation for banks to invest in a bank management distributed system as they seek to keep up with the evolving needs of their customers and the demands of the modern banking industry.

2. BACKGROUND THEORY

2.1 Abstract

This project proposes the development of a bank management distributed system application using Twilio API. The application will enable banks to provide their customers with real-time updates and information regarding transactions on their accounts through SMS notifications.

The proposed system will be distributed, fault-tolerant, and scalable, allowing banks to handle a larger volume of transactions and serve more customers. The system will be secured using the Twilio API, which provides advanced security features such as two-factor authentication and encrypted communication.

The bank management distributed system application will be designed using the Model-View-Controller (MVC) architecture pattern. The model will be responsible for handling the data storage and retrieval, while the view will handle the user interface. The controller will act as the intermediary between the model and the view, handling the business logic of the application.

The Twilio API will be used to send automated SMS notifications to customers whenever transactions are made on their accounts. This will enable customers to stay informed and in control of their finances. Additionally, the Twilio API will be used to provide secure authentication and communication between the bank management system and customers.

In conclusion, the proposed bank management distributed system application using Twilio API is expected to provide a secure and efficient way for banks to manage their operations and serve their customers. The application will enable banks to provide real-time updates and information to their customers through SMS notifications while ensuring the security of the data and communication.

2.2 Theory

In this section, we will explore the technical aspects of the Banking Management System (BMS) called MoneyFlo. We will discuss the message passing mechanism used in MoneyFlo, and how it leverages the Twilio API to send SMS notifications to users. We will also examine the scalability and fault-tolerance features that enable MoneyFlo to handle a large number of users and transactions. In addition, we will look at the consistency and security mechanisms that ensure the integrity of the system's data and protect it from unauthorized access. Let's dive in and explore the technical underpinnings of MoneyFlo in more detail.

2.2.1 Message Passing and Twilio API

Message passing is an important aspect of BMS, as it enables communication between the application and users. BMS uses the Twilio API to send SMS notifications to users. Twilio is a cloud communication platform that allows developers to build communication features into their applications, such as SMS, voice, and video.

The Twilio API is a RESTful API, which means that it uses HTTP requests to interact with the Twilio platform. REST stands for Representational State Transfer, and is a design pattern for building web services that are scalable, flexible, and easy to maintain. REST APIs use HTTP methods such as GET, POST, PUT, and DELETE to perform operations on resources.

In MoneyFlo, the Twilio API is used to send SMS notifications to users when they open an account or perform a transaction. To use the Twilio API, developers need to sign up for a Twilio account and obtain an API key and token. Developers can then use the Twilio Python library to interact with the Twilio API and send SMS notifications.

2.2.2 Scalability

Although MoneyFlo is a single-node application, it can handle a large number of users and transactions. MoneyFlo is designed to handle concurrent requests using threads, ensuring that multiple users can interact with the system simultaneously. This design allows MoneyFlo to scale vertically, by increasing the resources allocated to the server. For example, MoneyFlo can run on a larger server with more CPU cores and memory, allowing it to handle more users and transactions.

2.2.3 Fault-tolerance

MoneyFlo is designed to keep running even in the face of failures. Although MoneyFlo is a single-node application, it can use techniques such as data replication and backup mechanisms to improve fault tolerance. MoneyFlo can replicate the MySQL database to a secondary server, ensuring that data is not lost in the event of a hardware failure or network outage. MoneyFlo can also use backups to restore the system to a previous state in the event of data corruption or other failures.

2.2.4 Consistency

MoneyFlo is designed to ensure consistency of data within the MySQL database. The system uses a strong consistency model to ensure that all transactions are processed in a serializable manner. MoneyFlo uses locking mechanisms to prevent concurrent access to the same data, ensuring that only one transaction can modify a given piece of data at a time.

2.2.5 Security

MoneyFlo is designed to be secure, protecting user data from unauthorized access and ensuring the integrity of data. MoneyFlo uses encryption to protect sensitive data such as passwords and user information. MoneyFlo uses access control mechanisms to ensure that only authorized users can access the system. MoneyFlo also uses monitoring and logging mechanisms to detect and respond to security threats.

3. METHODOLOGY AND IMPLEMENTATION

The methodology for the distributed system mini project will involve the following steps:

- Design the architecture for the MoneyFlo Application using distributed system concepts such as message passing, fault tolerance, scalability, and consistency.
- Develop the server-side code in Python and the Twilio Python library for SMS notifications.
- Set up a MySQL database to store user data and integrate it with the server-side code.
- Deploy the application on a single-node server and test its performance and scalability.
- Evaluate the fault tolerance and consistency of the system by simulating failures and testing the backup and replication mechanisms.
- Ensure the security of the system by implementing access controls, encryption, monitoring, and logging mechanisms.

Implementation

- Use a three-tier architecture with a presentation layer, application layer, and data layer:

  This project can be enhanced by utilizing a RESTful API to streamline communication between the client-side and server-side code, making the application more efficient and easier to maintain. Additionally, message passing can be utilized to send SMS notifications to users using the Twilio API, providing an effective and reliable means of communication for important updates or alerts. Storing user data in a MySQL database can ensure that information is securely stored and easily accessible when required.

- Develop the server-side code in Python and the Twilio Python library for SMS notifications:

  To improve the functionality of the MoneyFlo Application, the Twilio Python library can be used to send SMS notifications to users, providing an efficient and reliable means of

communication for important updates or alerts. Additionally, the logic for opening accounts, depositing and withdrawing money, and sending SMS notifications can be implemented to create a comprehensive system that provides users with an easy and convenient way to manage their accounts.

```
# functions to handle transactions
def sendSMS(connection, acc_num, cat, amount=0):
    account_sid = 'AC6d991f3907307971bb69e112e20d9219'
    auth_token = 'c01cbb7123c89e5ebdbfe21959116aae'
    to_num = '+91'+getDetail(connection, "customers",
                             ("acc_num", acc_num, 'int'), "phone_num")
    client = Client(account_sid, auth_token)
    f_name = getDetail(connection, "customers",
                       ("acc_num", acc_num, 'int'), "f_name")
    l_name = getDetail(connection, "customers",
                       ("acc_num", acc_num, 'int'), "l_name")
    message = client.messages.create(
        from_='+16075233189',
        body='Dear ' + str(f_name) + " " + str(l_name) + ', your account number ' +
        str(acc_num) + " is " + str(cat) + str(amount) + ".",
        to=to_num
    )
```

Fig.1: Server Side Code for sending SMS

- Set up a MySQL database to store user data and integrate it with the server-side code:

  To create a robust and efficient application, it is essential to install MySQL on the server and create a dedicated database for the application. Once the database is created, the schema can be defined, and tables can be created for storing user data such as account details, transaction history, and personal information. Integration with the server-side code can be achieved by utilizing Python libraries like SQLAlchemy, which provides an easy and efficient way to connect to the database and execute SQL queries. By integrating the database with the server-side code, the application can become more efficient and scalable, ensuring that user data is securely stored and easily accessible when required.

- Implement access controls to ensure that only authorized users can access the system and also implement monitoring and logging mechanisms to detect and respond to security threats.

```
def authenticate(connection, level):

    usr_id = int(input("Enter your id : "))
    pass_word = getpass("Enter your password : ")

    if level == 'admin':
        if usr_id == admin_id and pass_word == admin_passwd:
            return True

    elif level == 'customer':
        p = getDetail(connection, "auth",
                      ("acc_num", usr_id, 'int'), "password")
        if pass_word == p:
            return True
    return False
```

Fig.2: Code for Authentication and Security

By incorporating these features, the banking management system can become more efficient, reliable, and user-friendly, improving the experience for both bank employees and customers.

Table Information

The system uses 3 tables: CUSTOMERS, AUTH and TRANSACTIONS. The CUSTOMERS table contains details about each customer like their name, address, phone number, aadhar number, whether they want SMS,etc with an auto incrementing field for account number which acts as the primary key. As it is auto incrementing, every customer will have a unique account number The CUSTOMERS table however does not contain passwords. The passwords are instead stored in the AUTH table along with a foreign key to account number in CUSTOMERS. While it could have been kept in the CUSTOMERS table, it is better to separate it. By doing this, we have the ability to encrypt the CUSTOMERS table and decrypt only if a correct password for the corresponding account number is provided. The AUTH table could use a different encryption than the CUSTOMERS table making breaking into the system even harder. The TRANSACTIONS table contains a list of all transactions that have taken place. It has fields for

type (whether credit or debit), amount, date as well as a foreign key to account number and similar to CUSTOMERS, an auto incrementing field for transaction id that acts as a primary key thus always unique.

```
# query to create customers table
query_create_table_customers = '''CREATE TABLE IF NOT EXISTS customers(
    acc_num int AUTO_INCREMENT,
    f_name varchar(20) NOT NULL,
    l_name varchar(20) NOT NULL,
    aadhar_num varchar(20) NOT NULL,
    dob varchar(10) NOT NULL,
    city varchar(20) NOT NULL,
    area varchar(20) NOT NULL,
    pincode varchar(12) NOT NULL,
    phone_num varchar(12) NOT NULL,
    email_id varchar(30) NOT NULL,
    account_type varchar(10) NOT NULL,
    sms_banking varchar(2),
    current_amount float NOT NULL,
    PRIMARY KEY(acc_num)
);'''

# query to create transactions table
query_create_table_transactions = '''
CREATE TABLE IF NOT EXISTS transactions(
    trans_id int AUTO_INCREMENT,
    acc_num int,
    amount float,
    type varchar(10),
    date varchar(20),
    PRIMARY KEY(trans_id),
    FOREIGN KEY (acc_num) REFERENCES customers(acc_num)
);'''

# query to create auth table
query_create_table_auth = '''
CREATE TABLE IF NOT EXISTS auth(
    acc_num int,
    password varchar(100) NOT NULL,
    FOREIGN KEY (acc_num) REFERENCES customers(acc_num)
);'''
```

Fig.3: Tables with their Keys

4. RESULTS AND SCREENSHOTS

```
➜  Banking-Management-System git:(main) ✗ python3 Bank-Management-System.py
Enter hostname : localhost
Enter username : root
Enter user password :
Enter the name of database : bank_db
agakshay304 root localhost bank_db

***Database is successfully created***


***Connection to MySQL database is successfull***


################################################
##        Tables Initialized - Ready to Go      ##
################################################


                    Login Panel

    1. Admin Login
    2. Customer Login
```

Fig.4: Login Panel

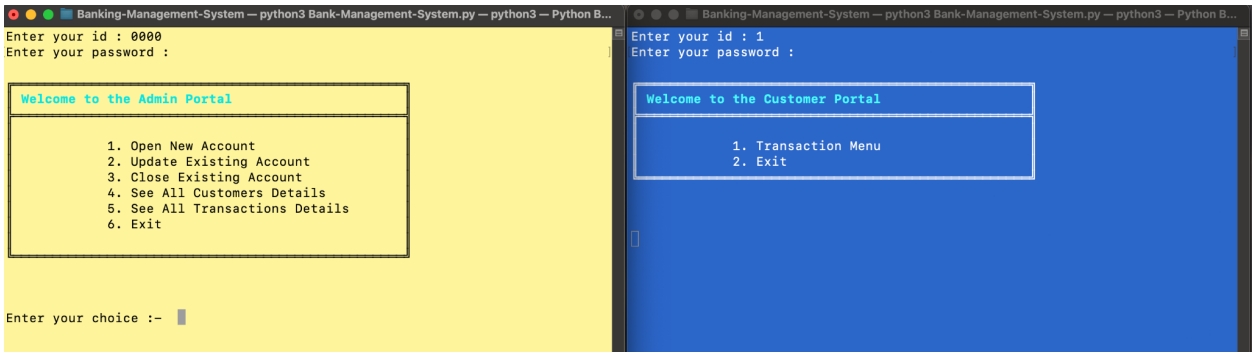

Fig.5: Admin & Customer Portal

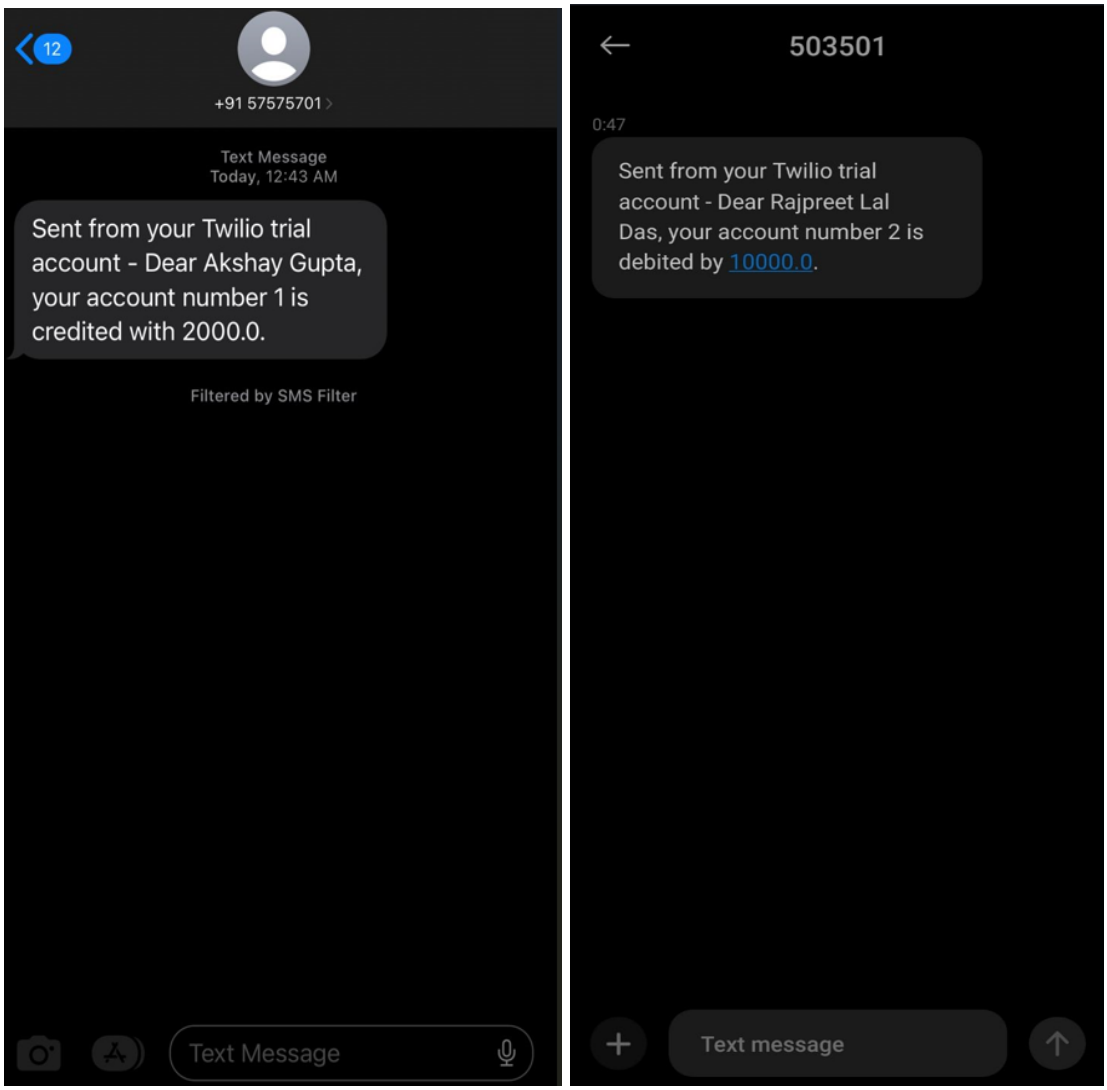

Fig.6: Transaction Messages

5. CONCLUSION AND FUTURE ENHANCEMENTS

Conclusion

MoneyFlo is a distributed system that demonstrates the importance of message passing and the Twilio API in enabling communication between the application and users. The system is designed to handle a large number of users and transactions, and is fault-tolerant through replication and backups. The system uses a strong consistency model to ensure that all transactions are processed correctly, and is designed to be secure, protecting user data from unauthorized access and ensuring the integrity of data. MoneyFlo can serve as an example of a distributed system mini-project that incorporates various distributed system concepts and considerations, including message passing and the use of third-party APIs like Twilio.

Future Enhancements

There are several potential future enhancements for a bank management distributed system, some of which are:

Mobile App: Developing a mobile app for the bank management system could enhance customer experience and provide an added convenience for customers to access their accounts and perform transactions on the go.

Automated Decision Making: Implementing automated decision-making capabilities using machine learning and artificial intelligence algorithms can improve the speed and accuracy of decision-making processes, such as credit approvals, fraud detection, and risk management.

Integration with Blockchain: Integrating the bank management system with blockchain technology can provide additional security and transparency for transactions, reducing the risk of fraud and errors.

Biometric Authentication: Implementing biometric authentication methods such as fingerprint, face recognition, or iris scanning can provide enhanced security and convenience for customers to access their accounts.

Chatbots: Incorporating chatbots into the bank management system can improve customer service by providing instant support and assistance for customer queries and concerns.

Advanced Analytics: Implementing advanced analytics tools can help bank management to gain deeper insights into customer behavior, market trends, and operational efficiency, which can aid in making data-driven decisions.

Overall, future enhancements for a bank management distributed system aim to provide improved functionality, enhanced security, and convenience for customers, while also optimizing the operational efficiency of the bank.

6. REFERENCES

- https://www.twilio.com/docs
- M. van Steen and A.S. Tanenbaum, Distributed Systems, 3rd ed., distributed-systems.net, 2017.
- Twilio Python library: https://www.twilio.com/docs/libraries/python
- MySQL database: https://www.mysql.com/
- SQLAlchemy Python library: https://www.sqlalchemy.org/
- "Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems" by Martin Kleppmann, O'Reilly Media, Inc., 2017.
- "Building Microservices: Designing Fine-Grained Systems" by Sam Newman, O'Reilly Media, Inc., 2015.