Diego Paredes

May 22, 2023

CIS 344

Introduction

In this project, "Bank's Portal" I will be showing the way I was able to build the database with MySQL, connect it with python using visual studio as my coding editor and will be explaining how I was able to make it a function platform that by the end of the coding, anyone would be able to add an account and it will update in Realtime showing on the home page of the localhost. I will also explain the challenges I faced during the project and how I was able to fix and learn from my mistake. By working on this project, I was able to get more hands-on experience and help me put what I've learn from class to work and gain a better understanding of MySQL.

Project Overview

The objective of this project is to finalize the development of a web platform that enables bank tellers to manage bank accounts and execute financial transactions. With the started code in python with a platform that runs on a python HTTP Server, the task is to connect to the MySQL server that will be created for bank tellers will have the ability to access the platform, add or modify account information, and process transactions such as deposits and withdrawals. The finalized source code, alongside the PDF version of the report, will be uploaded to a public GitHub repository.

MySQL Design and Implementation

For the start of MySQL design, it was to create the database and create two tables, the first one was where the information for accounts would go to and the second one was for information for Transactions.

```
1  create database banks_portal;
2  use banks_portal;
3  create table Accounts
4  (
5      accountId int not null unique auto_increment,
6      ownerName varchar(45) not null,
7      owner_ssn int not null,
8      balance decimal(10,2) default 0.00,
9      account_status varchar(45)
10 );
11
12 alter table Accounts
13 add Primary key (accountId);
14
15 create table IF not exists Transactions (
16     transactionId int not null unique auto_increment primary key,
17     accountId int not null,
18     transactionType varchar(45) not null,
19     transactionAmount decimal(10,2) not null
20 );
```

In the first table I didn't add the primary key to the "accountId" before I saved the code. Instead of deleting the table and starting new, I added another code to alter the table for "Accounts" and add the Primary Key to the correct attribute. For the second table, the instructions were to create a table but to also put in the command for "if the table doesn't exist" it won't create it again.

For the next two task, it was to put in values into the table accounts and transactions. Which are pretty direct on adding information to the account and transactions by adding it using the "Insert into" then the name of the table and put values in the correct order by the way it was set up on the table.

```
22 •  Insert into Accounts (ownerName, owner_ssn, balance, account_status)
23    values
24    ('Maria Jozef', 123456789, 10000.00, 'active'),
25    ('Linda Jones', 987654321, 2600.00, 'inactive'),
26    ('John McGrail', 222222222, 100.50, 'active'),
27    ('Patty Luna', 111111111, 509.75, 'inactive');
28
29 •  Insert into Transactions (accountId, transactionType, transactionAmount)
30    values
31    (1, 'deposit', 650.98),
32    (3, 'withdraw', 899.87),
33    (3, 'deposit', 350.00);
```

In the following steps, I had to put in three procedures. This I had some challenge because I wasn't able to completely understand what it was and after reviewing the lessons and seeing how they are basically shortcuts. I had to create "deposit", "withdraw" and "accountTransactions". This was probably my favorite thing in coding such as having things organized and it will show later on when working with python.

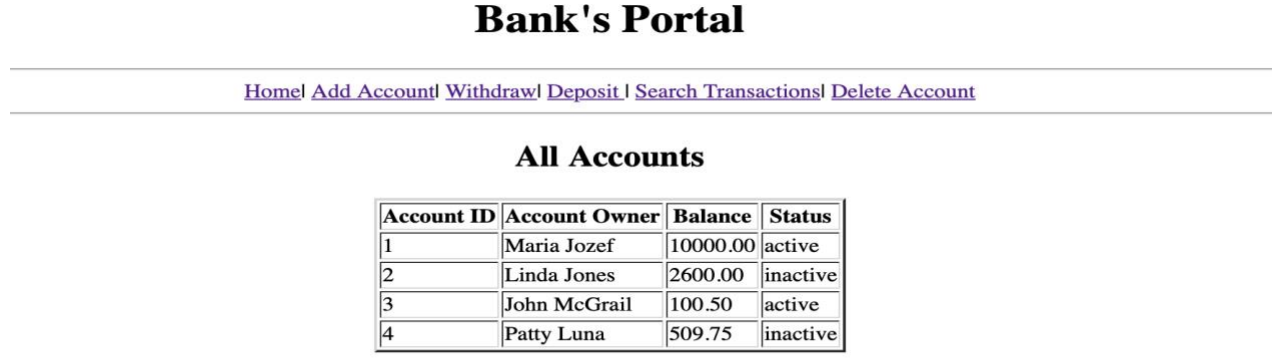
```
35    Delimiter //
36 •  Create procedure accountTransactions(in accountId int)
37    begin
38        select * from transactions
39        where accountId = accountId;
40    End//
41    Delimiter ;
42
43    Delimiter //
44 •  Create procedure deposit(in accountId int, amount decimal(10,2))
45    Begin
46        Start Transaction;
47        Insert into Transactions ( accountId, transactionType, transactionAmount)
48        value (accountId, 'deposit', amount);
49
50        update accounts
51        set balance = balance + amount
52        where accountId = accountId;
53
54        commit;
55    End//
56    Delimiter ;
```

Based on the descriptions of accountTransactions, the purpose of the procedure was to select a specific account, to do that I had to show that this will be a single command by putting the

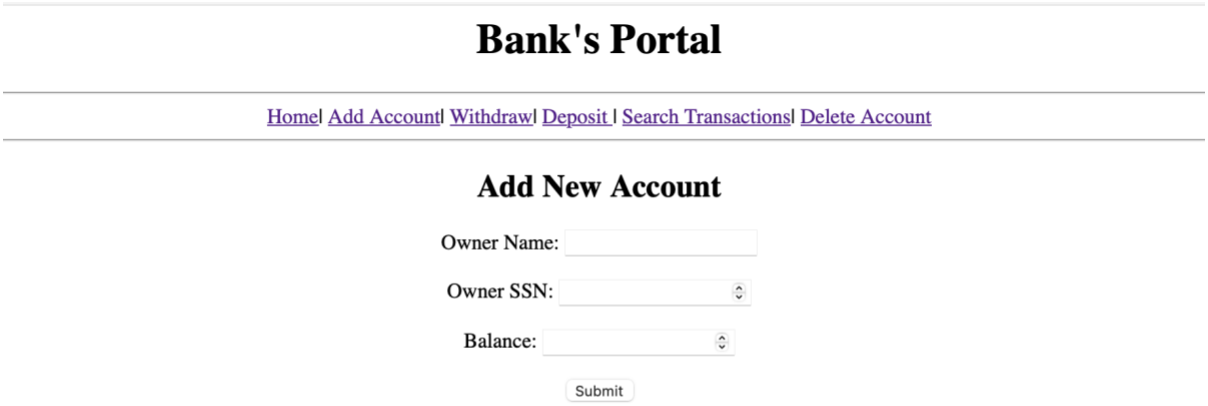
delimiter on the being to the end showing where it stops. We will then name our procedure based on the directions so that we can call upon it when we need it by that name and run the code. These procedures are very useful when needing to deal with a lot of code, it will shorten things you have to type and will make it look cleaner.

Python Server Design and Implementation

The server-side functionality of the project was developed using Python, with the portalDatabase.py file being central to this process. This file manages connections and interactions with the MySQL database. This file used the host, port, database, username, and password of the MySQL database and will work with the portalServer.py file so that it would be able to do a local host on the computer. We first entered the credentials for the MySQL database to the portalDatabase.py and run the portalServer.py on the terminal and go to the browser using localhost:8000/ and it will show use the accounts that were put in during the second step of the MySQL designing.



The next objective was to have the "add account form" actually add account and have it update on the home page. When we click on submit.



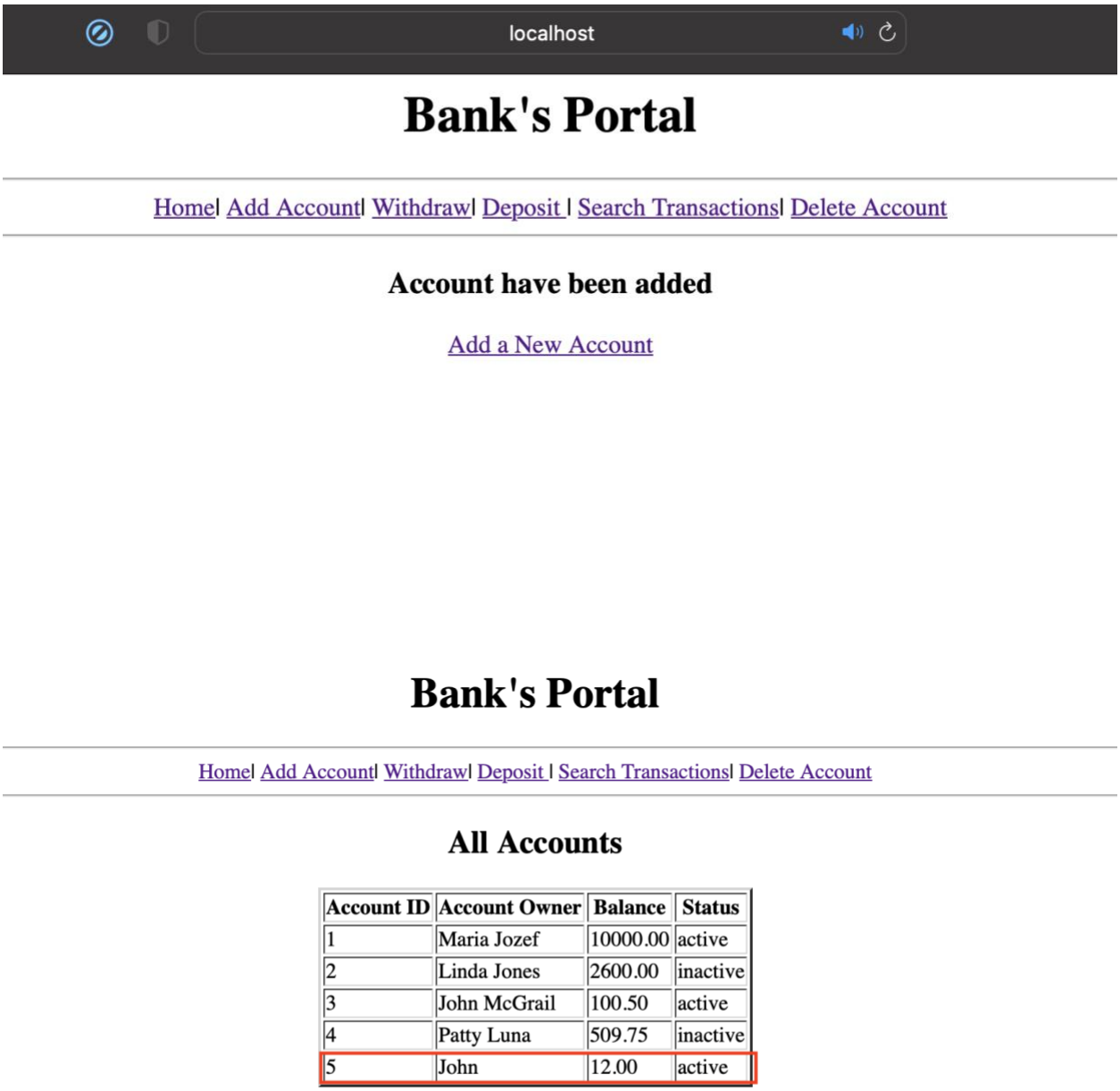
In order to have the form functional, I decided to create another procedure on MySQL database and add the code to call on the procedure on the python file.

```
89    Delimiter //
90    CREATE Procedure addAccount(IN ownerName varchar(45), IN owner_ssn INT, IN balance DECIMAL(10,2),  account_status varchar(45))
91    BEGIN
92        INSERT INTO accounts (ownerName, owner_ssn, balance, account_status)
93        VALUES (ownerName, owner_ssn, balance, account_status);
94    END //
95    DELIMITER ;
```

This was the code on MySQL, the way it works is by the information that is entered on the form will be added into the account table and will give an automatic status of active because it's a new account with cash. Back in the python file, I defined Add account on the portalDatabase.py to call on the procedure and add the information to the home page. This is how it would look.

```
5       def addAccount(self, ownerName, owner_ssn, balance, status):
6           if self.connection.is_connected():
7               self.cursor = self.connection.cursor()
8               query = "CALL addAccount(%s, %s, %s, %s)"
9               self.cursor.execute(query, (ownerName, owner_ssn, balance, status))
0               self.connection.commit()
```

Now back to the website, I was able save the changes enter the information and add the new account to the database.



The page would be automatically updated and will show their status and can continue to add more account if needed.

Conclusion

Through the completion of this project, we have not only created a functional and responsive platform but also have demonstrated a practical application of several core concepts in web development and database management. These include HTTP server management, SQL database creation, server-side scripting with Python, and integration of SQL with Python using MySQL Connector/Python. This project gave me the opportunity to develop skills on coding with Python and working with connecting MySQL, it was able to get more familiar with coding on the database side and it was really fun learning experience. It was also a learning experience on coding on a mac and refresher on visual studio code. Another learning experience was working with GitHub, I've only used it a couple of time but never submitted on the repository. Overall this class was a very fun class and good learning experience.

Appendix

The appendix of this report includes additional supporting details, such as direct links to the project repository and illustrative screenshots.

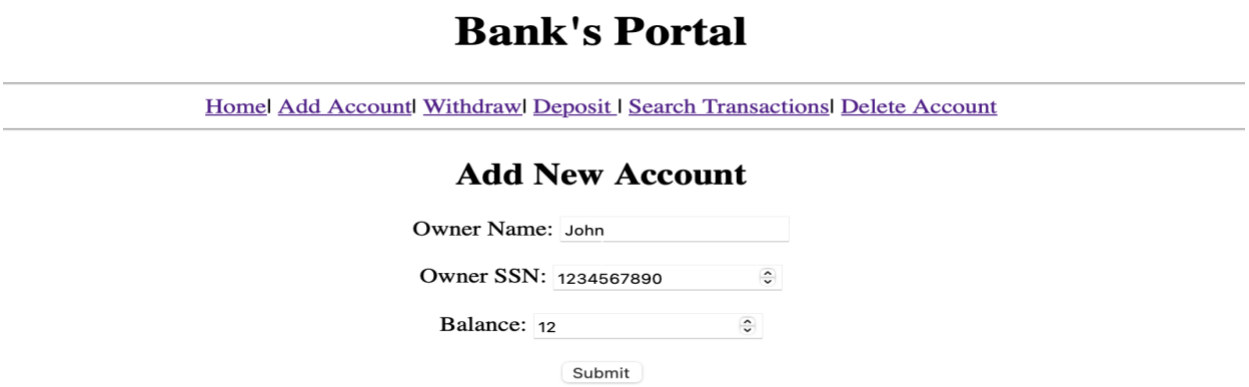
Project Repository Link

The complete source code of the project, along with the associated documentation, is available in a publicly accessible GitHub repository. The link to the repository is:

https://github.com/Dieblued1/Banks_portal

Additional Screenshots

The following are supplementary screenshots demonstrating various aspects and functionalities of the developed web platform:



This is the information entered when adding the account to the database.

```
def withdraw(self, accountID, amount):
    if self.connection.is_connected():
        self.cursor = self.connection.cursor()
        query = "CALL withdraw(%s, %s)"
        self.cursor.execute(query, (accountID, amount))
        result = self.cursor.fetchone()
        if result[0] == 0:
            print(f"Error: {result[0]}")
        else:
          self.connection.commit()
          print("{result[1]}")

def addAccount(self, ownerName, owner_ssn, balance, status):
    if self.connection.is_connected():
       self.cursor = self.connection.cursor()
       query = "CALL addAccount(%s, %s, %s, %s)"
       self.cursor.execute(query, (ownerName, owner_ssn, balance, status))
       self.connection.commit()

def accountTransactions(self, accountID):
    if self.connection.is_connected():
       self.cursor = self.connection.cursor()
       query = "CALL accountTransactions(%s)"
       self.cursor.execute(query, (accountID,))
       records = self.cursor.fetchall()
       return records

def deleteAccount(self, AccountID):
    if self.connection.is_connected():
       self.cursor = self.connection.cursor()
       query = "CALL deleteAccount(%s)"
       self.cursor.execute(query, (AccountID,))
       self.connection.commit()
```

This were another code that was defined on the portalDatabase.py. I was able to connect the procedures that was created on the MySQL file but wasn't able to create a page for they to be used. I decided to create multiple procedures to keep the code clean and easy to read. I input many calls and they will be able to update the database.

```
Delimiter //
Create procedure withdraw(in accountId int, amount decimal(10,2))
Begin
    declare currentBalance decimal(10,2);
    start transaction;
    select balance into currentBalance from accounts where accountId = accountId;

    If currentBalance >= amount Then
        insert into Transacations (accountId, transactionType, transactionAmount)
        values (accountId, 'withdraw', amount);

        update accounts
        set balance = balance - amount
        where accountId = accountId;

        commit;
        SELECT 1 as Status, 'Transaction successful' as Message;
    Else
        Rollback;
         SELECT 0 as Status, 'Insufficient balance' as Message;
    End If;
End //
Delimiter ;
```

This was the code that was used for the withdraw procedure. This code was able to update the balance and if it doesn't have enough based on the amount put in on the python file, it will be declined, and the amount won't be charged and it will give out a message saying Insufficient balance.

```
Delimiter //
Create Procedure deleteAccount(in accountId int)
Begin
    DELETE FROM Accounts WHERE accountId = accountId;
End//
Delimiter ;
```

This is the procedure for deleting an account and the it will be able to find the account based on the account Id # and delete it.