****

**CMPE 207 - Network Programming and Application**

**Distributed Banking System**

By

Gnana Deep Pallela

Kanti Bhat

Usha Veguru

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **S No** | **Topic** | **Page Number** |
| 1. | Introduction and Overview | 4 |
| 2. | Design Architecture | 5 |
| 3. | Software Platform and Tools | 6 |
| 4. | Features | 10 |
| 5. | Modules and Functionalities | 15 |
| 6. | Future Work | 16 |
| 7. | Conclusion | 17 |
| 8. | References | 17 |
| 9. | Contribution | 18 |
| 10. | Appendix | 18 |

**ABSTRACT**

Objective of the project is to design a client – server based Distributed Banking System Application with many players including Tellers, Customers, and System Administrators.

Built system supports following operations with respective players:

1. Administrator:

1.1 View teller accounts

1.2 Create teller accounts

1.3 Delete teller accounts

1.4 Approve customer transactions

1.5 View monthly transactions

2. Teller:

2.1 View and manage customer accounts

2.2 View and manage customer profile

2.3 Create new customer accounts and records

2.4 Delete customer accounts and records

3. Customer:

3.1 View Checking account balance

3.2 View Saving account balance

3.3 Deposit to Checking and Saving accounts

3.4 Withdraw from Checking and Saving accounts

3.5 Transfer funds to another bank accounts

Report contains the details of design and implementation carried out during the development of Distributed Banking application.

**1. INTRODUCTION AND OVERVIEW**

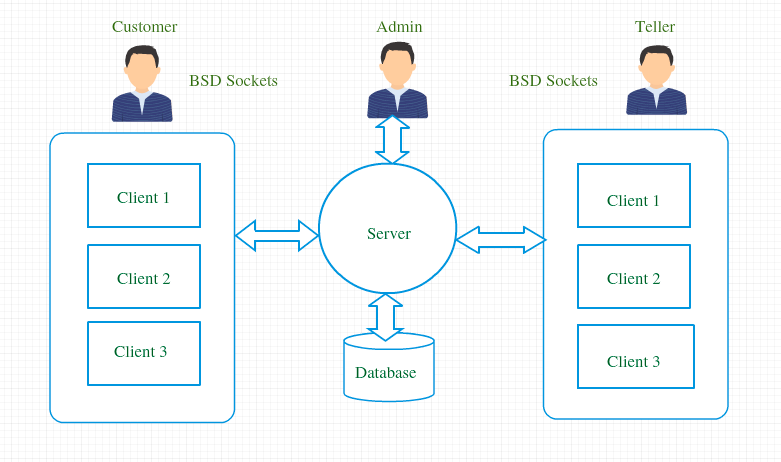
Goal of this project is a Distributed Banking System which provides online banking facility to its customers to check their accounts and do the transactions on-line any time from anyplace. System provides all the facilities to the customer when their authentication credentials match, including viewing account information, performing transfers, option to view previous transactions, giving the customer an option of changing their personal details like their address, phone number, email address etc.

System consist of centralized application server which handles multiple client connections at the same time. Bank tellers and administrator is also supported and served as clients by this application server. Tellers support bank customers by assisting them with opening new customer accounts, closing customer accounts, helping customers with money deposit and withdrawals.

Administrator supervises the customer activities and manages the bank crew with privileges to add and remove tellers to the system.

System is developed using python flask and socket programming APIs. It is written as client and server modules to support distributed idiom of the banking system. Client module can be run from anywhere and it supports different clients to support customer, teller and administrator roles. Server module on the other hand is more complicated and involves MySQL database handling, creation of threads to manage concurrency.

**2. Architecture of online banking:**



The 3-tier architecture shown above has the following major components:

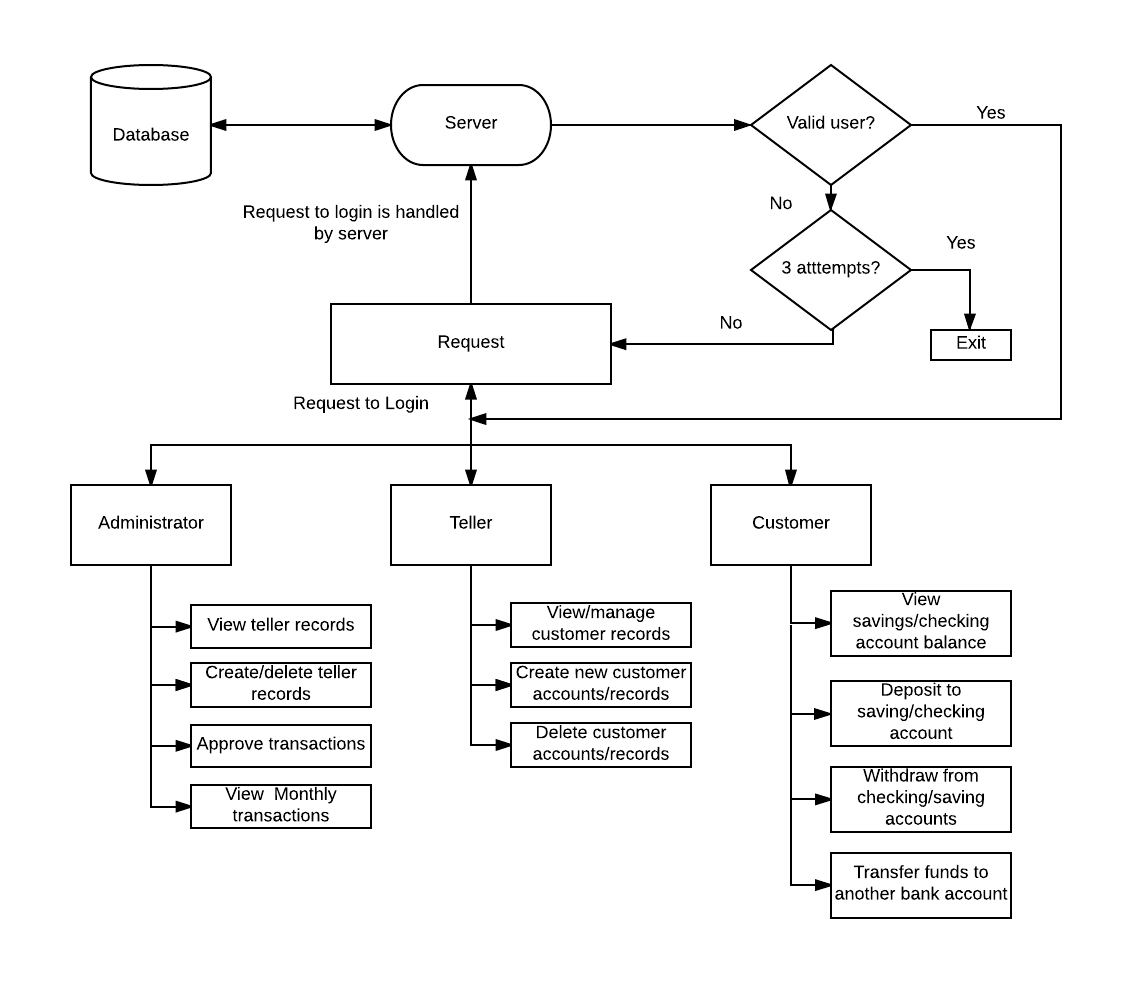
**1. Client** – There are three different types of clients for the application. These are web-based user-friendly front end applications serving different client roles.

**2. Application Server** – It takes care of the server script, connection with MySQL server, connectivity through MySQLdb connector.

**3. Database** – MySQL server stores customers and bank data.

Application works based on request / response protocol. Client initiates a request to the server. Server processes the request, accesses the database to retrieve the data and send this data back to the client. Server verifies client credentials to give access to the system. Server responds to the client either with results or with an error message.

**Flow Diagram:**

****

**3. Software Platform and Tools**

**3.1 MySQL Database**

A database is a collection of tables which allows us to view, create, delete and update information in the tables. MySQL is a fast and “easy-to-use” database management system which is implemented along with Python, PHP, PERL, C, C++, JAVA, etc. Therefore, the MySQL is used to manage large amount of structured data and it is also known as a relational database.

**Creating a Database:**

This command is required to create a database. To create a data base, we require a SQL command “CREATE DATABASE database\_name”.

**Deleting a Database:**

This command is required to delete a database. To delete a database, the required command is “DROP DATABASE database\_name”.

**Creating a table:**

This command is required to create a MySQL table. To create a table, the required syntax is “CREATE TABLE table\_name(column\_name column\_type)”.

**Dropping a table:**

This command is required to drop the MySQL table. To drop a table, the required command is “DROP TABLE table\_name”.

**Inserting data into the table:**

This command is required to insert data in the MySQL table. To insert data into table, therequired syntax is “INSERT INTO table\_name ( field1, field2,... fieldN ) VALUES ( value1, value2,... valueN )”.

**Fetching data from the database:**

This command is required to fetch data from the MySQL table. To fetch data from the MySQL database, we require the following syntax:

“SELECT field1, field2,... fieldN table\_name1, table\_name2... [WHERE Clause] [OFFSET M ][LIMIT N]”.

**Updating the data in table:**This command is required to update the existing data in the MySQL table. The syntax is as follows: “UPDATE table\_name SET field1=new-value1, field2=new-value2 [WHERE Clause]”.

**Deleting the Data from the table:**

This is required to delete a record from the MySQL table. The syntax is as follows: “DELETE FROM table\_name [WHERE Clause]”.

**3.2 Python socket programming**

The sockets are the endpoints of communication between the same machine, different machines or within a same process. To communicate between the server and the client, we require sockets on both the sides. Therefore, sockets play an important role in the transportation of the bytes between same machine, different machines or within a same process.

The important functions from the Python socket programming are as follows:

1. **socket.socket():**

To create a new socket, we require the following parameters such as: address family, socket type and protocol number.

1. **socket.bind(address, port):**

This is used to bind the socket to the particular address and the port.

1. **socket.listen(backlog):**

The function is used to listen to the connections on the socket. The backlog is used to specify the queue size where 0 is the minimum number and 5 can be the maximum number.

1. **socket.accept():**

The function is used to accept the client connection when it arrives, until then the socket is in the waiting state. In this, the return value has the socket object to send and receive information, and the address of other end socket.

1. **socket.connect(host,port):**

This is system call to connect to the remote socket at the given host and the port mentioned as parameters in the function.

1. **socket.send(bytes[, flags]):**

This function is used to send data and it returns the number of bytes sent to the other side. If the data is not sent completely, the program has to transmit the remaining data to the remote host.

1. **socket.recv(recv\_size):**

This function is a blocking call and it reads the recv\_size mentioned as a parameter in the function. If there is no data waiting in the buffer, then it is in the blocking state. If there is data remaining then the next call won’t be blocked.

1. **socket.close():**

This is used to close the socket, and the function is executed when there is no more data to send to the remote host or the remote host is not asking for the data. Therefore, this function is used to close the socket after the queued data is flushed.

**3.3 Python Flask - GUI**

Flask is a micro-web framework for Python developed to build web applications easily for any user with basic knowledge in Python. It is easy to use and it can be implemented by using Jinja2 templating and Restful API’s.

Steps to develop a basic web application:

1. **Create an Environment:** To install the virtual environment for the Flask, we should create a directory and execute the following command: “virtualenv venv”. To active the environment, we should execute the command: “venv\Scripts\activate”.
2. **Install Flask:** To install Flask with the Pip package manager, we can use the following command: “pip install Flask”.
3. **Create a folder:** After installing Flask, we should create a folder named “Flaskapplication” and create a file named “webapp.py”.
4. **Import the module:** In the program, to import the flask module and create a app variable using Flask. We should enter the following: “from flask import Flask” and “app=Flask(\_\_name\_\_)”.
5. **Create the basic program**: For the request handler, we should enter “@app.route”. To create a main function, we need this code: “def main(): return “welcome”.

In the main program, we should enter the following command to run the application: ‘if \_\_name\_\_ == “\_\_main\_\_”: app.run()‘

1. **Execute the program:** To execute the program in the command prompt, we should be the current directory and enter the following commmand: “python app.py”. This command starts the server on the port ‘5000’.

Finally, we can execute the following URL in the browser to see the welcome webpage: “<http://localhost:5000/>”

In this project, we have implemented the following steps:

1. Home page
2. Signup Page

**4. Features**

**4.1 Concurrent multithreaded Application server**

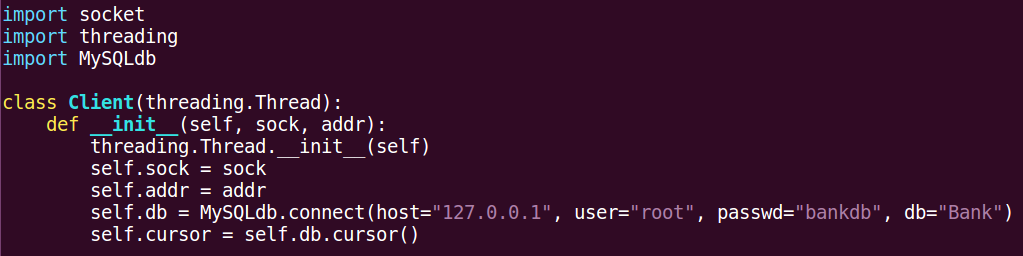
Python threading module and allows to create new client thread connections on each client connection request. Server thread hence accepts new client connections and passes this new connection socket to the newly created client thread to handle the client request. Thread start() method starts the new thread and serves the client.



**4.2 MySQL Database connectivity with the Application server**

Application server connects with the MySQL server using MySQLdb interface. This interface is thread-compatible and is one among the popular interfaces provided between MySQL database and python.

Each client thread requests a new database connectivity at the application server. Client is served using request / response protocol specifications. Client requests the application server and application server retrieves data from MySQL database server. Once data is retrieved from the database application server builds the response and send it to the client.

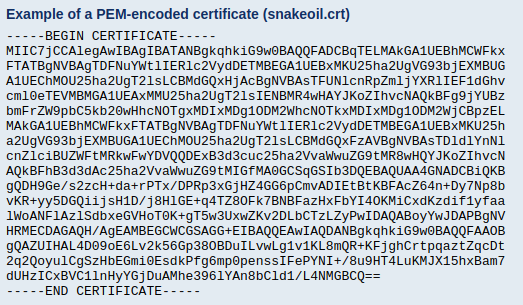


**4.3 SSL/TLS Encryption between Clients and Server**

System uses SSL/TLS encryption facilities with PEM-encoded certificate to secure the transactions between a bank customer and the application server. The (SSL) Secure Sockets Layer protocol is a protocol layer which may be placed between a reliable connection-oriented network layer protocol (e.g. TCP/IP) and the application protocol layer (e.g. HTTP). SSL provides for secure communication between client and server by allowing mutual authentication, the use of digital signatures for integrity and encryption for privacy.

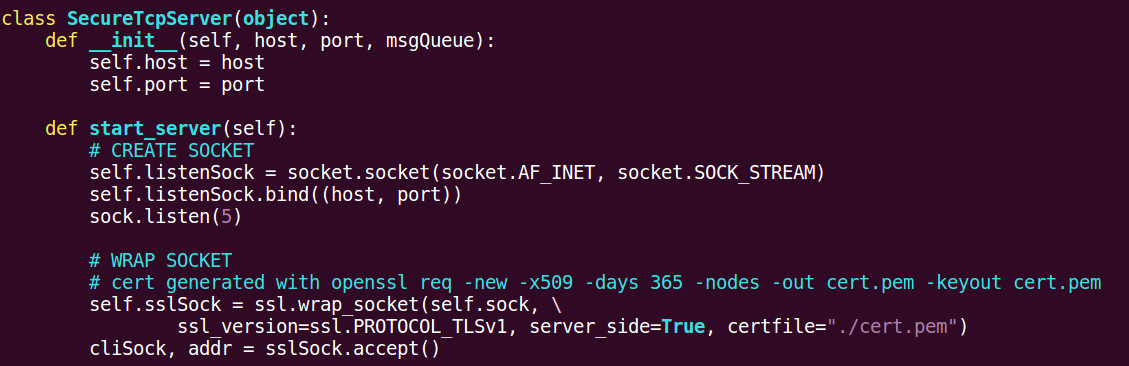
The protocol is designed to support a range of choices for specific algorithms used for cryptography, digests and signatures. This allows algorithm selection for specific servers to be made based on legal, export or other concerns and also enables the protocol to take advantage of new algorithms. Choices are negotiated between client and server when establishing a protocol session.

There are a number of versions of the SSL protocol. One of the benefits in SSL 3.0 is that it adds support of certificate chain loading.SSL 3.0 is the basis for the Transport Layer Security (TLS) protocol standard.

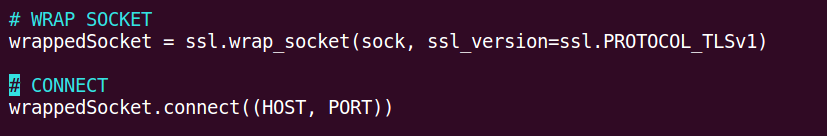


**SSL / TLS encryption at the Application server:**

Note: cert.pem certificate is saved in the Application Server Directory.



**SSL / TLS encryption at the Client:**

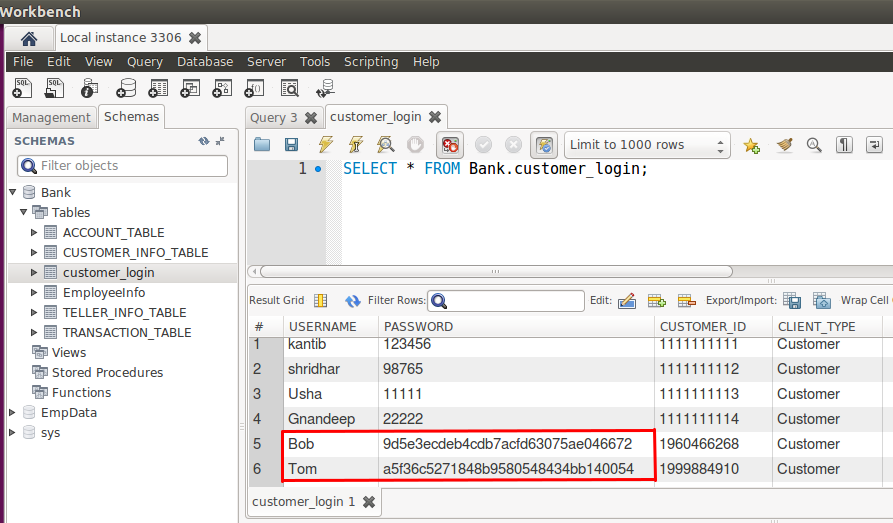


**4.4 MD5 Encrypted password in the database**

Application server secures client passwords stored in the database using MD5 hash function. MySQL language provides this MD5 functionality to securely store password information. However once password is stored with MD5 Hash function it is impossible to see the password. Application server can only encrypt the password provided by the clients and check if it matches the stored MD5 function.

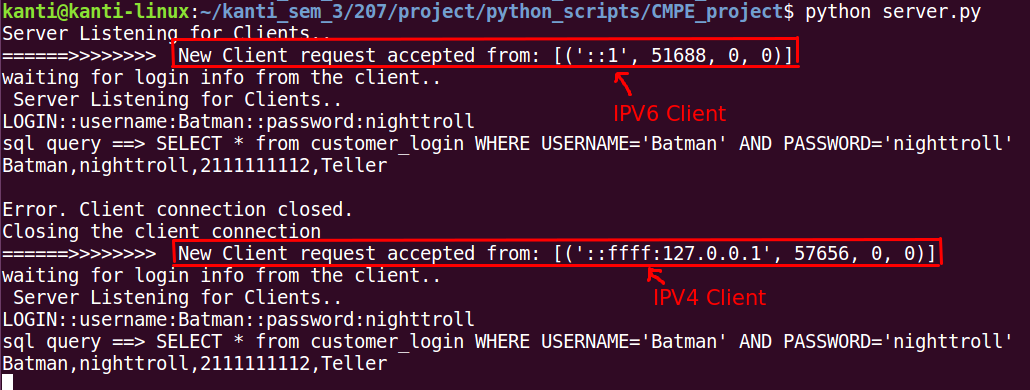
MySQL MD5() Calculates an MD5 128-bit checksum for a string. The value is returned as a binary string of 32 hex digits, or NULL if the argument was NULL. The return value can, for example, be used as a hash key.

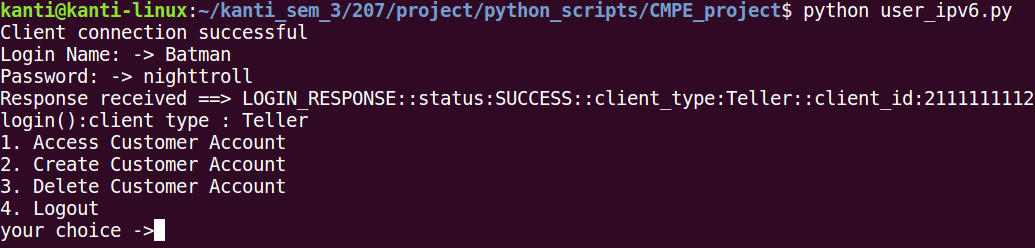


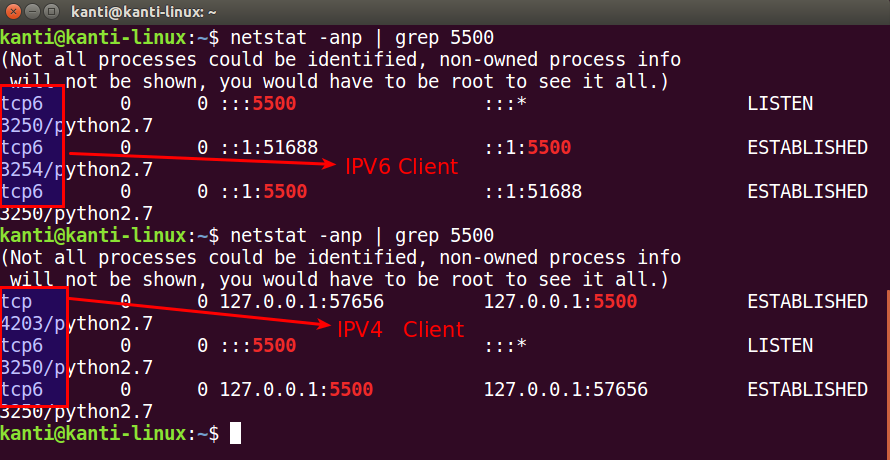


**4.5 IPV6 support between client and the server**

Server is running on IPV6 address in python has a provision to support both IPV4 and IPV6 clients.[[1]](#footnote-1)







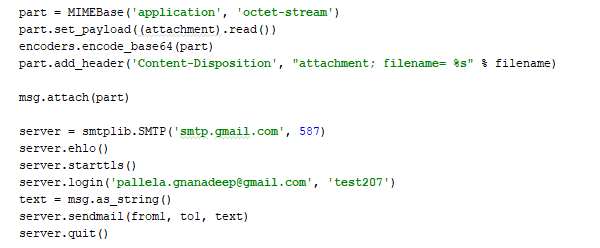
**4.6 Email facility via SMTP protocol**

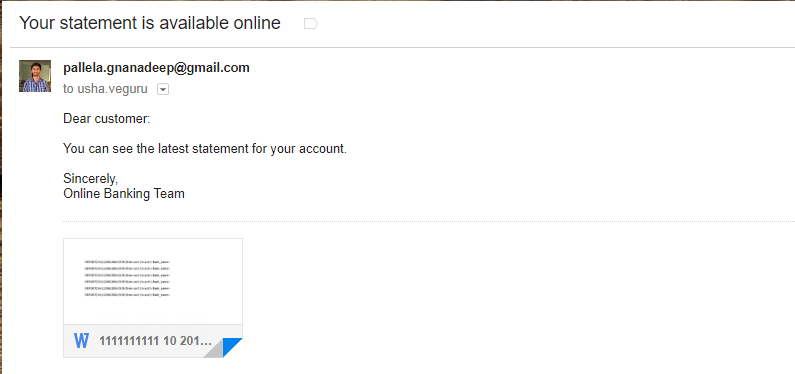
The SMTP stands for simple mail transfer protocol. In Python, we can find several modules to send an email using the SMTP Protocol, in fact, Python has a module called smtplib to send an email by communicating with mail server (i.e., SMTP server).

In this project, we are sending email using Gmail by changing the settings of the SMTP server.

The steps for sending monthly email to the customers using Python:

1. At first, we have to “setup the SMTP server and log into the account”.
2. Next, we need to “create the MIMEMultipart message object and load it with appropriate headers for From, To, and Subject fields”.
3. Then, we have to “add the email body”.
4. Finally, we send the “message using the SMTP server object”.





**5. Modules and Functionalities**

**Server Module:**

Centralized application server module with access to database server allows authentication of the clients requesting an access to the system. Server retrieves login credentials of the user from the database and matches with the login credentials sent by the client. If this authorization fails server disconnects the client immediately and secures any unauthenticated access to the Banking system. If client credentials matches then application server allows the client to access the facilities and the database based on the logged in client (Customer, Teller, Admin) privileges.

**Client Modules:**

Client module is divided into three major classes named Customer, Teller and Administrator. Based on the role, restrictions following operations are allowed and implemented in their respective classes.

1. Customer

1.1 View Checking Account

1.2 View Saving account

1.3 View transactions

1.4 View user information

1.5 Deposit to Checking account

1.6 Deposit to Saving account

1.7 Withdraw from Checking account

1.8 Withdraw from saving account

1.9 Transfer funds to another account

1.10 Update user information

2. Teller

2.1 View Customer accounts

2.2 Deposit to customer checking account

2.3 Deposit to customer savings account

2.4 Withdraw from customer checking account

2.5 Withdraw from customer saving account

2.6 View Customer Transactions

2.7 Create new customer accounts

2.8 Delete customer accounts

2.9 View customer profile

2.10 Update customer profile

3. Administrator

3.1 Create Teller record

3.2 Delete Teller record

3.3 View Customer transactions.

**6. Future Work**

Online banking is the most useful feature implemented by all the banks. All the features of bank are implemented. The other features that could be included for future scope are:

1. Use Winsock to support the implementation for windows.
2. Serve multiple protocols by client-server model.

**7. Conclusion**

This project implements basic operations and functionalities of a distributed banking system supporting different type of clients to communicate with the Banking System. Server stores all the client details and records in the database and allows clients to access the bank facilities and the database records only when the client authorizes with correct login and password credentials. As a customer, client can have checking and saving accounts and client has the ability to perform amount withdrawal, deposit and transferring the amount to another account. As a Teller client can perform above operations on any of the customer accounts. Teller can also perform new Account creation and close existing accounts. Admin client monitors the bank transactions and manages the Teller crew and hence can add and remove teller records as per requirements. Admin can also approve transactions involving amounts more than bank cut off on fund transactions.

**8. References**

1. Internetworking with TCP/IP Vol. 3, Client-Server programming and applications, Comer and Stevens, Linux/POSIX Sockets version, ISBN: 0-13-032071-4, 2001.
2. UNIX Network Programming Vol. 1, 3/e: The Sockets Networking API, Stevens, Fenner & Rudoff, ISBN: 0-13-141155-1, 2004.
3. UNIX Network Programming Vol. 1, 2/e: Networking APIs - Sockets and XTI, W. Stevens, ISBN: 0-13-490012-X, 1998.
4. UNIX Network Programming, Vol. 2, 2/e: Interprocess Communications, W. Stevens, ISBN 0-13-081081-9, 1999.
5. UNIX Network Programming, Stevens, ISBN: 0-13-949876-1, 1990.
6. Python Network sockets programming <https://pythonspot.com/en/python-network-sockets-programming-tutorial/>
7. Python socket programming How to documentation <https://docs.python.org/2/howto/sockets.html>
8. TLS/SSL wrapper for socket objects <https://docs.python.org/2/library/ssl.html>
9. MySQL with Python

<https://pythonspot.com/en/mysql-with-python/>

1. Reading and Writing Files in python

<http://www.pythonforbeginners.com/files/reading-and-writing-files-in-python>

1. IPV6 support in python

<https://docs.python.org/2.4/lib/socket-example.html>

13. Flask tutorial

<https://code.tutsplus.com/tutorials/creating-a-web-app-from-scratch-using-python-flask-and-mysql--cms-22972>

14. MySQL tutorial

<https://www.tutorialspoint.com/mysql/mysql_tutorial.pdf>

15. Python Socket Programming

<http://www.bogotobogo.com/python/python_network_programming_s>erver\_client.php

**9. Contribution**

All the team members contributed for design and documentation work. The implementation part is divided as follows:

Kanthi - client and server of teller, SSL,IPv6

Gnanadeep - client and server of admin, SMTP

Usha - client and server of customer, GUI

**10. Appendix**

1. **Code**
   1. **Server**

**import time**

**import ssl**

**import socket**

**import threading**

**import MySQLdb**

**import fpdf**

**from fpdf import FPDF**

**import smtplib**

**import calendar**

**from email.mime.text import MIMEText**

**import sys**

**import os**

**from email.MIMEMultipart import MIMEMultipart**

**from email.MIMEText import MIMEText**

**from email.MIMEBase import MIMEBase**

**from email import encoders**

**AUTHENTICATION\_FAILURE = "authentication\_failed"**

**AUTHENTICATION\_SUCCESSFUL = "authentication\_successfull"**

**SUCCESS = 0**

**ERROR = 1**

**LOGOUT = 2**

**class Response(object):**

**def \_\_init\_\_(self):**

**self.resType = ''**

**self.resParams = {}**

**def toString(self):**

**s = "{}".format(self.resType)**

**for k,v in self.resParams.items():**

**s = "{}::{}:{}".format(s, k, v)**

***#print"toString(): s = " + s***

**return s**

**class Request(object):**

**def \_\_init\_\_(self, buf):**

**print "\nNew Request ["+buf+"]"**

**print "\n"**

**values = buf.split('::')**

**self.reqType = values[0]**

***#print "Request Received ==> "+self.reqType***

**self.reqParams = {}**

**for elem in values[1:]:**

**k, v = elem.split(':')**

**self.reqParams[k] = v**

***#print "\""+k+":"+v+"\" "***

***######################### CLIENT THREAD CLASS #######################***

**class Client(threading.Thread):**

**def \_\_init\_\_(self, ssock, addr):**

**threading.Thread.\_\_init\_\_(self)**

**self.sock = ssock**

**self.addr = addr**

**self.db = MySQLdb.connect(host="127.0.0.1", user="root", passwd="vaibhav", db="Bank")**

**self.cursor = self.db.cursor()**

**def run(self):**

**print"waiting for login info from the client.."**

**while True:**

***#receive login request from the client***

**try:**

**data = self.sock.recv(1024)**

**print data**

**except:**

**break**

**result = self.process\_request(data)**

**if(result == ERROR):**

**print "Error. Client connection closed."**

**break**

**elif(result == LOGOUT):**

**print "Client done. Connection closed."**

**break**

**print "Closing the client connection"**

**self.sock.close()**

**self.db.close()**

**return**

**def send\_error(self,msgtype,errmsg):**

***#create response to send an error***

**resp = Response()**

**resp.resType = msgtype**

**resp.resParams['status']='FAILED'**

**resp.resParams['error']=errmsg**

**self.sock.send(resp.toString())**

**def process\_request(self,buf):**

**req = Request(buf)**

**print"process\_request(): reqType ==>"+req.reqType**

**if req.reqType == 'LOGIN':**

**return self.check\_login(req)**

**elif req.reqType == 'GET':**

**self.service\_get\_request(req)**

**elif req.reqType == 'SET':**

**self.service\_set\_request(req)**

**elif req.reqType == 'INSERT':**

**self.service\_insert\_request(req)**

**elif req.reqType == 'DELETE':**

**self.service\_delete\_request(req)**

**else:**

***#create response to send an error***

**resp = Response()**

**resp.resType = 'UNKNOWN\_REQUEST'**

**resp.resParams['status']='FAILED'**

**resp.resParams['error']="invalid request type."**

**self.sock.send(resp.toString())**

**return ERROR**

**def check\_login(self, req):**

**resp = Response()**

**if "username" not in req.reqParams or \**

**"password" not in req.reqParams or \**

**req.reqParams["username"] == "" or \**

**req.reqParams["password"] == "":**

**self.send\_error('LOGIN\_RESPONSE','Invalid Username or password')**

**return ERROR**

**else:**

**sql = "SELECT \* from customer\_login WHERE USERNAME='{x}' AND PASSWORD=MD5('{y}')".format(x=req.reqParams["username"],y=req.reqParams["password"])**

**print "sql query ==> " + sql**

**try:**

**self.cursor.execute(sql)**

**except Exception as e:**

**print e**

**print "calling auth failure server side error"**

**self.send\_error('LOGIN\_RESPONSE', 'Server side error')**

***#self.sock.close()***

**return ERROR**

**print"db executed successfully"**

***# fetch results (its a list)***

**results = self.cursor.fetchone()**

**if not results:**

**print "calling auth failure user does not exist"**

**self.send\_error('LOGIN\_RESPONSE','User does not exist')**

**return ERROR**

**else:**

**print"inside login else block"**

***#print results[0]+","+results[1]+","+str(results[2])+","+results[3]***

**resp.resType = 'LOGIN\_RESPONSE'**

**resp.resParams['status']='SUCCESS'**

**resp.resParams['client\_type']=results[3]**

**resp.resParams['client\_id']=str(results[2])**

**print"before send"**

**self.sock.send(resp.toString())**

**print "after send"**

**return SUCCESS**

**def service\_get\_request(self, req):**

**print"called service\_get\_request()\n"**

**resp = Response()**

**if req.reqParams['subreq\_type'] == 'CUSTOMER\_ID':**

**self.get\_customer\_id(req)**

**elif req.reqParams['subreq\_type'] == 'CUSTOMER\_ACCT':**

**self.get\_customer\_acct(req)**

**elif req.reqParams['subreq\_type'] == 'CUSTOMER\_TRANSACTION':**

**self.get\_customer\_transactions(req)**

**elif req.reqParams['subreq\_type'] == 'CUSTOMER\_PROFILE':**

**self.get\_customer\_profile(req)**

**elif req.reqParams['subreq\_type'] == 'ALL\_TELLER\_ID':**

**self.get\_teller\_ids(req)**

**elif req.reqParams['subreq\_type'] == 'TELLER\_PROFILE':**

**self.get\_teller\_profile(req)**

**elif req.reqParams['subreq\_type'] == 'MONTH\_TRANSACTIONS':**

**self.get\_month\_transactions(req)**

**elif req.reqParams['subreq\_type'] == 'MONTHLY\_STATEMENT':**

**self.get\_monthly\_statement(req)**

**else:**

**self.send\_error('GET\_RESPONSE','Invalid Sub request sent.')**

**return ERROR**

**def service\_set\_request(self, req):**

**print"service\_set\_request(): subreq\_type ==>" + req.reqParams['subreq\_type']**

**if req.reqParams['subreq\_type'] == 'UPDATE\_CHK\_ACCT':**

**self.update\_customer\_acct(req)**

**elif req.reqParams['subreq\_type'] == 'UPDATE\_SAV\_ACCT':**

**self.update\_customer\_acct(req)**

**elif req.reqParams['subreq\_type'] == 'UPDATE\_CUSTOMER\_PROFILE':**

**self.update\_customer\_profile(req)**

**elif req.reqParams['subreq\_type'] == 'UPDATE\_TELLER\_PROFILE':**

**self.update\_teller\_profile(req)**

**elif req.reqParams['subreq\_type'] == 'TRANSFER\_MONEY':**

**self.transfer\_money(req)**

**else:**

**self.send\_error('GET\_RESPONSE','Invalid Sub request sent.')**

**return ERROR**

**def service\_insert\_request(self, req):**

**print"called service\_insert\_request()"**

**if req.reqParams['subreq\_type'] == 'INSERT\_LOGIN\_RECORD':**

**self.insert\_login\_record(req)**

**elif req.reqParams['subreq\_type'] == 'INSERT\_ACCT\_RECORD':**

**self.insert\_accounts\_record(req)**

**elif req.reqParams['subreq\_type'] == 'INSERT\_PROFILE\_RECORD':**

**self.insert\_profile\_record(req)**

**else:**

**self.send\_error('INSERT\_RESPONSE','Invalid Subreq\_type sent.')**

**return ERROR**

**def service\_delete\_request(self, req):**

**print "service\_delete\_request()"**

**if req.reqParams['subreq\_type'] == 'DELETE\_LOGIN\_RECORD':**

**self.delete\_login\_record(req)**

**elif req.reqParams['subreq\_type'] == 'DELETE\_ACCT\_RECORD':**

**self.delete\_accounts\_record(req)**

**elif req.reqParams['subreq\_type'] == 'DELETE\_PROFILE\_RECORD':**

**self.delete\_profile\_record(req)**

**else:**

**self.send\_error('DELETE\_RESPONSE','Invalid Subreq\_type sent.')**

**return ERROR**

***########################### GET FUNCTIONS ############################***

**def get\_customer\_profile(self, req):**

**print "called get\_customer\_profile()"**

**if "customer\_id" not in req.reqParams or \**

**req.reqParams["customer\_id"] == "" :**

**self.send\_error('GET\_RESPONSE','Invalid Customer ID')**

**return ERROR**

**else:**

**resp = Response()**

**sql = "SELECT \* from CUSTOMER\_INFO\_TABLE WHERE CUSTOMER\_ID='{x}'".format(x=req.reqParams["customer\_id"])**

***#print "sql query ==> " + sql***

**try:**

**self.cursor.execute(sql)**

**except Exception as e:**

**print e**

**self.send\_error('GET\_RESPONSE', 'Server error: DB operation failed.')**

**return ERROR**

***# fetch results (its a list)***

**results = self.cursor.fetchone()**

**if not results:**

***#print "get\_customer\_profile():Request failure-Record does not exist"***

**self.send\_error('GET\_RESPONSE','Record does not exist')**

**return ERROR**

**else:**

***#print str(results[0])+","+results[1]+","+results[2]+","+results[3]+ \***

***#","+results[4]+","+results[5]+","+results[6]***

**resp.resType = 'GET\_RESPONSE'**

**resp.resParams['status']='SUCCESS'**

**resp.resParams['customer\_id']=results[0]**

**resp.resParams['first\_name']=results[1]**

**resp.resParams['last\_name']=results[2]**

**resp.resParams['DOB']=results[3]**

**resp.resParams['email']=results[4]**

**resp.resParams['phone']=results[5]**

**resp.resParams['address']=str(results[6])+","+str(results[7])+","+str(results[8]) \**

**+","+str(results[9])+","+str(results[10])+","+str(results[11])**

**self.sock.send(resp.toString())**

**return SUCCESS**

**def get\_customer\_id(self, req):**

**print"called get\_customer\_id()"**

**if "customer\_name" not in req.reqParams or \**

**req.reqParams["customer\_name"] == "" :**

**self.send\_error('GET\_RESPONSE','Invalid Username or password')**

**return ERROR**

**else:**

**resp = Response()**

**sql = "SELECT \* from customer\_login WHERE USERNAME='{x}'".format\**

**(x=req.reqParams["customer\_name"])**

***#print "sql query ==> " + sql***

**try:**

**self.cursor.execute(sql)**

**except Exception as e:**

**print e**

***#print " DB execute error"***

**self.send\_error('GET\_RESPONSE', 'Server side error')**

**return ERROR**

***# fetch results (its a list)***

**results = self.cursor.fetchone()**

**if not results:**

**print "get\_customer\_id:Request failure-Record does not exist"**

**self.send\_error('GET\_RESPONSE','Record does not exist')**

**return ERROR**

**else:**

**print "User\_name: "+ req.reqParams['customer\_name'] +","+"Id: "+ str(results[2])**

**resp.resType = 'GET\_RESPONSE'**

**resp.resParams['status']='SUCCESS'**

**resp.resParams['client\_id']=results[2]**

**self.sock.send(resp.toString())**

**return SUCCESS**

**def get\_customer\_chk\_acct(self,customer\_id):**

**print"called get\_customer\_chk\_acct()"**

**sql = "SELECT \* from ACCOUNT\_TABLE WHERE CUSTOMER\_ID='{x}'".format(x=customer\_id)**

***#print "sql query ==> " + sql***

**try:**

**self.cursor.execute(sql)**

**except Exception as e:**

**print e**

**return ERROR**

***# fetch results (its a list)***

**results = self.cursor.fetchone()**

**if not results:**

**return ERROR**

**else:**

**return results[1], results[3]**

**def get\_customer\_sav\_acct(self,customer\_id):**

**print "called get\_customer\_sav\_acct()"**

**sql = "SELECT \* from ACCOUNT\_TABLE WHERE CUSTOMER\_ID='{x}'".format(x=customer\_id)**

***#print "sql query ==> " + sql***

**try:**

**self.cursor.execute(sql)**

**except Exception as e:**

**print e**

**return ERROR**

***# fetch results (its a list)***

**results = self.cursor.fetchone()**

**if not results:**

**return ERROR**

**else:**

**return results[2], results[4]**

**def get\_customer\_acct(self, req):**

**print"called get\_customer\_acct()"**

**if "customer\_id" not in req.reqParams or \**

**req.reqParams["customer\_id"] == "":**

**self.send\_error('GET\_RESPONSE','Invalid key-value entries for customer\_id')**

**return ERROR**

**else:**

**chk\_acct\_num, chk\_acct\_bal = self.get\_customer\_chk\_acct(req.reqParams['customer\_id'])**

**sav\_acct\_num, sav\_acct\_bal = self.get\_customer\_sav\_acct(req.reqParams['customer\_id'])**

**if chk\_acct\_num == '' or chk\_acct\_bal == '' or \**

**sav\_acct\_num == '' or sav\_acct\_num == '':**

***#print "Failed to GET Records"***

**self.send\_error('GET\_RESPONSE','Get operation failed')**

**return ERROR**

**else:**

**resp = Response()**

**resp.resType = 'GET\_RESPONSE'**

**resp.resParams['status']='SUCCESS'**

**resp.resParams['customer\_id'] = req.reqParams['customer\_id']**

**resp.resParams['chk\_acct'] = chk\_acct\_num**

**resp.resParams['chk\_bal'] = chk\_acct\_bal**

**resp.resParams['sav\_acct'] = sav\_acct\_num**

**resp.resParams['sav\_bal'] = sav\_acct\_bal**

**self.sock.send(resp.toString())**

**return SUCCESS**

**def get\_customer\_transactions(self, req):**

**transaction\_list = []**

**print "Function:get\_customer\_transactions() ==>"**

**if "customer\_id" not in req.reqParams or \**

**req.reqParams["customer\_id"] == "" :**

**self.send\_error('GET\_RESPONSE','Invalid Customer ID')**

**return ERROR**

**else:**

**cur\_date = time.strftime("%m/%d/%Y")**

**cur\_month,cur\_day,cur\_year = cur\_date.split('/')**

**print"cur\_month = "+str(cur\_month)**

**sql = "SELECT \* FROM TRANSACTION\_TABLE"**

**print"sql -> "+sql**

**try:**

**self.cursor.execute(sql)**

**for row in self.cursor.fetchall():**

**print"Next Row = "+ str(row)**

**print"req.reqParams['customer\_id'] = "+req.reqParams['customer\_id']**

**print"row[0] = "+str(row[0])**

**dmonth, ddate, dyear=row[1].split('/')**

**print str(dmonth)+" "+ddate+ " "+dyear**

**if cur\_month == dmonth and req.reqParams['customer\_id'] == str(row[0]):**

**print"Above row matches criteria"**

**temp\_str = "CUSTOMER\_ID-"+str(row[0])+" DATE-"+ \**

**str(row[1])+" TIME-"+str(row[2])+" ACCOUNT-"+str(row[3]) \**

**+" TRNSTYPE-"+str(row[4])+" AMOUNT-"+str(row[5]) + \**

**" FROM\_ACCT-"+str(row[6])+" TO\_ACCT-"+str(row[7])**

**print"temp\_str = "+temp\_str**

**transaction\_list.append(temp\_str)**

**print transaction\_list**

**except Exception as e:**

**print e**

**self.send\_error('GET\_RESPONSE', 'Server error: DB operation failed.')**

**return ERROR**

**resp = Response()**

**resp.resType = 'GET\_RESPONSE'**

**resp.resParams['status']='SUCCESS'**

**print "Transaction list = "**

**print transaction\_list**

**trnslist\_str = ' '.join(transaction\_list)**

**print trnslist\_str**

**resp.resParams['trns\_list']=' '.join(transaction\_list)**

**self.sock.send(resp.toString())**

**return SUCCESS**

**def get\_transaction\_record(self, customer\_id):**

**print"called get\_transaction\_record()"**

**sql = "SELECT \* from TRANSACTION\_TABLE WHERE CUSTOMER\_ID='{x}'".format(x=customer\_id)**

**print "sql query ==> " + sql**

**try:**

**self.cursor.execute(sql)**

**except Exception as e:**

**print e**

**return ERROR**

***# fetch results (its a list)***

**results = self.cursor.fetchone()**

**print "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"**

**print results**

**print "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"**

**if not results:**

**print "get\_transaction\_record():Request failure - Record does not exist"**

**return results**

**else:**

***#print str(results[0])+","+str(results[1])+","+str(results[2])+"," \***

***# +str(results[3])+","+str(results[4])+","+str(results[5])+","+str(results[6])***

**print "get\_transaction\_record(): results = "**

**print results**

**print "returning from get\_transaction\_record()"**

**return results**

**def get\_teller\_ids(self, req):**

**print"Function: get\_teller\_ids()==>"**

**sql = "SELECT \* FROM customer\_login WHERE CLIENT\_TYPE = \'Teller\'"**

**print "sql query ==> " + sql**

**try:**

**self.cursor.execute(sql)**

**except Exception as e:**

**print e**

**return ERROR**

**total\_num = 0**

**resp = Response()**

**resp.resType = 'GET\_RESPONSE'**

**resp.resParams['status']='SUCCESS'**

***# fetch results (its a list)***

**for row in self.cursor.fetchall():**

**total\_num = total\_num + 1**

**teller = 'teller'+str(total\_num)**

**resp.resParams[teller] = str(row[0])+"/"+str(row[2])**

**print"total\_num after loop = " + str(total\_num)**

**resp.resParams['total\_teller\_num'] = total\_num**

**self.sock.send(resp.toString())**

**return SUCCESS**

**def get\_teller\_profile(self,req):**

**print "called get\_teller\_profile()"**

**if "teller\_id" not in req.reqParams or \**

**req.reqParams["teller\_id"] == "" :**

**self.send\_error('GET\_RESPONSE','Invalid Teller ID')**

**return ERROR**

**else:**

**resp = Response()**

**sql = "SELECT \* from TELLER\_INFO\_TABLE WHERE TELLER\_ID='{x}'".format(x=req.reqParams["teller\_id"])**

**print "sql query ==> " + sql**

**try:**

**self.cursor.execute(sql)**

**except Exception as e:**

**print e**

**self.send\_error('GET\_RESPONSE', 'Server error: DB operation failed.')**

**return ERROR**

***# fetch results (its a list)***

**results = self.cursor.fetchone()**

**if not results:**

***#print "get\_teller\_profile():Request failure-Record does not exist"***

**self.send\_error('GET\_RESPONSE','Record does not exist')**

**return ERROR**

**else:**

***#print str(results[0])+","+results[1]+","+results[2]+","+results[3]+ \***

***#","+results[4]+","+results[5]+","+results[6]***

**resp.resType = 'GET\_RESPONSE'**

**resp.resParams['status']='SUCCESS'**

**resp.resParams['teller\_id']=results[0]**

**resp.resParams['first\_name']=results[1]**

**resp.resParams['last\_name']=results[2]**

**resp.resParams['DOB']=results[3]**

**resp.resParams['email']=results[4]**

**resp.resParams['phone']=results[5]**

**resp.resParams['address']=str(results[6])+","+str(results[7])+","+str(results[8]) \**

**+","+str(results[9])+","+str(results[10])+","+str(results[11])**

**self.sock.send(resp.toString())**

**return SUCCESS**

**def get\_month\_transactions(self, req):**

***# Form a list of transactions (Each transaction is a list)***

***# to send it to the client***

**transaction\_list = []**

**print "called get\_month\_transactions()"**

**if "month" not in req.reqParams or \**

**req.reqParams["month"] == "" :**

**self.send\_error('GET\_RESPONSE','Invalid month field')**

**return ERROR**

**else:**

**sql = "SELECT \* FROM TRANSACTION\_TABLE"**

**print"sql -> "+sql**

**try:**

**self.cursor.execute(sql)**

**for row in self.cursor.fetchall():**

**print"Next Row = "+ str(row)**

**dmonth, ddate, dyear=row[1].split('/')**

**if req.reqParams["month"] == dmonth:**

***#sql = "SELECT \* FROM TRANSACTION\_TABLE WHERE CUSTOMER\_ID="+str(row[0])***

***#print"sql -> "+sql***

***#self.cursor.execute(sql)***

***# fetch results (its a list)***

***#results = self.cursor.fetchone()***

**print"Above row matches criteria"**

**temp\_str = "CUSTOMER\_ID-"+str(row[0])+" DATE-"+ \**

**str(row[1])+" TIME-"+str(row[2])+" ACCOUNT-"+str(row[3]) \**

**+" TRNSTYPE-"+str(row[4])+" AMOUNT-"+str(row[5]) + \**

**" FROM\_ACCT-"+str(row[6])+" TO\_ACCT-"+str(row[7])**

**print"temp\_str = "+temp\_str**

**transaction\_list.append(temp\_str)**

**print transaction\_list**

**except Exception as e:**

**print e**

**self.send\_error('GET\_RESPONSE', 'Server error: DB operation failed.')**

**return ERROR**

**resp = Response()**

**resp.resType = 'GET\_RESPONSE'**

**resp.resParams['status']='SUCCESS'**

**print "Transaction list = "**

***#print transaction\_list***

**trnslist\_str = ' '.join(transaction\_list)**

**print trnslist\_str**

**resp.resParams['trns\_list']=' '.join(transaction\_list)**

**self.sock.send(resp.toString())**

**return SUCCESS**

**def send\_pdf\_statement(self, customer\_id\_fetched):**

***#db = MySQLdb.connect(host="127.0.0.1", user="root", passwd="vaibhav", db="Bank")***

**print"Function:send\_pdf\_statement() ==>"**

**cursor\_month = self.db.cursor()**

***#date\_of\_txn = (datetime.datetime.now()).strftime("%Y-%m-%d")***

***#time\_of\_txn = (datetime.datetime.now()).strftime("%H:%M")***

**date\_of\_txn = time.strftime("%Y-%m-%d")**

**time\_of\_txn = time.strftime("%H:%M")**

***#***

**cursor\_month.execute("SELECT FIRST\_NAME, LAST\_NAME FROM CUSTOMER\_INFO\_TABLE \**

**WHERE CUSTOMER\_ID=%s LIMIT 1" %customer\_id\_fetched)**

**tup\_name = cursor\_month.fetchone()**

**first\_name = tup\_name[0]**

**last\_name = tup\_name[1]**

**customer\_name = first\_name + " " + last\_name**

**cursor\_month.execute("SELECT CHECKING\_ACCOUNT\_NUM,SAVING\_ACCOUNT\_NUM FROM \**

**ACCOUNT\_TABLE WHERE CUSTOMER\_ID=%s LIMIT 1" %customer\_id\_fetched)**

**tup\_name\_account = cursor\_month.fetchone()**

**checking\_acct = tup\_name\_account[0]**

**saving\_acct = tup\_name\_account[1]**

**cursor\_month.execute("SELECT \* FROM TRANSACTION\_TABLE WHERE CUSTOMER\_ID=%s \**

**ORDER BY DATE" %customer\_id\_fetched)**

**tup\_fetched\_name = cursor\_month.fetchall()**

**pdf = FPDF()**

**pdf.add\_page()**

**pdf.set\_font('Arial','B',16)**

**pdf.cell(5,10,"Monthly statements ")**

**pdf.ln(5)**

**pdf.cell(5,10,"Name "+'{:>42}'.format(" : ") + customer\_name)**

**pdf.ln(5)**

**pdf.cell(5,10,"Customer ID " + '{:>31}'.format(" : ") + str(customer\_id\_fetched))**

**pdf.ln(5)**

**pdf.cell(5,10,"Checking Account Number " + '{:>5}'.format(" : ") + str(checking\_acct))**

**pdf.ln(5)**

**pdf.cell(5,10,"Saving Account Number " + '{:>10}'.format(" : ") + str(saving\_acct))**

**pdf.ln(5)**

**pdf.cell(5,10,"Date of transaction " + '{:>19}'.format(" : ") + date\_of\_txn)**

**pdf.ln(5)**

**pdf.cell(5,10,"Time of transaction " + '{:>19}'.format(" : " )+ time\_of\_txn)**

**pdf.ln(5)**

**pdf.set\_font('Arial','B',12)**

**pdf.cell(5,30,"Date of transaction " + " Time of transaction " + " From acct number \**

**" + " To acct number " + " Amount ")**

**print tup\_fetched\_name[0][1]**

**initial\_month\_year\_of\_txn = str(calendar.month\_name[int(str(tup\_fetched\_name[0][1]).split("/")[0])])**

**print initial\_month\_year\_of\_txn**

**date\_list = str(tup\_fetched\_name[0][1]).split("/")**

**print date\_list**

**file\_name\_prev = str(customer\_id\_fetched) + "\_" + \**

**str(calendar.month\_name[int(date\_list[0])]) + " " + date\_list[2]**

**print file\_name\_prev**

**for val in tup\_fetched\_name:**

**date\_list = str(val[1]).split("/")**

**month\_year\_of\_txn = str(calendar.month\_name[int(date\_list[0])]) + " " + date\_list[2]**

**file\_name = str(customer\_id\_fetched) + "\_" + month\_year\_of\_txn + ".pdf"**

**date\_of\_transfer\_txn = val[1]**

**time\_of\_transfer\_txn = val[2]**

**acct\_type = val[3]**

**trns\_type = val[4]**

**from\_account\_number = val[6]**

**to\_account\_number =val[7]**

**amount\_txn = val[5]**

**if initial\_month\_year\_of\_txn != str(calendar.month\_name[int(date\_list[0])]) :**

**pdf.output(file\_name\_prev,'F')**

**self.smtpmail(customer\_id\_fetched,file\_name\_prev)**

**break**

**else:**

**pdf.ln(5)**

**pdf.cell(5,35,'{:>15}'.format(str(date\_of\_transfer\_txn)) + \**

**'{:>18}'.format(str(time\_of\_transfer\_txn)) + \**

**'{:>20}'.format(str(acct\_type))+ \**

**'{:>20}'.format(str(trns\_type))+ \**

**'{:>37}'.format(str(from\_account\_number)) + \**

**'{:>30}'.format(str(to\_account\_number))+ \**

**'{:>20}'.format(str(amount\_txn)))**

**pdf.ln(5)**

**pdf.output(file\_name,'F')**

**self.smtpmail(customer\_id\_fetched,file\_name)**

***#db.close()***

**cursor\_month.close()**

**def smtpmail(self, customer\_id\_fetched, file\_name):**

**cursor = self.db.cursor()**

**cursor.execute("SELECT EMAIL\_ID FROM CUSTOMER\_INFO\_TABLE WHERE CUSTOMER\_ID = {} ".format(customer\_id\_fetched))**

**to\_mail\_id = cursor.fetchone()[0]**

**print to\_mail\_id**

**msg = MIMEMultipart()**

**msg['From'] = 'kgus.bank@gmail.com'**

**msg['To'] = to\_mail\_id**

**msg['Subject'] = "KGUS BANK ONLINE STATEMENT"**

**body = "Dear customer:" \**

**"\n\nYou can see the latest statement for your account." \**

**"\n\nSincerely," \**

**"\nKgus bank Team"**

**msg.attach(MIMEText(body, 'plain'))**

**attachment = open(file\_name, "rb")**

**part = MIMEBase('application', 'octet-stream')**

**part.set\_payload((attachment).read())**

**encoders.encode\_base64(part)**

**part.add\_header('Content-Disposition', "attachment; filename= %s" % file\_name)**

**msg.attach(part)**

**server = smtplib.SMTP('smtp.gmail.com', 587)**

**server.ehlo()**

**server.starttls()**

**server.login('kgus.bank@gmail.com', 'kguskgus')**

**text = msg.as\_string()**

**server.sendmail('kgus.bank@gmail.com', to\_mail\_id, text)**

**server.quit()**

**def get\_monthly\_statement(self,req):**

**transaction\_list = []**

**print "Function:get\_monthly\_statement() ==>"**

**if "customer\_id" not in req.reqParams or \**

**req.reqParams["customer\_id"] == "" :**

**self.send\_error('GET\_RESPONSE','Invalid Customer ID')**

**return ERROR**

**else:**

**cur\_date = time.strftime("%m/%d/%Y")**

**cur\_month,cur\_day,cur\_year = cur\_date.split('/')**

**print"cur\_month = "+str(cur\_month)**

**sql = "SELECT \* FROM TRANSACTION\_TABLE"**

**print"sql -> "+sql**

**try:**

**self.cursor.execute(sql)**

**for row in self.cursor.fetchall():**

**print"Next Row = "+ str(row)**

**print"req.reqParams['customer\_id'] = "+req.reqParams['customer\_id']**

**print"row[0] = "+str(row[0])**

**dmonth, ddate, dyear=row[1].split('/')**

**print str(dmonth)+" "+ddate+ " "+dyear**

**if cur\_month == dmonth and req.reqParams['customer\_id'] == str(row[0]):**

**print"Above row matches criteria"**

**temp\_str = "CUSTOMER\_ID-"+str(row[0])+" DATE-"+ \**

**str(row[1])+" TIME-"+str(row[2])+" ACCOUNT-"+str(row[3]) \**

**+" TRNSTYPE-"+str(row[4])+" AMOUNT-"+str(row[5]) + \**

**" FROM\_ACCT-"+str(row[6])+" TO\_ACCT-"+str(row[7])**

***#print"temp\_str = "+temp\_str***

**transaction\_list.append(temp\_str)**

***#print transaction\_list***

**except Exception as e:**

**print e**

**self.send\_error('GET\_RESPONSE', 'Server error: DB operation failed.')**

**return ERROR**

**resp = Response()**

**resp.resType = 'GET\_RESPONSE'**

**resp.resParams['status']='SUCCESS'**

**print "Transaction list = "**

**print transaction\_list**

**trnslist\_str = ' '.join(transaction\_list)**

**print trnslist\_str**

**resp.resParams['trns\_list']=' '.join(transaction\_list)**

**self.sock.send(resp.toString())**

**print"calling send\_pdf"**

**self.send\_pdf\_statement(req.reqParams["customer\_id"] )**

**return SUCCESS**

***################################# UPDATE FUNCTIONS ###############################***

**def update\_customer\_acct(self, req):**

**print"called update\_customer\_acct()"**

**if 'customer\_id' not in req.reqParams or \**

**req.reqParams['customer\_id'] == "" or \**

**'chk\_acct\_num' not in req.reqParams or \**

**req.reqParams['chk\_acct\_num'] == "" or \**

**'op\_type' not in req.reqParams or \**

**req.reqParams['op\_type'] == "" or \**

**'amt' not in req.reqParams or \**

**req.reqParams['amt'] == "":**

**self.send\_error('UPDATE\_RESPONSE','Multiple Invalid key-value entries')**

**return ERROR**

**amt = req.reqParams['amt']**

**amt = float(amt)**

**if req.reqParams['subreq\_type'] == 'UPDATE\_CHK\_ACCT':**

**acct, bal = self.get\_customer\_chk\_acct(req.reqParams['customer\_id'])**

**else:**

**acct, bal = self.get\_customer\_sav\_acct(req.reqParams['customer\_id'])**

**bal = float(bal)**

**if(req.reqParams['op\_type'] == 'SUBTRACT'):**

**trns\_type = 'WITHDRAW'**

**bal = bal - amt**

**if bal < 10:**

***# send update failure***

**self.send\_error('UPDATE\_RESPONSE', 'Minimum balance should be $10')**

**return ERROR**

**elif(req.reqParams['op\_type'] == 'ADD'):**

**trns\_type = 'DEPOSIT'**

**bal = bal + amt**

**else:**

**self.send\_error('UPDATE\_RESPONSE', 'Unknown operation type sent')**

**return ERROR**

**if req.reqParams['subreq\_type'] == 'UPDATE\_CHK\_ACCT':**

**acct\_type = 'CHECKING'**

**sql = "UPDATE ACCOUNT\_TABLE SET CHECKING\_ACCOUNT\_BAL = '{x}' WHERE CHECKING\_ACCOUNT\_NUM ='{y}'".format(x=bal, y=acct)**

**else:**

**acct\_type = 'SAVING'**

**sql = "UPDATE ACCOUNT\_TABLE SET SAVING\_ACCOUNT\_BAL = '{x}' WHERE SAVING\_ACCOUNT\_NUM ='{y}'".format(x=bal, y=acct)**

**print "sql query ==> " + sql**

**try:**

**self.cursor.execute(sql)**

**self.db.commit()**

**except Exception as e:**

**print e**

**print "UPDATE request failure: server side error"**

**self.db.rollback()**

**self.send\_error('UPDATE\_RESPONSE', 'Server error: DB operation could not be completed')**

**return ERROR**

**resp = Response()**

**resp.resType = 'UPDATE\_RESPONSE'**

**resp.resParams['status']='SUCCESS'**

**self.sock.send(resp.toString())**

**print"User ACCOUNT record updated successfully"**

**status = self.insert\_transaction\_record(req.reqParams['customer\_id'],trns\_type,acct\_type,amt,"","")*#acct,req.reqParams['to\_acct'])***

***#status = self.update\_transactions(req.reqParams['customer\_id'],trns\_type,acct\_type,amt)***

**return status**

**def update\_customer\_profile(self,req):**

**print"called update\_customer\_profile()"**

**if 'customer\_id' not in req.reqParams or \**

**req.reqParams['customer\_id'] == "":**

**self.send\_error('UPDATE\_RESPONSE','Multiple Invalid key-value entries')**

**return ERROR**

**elems = []**

**if 'first\_name' in req.reqParams and req.reqParams['first\_name'] != '':**

**elems.append("FIRST\_NAME='{}'".format(req.reqParams['first\_name']))**

**print elems**

**if 'last\_name' in req.reqParams and req.reqParams['last\_name'] != '':**

**elems.append("LAST\_NAME='{}'".format(req.reqParams['last\_name']))**

**print elems**

**if 'DOB' in req.reqParams and req.reqParams['DOB'] != '':**

**elems.append("DATE\_OF\_BIRTH='{}'".format(req.reqParams['DOB']))**

**print elems**

**if 'email' in req.reqParams and req.reqParams['email'] != '':**

**elems.append("EMAIL\_ID='{}'".format(req.reqParams['email']))**

**print elems**

**if 'phone' in req.reqParams and req.reqParams['phone'] != '':**

**elems.append("PHONE\_NUMBER='{}'".format(req.reqParams['phone']))**

**print elems**

**if 'apt\_num' in req.reqParams and req.reqParams['apt\_num'] != '':**

**elems.append("APT\_NUM='{}'".format(req.reqParams['apt\_num']))**

**print elems**

**if 'street\_name' in req.reqParams and req.reqParams['street\_name']:**

**elems.append("STREET\_NAME='{}'".format(req.reqParams['street\_name']))**

**print elems**

**if 'city' in req.reqParams and req.reqParams['city'] != '':**

**elems.append("CITY='{}'".format(req.reqParams['city']))**

**print elems**

**if 'state' in req.reqParams and req.reqParams['state'] != '':**

**elems.append("STATE='{}'".format(req.reqParams['state']))**

**print elems**

**if 'country' in req.reqParams and req.reqParams['country'] != '':**

**elems.append("COUNTRY='{}'".format(req.reqParams['country']))**

**print elems**

**if 'zipcode' in req.reqParams and req.reqParams['zipcode']:**

**elems.append("ZIPCODE='{}'".format(req.reqParams['zipcode']))**

**print elems**

**if 'gender' in req.reqParams and req.reqParams['gender']:**

**elems.append("GENDER='{}'".format(req.reqParams['gender']))**

**print elems**

**updates = ','.join(elems)**

**print "elements Joined:" + updates**

**sql = "UPDATE CUSTOMER\_INFO\_TABLE SET {e} WHERE \**

**CUSTOMER\_ID ='{y}'".format(e=updates, y=req.reqParams['customer\_id'])**

**print "sql -> "+sql**

**try:**

**self.cursor.execute(sql)**

**self.db.commit()**

**except Exception as e:**

**print e**

***#print "UPDATE request failure: server side error"***

**self.db.rollback()**

**self.send\_error('UPDATE\_RESPONSE', 'Server error: DB operation could not be completed')**

**return ERROR**

**resp = Response()**

**resp.resType = 'UPDATE\_RESPONSE'**

**resp.resParams['status']='SUCCESS'**

**self.sock.send(resp.toString())**

**return SUCCESS**

**def transfer\_money(self, req):**

**print"transfer\_money(): ==>"**

**if 'to\_bank' not in req.reqParams or \**

**req.reqParams['to\_bank'] == "" or \**

**'to\_acct' not in req.reqParams or \**

**req.reqParams['to\_acct'] == "":**

**self.send\_error('UPDATE\_RESPONSE','Multiple Invalid key-value entries')**

**return ERROR**

**if(req.reqParams['acct\_type'] == 'checking'):**

**req.reqParams['subreq\_type'] = 'UPDATE\_CHK\_ACCT'**

**print "transfer\_money(): subrequest = UPDATE\_CHK\_ACCT & acct\_type = checking"**

**else:**

**print"transfer\_money(): inside else block"**

**print "acct\_type set to " + req.reqParams['acct\_type']**

**req.reqParams['subreq\_type'] = 'UPDATE\_SAV\_ACCT'**

***# Here goes the code to take care of to\_acct and to\_bank***

***# Now update user account***

**trns\_type = 'TRANSFER'**

**self.update\_user\_acct(req, trns\_type)**

**def update\_user\_acct(self, req, trns\_type):**

**print"update\_user\_acct() ==> "**

**if 'customer\_id' not in req.reqParams or \**

**req.reqParams['customer\_id'] == "" or \**

**'chk\_acct\_num' not in req.reqParams or \**

**req.reqParams['chk\_acct\_num'] == "" or \**

**'op\_type' not in req.reqParams or \**

**req.reqParams['op\_type'] == "" or \**

**'amt' not in req.reqParams or \**

**req.reqParams['amt'] == "":**

**self.send\_error('UPDATE\_RESPONSE','Multiple Invalid key-value entries')**

**return ERROR**

**amt = req.reqParams['amt']**

**amt = float(amt)**

**if req.reqParams['subreq\_type'] == 'UPDATE\_CHK\_ACCT':**

**acct, bal = self.get\_customer\_chk\_acct(req.reqParams['customer\_id'])**

**print "acct = "+ str(acct) + " bal = "+ str(bal)**

**else:**

**acct, bal = self.get\_customer\_sav\_acct(req.reqParams['customer\_id'])**

**print "acct = "+ str(acct) + " bal = "+ str(bal)**

**bal = float(bal)**

**if(req.reqParams['op\_type'] == 'SUBTRACT'):**

***#trns\_type = 'WITHDRAW'***

**bal = bal - amt**

**if bal < 10:**

***# send update failure***

**self.send\_error('UPDATE\_RESPONSE', 'Minimum balance should be $10')**

**return ERROR**

**elif(req.reqParams['op\_type'] == 'ADD'):**

***#trns\_type = 'DEPOSIT'***

**bal = bal + amt**

**else:**

**self.send\_error('UPDATE\_RESPONSE', 'Unknown operation type sent')**

**return ERROR**

**if req.reqParams['subreq\_type'] == 'UPDATE\_CHK\_ACCT':**

**print"acct\_type getting set to CHECKING"**

**acct\_type = 'CHECKING'**

**sql = "UPDATE ACCOUNT\_TABLE SET CHECKING\_ACCOUNT\_BAL = '{x}' WHERE CHECKING\_ACCOUNT\_NUM ='{y}'".format(x=bal, y=acct)**

**else:**

**print "acct\_type getting set to SAVIG"**

**acct\_type = 'SAVING'**

**sql = "UPDATE ACCOUNT\_TABLE SET SAVING\_ACCOUNT\_BAL = '{x}' WHERE SAVING\_ACCOUNT\_NUM ='{y}'".format(x=bal, y=acct)**

**print "sql query ==> " + sql**

**try:**

**self.cursor.execute(sql)**

**self.db.commit()**

**except Exception as e:**

**print e**

**print "UPDATE request failure: server side error"**

**self.db.rollback()**

**self.send\_error('UPDATE\_RESPONSE', 'Server error: DB operation could not be completed')**

**return ERROR**

**resp = Response()**

**resp.resType = 'UPDATE\_RESPONSE'**

**resp.resParams['status']='SUCCESS'**

**self.sock.send(resp.toString())**

**print"User ACCOUNT record updated successfully"**

**print"Before calling insert transaction record: (trns\_type,acct\_type,amt,acct,to\_actt) "+ \**

**str(trns\_type)+" "+str(acct\_type)+" "+str(amt)+" "+str(acct)**

**status = self.insert\_transaction\_record(req.reqParams['customer\_id'],trns\_type,acct\_type,amt,acct,req.reqParams['to\_acct'])**

**return *#status***

**def update\_teller\_profile(self,req):**

**print"called update\_teller\_profile()"**

**if 'teller\_id' not in req.reqParams or \**

**req.reqParams['teller\_id'] == "":**

**self.send\_error('UPDATE\_RESPONSE','Multiple Invalid key-value entries')**

**return ERROR**

**elems = []**

**if 'first\_name' in req.reqParams and req.reqParams['first\_name'] != '':**

**elems.append("FIRST\_NAME='{}'".format(req.reqParams['first\_name']))**

**print elems**

**if 'last\_name' in req.reqParams and req.reqParams['last\_name'] != '':**

**elems.append("LAST\_NAME='{}'".format(req.reqParams['last\_name']))**

**print elems**

**if 'DOB' in req.reqParams and req.reqParams['DOB'] != '':**

**elems.append("DATE\_OF\_BIRTH='{}'".format(req.reqParams['DOB']))**

**print elems**

**if 'email' in req.reqParams and req.reqParams['email'] != '':**

**elems.append("EMAIL\_ID='{}'".format(req.reqParams['email']))**

**print elems**

**if 'phone' in req.reqParams and req.reqParams['phone'] != '':**

**elems.append("PHONE\_NUMBER='{}'".format(req.reqParams['phone']))**

**print elems**

**if 'apt\_num' in req.reqParams and req.reqParams['apt\_num'] != '':**

**elems.append("APT\_NUM='{}'".format(req.reqParams['apt\_num']))**

**print elems**

**if 'street\_name' in req.reqParams and req.reqParams['street\_name']:**

**elems.append("STREET\_NAME='{}'".format(req.reqParams['street\_name']))**

**print elems**

**if 'city' in req.reqParams and req.reqParams['city'] != '':**

**elems.append("CITY='{}'".format(req.reqParams['city']))**

**print elems**

**if 'state' in req.reqParams and req.reqParams['state'] != '':**

**elems.append("STATE='{}'".format(req.reqParams['state']))**

**print elems**

**if 'country' in req.reqParams and req.reqParams['country'] != '':**

**elems.append("COUNTRY='{}'".format(req.reqParams['country']))**

**print elems**

**if 'zipcode' in req.reqParams and req.reqParams['zipcode']:**

**elems.append("ZIPCODE='{}'".format(req.reqParams['zipcode']))**

**print elems**

**if 'gender' in req.reqParams and req.reqParams['gender']:**

**elems.append("GENDER='{}'".format(req.reqParams['gender']))**

**print elems**

**updates = ','.join(elems)**

**print "elements Joined:" + updates**

**sql = "UPDATE TELLER\_INFO\_TABLE SET {e} WHERE \**

**TELLER\_ID ='{y}'".format(e=updates, y=req.reqParams['teller\_id'])**

**print "sql -> "+sql**

**try:**

**self.cursor.execute(sql)**

**self.db.commit()**

**except Exception as e:**

**print e**

***#print "UPDATE request failure: server side error"***

**self.db.rollback()**

**self.send\_error('UPDATE\_RESPONSE', 'Server error: DB operation could not be completed')**

**return ERROR**

**resp = Response()**

**resp.resType = 'UPDATE\_RESPONSE'**

**resp.resParams['status']='SUCCESS'**

**self.sock.send(resp.toString())**

**return SUCCESS**

***################################## INSERT FUNCTIONS ####################################***

**def insert\_login\_record(self, req):**

**print"called insert\_login\_record()"**

**if 'user\_name' not in req.reqParams or \**

**'password' not in req.reqParams or \**

**'customer\_id' not in req.reqParams or \**

**'record\_client\_type' not in req.reqParams or \**

**req.reqParams['user\_name'] == "" or \**

**req.reqParams['password'] == "" or \**

**req.reqParams['customer\_id'] == "" or \**

**req.reqParams['record\_client\_type'] == "":**

**self.send\_error('INSERT\_RESPONSE','Multiple Invalid key-value entries')**

**return ERROR**

**sql = "INSERT INTO customer\_login(USERNAME,PASSWORD,CUSTOMER\_ID,CLIENT\_TYPE) \**

**VALUES('{a}',MD5('{b}'),'{c}','{d}')".format(a=req.reqParams['user\_name'], \**

**b=req.reqParams['password'], c=req.reqParams['customer\_id'], \**

**d = req.reqParams['record\_client\_type'])**

**try:**

**self.cursor.execute(sql)**

**self.db.commit()**

**except Exception as e:**

**print e**

***#print "INSERT request failure: server side error"***

**self.db.rollback()**

**self.send\_error('INSERT\_RESPONSE', 'Server error: DB operation could not be completed')**

**return ERROR**

**resp = Response()**

**resp.resType = 'INSERT\_RESPONSE'**

**resp.resParams['status']='SUCCESS'**

**self.sock.send(resp.toString())**

**return SUCCESS**

**def insert\_accounts\_record(self, req):**

**print"called insert\_accounts\_record()"**

**if 'customer\_id' not in req.reqParams or \**

**'customer\_chk\_acct' not in req.reqParams or \**

**'customer\_sav\_acct' not in req.reqParams or \**

**'customer\_chk\_bal' not in req.reqParams or \**

**'customer\_sav\_bal' not in req.reqParams or \**

**req.reqParams['customer\_id'] == "" or \**

**req.reqParams['customer\_chk\_acct'] == "" or \**

**req.reqParams['customer\_sav\_acct'] == "" or \**

**req.reqParams['customer\_chk\_bal'] == "" or \**

**req.reqParams['customer\_sav\_bal'] == "":**

**self.send\_error('INSERT\_RESPONSE','Multiple Invalid key-value entries')**

**return ERROR**

**sql = "INSERT INTO ACCOUNT\_TABLE(CUSTOMER\_ID, CHECKING\_ACCOUNT\_NUM, SAVING\_ACCOUNT\_NUM, \**

**CHECKING\_ACCOUNT\_BAL, SAVING\_ACCOUNT\_BAL) VALUES('{a}','{b}','{c}','{d}','{e}')".format \**

**(a=req.reqParams['customer\_id'],b=req.reqParams['customer\_chk\_acct'], \**

**c=req.reqParams['customer\_sav\_acct'],d=req.reqParams['customer\_chk\_bal'], \**

**e=req.reqParams['customer\_sav\_bal'])**

***#print "sql -> "+sql***

**try:**

**self.cursor.execute(sql)**

**self.db.commit()**

**except Exception as e:**

**print e**

***#print "INSERT request failure: server side error"***

**self.db.rollback()**

**self.send\_error('INSERT\_RESPONSE', 'Server error: DB operation could not be completed')**

**return ERROR**

**resp = Response()**

**resp.resType = 'INSERT\_RESPONSE'**

**resp.resParams['status']='SUCCESS'**

**self.sock.send(resp.toString())**

**return SUCCESS**

**def insert\_profile\_record(self, req):**

**print "called insert\_profile\_record()"**

**if 'first\_name' not in req.reqParams or 'last\_name' not in req.reqParams or \**

**'DOB' not in req.reqParams or 'email' not in req.reqParams or \**

**'phone' not in req.reqParams or 'apt\_num' not in req.reqParams \**

**or 'street\_name' not in req.reqParams or 'city' not in req.reqParams or \**

**'state' not in req.reqParams or 'country' not in req.reqParams or \**

**'zipcode' not in req.reqParams or 'gender' not in req.reqParams or \**

**req.reqParams['first\_name'] == "" or req.reqParams['last\_name'] == "" or \**

**req.reqParams['DOB'] == "" or req.reqParams['email'] == "" or \**

**req.reqParams['phone'] == "" or req.reqParams['apt\_num'] == "" \**

**or req.reqParams['street\_name'] == "" or req.reqParams['city'] == "" or \**

**req.reqParams['state'] == "" or req.reqParams['country'] == "" or \**

**req.reqParams['zipcode'] == "" or req.reqParams['gender'] == "":**

**self.send\_error('INSERT\_RESPONSE','Multiple Invalid key-value entries')**

**return ERROR**

**if 'customer\_id' in req.reqParams:**

**print"in customer\_id block"**

**sql = "INSERT INTO CUSTOMER\_INFO\_TABLE(CUSTOMER\_ID, FIRST\_NAME, LAST\_NAME, \**

**DATE\_OF\_BIRTH, EMAIL\_ID,PHONE\_NUMBER,APT\_NUM,STREET\_NAME,CITY,STATE, \**

**COUNTRY, ZIPCODE, GENDER) VALUES('{a}','{b}','{c}','{d}','{e}','{f}', \**

**'{g}','{h}','{i}','{j}','{k}','{l}','{m}')".format(a=req.reqParams['customer\_id'], \**

**b=req.reqParams['first\_name'], c=req.reqParams['last\_name'] ,\**

**d=req.reqParams['DOB'],e=req.reqParams['email'], f=req.reqParams['phone'], \**

**g=req.reqParams['apt\_num'], h=req.reqParams['street\_name'], \**

**i=req.reqParams['city'], j=req.reqParams['state'], k=req.reqParams['country'], \**

**l=req.reqParams['zipcode'], m=req.reqParams['gender'])**

**print "sql -> "+sql**

**elif 'teller\_id' in req.reqParams:**

**print"in teller id block"**

**sql = "INSERT INTO TELLER\_INFO\_TABLE(TELLER\_ID, FIRST\_NAME, LAST\_NAME, \**

**DATE\_OF\_BIRTH, EMAIL\_ID,PHONE\_NUMBER,APT\_NUM,STREET\_NAME,CITY,STATE, \**

**COUNTRY, ZIPCODE, GENDER) VALUES('{a}','{b}','{c}','{d}','{e}','{f}', \**

**'{g}','{h}','{i}','{j}','{k}','{l}','{m}')".format(a=req.reqParams['teller\_id'], \**

**b=req.reqParams['first\_name'], c=req.reqParams['last\_name'] ,\**

**d=req.reqParams['DOB'],e=req.reqParams['email'], f=req.reqParams['phone'], \**

**g=req.reqParams['apt\_num'], h=req.reqParams['street\_name'], \**

**i=req.reqParams['city'], j=req.reqParams['state'], k=req.reqParams['country'], \**

**l=req.reqParams['zipcode'], m=req.reqParams['gender'])**

**print "sql -> "+sql**

**try:**

**self.cursor.execute(sql)**

**self.db.commit()**

**except Exception as e:**

**print e**

***#print "INSERT request failure: server side error"***

**self.db.rollback()**

**self.send\_error('INSERT\_RESPONSE', 'Server error: DB operation could not be completed')**

**return ERROR**

**resp = Response()**

**resp.resType = 'INSERT\_RESPONSE'**

**resp.resParams['status']='SUCCESS'**

**self.sock.send(resp.toString())**

**return SUCCESS**

**def insert\_transaction\_record(self,customer\_id,trns\_type,acct\_type,amount,from\_acct,to\_acct):**

**print"insert\_transaction\_record() ==>"**

**cur\_date = time.strftime("%m/%d/%Y")**

***#print date***

**cur\_time = time.strftime("%X")**

***#print cur\_time***

**cur\_time = cur\_time.replace(':', '.')**

***#print cur\_time***

**sql = "INSERT INTO TRANSACTION\_TABLE(CUSTOMER\_ID,DATE,TIME,ACCOUNT\_TYPE, \**

**TRNS\_TYPE,AMOUNT,FROM\_ACCT,TO\_ACCT) \**

**VALUES('{a}','{b}','{c}','{d}','{e}','{f}','{g}','{h}')".format( \**

**a=customer\_id, b=cur\_date,c=cur\_time, d=acct\_type, e=trns\_type,f=amount, \**

**g=from\_acct, h=to\_acct)**

**print "insert transaction record sql ==>" + sql**

**try:**

**self.cursor.execute(sql)**

**self.db.commit()**

**except Exception as e:**

**print e**

***#print "INSERT request failure: server side error"***

**self.db.rollback()**

**return ERROR**

**print"Transaction record updated successfully"**

**return SUCCESS**

***################################# DELETE FUNCTIONS ################################***

**def delete\_login\_record(self, req):**

**print"called delete\_login\_record()"**

**if "customer\_id" not in req.reqParams or \**

**req.reqParams["customer\_id"] == "" :**

**self.send\_error('DELETE\_RESPONSE','Invalid Customer ID')**

**return ERROR**

**sql = "DELETE FROM customer\_login WHERE CUSTOMER\_ID='{a}'".format \**

**(a=req.reqParams['customer\_id'])**

***#print "sql -> "+sql***

**try:**

**self.cursor.execute(sql)**

**self.db.commit()**

**except Exception as e:**

**print e**

***#print "DELETE request failure: server side error"***

**self.db.rollback()**

**self.send\_error('DELETE\_RESPONSE', 'Server error: DB operation could not be completed')**

**return ERROR**

**resp = Response()**

**resp.resType = 'DELETE\_RESPONSE'**

**resp.resParams['status']='SUCCESS'**

**self.sock.send(resp.toString())**

**return SUCCESS**

**def delete\_accounts\_record(self, req):**

**print"called delete\_accounts\_record()"**

**if "customer\_id" not in req.reqParams or \**

**req.reqParams["customer\_id"] == "" :**

**self.send\_error('DELETE\_RESPONSE','Invalid Customer ID')**

**return ERROR**

**sql = "DELETE FROM ACCOUNT\_TABLE WHERE CUSTOMER\_ID='{a}'".format \**

**(a=req.reqParams['customer\_id'])**

***#print "sql -> "+sql***

**try:**

**self.cursor.execute(sql)**

**self.db.commit()**

**except Exception as e:**

**print e**

***#print "DELETE request failure: server side error"***

**self.db.rollback()**

**self.send\_error('DELETE\_RESPONSE', 'Server error: DB operation could not be completed')**

**return ERROR**

**resp = Response()**

**resp.resType = 'DELETE\_RESPONSE'**

**resp.resParams['status']='SUCCESS'**

**self.sock.send(resp.toString())**

**return SUCCESS**

**def delete\_profile\_record(self, req):**

**print"\ndelete\_profile\_record()"**

**if "customer\_id" in req.reqParams:**

**print "customer id in request ==> "**

**if req.reqParams["customer\_id"] == "" :**

**self.send\_error('DELETE\_RESPONSE','Invalid Customer ID')**

**return ERROR**

**sql = "DELETE FROM CUSTOMER\_INFO\_TABLE WHERE CUSTOMER\_ID='{a}'".format \**

**(a=req.reqParams['customer\_id'])**

**elif "teller\_id" in req.reqParams:**

**print "teller id in request ==> "**

**if req.reqParams["teller\_id"] == "":**

**self.send\_error('DELETE\_RESPONSE', 'Invalid Teller ID')**

**return ERROR**

**sql = "DELETE FROM TELLER\_INFO\_TABLE WHERE TELLER\_ID='{a}'".format \**

**(a=req.reqParams['teller\_id'])**

**print "sql -> "+sql**

**try:**

**self.cursor.execute(sql)**

**self.db.commit()**

**except Exception as e:**

**print e**

**print "DELETE request failure: server side error"**

**self.db.rollback()**

**self.send\_error('DELETE\_RESPONSE', 'Server error: DB operation could not be completed')**

**return ERROR**

**resp = Response()**

**resp.resType = 'DELETE\_RESPONSE'**

**resp.resParams['status']='SUCCESS'**

**self.sock.send(resp.toString())**

**print"profile successfully deleted\n"**

**return SUCCESS**

**def delete\_transaction\_record(self, customer\_id):**

**print"called delete\_transaction\_record()"**

**sql = "DELETE FROM TRANSACTION\_TABLE WHERE CUSTOMER\_ID='{a}'".format(a=customer\_id)**

***#print "sql -> "+sql***

**try:**

**self.cursor.execute(sql)**

**self.db.commit()**

**except Exception as e:**

**print e**

***#print "DELETE request failure: server side error"***

**self.db.rollback()**

**return ERROR**

**print"delete\_transaction\_record(): Record deleted successfully"**

**return SUCCESS**

***######################### SERVER CLASS #########################***

**class Server(threading.Thread):**

**def \_\_init\_\_(self, family, host, port):**

***# store input values for future reference***

**threading.Thread.\_\_init\_\_(self)**

**self.host = host**

**self.port = port**

**self.clientThreads = []**

***# create new server socket and bind to host:port***

**self.ssock = socket.socket(family)**

**self.ssock.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)**

**self.ssock.bind((self.ssock.getsockname()[0], port))**

***#self.ssock.settimeout(30)***

**self.ssock.listen(5)**

***#---------------------------------------------------------------------------------------------------***

***# To enable SSL uncomment ssl.wrap\_socket() call***

***# Wrap Socket with SSL/TLS encryption function***

***# cert generated with openssl req -new -x509 -days 365 -nodes -out cert.pem -keyout cert.pem***

**self.sslSock = ssl.wrap\_socket(self.ssock, \**

**ssl\_version=ssl.PROTOCOL\_TLSv1, server\_side=True, certfile="./cert.pem")**

***#---------------------------------------------------------------------------------------------------***

**def run(self):**

**while True:**

**print "Server Listening for Clients.."**

***# c,addr is a <connected\_socket , client\_address> tuple***

***#------------------------------------------------------***

***# To enable SSL uncomment this line***

**cliSock, cliAddr = self.sslSock.accept()**

***#------------------------------------------------------***

***# To disable SSL uncomment this line***

***#cliSock, cliAddr = self.ssock.accept()***

***#------------------------------------------------------***

**print "======>>>>>>>> New Client request accepted from: [" + str(cliAddr) + "]"**

**cli = Client(cliSock, cliAddr)**

**cli.setDaemon(True)**

**cli.start()**

***######################### MAIN #########################***

**def main():**

**host = '127.0.0.1'**

**port = 5500**

***# Create DataBase object***

**server = Server(socket.AF\_INET, "0.0.0.0", port)**

***#server.start()***

***#server = Server(socket.AF\_INET6, "::", port)***

**server.setDaemon(True)**

**server.start()**

**while True:**

**time.sleep(10)**

**if \_\_name\_\_=='\_\_main\_\_':**

**main()**

* 1. **Client - Terminal**

**import socket**

**import ssl**

**from random import randint**

**import getpass**

**ERROR = 'FAILED'**

**SUCCESS = 'SUCCESS'**

**LOGOUT = 1**

**class Response(object):**

**def \_\_init\_\_(self, buf):**

***#print"Response object created"***

***#print "Response received ==> " + buf***

**print "\n"**

**values = buf.split('::')**

***#print values***

**self.resType = values[0]**

**self.resParams = {}**

**for elem in values[1:]:**

**k, v = elem.split(':')**

***#print"k = "+k+":"+"v = "+v***

**self.resParams[k] = v**

**class Request:**

**def \_\_init\_\_(self):**

**self.reqType = ''**

**self.reqParams = {}**

**def toString(self):**

**s = "{}".format(self.reqType)**

**for k,v in self.reqParams.items():**

**s = "{}::{}:{}".format(s, k, v)**

***#print "Request Sent ==> " + s***

***#print "\n"***

**return s**

***######################### CUSTOMER CLASS #########################***

**class Customer(object):**

**def \_\_init\_\_(self, cli\_obj,\_name,\_pwd,cli\_id):**

**self.cli\_obj = cli\_obj**

**self.cli\_type = 'Customer'**

**self.cli\_id = cli\_id**

**self.cli\_name = \_name**

**self.cli\_pwd = \_pwd**

**tmp\_user\_chk\_act = ''**

**tmp\_user\_sav\_act = ''**

**tmp\_user\_chk\_bal = ''**

**tmp\_user\_sav\_bal = ''**

**def display\_customer\_options(self):**

**while True:**

**print "1. Display checking AND saving acct balance"**

**print "2. Display profile"**

**print "3. Change Profile"**

**print "4. Transfer funds"**

**print "5. Monthly Transactions"**

**print "6. Withdraw"**

**print "7. Deposit"**

**print "8. Log out"**

**choice = raw\_input("your choice ->")**

**if(choice == '1'):**

***# view account balance***

**self.view\_user\_accounts(1)**

**elif(choice == '2'):**

***# view personal information***

**self.view\_user\_profile()**

**elif(choice == '3'):**

***# Update personal information***

**self.update\_user\_profile()**

**elif(choice == '4'):**

***# transfer money***

**self.transfer\_money()**

**elif(choice == '5'):**

**self.view\_monthly\_statements()**

**elif(choice == '6'):**

**print "\n==> Please approach the Teller for withdrawal\n"**

**elif(choice == '7'):**

**print "\n==> Please approach the Teller for Deposital\n"**

**elif(choice == '8'):**

***# Logout***

**return LOGOUT**

**else:**

***# Update Customer profile***

**print "Invalid choice"**

**def view\_monthly\_statements(self):**

***#print"\nFunction: view\_monthly\_statements() ==>"***

***# create request object***

**req\_obj = Request()**

**req\_obj.reqType = 'GET'**

**req\_obj.reqParams['client\_type'] = self.cli\_type**

**req\_obj.reqParams['subreq\_type'] = 'MONTHLY\_STATEMENT'**

**req\_obj.reqParams['customer\_id'] = self.cli\_id**

***#print"sending GET Acct request.."***

**err = self.cli\_obj.sock.sendall(req\_obj.toString())**

***#if err != None:***

***# print" Send ERROR"***

**data = self.cli\_obj.sock.recv(4096)**

**res\_obj = Response(data)**

**if(res\_obj.resParams['status'] == 'SUCCESS'):**

**if(res\_obj.resParams['trns\_list'] == ''):**

**print"==> NO TRANSACTIONS FOUND FOR GIVEN MONTH\n"**

**else:**

**self.print\_transactions(res\_obj.resParams['trns\_list'])**

**else:**

**print"Server error. Failed to get transactions at this time"**

**def print\_transactions(self, trns\_str):**

***#print "\nFunction: print\_transactions() ==> "***

**trns\_list = trns\_str.split()**

***#print trns\_list***

***#print"\n"***

**print"%-3s %-14s %-12s %-10s %-8s %-18s %-12s %-13s %-15s" % ('NUM','CUSTOMER ID', \**

**'DATE','TIME','ACCOUNT','TRANSACTION TYPE','AMOUNT','FROM ACCOUNT', 'TO\_ACCOUNT')**

**print"---------------------------------------------------------------------------------------------------------------"**

**for i in range(0,len(trns\_list),8):**

**for j in range(0,8):**

***#print trns\_list[i+j]***

**key, t\_id = trns\_list[i+0].split('-')**

**key, t\_date = trns\_list[i+1].split('-')**

**key, t\_time = trns\_list[i+2].split('-')**

**key, t\_acct = trns\_list[i+3].split('-')**

**key, t\_trtype = trns\_list[i+4].split('-')**

**key, t\_amt = trns\_list[i+5].split('-')**

**key, t\_facct = trns\_list[i+6].split('-')**

**if t\_facct is None:**

**t\_facct = ' '**

**key, t\_tacct = trns\_list[i+7].split('-')**

**if t\_facct is None:**

**t\_facct = ' '**

**print "%-3s %-14s %-12s %-10s %-11s %-16s %-12s %-13s %-15s" % (str((i/8)+1),t\_id,t\_date, \**

**str(t\_time),str(t\_acct),str(t\_trtype),str(t\_amt),str(t\_facct),str(t\_tacct))**

***#print "\n"***

**print"---------------------------------------------------------------------------------------------------------------"**

**def view\_user\_accounts(self, display\_flag):**

***#print "view\_customer\_accounts - display\_flag = "+ str(display\_flag)***

***#print"\nFunction: view\_user\_accounts() ==>"***

***# create request object***

**req\_obj = Request()**

**req\_obj.reqType = 'GET'**

**req\_obj.reqParams['client\_type'] = self.cli\_type**

**req\_obj.reqParams['subreq\_type'] = 'CUSTOMER\_ACCT'**

**req\_obj.reqParams['customer\_id'] = self.cli\_id**

***#print"sending GET Acct request.."***

**err = self.cli\_obj.sock.sendall(req\_obj.toString())**

***#if err != None:***

***# print" Send ERROR"***

***#print "Waiting for response.."***

**data = self.cli\_obj.sock.recv(1024)**

***#print "creating Response object"***

**res\_obj = Response(data)**

**if(res\_obj.resParams['status'] == 'SUCCESS'):**

**self.tmp\_user\_chk\_act = res\_obj.resParams['chk\_acct']**

**self.tmp\_user\_chk\_bal = res\_obj.resParams['chk\_bal']**

**self.tmp\_user\_sav\_act = res\_obj.resParams['sav\_acct']**

**self.tmp\_user\_sav\_bal = res\_obj.resParams['sav\_bal']**

**if(display\_flag == 1):**

***#print "display flag 1 block"***

**print"------------------------------------------------"**

**print "Name: "+self.cli\_name+" "+"Customer ID: "+self.cli\_id+"\n"**

**print "Checking Account Balance "**

**print res\_obj.resParams['chk\_acct']+ " $" + res\_obj.resParams['chk\_bal']+"\n"**

**print "Saving Account Balance "**

**print res\_obj.resParams['sav\_acct']+" $"+ res\_obj.resParams['sav\_bal']**

**print"------------------------------------------------\n"**

**elif(display\_flag == 2):**

***#print "display flag 2 block"***

**print"------------------------------------------------"**

**print "Name: "+self.cli\_name+" "+"Customer ID: "+self.cli\_id+"\n"**

**print "Checking Account Balance "**

**print res\_obj.resParams['chk\_acct']+ " $" + res\_obj.resParams['chk\_bal']+"\n"**

**print"------------------------------------------------\n"**

**else:**

***#print "Last else block"***

**print"------------------------------------------------"**

**print "Name: "+self.cli\_name+" "+"Customer ID: "+self.cli\_id+"\n"**

**print "Saving Account Balance "**

**print res\_obj.resParams['sav\_acct']+ " $" + res\_obj.resParams['sav\_bal']+"\n"**

**print"------------------------------------------------\n"**

**else:**

**print "GET :operation failed"**

**def view\_user\_profile(self):**

***#print"\nFunction view\_user\_profile() ==>"***

***# create request object***

**req\_obj = Request()**

**req\_obj.reqType = 'GET'**

**req\_obj.reqParams['client\_type'] = self.cli\_type**

**req\_obj.reqParams['subreq\_type'] = 'CUSTOMER\_PROFILE'**

**req\_obj.reqParams['customer\_id'] = self.cli\_id**

***#print"sending GET Acct request.."***

**err = self.cli\_obj.sock.sendall(req\_obj.toString())**

***#if err != None:***

***# print" Send ERROR"***

**data = self.cli\_obj.sock.recv(1024)**

**res\_obj = Response(data)**

**if(res\_obj.resParams['status'] == 'SUCCESS'):**

**print"\n---------------------------------------------"**

**print"Customer ID : "+res\_obj.resParams['customer\_id']**

**print"First Name : "+res\_obj.resParams['first\_name']**

**print"Last Name : "+res\_obj.resParams['last\_name']**

**print"Date of Birth: "+res\_obj.resParams['DOB']**

**print"Email Address: "+res\_obj.resParams['email']**

**print"Phone Number : "+res\_obj.resParams['phone']**

**print"Address : "+res\_obj.resParams['address']**

**print"---------------------------------------------\n"**

**def update\_user\_profile(self):**

**first\_name = last\_name = dob = email\_id = phone\_num = address = ''**

**state = apt\_num = street\_name = city = country = zipcode = gender = ''**

***#print"\n Function: update\_user\_profile ==>"***

**print"Press Y if you wish to change ==>"**

**choice = raw\_input("First name? ")**

**if choice == 'Y'or choice =='y':**

**first\_name = raw\_input("Enter New First name: ")**

**choice = raw\_input("Last name? ")**

**if choice == 'Y' or choice == 'y':**

**last\_name = raw\_input("Enter New Last name: ")**

**choice = raw\_input("Date of Birth? ")**

**if choice == 'Y' or choice == 'y':**

**dob = raw\_input("Enter New DOB: ")**

**choice = raw\_input("Email ID? ")**

**if choice == 'Y' or choice == 'y':**

**email\_id = raw\_input("Enter New Email ID: ")**

**choice = raw\_input("Phone number? ")**

**if choice == 'Y' or choice == 'y':**

**phone\_num = raw\_input("Enter New Phone number: ")**

**choice = raw\_input("APT number? ")**

**if choice == 'Y' or choice == 'y':**

**apt\_num = raw\_input("Enter New APT number: ")**

**choice = raw\_input("Street Name? ")**

**if choice == 'Y' or choice == 'y':**

**street\_name = raw\_input("Enter New Street name: ")**

**choice = raw\_input("City? ")**

**if choice == 'Y' or choice == 'y':**

**city = raw\_input("Enter New City: ")**

**choice = raw\_input("State? ")**

**if choice == 'Y' or choice == 'y':**

**state = raw\_input("Enter New State: ")**

**choice = raw\_input("Country? ")**

**if choice == 'Y' or choice == 'y':**

**country = raw\_input("Enter New Country: ")**

**choice = raw\_input("Zipcode? ")**

**if choice == 'Y' or choice == 'y':**

**zipcode = raw\_input("Enter New Zipcode: ")**

**choice = raw\_input("Gender? ")**

**if choice == 'Y' or choice == 'y':**

**gender = raw\_input("Enter New Gender: ")**

**if (first\_name == '' and last\_name == '' and dob == '' \**

**and email\_id == '' and phone\_num == '' and address == '' \**

**and apt\_num == '' and street\_name == '' and city == '' \**

**and state == '' and country == '' and zipcode == '' and gender == ''):**

**print"\n-----------------------"**

**print" => No changes opted."**

**print"-------------------------\n"**

**return**

***# create request object***

**req\_obj = Request()**

**req\_obj.reqType = 'SET'**

**req\_obj.reqParams['client\_type'] = self.cli\_type**

**req\_obj.reqParams['customer\_id'] = self.cli\_id**

**req\_obj.reqParams['subreq\_type'] = 'UPDATE\_CUSTOMER\_PROFILE'**

**if first\_name != '':**

**req\_obj.reqParams['first\_name']=first\_name**

**if last\_name != '':**

**req\_obj.reqParams['last\_name']=last\_name**

**if dob != '':**

**req\_obj.reqParams['DOB']=dob**

**if email\_id != '':**

**req\_obj.reqParams['email']=email\_id**

**if phone\_num != '':**

**req\_obj.reqParams['phone']=phone\_num**

**if apt\_num != '':**

**req\_obj.reqParams['apt\_num']=apt\_num**

**if street\_name != '':**

**req\_obj.reqParams['street\_name']=street\_name**

**if city != '':**

**req\_obj.reqParams['city'] = city**

**if state != '':**

**req\_obj.reqParams['state'] = state**

**if country != '':**

**req\_obj.reqParams['country'] = country**

**if zipcode != '':**

**req\_obj.reqParams['zipcode'] = zipcode**

**if gender != '':**

**req\_obj.reqParams['gender'] = gender**

***#print"sending UPDATE profile request.."***

***#print"==> "+req\_obj.toString()***

**err = self.cli\_obj.sock.sendall(req\_obj.toString())**

***#if err != None:***

***# print" Send ERROR"***

***#print "Waiting for response.."***

**data = self.cli\_obj.sock.recv(1024)**

***#print "creating Response object"***

**res\_obj = Response(data)**

**if(res\_obj.resParams['status'] == 'SUCCESS'):**

**print "Customer account updated successfully."**

**self.view\_user\_profile()**

**else:**

**print "Update failure: Customer account could not be updated at this time"**

**def transfer\_money(self):**

***#print"\nFunction: transfer\_money() ==>"***

**print"select account type for transfer:"**

**while True:**

**print "1. Checking"**

**print "2. Saving"**

**choice = raw\_input("choice -> ")**

**if (choice == '1'):**

***#print "tmp\_user\_chk\_act = " + self.tmp\_user\_chk\_act***

**from\_acct = self.tmp\_user\_chk\_act**

***#print"Dispaly flag set to 2"***

**display\_flag = 2**

**break**

**elif (choice == '2'):**

**from\_acct = self.tmp\_user\_sav\_act**

***#print"Display flag set to 3"***

**display\_flag = 3**

**break**

**else:**

**print"Invalid choice"**

**to\_bank = raw\_input("Bank Name: ")**

**to\_acct = raw\_input("Bank Account number: ")**

**amount = raw\_input("Amount: ")**

**req\_obj = Request()**

**req\_obj.reqType = 'SET'**

**req\_obj.reqParams['client\_type'] = self.cli\_type**

**req\_obj.reqParams['customer\_id'] = self.cli\_id**

**req\_obj.reqParams['subreq\_type'] = 'TRANSFER\_MONEY'**

**if (display\_flag == 2):**

**req\_obj.reqParams['acct\_type'] = 'checking'**

**elif(display\_flag ==3):**

**req\_obj.reqParams['acct\_type'] = 'saving'**

**req\_obj.reqParams['chk\_acct\_num'] = from\_acct**

**req\_obj.reqParams['op\_type'] = 'SUBTRACT'**

**req\_obj.reqParams['to\_bank'] = to\_bank**

**req\_obj.reqParams['to\_acct'] = to\_acct**

**req\_obj.reqParams['amt'] = amount**

**err = self.cli\_obj.sock.sendall(req\_obj.toString())**

***#if err != None:***

***# print" Send ERROR"***

***#print "Waiting for response.."***

**data = self.cli\_obj.sock.recv(1024)**

***#print "creating Response object"***

**res\_obj = Response(data)**

**if(res\_obj.resParams['status'] == 'SUCCESS'):**

**print "TRANSFER SUCCESSFUL."**

**self.view\_user\_accounts(display\_flag)**

**else:**

**print "\nTRANSFER FAILED: Error:"+res\_obj.resParams['error']**

***######################### TELLER CLASS #########################***

**class Teller(object):**

**def \_\_init\_\_(self, cli\_obj, cli\_id):**

**self.cli\_obj = cli\_obj**

**self.cli\_type = 'Teller'**

**self.cli\_id = cli\_id**

**self.temp\_cust\_id = ''**

**self.temp\_cust\_name = ''**

**self.temp\_cust\_chk\_acct = ''**

**self.temp\_cust\_chk\_bal = ''**

**self.temp\_cust\_sav\_acct = ''**

**self.temp\_cust\_sav\_bal = ''**

**def display\_teller\_options(self):**

**while True:**

**print "1. Access Customer Account"**

**print "2. Create Customer Account"**

**print "3. Delete Customer Account"**

**print "4. Logout"**

**choice = raw\_input("your choice ->")**

**if(choice == '1'):**

***# Access Customer account***

**self.access\_customer\_account()**

**elif(choice == '2'):**

***# Create Customer account***

**self.create\_customer\_login\_record()**

***#self.create\_customer\_account\_record()***

***#self.create\_customer\_profile\_record()***

**elif(choice == '3'):**

***# Delete Customer account***

**self.delete\_customer\_account()**

**elif(choice == '4'):**

***# Logout***

**return LOGOUT**

**else:**

***# Update Customer profile***

**print "Invalid choice"**

**def access\_customer\_account(self):**

***#print"Function: access\_customer\_account() ==>"***

**customer\_name = raw\_input("Login Name: -> ")**

**customer\_id = 0**

**customer\_id = self.get\_customer\_id(customer\_name)**

**if customer\_id:**

**self.temp\_cust\_name = customer\_name**

**self.temp\_cust\_id = customer\_id**

**else:**

**print"\nCUSTOMER RECORD NOT FOUND\n"**

**return**

**while True:**

**print "1. Manage customer accounts"**

**print "2. Manage customer profile"**

**print "3. Exit Menu"**

**choice = raw\_input("choice -> ")**

**if (choice == '1'):**

**self.manage\_customer\_accounts()**

**elif(choice == '2'):**

**self.manage\_customer\_profile()**

**elif(choice == '3'):**

**return**

**def manage\_customer\_accounts(self):**

***#print"Function:manage\_customer\_accounts() ==>"***

**while True:**

**print "1. View customer account"**

**print "2. View customer Transactions"**

**print "3. Update customer Checking account"**

**print "4. Update customer Saving account"**

**print "5. Exit"**

**choice = raw\_input("choice -> ")**

**if (choice == '1'):**

**self.view\_customer\_accounts('1')**

**elif(choice == '2'):**

**self.view\_customer\_transactions()**

**elif(choice == '3'):**

**self.update\_acct('checking')**

**elif(choice == '4'):**

**self.update\_acct('saving')**

**elif(choice == '5'):**

**return**

**else:**

**print "Invalid choice. Enter again"**

**def manage\_customer\_profile(self):**

***#print"Function:manage\_customer\_profile() ==>"***

**while True:**

**print "1. View customer profile"**

**print "2. Update customer profile"**

**print "3. Exit"**

**choice = raw\_input("choice -> ")**

**if (choice == '1'):**

**self.view\_customer\_profile()**

**elif(choice == '2'):**

**self.update\_customer\_profile()**

**elif(choice == '3'):**

**return**

**else:**

**print "Invalid choice. Enter again"**

**def get\_customer\_id(self, customer\_name):**

***#print"Function:get\_customer\_id() ==>"***

***# create request object***

**req\_obj = Request()**

**req\_obj.reqType = 'GET'**

**req\_obj.reqParams['client\_type'] = self.cli\_type**

**req\_obj.reqParams['subreq\_type'] = 'CUSTOMER\_ID'**

**req\_obj.reqParams['customer\_name'] = customer\_name**

**err = self.cli\_obj.sock.sendall(req\_obj.toString())**

***#if err != None:***

***# print" Send ERROR"***

***#print "Waiting for response.."***

**data = self.cli\_obj.sock.recv(1024)**

***#print "creating Response object"***

**res\_obj = Response(data)**

**if(res\_obj.resParams['status'] == 'SUCCESS'):**

**return res\_obj.resParams['client\_id']**

**else:**

**print "GET :operation failed"**

**def view\_customer\_accounts(self, display\_flag):**

***#print"Function:view\_customer\_accounts() ==>"***

***# create request object***

**req\_obj = Request()**

**req\_obj.reqType = 'GET'**

**req\_obj.reqParams['client\_type'] = self.cli\_type**

**req\_obj.reqParams['subreq\_type'] = 'CUSTOMER\_ACCT'**

**req\_obj.reqParams['customer\_id'] = self.temp\_cust\_id**

***#print"sending GET Acct request.."***

**err = self.cli\_obj.sock.sendall(req\_obj.toString())**

***#if err != None:***

***# print" Send ERROR"***

***#print "Waiting for response.."***

**data = self.cli\_obj.sock.recv(1024)**

***#print "creating Response object"***

**res\_obj = Response(data)**

**if(res\_obj.resParams['status'] == 'SUCCESS'):**

**self.temp\_cust\_chk\_acct = res\_obj.resParams['chk\_acct']**

**self.temp\_cust\_chk\_bal = res\_obj.resParams['chk\_bal']**

**self.temp\_cust\_sav\_acct = res\_obj.resParams['sav\_acct']**

**self.temp\_cust\_sav\_bal = res\_obj.resParams['sav\_bal']**

**if(display\_flag == '1'):**

**print"------------------------------------------------"**

**print "Name: "+self.temp\_cust\_name+" "+"Customer ID: "+self.temp\_cust\_id+"\n"**

**print "Checking Account Balance "**

**print res\_obj.resParams['chk\_acct']+ " $" + res\_obj.resParams['chk\_bal']+"\n"**

**print "Saving Account Balance "**

**print res\_obj.resParams['sav\_acct']+" $"+ res\_obj.resParams['sav\_bal']**

**print"------------------------------------------------\n"**

**elif(display\_flag == '2'):**

**print"------------------------------------------------"**

**print "Name: "+self.temp\_cust\_name+" "+"Customer ID: "+self.temp\_cust\_id+"\n"**

**print "Checking Account Balance "**

**print res\_obj.resParams['chk\_acct']+ " $" + res\_obj.resParams['chk\_bal']+"\n"**

**print"------------------------------------------------\n"**

**else:**

**print"------------------------------------------------"**

**print "Name: "+self.temp\_cust\_name+" "+"Customer ID: "+self.temp\_cust\_id+"\n"**

**print "Saving Account Balance "**

**print res\_obj.resParams['sav\_acct']+ " $" + res\_obj.resParams['sav\_bal']+"\n"**

**print"------------------------------------------------\n"**

**else:**

**print "GET :operation failed"**

**def view\_customer\_profile(self):**

***#print"Function:view\_customer\_profile()==>"***

***# create request object***

**req\_obj = Request()**

**req\_obj.reqType = 'GET'**

**req\_obj.reqParams['client\_type'] = self.cli\_type**

**req\_obj.reqParams['subreq\_type'] = 'CUSTOMER\_PROFILE'**

**req\_obj.reqParams['customer\_id'] = self.temp\_cust\_id**

***#print"sending GET Acct request.."***

**err = self.cli\_obj.sock.sendall(req\_obj.toString())**

***#if err != None:***

***# print" Send ERROR"***

**data = self.cli\_obj.sock.recv(1024)**

**res\_obj = Response(data)**

**if(res\_obj.resParams['status'] == 'SUCCESS'):**

**print"\n---------------------------------------------"**

**print"Customer ID : "+res\_obj.resParams['customer\_id']**

**print"First Name : "+res\_obj.resParams['first\_name']**

**print"Last Name : "+res\_obj.resParams['last\_name']**

**print"Date of Birth: "+res\_obj.resParams['DOB']**

**print"Email Address: "+res\_obj.resParams['email']**

**print"Phone Number : "+res\_obj.resParams['phone']**

**print"Address : "+res\_obj.resParams['address']**

**print"---------------------------------------------\n"**

**def view\_customer\_transactions(self):*#, customer\_id):***

***#print"Function:view\_customer\_transactions() ==>"***

***# create request object***

**req\_obj = Request()**

**req\_obj.reqType = 'GET'**

**req\_obj.reqParams['client\_type'] = self.cli\_type**

**req\_obj.reqParams['subreq\_type'] = 'CUSTOMER\_TRANSACTION'**

**req\_obj.reqParams['customer\_id'] = self.temp\_cust\_id**

***#print"sending GET Acct request.."***

**err = self.cli\_obj.sock.sendall(req\_obj.toString())**

***#if err != None:***

***# print" Send ERROR"***

**data = self.cli\_obj.sock.recv(4096)**

**res\_obj = Response(data)**

**if(res\_obj.resParams['status'] == 'SUCCESS'):**

**if(res\_obj.resParams['trns\_list'] == ''):**

**print"==> NO TRANSACTIONS FOUND FOR GIVEN MONTH\n"**

**else:**

**self.print\_transactions(res\_obj.resParams['trns\_list'])**

**else:**

**print"Server error. Failed to get transactions at this time"**

**def print\_transactions(self, trns\_str):**

***#print "\nFunction: print\_transactions() ==> "***

**trns\_list = trns\_str.split()**

***#print trns\_list***

***#print"\n"***

**print"%-3s %-14s %-12s %-10s %-8s %-18s %-12s %-13s %-15s" % ('NUM','CUSTOMER ID', \**

**'DATE','TIME','ACCOUNT','TRANSACTION TYPE','AMOUNT','FROM ACCOUNT', 'TO\_ACCOUNT')**

**print"---------------------------------------------------------------------------------------------------------------"**

**for i in range(0,len(trns\_list),8):**

**for j in range(0,8):**

***#print trns\_list[i+j]***

**key, t\_id = trns\_list[i+0].split('-')**

**key, t\_date = trns\_list[i+1].split('-')**

**key, t\_time = trns\_list[i+2].split('-')**

**key, t\_acct = trns\_list[i+3].split('-')**

**key, t\_trtype = trns\_list[i+4].split('-')**

**key, t\_amt = trns\_list[i+5].split('-')**

**key, t\_facct = trns\_list[i+6].split('-')**

**if t\_facct is None:**

**t\_facct = ' '**

**key, t\_tacct = trns\_list[i+7].split('-')**

**if t\_facct is None:**

**t\_facct = ' '**

**print "%-3s %-14s %-12s %-10s %-11s %-16s %-12s %-13s %-15s" % (str(i/8+1),t\_id,t\_date, \**

**str(t\_time),str(t\_acct),str(t\_trtype),str(t\_amt),str(t\_facct),str(t\_tacct))**

***#print "\n"***

**print"---------------------------------------------------------------------------------------------------------------"**

**def update\_acct(self,acct\_type):**

***#print"Function:update\_acct() ==>"***

**if (self.temp\_cust\_chk\_acct == '' and self.temp\_cust\_sav\_acct == ''):**

**if acct\_type == 'checking':**

**self.view\_customer\_accounts('2')**

**elif acct\_type == 'saving':**

**self.view\_customer\_accounts('3')**

**while True:**

**print"1. Withdrawal"**

**print"2. Deposit"**

**print"3. Exit"**

**choice = raw\_input("Select your option: -> ")**

**if choice >= '1' or choice <= '3':**

**break**

**else:**

**print"Invalid Input. Enter again"**

**if choice == '3':**

**return**

**if(choice =='1' or choice == '2'):**

**while True:**

**balance = raw\_input("Amount -> ")**

**balance = float(balance)**

**if balance != '':**

**break**

**else:**

**print "Amount Invalid"**

***# create request object***

**req\_obj = Request()**

**req\_obj.reqType = 'SET'**

**req\_obj.reqParams['client\_type'] = self.cli\_type**

**req\_obj.reqParams['customer\_id'] = self.temp\_cust\_id**

**if acct\_type == 'checking':**

**req\_obj.reqParams['subreq\_type'] = 'UPDATE\_CHK\_ACCT'**

**req\_obj.reqParams['chk\_acct\_num'] = self.temp\_cust\_chk\_acct**

**else:**

**req\_obj.reqParams['subreq\_type'] = 'UPDATE\_SAV\_ACCT'**

**req\_obj.reqParams['chk\_acct\_num'] = self.temp\_cust\_sav\_acct**

**if (choice == '1'):**

**req\_obj.reqParams['op\_type'] = 'SUBTRACT'**

**req\_obj.reqParams['amt'] = balance**

**elif (choice == '2'):**

**req\_obj.reqParams['op\_type'] = 'ADD'**

**req\_obj.reqParams['amt'] = balance**

***#print"sending UPDATE Acct request.."***

**err = self.cli\_obj.sock.sendall(req\_obj.toString())**

***#if err != None:***

***# print" Send ERROR"***

***#print "Waiting for response.."***

**data = self.cli\_obj.sock.recv(1024)**

***#print "creating Response object"***

**res\_obj = Response(data)**

**if(res\_obj.resParams['status'] == 'SUCCESS'):**

**print "Customer account updated successfully. New balance =>"**

**if acct\_type == 'checking':**

**self.view\_customer\_accounts('2')**

**elif acct\_type == 'saving':**

**self.view\_customer\_accounts('3')**

**else:**

**print "Update failure: Customer account could not be updated at this time"**

**def update\_customer\_profile(self):**

***#print"Function:update\_customer\_profile() ==>"***

**first\_name = last\_name = dob = email\_id = phone\_num = address = ''**

**state = apt\_num = street\_name = city = country = zipcode = gender = ''**

**print"Press Y if you wish to change ==>"**

**choice = raw\_input("First name? ")**

**if choice == 'Y'or choice =='y':**

**first\_name = raw\_input("Enter New First name: ")**

**choice = raw\_input("Last name? ")**

**if choice == 'Y' or choice == 'y':**

**last\_name = raw\_input("Enter New Last name: ")**

**choice = raw\_input("Date of Birth? ")**

**if choice == 'Y' or choice == 'y':**

**dob = raw\_input("Enter New DOB: ")**

**choice = raw\_input("Email ID? ")**

**if choice == 'Y' or choice == 'y':**

**email\_id = raw\_input("Enter New Email ID: ")**

**choice = raw\_input("Phone number? ")**

**if choice == 'Y' or choice == 'y':**

**phone\_num = raw\_input("Enter New Phone number: ")**

**choice = raw\_input("APT number? ")**

**if choice == 'Y' or choice == 'y':**

**apt\_num = raw\_input("Enter New APT number: ")**

**choice = raw\_input("Street Name? ")**

**if choice == 'Y' or choice == 'y':**

**street\_name = raw\_input("Enter New Street name: ")**

**choice = raw\_input("City? ")**

**if choice == 'Y' or choice == 'y':**

**city = raw\_input("Enter New City: ")**

**choice = raw\_input("State? ")**

**if choice == 'Y' or choice == 'y':**

**state = raw\_input("Enter New State: ")**

**choice = raw\_input("Country? ")**

**if choice == 'Y' or choice == 'y':**

**country = raw\_input("Enter New Country: ")**

**choice = raw\_input("Zipcode? ")**

**if choice == 'Y' or choice == 'y':**

**zipcode = raw\_input("Enter New Zipcode: ")**

**choice = raw\_input("Gender? ")**

**if choice == 'Y' or choice == 'y':**

**gender = raw\_input("Enter New Gender: ")**

**if (first\_name == '' and last\_name == '' and dob == '' \**

**and email\_id == '' and phone\_num == '' and address == '' \**

**and apt\_num == '' and street\_name == '' and city == '' \**

**and state == '' and country == '' and zipcode == '' and gender == ''):**

**print"\n-----------------------"**

**print" => No changes opted."**

**print"-------------------------\n"**

**return**

***# create request object***

**req\_obj = Request()**

**req\_obj.reqType = 'SET'**

**req\_obj.reqParams['client\_type'] = self.cli\_type**

**req\_obj.reqParams['customer\_id'] = self.temp\_cust\_id**

**req\_obj.reqParams['subreq\_type'] = 'UPDATE\_CUSTOMER\_PROFILE'**

**if first\_name != '':**

**req\_obj.reqParams['first\_name']=first\_name**

**if last\_name != '':**

**req\_obj.reqParams['last\_name']=last\_name**

**if dob != '':**

**req\_obj.reqParams['DOB']=dob**

**if email\_id != '':**

**req\_obj.reqParams['email']=email\_id**

**if phone\_num != '':**

**req\_obj.reqParams['phone']=phone\_num**

**if apt\_num != '':**

**req\_obj.reqParams['apt\_num']=apt\_num**

**if street\_name != '':**

**req\_obj.reqParams['street\_name']=street\_name**

**if city != '':**

**req\_obj.reqParams['city'] = city**

**if state != '':**

**req\_obj.reqParams['state'] = state**

**if country != '':**

**req\_obj.reqParams['country'] = country**

**if zipcode != '':**

**req\_obj.reqParams['zipcode'] = zipcode**

**if gender != '':**

**req\_obj.reqParams['gender'] = gender**

***#print"sending UPDATE profile request.."***

***#print"==> "+req\_obj.toString()***

**err = self.cli\_obj.sock.sendall(req\_obj.toString())**

***#if err != None:***

***# print" Send ERROR"***

***#print "Waiting for response.."***

**data = self.cli\_obj.sock.recv(1024)**

***#print "creating Response object"***

**res\_obj = Response(data)**

**if(res\_obj.resParams['status'] == 'SUCCESS'):**

**print "Customer account updated successfully."**

**self.view\_customer\_profile()**

**else:**

**print "Update failure: Customer account could not be updated at this time"**

**def create\_customer\_login\_record(self):**

***#print "Function:create\_customer\_login\_record() ==>"***

**user\_name = raw\_input("User Name: ")**

**password = getpass.getpass("Password: ")**

**customer\_id = randint(1000000000, 2147483648)**

***#print "Generated Customer id = " + str(customer\_id)***

**client\_type = 'Customer'**

**req\_obj = Request()**

**req\_obj.reqType = 'INSERT'**

**req\_obj.reqParams['client\_type'] = self.cli\_type**

**req\_obj.reqParams['subreq\_type'] = 'INSERT\_LOGIN\_RECORD'**

**req\_obj.reqParams['user\_name'] = user\_name**

**req\_obj.reqParams['password'] = password**

**req\_obj.reqParams['customer\_id'] = customer\_id**

**req\_obj.reqParams['record\_client\_type'] = client\_type**

**err = self.cli\_obj.sock.sendall(req\_obj.toString())**

***#if err != None:***

***# print" Send ERROR"***

***#print "Waiting for response.."***

**data = self.cli\_obj.sock.recv(1024)**

***#print "creating Response object"***

**res\_obj = Response(data)**

**if(res\_obj.resParams['status'] == 'SUCCESS'):**

**print "New Customer login Created successfully"**

**self.create\_customer\_account\_record(customer\_id)**

**else:**

**print "Create Failed - Error: " + res\_obj.resParams['error']**

**def create\_customer\_account\_record(self, customer\_id):**

***#print"Function:create\_customer\_account\_record() ==>"***

**chk\_acct\_num = randint(100000000, 2147483648)**

**sav\_acct\_num = randint(100000000, 2147483648)**

**req\_obj = Request()**

**req\_obj.reqType = 'INSERT'**

**req\_obj.reqParams['client\_type'] = self.cli\_type**

**req\_obj.reqParams['subreq\_type'] = 'INSERT\_ACCT\_RECORD'**

**req\_obj.reqParams['customer\_chk\_acct'] = chk\_acct\_num**

**req\_obj.reqParams['customer\_sav\_acct'] = sav\_acct\_num**

**req\_obj.reqParams['customer\_id'] = customer\_id**

**req\_obj.reqParams['customer\_chk\_bal'] = 0**

**req\_obj.reqParams['customer\_sav\_bal'] = 0**

**err = self.cli\_obj.sock.sendall(req\_obj.toString())**

***#if err != None:***

***# print" Send ERROR"***

***#print "Waiting for response.."***

**data = self.cli\_obj.sock.recv(1024)**

***#print "creating Response object"***

**res\_obj = Response(data)**

**if(res\_obj.resParams['status'] == 'SUCCESS'):**

**print "Customer Account record created successfully"**

**self.create\_customer\_profile\_record(customer\_id)**

**else:**

**print"Create Failed - Error: "+res\_obj.resParams['error']**

***# Delete Corresponding entry in the Customer login here***

**def create\_customer\_profile\_record(self, customer\_id):**

***#print"Function:create\_customer\_profile\_record() ==>"***

**while True:**

**first\_name = raw\_input("First Name: ")**

**last\_name = raw\_input("Last\_name: ")**

**dob = raw\_input("Date Of Birth: ")**

**email = raw\_input("Email: ")**

**phone\_num = raw\_input("Phone number: ")**

**apt\_num = raw\_input("Apt number: ")**

**street\_name = raw\_input("Street num: ")**

**city = raw\_input("City: ")**

**state = raw\_input("State: ")**

**country = raw\_input("Country: ")**

**zipcode = raw\_input("Zipcode: ")**

**gender = raw\_input("Gender: ")**

**if first\_name != '' and last\_name != '' and dob != '' \**

**and email != '' and phone\_num != '' and apt\_num != '' \**

**and street\_name != '' and city != '' and state != '' \**

**and country != '' and zipcode != '' and gender != '':**

**break**

**else: print "Some fields are empty. Enter all the details:"**

**req\_obj = Request()**

**req\_obj.reqType = 'INSERT'**

**req\_obj.reqParams['client\_type'] = self.cli\_type**

**req\_obj.reqParams['subreq\_type'] = 'INSERT\_PROFILE\_RECORD'**

**req\_obj.reqParams['customer\_id'] = customer\_id**

**req\_obj.reqParams['first\_name'] = first\_name**

**req\_obj.reqParams['last\_name'] = last\_name**

**req\_obj.reqParams['DOB'] = dob**

**req\_obj.reqParams['email'] = email**

**req\_obj.reqParams['phone'] = phone\_num**

**req\_obj.reqParams['apt\_num'] = apt\_num**

**req\_obj.reqParams['street\_name'] = street\_name**

**req\_obj.reqParams['city'] = city**

**req\_obj.reqParams['state'] = state**

**req\_obj.reqParams['country'] = country**

**req\_obj.reqParams['zipcode'] = zipcode**

**req\_obj.reqParams['gender'] = gender**

**err = self.cli\_obj.sock.sendall(req\_obj.toString())**

***#if err != None:***

***# print" Send ERROR"***

***#print "Waiting for response.."***

**data = self.cli\_obj.sock.recv(1024)**

***#print "creating Response object"***

**res\_obj = Response(data)**

**if(res\_obj.resParams['status'] == 'SUCCESS'):**

**print "Customer Profile Created sucessfully"**

**else:**

**print "Create Failed - Error: "+ res\_obj+resParams['error']**

***# Send Delete request to delete the corresponding entries***

***# in customer\_login and ACCOUNT\_TABLE table***

**def delete\_customer\_account(self):**

***#print"Function:delete\_customer\_account() ==>"***

**customer\_id = raw\_input("Enter customer id: ")**

**req\_obj = Request()**

**req\_obj.reqType = 'DELETE'**

**req\_obj.reqParams['client\_type'] = self.cli\_type**

**req\_obj.reqParams['subreq\_type'] = 'DELETE\_LOGIN\_RECORD'**

**req\_obj.reqParams['customer\_id'] = customer\_id**

**err = self.cli\_obj.sock.sendall(req\_obj.toString())**

***#if err != None:***

***# print" Send ERROR"***

***#print "Waiting for response.."***

**data = self.cli\_obj.sock.recv(1024)**

***#print "creating Response object"***

**res\_obj = Response(data)**

**if(res\_obj.resParams['status'] == 'SUCCESS'):**

**print "Customer login Deleted"**

**req\_obj.reqType = 'DELETE'**

**req\_obj.reqParams['client\_type'] = self.cli\_type**

**req\_obj.reqParams['subreq\_type'] = 'DELETE\_ACCT\_RECORD'**

**req\_obj.reqParams['customer\_id'] = customer\_id**

**err = self.cli\_obj.sock.sendall(req\_obj.toString())**

***#if err != None:***

***# print" Send ERROR"***

***#print "Waiting for response.."***

**data = self.cli\_obj.sock.recv(1024)**

***#print "creating Response object"***

**res\_obj = Response(data)**

**if(res\_obj.resParams['status'] == 'SUCCESS'):**

**print "Customer Account Deleted"**

**req\_obj.reqType = 'DELETE'**

**req\_obj.reqParams['client\_type'] = self.cli\_type**

**req\_obj.reqParams['subreq\_type'] = 'DELETE\_PROFILE\_RECORD'**

**req\_obj.reqParams['customer\_id'] = customer\_id**

**err = self.cli\_obj.sock.sendall(req\_obj.toString())**

***#if err != None:***

***# print" Send ERROR"***

***#print "Waiting for response.."***

**data = self.cli\_obj.sock.recv(1024)**

***#print "creating Response object"***

**res\_obj = Response(data)**

**if(res\_obj.resParams['status'] == 'SUCCESS'):**

**print "Customer Profile Deleted"**

**else:**

**print "Create Failed - Error: "+ res\_obj+resParams['error']**

***# Send Delete request to delete the corresponding entries***

***# in customer\_login and ACCOUNT\_TABLE table***

**else:**

**print"Create Failed - Error: "+res\_obj.resParams['error']**

***# Delete Corresponding entry in the Customer login here***

**else:**

**print "Create Failed - Error: " + res\_obj.resParams['error']**

***######################### ADMIN CLASS #########################***

**class Admin(object):**

**def \_\_init\_\_(self, cli\_obj, cli\_id):**

**self.cli\_obj = cli\_obj**

**self.cli\_type = 'Admin'**

**self.cli\_id = cli\_id**

**self.temp\_teller\_id = ''**

**self.temp\_teller\_name = ''**

**def display\_admin\_options(self):**

**while True:**

**print "1. View Tellers"**

**print "2. View Teller information"**

**print "3. View Transactions"**

**print "4. Update Teller information"**

**print "5. Create Teller record"**

**print "6. Delete Teller record"**

**print "7. Logout"**

**choice = raw\_input("your choice ->")**

**if(choice == '1'):**

**self.view\_all\_tellers()**

**elif(choice == '2'):**

**self.view\_teller\_info()**

**elif(choice == '3'):**

**self.view\_transactions()**

**elif(choice == '4'):**

**self.update\_teller\_info()**

**elif(choice == '5'):**

**self.create\_teller\_login\_record()**

**elif(choice == '6'):**

**self.delete\_teller\_record()**

**elif(choice == '7'):**

***# Logout***

**return LOGOUT**

**else:**

***# Update Customer profile***

**print "Invalid choice"**

**def view\_all\_tellers(self):**

***#print "\nFunction: view\_all\_tellers() ==>"***

***# create request object***

**req\_obj = Request()**

**req\_obj.reqType = 'GET'**

**req\_obj.reqParams['client\_type'] = self.cli\_type**

**req\_obj.reqParams['subreq\_type'] = 'ALL\_TELLER\_ID'**

***#print"sending GET Acct request.."***

**err = self.cli\_obj.sock.sendall(req\_obj.toString())**

***#if err != None:***

***# print" Send ERROR"***

**data = self.cli\_obj.sock.recv(1024)**

**res\_obj = Response(data)**

**num\_of\_tellers = 0**

**if(res\_obj.resParams['status'] == 'SUCCESS'):**

***#print tellers info***

**num\_of\_tellers = int(res\_obj.resParams['total\_teller\_num'])**

**if num\_of\_tellers == 0:**

**print"TELLER RECORDS NOT FOUND"**

**else:**

**print"----------------------------------"**

**print "%-17s %-17s" % ('TELLER NAME', 'TELLER ID')**

**print"----------------------------------"**

**for x in range(0, num\_of\_tellers):**

**teler = 'teller'+str(x+1)**

**teller\_name, teller\_id = res\_obj.resParams[teler].split("/")**

***#print "'{:>20}'.format(teller\_name)" +"'{:>20}'.format(teller\_id)"***

**print"%-17s %-17s" % (teller\_name, teller\_id)**

**print"----------------------------------"**

**else:**

**print"Server Error. Unable to get Tellers information"**

**def view\_teller\_info(self):**

***#print"\nFunction:view\_teller\_info() ==>"***

**self.view\_all\_tellers()**

**teller\_id = raw\_input("Teller ID ->")**

***# create request object***

**req\_obj = Request()**

**req\_obj.reqType = 'GET'**

**req\_obj.reqParams['client\_type'] = self.cli\_type**

**req\_obj.reqParams['subreq\_type'] = 'TELLER\_PROFILE'**

**req\_obj.reqParams['teller\_id'] = teller\_id**

***#print"sending GET Acct request.."***

**err = self.cli\_obj.sock.sendall(req\_obj.toString())**

***#if err != None:***

***# print" Send ERROR"***

**data = self.cli\_obj.sock.recv(1024)**

**res\_obj = Response(data)**

**if(res\_obj.resParams['status'] == 'SUCCESS'):**

**self.temp\_teller\_id = res\_obj.resParams['teller\_id']**

**print"\n---------------------------------------------"**

**print"Teller ID : "+res\_obj.resParams['teller\_id']**

**print"First Name : "+res\_obj.resParams['first\_name']**

**print"Last Name : "+res\_obj.resParams['last\_name']**

**print"Date of Birth: "+res\_obj.resParams['DOB']**

**print"Email Address: "+res\_obj.resParams['email']**

**print"Phone Number : "+res\_obj.resParams['phone']**

**print"Address : "+res\_obj.resParams['address']**

**print"---------------------------------------------\n"**

**else:**

**print"==> RECORD NOT FOUND\n"**

**def view\_transactions(self):**

***#print"Function: view\_transactions() ==> "***

**while True:**

**month = raw\_input("Enter Month -> ")**

**try:**

**month\_int = int(month)**

**except:**

**print "Enter a valid numeric month value between 1-12"**

**continue**

**if((int(month)) >= 1 and (int(month)) <= 12):**

**break**

**else:**

**print"Enter a valid month of the year(1 >= range <= 12) "**

**continue**

***# create request object***

**req\_obj = Request()**

**req\_obj.reqType = 'GET'**

**req\_obj.reqParams['client\_type'] = self.cli\_type**

**req\_obj.reqParams['subreq\_type'] = 'MONTH\_TRANSACTIONS'**

**req\_obj.reqParams['month'] = month**

***#print"sending GET Acct request.."***

**err = self.cli\_obj.sock.sendall(req\_obj.toString())**

***#if err != None:***

***# print" Send ERROR"***

**data = self.cli\_obj.sock.recv(4096)**

**res\_obj = Response(data)**

**if(res\_obj.resParams['status'] == 'SUCCESS'):**

**if(res\_obj.resParams['trns\_list'] == ''):**

**print"==> NO TRANSACTIONS FOUND FOR GIVEN MONTH\n"**

**else:**

**self.print\_transactions(res\_obj.resParams['trns\_list'])**

**else:**

**print"Server error. Failed to get transactions at this time"**

**def print\_transactions(self, trns\_str):**

***#print "\nFunction: print\_transactions() ==> "***

**trns\_list = trns\_str.split()**

***#print trns\_list***

***#print"\n"***

**print"%-3s %-14s %-12s %-10s %-8s %-18s %-12s %-13s %-15s" % ('NUM','CUSTOMER ID', \**

**'DATE','TIME','ACCOUNT','TRANSACTION TYPE','AMOUNT','FROM ACCOUNT', 'TO\_ACCOUNT')**

**print"---------------------------------------------------------------------------------------------------------------"**

**for i in range(0,len(trns\_list),8):**

**for j in range(0,8):**

***#print trns\_list[i+j]***

**key, t\_id = trns\_list[i+0].split('-')**

**key, t\_date = trns\_list[i+1].split('-')**

**key, t\_time = trns\_list[i+2].split('-')**

**key, t\_acct = trns\_list[i+3].split('-')**

**key, t\_trtype = trns\_list[i+4].split('-')**

**key, t\_amt = trns\_list[i+5].split('-')**

**key, t\_facct = trns\_list[i+6].split('-')**

**if t\_facct is '':**

**t\_facct = ' '**

**key, t\_tacct = trns\_list[i+7].split('-')**

**if t\_facct is '':**

**t\_facct = ' '**

**print "%-3s %-14s %-12s %-10s %-11s %-16s %-12s %-13s %-15s" % (str(i/8+1),t\_id,t\_date, \**

**str(t\_time),str(t\_acct),str(t\_trtype),str(t\_amt),str(t\_facct),str(t\_tacct))**

***#print "\n"***

**print"---------------------------------------------------------------------------------------------------------------"**

**def update\_teller\_info(self):**

***#print"\nFunction: update\_teller\_info() ==> "***

**self.view\_all\_tellers()**

**first\_name = last\_name = dob = email\_id = phone\_num = address = ''**

**state = apt\_num = street\_name = city = country = zipcode = gender = ''**

**teller\_id = raw\_input("Teller\_ID ->")**

**print"Press Y if you wish to change ==>"**

**choice = raw\_input("First name? ")**

**if choice == 'Y'or choice =='y':**

**first\_name = raw\_input("Enter New First name: ")**

**choice = raw\_input("Last name? ")**

**if choice == 'Y' or choice == 'y':**

**last\_name = raw\_input("Enter New Last name: ")**

**choice = raw\_input("Date of Birth? ")**

**if choice == 'Y' or choice == 'y':**

**dob = raw\_input("Enter New DOB: ")**

**choice = raw\_input("Email ID? ")**

**if choice == 'Y' or choice == 'y':**

**email\_id = raw\_input("Enter New Email ID: ")**

**choice = raw\_input("Phone number? ")**

**if choice == 'Y' or choice == 'y':**

**phone\_num = raw\_input("Enter New Phone number: ")**

**choice = raw\_input("APT number? ")**

**if choice == 'Y' or choice == 'y':**

**apt\_num = raw\_input("Enter New APT number: ")**

**choice = raw\_input("Street Name? ")**

**if choice == 'Y' or choice == 'y':**

**street\_name = raw\_input("Enter New Street name: ")**

**choice = raw\_input("City? ")**

**if choice == 'Y' or choice == 'y':**

**city = raw\_input("Enter New City: ")**

**choice = raw\_input("State? ")**

**if choice == 'Y' or choice == 'y':**

**state = raw\_input("Enter New State: ")**

**choice = raw\_input("Country? ")**

**if choice == 'Y' or choice == 'y':**

**country = raw\_input("Enter New Country: ")**

**choice = raw\_input("Zipcode? ")**

**if choice == 'Y' or choice == 'y':**

**zipcode = raw\_input("Enter New Zipcode: ")**

**choice = raw\_input("Gender? ")**

**if choice == 'Y' or choice == 'y':**

**gender = raw\_input("Enter New Gender: ")**

**if (first\_name == '' and last\_name == '' and dob == '' \**

**and email\_id == '' and phone\_num == '' and address == '' \**

**and apt\_num == '' and street\_name == '' and city == '' \**

**and state == '' and country == '' and zipcode == '' and gender == ''):**

**print"\n-----------------------"**

**print" => No changes opted."**

**print"-------------------------\n"**

**return**

***# create request object***

**req\_obj = Request()**

**req\_obj.reqType = 'SET'**

**req\_obj.reqParams['client\_type'] = self.cli\_type**

**req\_obj.reqParams['teller\_id'] = teller\_id**

**req\_obj.reqParams['subreq\_type'] = 'UPDATE\_TELLER\_PROFILE'**

**if first\_name != '':**

**req\_obj.reqParams['first\_name']=first\_name**

**if last\_name != '':**

**req\_obj.reqParams['last\_name']=last\_name**

**if dob != '':**

**req\_obj.reqParams['DOB']=dob**

**if email\_id != '':**

**req\_obj.reqParams['email']=email\_id**

**if phone\_num != '':**

**req\_obj.reqParams['phone']=phone\_num**

**if apt\_num != '':**

**req\_obj.reqParams['apt\_num']=apt\_num**

**if street\_name != '':**

**req\_obj.reqParams['street\_name']=street\_name**

**if city != '':**

**req\_obj.reqParams['city'] = city**

**if state != '':**

**req\_obj.reqParams['state'] = state**

**if country != '':**

**req\_obj.reqParams['country'] = country**

**if zipcode != '':**

**req\_obj.reqParams['zipcode'] = zipcode**

**if gender != '':**

**req\_obj.reqParams['gender'] = gender**

***#print"sending UPDATE profile request.."***

***#print"==> "+req\_obj.toString()***

**err = self.cli\_obj.sock.sendall(req\_obj.toString())**

***#if err != None:***

***# print" Send ERROR"***

***#print "Waiting for response.."***

**data = self.cli\_obj.sock.recv(1024)**

***#print "creating Response object"***

**res\_obj = Response(data)**

**if(res\_obj.resParams['status'] == 'SUCCESS'):**

**print "Teller account updated successfully."**

**else:**

**print "Update failure: Teller account could not be updated at this time"**

**def create\_teller\_login\_record(self):**

***#print "\nFunction:create\_teller\_login\_record() ==>"***

**user\_name = raw\_input("Teller Name: ")**

**password = getpass.getpass("Password: ")**

**customer\_id = randint(1000, 9999)**

***#customer\_id = randint(1000000000, 2147483648)***

***#print "Generated Customer id = " + str(customer\_id)***

**client\_type = 'Teller'**

**req\_obj = Request()**

**req\_obj.reqType = 'INSERT'**

**req\_obj.reqParams['client\_type'] = self.cli\_type**

**req\_obj.reqParams['subreq\_type'] = 'INSERT\_LOGIN\_RECORD'**

**req\_obj.reqParams['user\_name'] = user\_name**

**req\_obj.reqParams['password'] = password**

**req\_obj.reqParams['customer\_id'] = customer\_id**

**req\_obj.reqParams['record\_client\_type'] = client\_type**

**err = self.cli\_obj.sock.sendall(req\_obj.toString())**

***#if err != None:***

***# print" Send ERROR"***

***#print "Waiting for response.."***

**data = self.cli\_obj.sock.recv(1024)**

***#print "creating Response object"***

**res\_obj = Response(data)**

**if(res\_obj.resParams['status'] == 'SUCCESS'):**

**print "New Teller login Created successfully"**

**self.create\_teller\_info\_record(customer\_id)**

**else:**

**print "Create Failed - Error: " + res\_obj.resParams['error']**

**def create\_teller\_info\_record(self, teller\_id):**

***#print"Function:create\_teller\_info\_record() ==>"***

**while True:**

**first\_name = raw\_input("First Name: ")**

**last\_name = raw\_input("Last\_name: ")**

**dob = raw\_input("Date Of Birth: ")**

**email = raw\_input("Email: ")**

**phone\_num = raw\_input("Phone number: ")**

**apt\_num = raw\_input("Apt number: ")**

**street\_name = raw\_input("Street num: ")**

**city = raw\_input("City: ")**

**state = raw\_input("State: ")**

**country = raw\_input("Country: ")**

**zipcode = raw\_input("Zipcode: ")**

**gender = raw\_input("Gender: ")**

**if first\_name != '' and last\_name != '' and dob != '' \**

**and email != '' and phone\_num != '' and apt\_num != '' \**

**and street\_name != '' and city != '' and state != '' \**

**and country != '' and zipcode != '' and gender != '':**

**break**

**else: print "Some fields are empty. Enter all the details:"**

**req\_obj = Request()**

**req\_obj.reqType = 'INSERT'**

**req\_obj.reqParams['client\_type'] = self.cli\_type**

**req\_obj.reqParams['subreq\_type'] = 'INSERT\_PROFILE\_RECORD'**

**req\_obj.reqParams['teller\_id'] = teller\_id**

**req\_obj.reqParams['first\_name'] = first\_name**

**req\_obj.reqParams['last\_name'] = last\_name**

**req\_obj.reqParams['DOB'] = dob**

**req\_obj.reqParams['email'] = email**

**req\_obj.reqParams['phone'] = phone\_num**

**req\_obj.reqParams['apt\_num'] = apt\_num**

**req\_obj.reqParams['street\_name'] = street\_name**

**req\_obj.reqParams['city'] = city**

**req\_obj.reqParams['state'] = state**

**req\_obj.reqParams['country'] = country**

**req\_obj.reqParams['zipcode'] = zipcode**

**req\_obj.reqParams['gender'] = gender**

**err = self.cli\_obj.sock.sendall(req\_obj.toString())**

***#if err != None:***

***# print" Send ERROR"***

***#print "Waiting for response.."***

**data = self.cli\_obj.sock.recv(1024)**

***#print "creating Response object"***

**res\_obj = Response(data)**

**if(res\_obj.resParams['status'] == 'SUCCESS'):**

**print "Teller Profile Created sucessfully"**

**else:**

**print "Create Failed - Error: "+ res\_obj+resParams['error']**

***# Send Delete request to delete the corresponding entries***

***# in customer\_login and ACCOUNT\_TABLE table***

**def delete\_teller\_record(self):**

***#print"\nFunction: delete\_teller\_record() ==>"***

**self.view\_all\_tellers()**

**teller\_id = raw\_input("Enter Teller id: ")**

**req\_obj = Request()**

**req\_obj.reqType = 'DELETE'**

**req\_obj.reqParams['client\_type'] = self.cli\_type**

**req\_obj.reqParams['subreq\_type'] = 'DELETE\_LOGIN\_RECORD'**

**req\_obj.reqParams['customer\_id'] = teller\_id**

**err = self.cli\_obj.sock.sendall(req\_obj.toString())**

**if err != None:**

**print" Send ERROR"**

***#print "Waiting for response.."***

**data = self.cli\_obj.sock.recv(1024)**

***#print "creating Response object"***

**res\_obj = Response(data)**

**if(res\_obj.resParams['status'] == 'SUCCESS'):**

**print "Teller Login record Deleted successfully"**

**self.delete\_teller\_info\_record(teller\_id)**

**def delete\_teller\_info\_record(self, teller\_id):**

**print"\nFunction:delete\_teller\_info\_record() ==>"**

**req\_obj = Request()**

**req\_obj.reqType = 'DELETE'**

**req\_obj.reqParams['client\_type'] = self.cli\_type**

**req\_obj.reqParams['subreq\_type'] = 'DELETE\_PROFILE\_RECORD'**

**req\_obj.reqParams['teller\_id'] = teller\_id**

**err = self.cli\_obj.sock.sendall(req\_obj.toString())**

***#if err != None:***

***# print" Send ERROR"***

***#print "Waiting for response.."***

**data = self.cli\_obj.sock.recv(1024)**

***#print "creating Response object"***

**res\_obj = Response(data)**

**if(res\_obj.resParams['status'] == 'SUCCESS'):**

**print "Teller Profile Deleted"**

**else:**

**print "Delete Failed - Error: "+ res\_obj+resParams['error']**

***# Send Delete request to delete the corresponding entries***

***# in customer\_login and ACCOUNT\_TABLE table***

**def create\_admin\_record(self):**

**pass**

**def delete\_admin\_record(self):**

**pass**

***#################### SOCKET CONNECTION CLASS ####################***

**class ClientConnection(object):**

**def \_\_init\_\_(self):**

**self.host = '127.0.0.1'**

**self.port = 5500**

**self.sock = 0**

**self.usr = ''**

**self.pwd = ''**

***#start a TCP connection***

**err = self.connect\_server()**

**if err != None:**

**raise Exception("Failed to connect to server")**

***# start a secure TCP connection with the server***

**def connect\_server(self):**

***#--------------------------------------------------------***

***# Below two lines enables normal connection without ssl***

***#self.sock = socket.socket()***

***#err = self.sock.connect((self.host,self.port))***

***#--------------------------------------------------------***

***#----------------------------------------------------------------------------------***

***# Below two lines enables ssl***

**self.orig\_sock = socket.socket()**

***# use SSL wrapped socket to connect to the server***

**self.sock = ssl.wrap\_socket(self.orig\_sock, ssl\_version=ssl.PROTOCOL\_TLSv1)**

**err = self.sock.connect((self.host, self.port))**

***#----------------------------------------------------------------------------------***

**return err**

***# login into the system***

**def login(self, usr, pwd):**

***#print"\nFunction login() ==>"***

**self.usr = usr**

**self.pwd = pwd**

***# create request object***

**req\_obj = Request()**

**req\_obj.reqType = 'LOGIN'**

**req\_obj.reqParams['username'] = self.usr**

**req\_obj.reqParams['password'] = self.pwd**

***#print"sending Login request.."***

**err = self.sock.sendall(req\_obj.toString())**

***#print "err = " + str(err)***

***#if err != None:***

***# print"login(): returning error here"***

***# return [ERROR,'','']***

***#print "login(): Waiting for response.."***

**data = self.sock.recv(1024)**

***#print "creating Response object"***

**res\_obj = Response(data)**

**if(res\_obj.resParams['status'] == 'SUCCESS'):**

***#print "login():client type : "+ res\_obj.resParams['client\_type']***

**res\_list = [res\_obj.resParams['status'],res\_obj.resParams['client\_type'],res\_obj.resParams['client\_id']]**

**return res\_list**

**else:**

***#print "login(): returning ERROR"***

**return [ERROR,'','']**

**def closeconnection(self):**

**self.sock.close()**

***######################### MAIN #########################***

**def main():**

**try:**

***# create TCP connection with the server***

**cc1 = ClientConnection()**

**except Exception, e:**

**print e**

**return**

**print "Client connection successful"**

**while True:**

**usr = raw\_input("Login Name: -> ")**

**pwd = getpass.getpass("Password: -> ")**

**if usr != '' and pwd != '':**

**break**

**else:**

**print"Error: Login / password empty"**

***# send login credentials***

**res\_list = cc1.login(usr, pwd)**

***#print"res\_list ="***

***#print res\_list***

**if (res\_list[0] == ERROR):**

**print "Authentication Failure"**

**else:**

**if (res\_list[1] == 'Customer'):**

***#create customer object***

**cst\_user = Customer(cc1,usr,pwd, res\_list[2])**

**cst\_user.display\_customer\_options()**

**elif (res\_list[1] == 'Teller'):**

***#create teller object***

**tel\_user = Teller(cc1, res\_list[2])**

**while True:**

**choice = tel\_user.display\_teller\_options()**

**if(choice == LOGOUT):**

**break**

**elif (res\_list[1] == 'Admin'):**

***#create admin user***

**admin\_user = Admin(cc1, res\_list[2])**

**while True:**

**choice = admin\_user.display\_admin\_options()**

**if(choice == LOGOUT):**

**break**

**else:**

***#user authentication failed***

**print "Unknown Client\_type.Closing the connection"**

**cc1.closeconnection()**

**if \_\_name\_\_ == "\_\_main\_\_":**

**main()**

* 1. **Client - GUI**

**from flask import Flask, render\_template, redirect, url\_for, request**

**import socket**

**import ssl**

**import random**

**ERROR = 'FAILED'**

**SUCCESS = 'SUCCESS'**

**LOGOUT = 1**

**host = '127.0.0.1'**

**port = 5500**

**sock = 0**

**usr = ''**

**pwd = ''**

**response\_dict = {}**

***#customer\_ID = 0***

***#----------------------------------------------------------------------------------***

***# Below two lines enables ssl***

**orig\_sock = socket.socket()**

***# use SSL wrapped socket to connect to the server***

**sock = ssl.wrap\_socket(orig\_sock, ssl\_version=ssl.PROTOCOL\_TLSv1)**

**sock.connect((host,port))**

**app = Flask(\_\_name\_\_,template\_folder="/Users/venkataponnaluri/Documents/CMPE-207/project\_new/templates/")**

**app.\_static\_folder = '/Users/venkataponnaluri/Documents/CMPE-207/project\_new/static'**

***#app.debug = debug***

**@app.route('/home',methods=['GET','POST'])**

**def home():**

**if request.method == 'POST':**

**return redirect(url\_for('login'))**

**return render\_template("home.html")**

**@app.route('/login',methods=['GET','POST'])**

**def login():**

**error = None**

**if request.method == 'POST':**

***# while True:***

**if request.form['username'] != '' and request.form['password'] != '':**

**usr = request.form['username']**

**pwd = request.form['password']**

**string\_send = "LOGIN::username:"+str(usr)+"::password:"+str(pwd)**

**print (string\_send)**

**sock.send(string\_send)**

**response = sock.recv(4096)**

**print (response)**

**res\_list = response.split("::")**

**for val in res\_list[1:]:**

**val\_list = val.split(':')**

**response\_dict[val\_list[0].rstrip()] = val\_list[1].rstrip()**

**print (response\_dict)**

**customer\_ID = response\_dict['client\_id']**

**if response\_dict['client\_type'] == 'Customer':**

**return redirect(url\_for('customer',customer\_ID=customer\_ID))**

**elif response\_dict['client\_type'] == 'Teller':**

**return redirect(url\_for('teller',customer\_ID=customer\_ID))**

**elif response\_dict['client\_type'] == 'Admin':**

**return redirect(url\_for('admin',customer\_ID=customer\_ID))**

**else:**

**error = "Invalid username or password! Please try again!"**

**return render\_template("login.html",error=error)**

**return render\_template("login.html",error=error)**

**@app.route('/customer',methods=['GET','POST'])**

**def customer():**

**error=None**

**customer\_ID = request.args['customer\_ID']**

**if request.method == 'POST':**

**choice = request.form['choice']**

**print (request.form['choice'])**

**if(request.form['choice'] == 'Logout'):**

**return redirect(url\_for('login'))**

**else:**

**return redirect(url\_for("customeropt",choice=choice,customer\_ID=customer\_ID))**

**return render\_template("customer.html",error=error)**

**@app.route('/teller',methods=['GET','POST'])**

**def teller():**

**error=None**

**customer\_ID = request.args['customer\_ID']**

**if request.method == 'POST':**

**choice = request.form['choice']**

**print (request.form['choice'])**

**if(request.form['choice'] == 'Logout'):**

**return redirect(url\_for('login'))**

**else:**

**return redirect(url\_for("telleropt",choice=choice,customer\_ID=customer\_ID))**

**return render\_template("teller.html",error=error)**

**@app.route('/admin',methods=['GET','POST'])**

**def admin():**

**error=None**

**customer\_ID = request.args['customer\_ID']**

**if request.method == 'POST':**

**choice = request.form['choice']**

**print (request.form['choice'])**

**if(request.form['choice'] == 'Logout'):**

**return redirect(url\_for('login'))**

**else:**

**return redirect(url\_for("adminopt",choice=choice,customer\_ID=customer\_ID))**

**return render\_template("admin.html",error=error)**

**@app.route('/customeropt',methods=['GET','POST'])**

**def customeropt():**

**error=None**

**choice = request.args['choice']**

**customer\_ID = request.args['customer\_ID']**

**if choice == "Checking and Savings account":**

**string\_send = "GET::" +"client\_type:Customer" +"::customer\_id:"+ response\_dict['client\_id'] + "::subreq\_type:CUSTOMER\_ACCT"**

**sock.send(string\_send)**

**response = sock.recv(4096)**

**print (response)**

**res\_list = response.split("::")**

**for val in res\_list[1:]:**

**val\_list = val.split(':')**

**response\_dict[val\_list[0].rstrip()] = val\_list[1]**

**print (response\_dict)**

**render\_template("customeropt.html",choice=choice,response\_dict=response\_dict,error=error)**

**elif choice == "Profile":**

**string\_send = "GET::" +"client\_type:Customer" +"::customer\_id:"+ response\_dict['client\_id'] + "::subreq\_type:CUSTOMER\_PROFILE"**

**sock.send(string\_send)**

**response = sock.recv(4096)**

**print (response)**

**chk\_acct\_num = response\_dict['chk\_acct']**

**res\_list = response.split("::")**

**for val in res\_list[1:]:**

**val\_list = val.split(':')**

**response\_dict[val\_list[0].rstrip()] = val\_list[1]**

**print (response\_dict)**

**render\_template("customeropt.html",choice=choice,response\_dict=response\_dict,error=error)**

**elif choice == "Transfer Funds":**

**if request.method == "POST":**

**if (request.form["checksave"] == "Checking"):**

**acct\_type = "chk\_acct\_num:"**

**checksave = 'checking::'**

**elif (request.form["checksave"] == "Saving"):**

**acct\_type = "sav\_acct\_num:"**

**checksave = 'saving::'**

**string\_send = "SET::to\_bank:" + str(request.form['bankname'])+"::subreq\_type:TRANSFER\_MONEY::"+acct\_type+str(request.form['frombankaccountnumber'])+"::to\_acct:"+str(request.form['tobankaccountnumber'])+"::client\_type:Customer::op\_type:SUBTRACT::acct\_type:"+checksave+"customer\_id:"+str(customer\_ID)+"::amt:"+str(request.form['Amount'])**

**sock.send(string\_send)**

**response = sock.recv(4096)**

**print (response)**

**res\_list = response.split("::")**

**for val in res\_list[1:]:**

**val\_list = val.split(':')**

**response\_dict[val\_list[0].rstrip()] = val\_list[1]**

**print (response\_dict)**

**if response\_dict['status'] == 'SUCCESS':**

**string\_send = "GET::client\_type:Customer::customer\_id:"+ str(customer\_ID) + "::subreq\_type:CUSTOMER\_ACCT"**

**sock.send(string\_send)**

**response = sock.recv(4096)**

**print (response)**

**res\_list = response.split("::")**

**for val in res\_list[1:]:**

**val\_list = val.split(':')**

**response\_dict[val\_list[0].rstrip()] = val\_list[1]**

**print (response\_dict)**

**elif choice == "Monthly Statements":**

**if (request.method == "POST"):**

**customer\_ID = str(request.form['Customer\_ID'])**

**string\_send = "GET::client\_type:Customer::customer\_id:"+ customer\_ID+ "::subreq\_type:MONTHLY\_STATEMENT"**

**sock.send(string\_send)**

**response = sock.recv(4096)**

**print (response)**

**res\_list = response.split("::")**

**for val in res\_list[1:]:**

**val\_list = val.split(':')**

**response\_dict[val\_list[0].rstrip()] = val\_list[1]**

**print (response\_dict)**

**return render\_template("customeropt.html",response\_dict=response\_dict,choice=choice,error=error)**

**elif choice == "Withdraw":**

**error = "==> Please approach the Teller for withdrawal"**

**render\_template("customeropt.html",choice=choice,response\_dict=response\_dict,error=error)**

**elif choice == "Deposit":**

**error = "==> Please approach the Teller for withdrawal"**

**render\_template("customeropt.html",choice=choice,response\_dict=response\_dict,error=error)**

**else:**

**error = "Failing in GET"**

**return render\_template("customeropt.html",choice=choice,response\_dict=response\_dict,error=error)**

**@app.route('/telleropt',methods=['GET','POST'])**

**def telleropt():**

**error=None**

**choice = request.args['choice']**

**teller\_ID = request.args['customer\_ID']**

**transaction\_dict = {}**

**response\_dict = {}**

**if choice == "Access Customer Account":**

**if request.method == "POST":**

**customer\_ID = request.form['Customer\_ID']**

**string\_send = "GET::client\_type:Teller::customer\_id:"+ str(customer\_ID)+"::subreq\_type:CUSTOMER\_ACCT"**

**sock.send(string\_send)**

**response = sock.recv(4096)**

**print (response)**

**res\_list = response.split("::")**

**for val in res\_list[1:]:**

**val\_list = val.split(':')**

**response\_dict[val\_list[0].rstrip()] = val\_list[1]**

**print (response\_dict)**

**string\_send = "GET::client\_type:Teller::customer\_id:"+ str(customer\_ID)+"::subreq\_type:CUSTOMER\_TRANSACTION"**

**sock.send(string\_send)**

**response = sock.recv(4096)**

**print (response)**

**res\_list = response.split("::")**

**for val in res\_list[1:]:**

**val\_list = val.split(':')**

**transaction\_dict[val\_list[0].rstrip()] = val\_list[1]**

**print (transaction\_dict)**

**chk\_acct\_num = response\_dict['chk\_acct']**

**render\_template("telleropt.html",choice=choice,response\_dict=response\_dict,transaction\_dict=transaction\_dict,error=error)**

**elif choice == "Update Customer Account":**

**if (request.method == "POST"):**

**amount = str(request.form['Amount'])**

**acct\_num = request.form['tobankaccountnumber']**

**customer\_ID = str(request.form['Customer\_ID'])**

**print (request.form['withdep'])**

**if request.form['withdep'] == "Withdraw":**

**op\_type = "SUBTRACT"**

**elif request.form['withdep'] == "Deposit":**

**op\_type = "ADD"**

**if (request.form["checksave"] == "Checking"):**

**acct\_type = "chk\_acct\_num:"**

**checksave = 'checking::'**

**elif (request.form["checksave"] == "Saving"):**

**acct\_type = "sav\_acct\_num:"**

**checksave = 'saving::'**

**string\_send = "SET::subreq\_type:UPDATE\_CHK\_ACCT::"+acct\_type+ str(acct\_num)+"::client\_type:Teller::op\_type:"+op\_type+"::customer\_id:"+customer\_ID+"::amt:"+amount**

**sock.send(string\_send)**

**response = sock.recv(4096)**

**print (response)**

**res\_list = response.split("::")**

**for val in res\_list[1:]:**

**val\_list = val.split(':')**

**response\_dict[val\_list[0].rstrip()] = val\_list[1]**

**print (response\_dict)**

**if response\_dict['status'] == "SUCCESS":**

**string\_send = "GET::client\_type:Teller::customer\_id:"+customer\_ID+"::subreq\_type:CUSTOMER\_ACCT"**

**sock.send(string\_send)**

**response = sock.recv(4096)**

**print (response)**

**res\_list = response.split("::")**

**for val in res\_list[1:]:**

**val\_list = val.split(':')**

**response\_dict[val\_list[0].rstrip()] = val\_list[1]**

**print (response\_dict)**

**render\_template("telleropt.html",choice=choice,response\_dict=response\_dict,transaction\_dict=transaction\_dict,error=error)**

**elif choice == "Create Customer Account":**

**if (request.method == "POST"):**

**customer\_ID = str(request.form['Customer\_ID'])**

**string\_send = "INSERT::record\_client\_type:Customer::customer\_id:"+ str(customer\_ID)+"::client\_type:Teller::subreq\_type:INSERT\_LOGIN\_RECORD::password:"+request.form['password']+"::user\_name:"+request.form['username']**

**print (string\_send)**

**sock.send(string\_send)**

**response = sock.recv(4096)**

**print (response)**

**res\_list = response.split("::")**

**for val in res\_list[1:]:**

**val\_list = val.split(':')**

**response\_dict[val\_list[0].rstrip()] = val\_list[1]**

**print (response\_dict)**

**if response\_dict['status'] == "SUCCESS":**

**string\_send = "INSERT::subreq\_type:INSERT\_ACCT\_RECORD::customer\_chk\_bal:0::customer\_sav\_bal:0::customer\_chk\_acct:"+ request.form['checkacct'] +"::client\_type:Teller::customer\_sav\_acct:"+request.form['savacct'] +"::customer\_id:"+str(customer\_ID)**

**sock.send(string\_send)**

**response = sock.recv(4096)**

**print (response)**

**res\_list = response.split("::")**

**for val in res\_list[1:]:**

**val\_list = val.split(':')**

**response\_dict[val\_list[0].rstrip()] = val\_list[1]**

**print (response\_dict)**

**if response\_dict['status'] == "SUCCESS":**

**string\_send = "INSERT::city:"+request.form['City']+"::first\_name:"+request.form['First\_name']+"::last\_name:"+request.form['Last\_name']+"::DOB:"+ request.form['dob']+"::country:"+request.form['Country']+"::street\_name:"+request.form['stnum']+"::zipcode:"+request.form['Zipcode']+"::phone:"+request.form['phnum']+"::state:"+request.form['State']+"::client\_type:Teller::subreq\_type:INSERT\_PROFILE\_RECORD::"+"gender:"+request.form['Gender']+"::customer\_id:"+str(customer\_ID)+"::email:"+request.form['Email']+"::apt\_num:"+request.form['Aptnumber']**

**sock.send(string\_send)**

**response = sock.recv(4096)**

**print (response)**

**res\_list = response.split("::")**

**for val in res\_list[1:]:**

**val\_list = val.split(':')**

**response\_dict[val\_list[0].rstrip()] = val\_list[1]**

**print (response\_dict)**

**return render\_template("telleropt.html",choice=choice,error=error)**

**elif choice == "Delete Customer Account":**

**if (request.method == "POST"):**

**customer\_ID = str(request.form['Customer\_ID'])**

**string\_send = "DELETE::client\_type:Teller::customer\_id:"+ customer\_ID +"::subreq\_type:DELETE\_LOGIN\_RECORD"**

**sock.send(string\_send)**

**response = sock.recv(4096)**

**print (response)**

**res\_list = response.split("::")**

**for val in res\_list[1:]:**

**val\_list = val.split(':')**

**response\_dict[val\_list[0].rstrip()] = val\_list[1]**

**print (response\_dict)**

**return render\_template("telleropt.html",choice=choice,error=error)**

**else :**

**error = "Failing in GET"**

**return render\_template("telleropt.html",choice=choice,response\_dict=response\_dict,transaction\_dict=transaction\_dict,error=error)**

**@app.route('/adminopt',methods=['GET','POST'])**

**def adminopt():**

**error=None**

**choice = request.args['choice']**

**admin\_ID = request.args['customer\_ID']**

**transaction\_dict = {}**

**response\_dict = {}**

**if choice == "View Tellers":**

**string\_send = "GET::client\_type:Admin::subreq\_type:ALL\_TELLER\_ID"**

**sock.send(string\_send)**

**response = sock.recv(4096)**

**print (response)**

**res\_list = response.split("::")**

**for val in res\_list[1:]:**

**val\_list = val.split(':')**

**response\_dict[val\_list[0].rstrip()] = val\_list[1]**

**print (response\_dict)**

**return render\_template("adminopt.html",choice=choice,response\_dict=response\_dict,error=error)**

**else :**

**error = "Failing in GET"**

**return render\_template("adminopt.html",choice=choice,response\_dict=response\_dict,error=error)**

***######################### MAIN #########################***

**def main(debug=False):**

***# send login credentials***

**print"res\_list ="**

**print res\_list**

**sock.close()**

**if \_\_name\_\_ == "\_\_main\_\_":**

**app.run(debug=True)**

***#main(app)***

* 1. **HTML** 
     1. **Home page**

<!DOCTYPE html>

<html>

<title>WELCOME TO KGUS BANK</title>

<body>

<center> <h1> Welcome to KGUS bank </h1> </center>

<center> <img src = "{{ url\_for('static', filename = 'bank.jpg') }}" > </center>

<form method="post">

<center> <input type='Submit' value="Login"><style="background:red", "cursor:pointer"></style></center>

</form>

</body>

</html>

* + 1. **Login page**

<!DOCTYPE html>

<html>

<head>

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.0/jquery.min.js"></script>

<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>

<style>

body {background-color: powderblue;}

h1 {color: black;}

</style>

</head>

<body>

<div class='container'>

<h1>Login Page</h1>

<br>

<form action="" method="post">

<label for="username"> Username </label>

<input type='text' placeholder="Username" name="username" value="{{ request.form.username }}">

<label for="password"> Password </label>

<input type='password' placeholder="Password" name="password" value="{{ request.form.password }}">

<br>

<input type='submit' value="Login" >

</form>

{% if error %}

<p class="error"><strong>Error!</strong> {{ error }}</p>

{% endif %}

</div>

</body>

</html>

* + 1. **Customer page 1**

<!DOCTYPE html>

<html>

<head>

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.0/jquery.min.js"></script>

<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>

<style>

body {

background-color: #dfc12a

}

.boxed{

margin-top:50px;

margin-left:auto;

margin-right:auto;

width:40%;

font-size: 24px;

text-align: center;

background-color: #4CAF50; /\* Green background \*/

border: 5px solid green; /\* Green border \*/

color: green; /\* White text \*/

padding: 0.8% 0.8%; /\* Some padding \*/

}

.Profile{

margin-top:16px;

margin-left:auto;

margin-right:auto;

width:40%;

font-size: 24px;

text-align: center;

background-color: #4CAF50; /\* Green background \*/

border: 5px solid green; /\* Green border \*/

color: green; /\* White text \*/

padding: 0.8% 0.8%; /\* Some padding \*/ }

.Transfer{

margin-top:16px;

margin-left:auto;

margin-right:auto;

width:40%;

font-size: 24px;

text-align: center;

background-color: #4CAF50; /\* Green background \*/

border: 5px solid green; /\* Green border \*/

color: green; /\* White text \*/

padding: 0.8% 0.8%; /\* Some padding \*/ }

.Withdraw{

margin-top:16px;

margin-left:auto;

margin-right:auto;

width:40%;

font-size: 24px;

text-align: center;

background-color: #4CAF50; /\* Green background \*/

border: 5px solid green; /\* Green border \*/

color: green; /\* White text \*/

padding: 0.8% 0.8%; /\* Some padding \*/ }

.Deposit{

margin-top:16px;

margin-left:auto;

margin-right:auto;

width:40%;

font-size: 24px;

text-align: center;

background-color: #4CAF50; /\* Green background \*/

border: 5px solid green; /\* Green border \*/

color: green; /\* White text \*/

padding: 0.8% 0.8%; /\* Some padding \*/ }

.Statement{

margin-top:16px;

margin-left:auto;

margin-right:auto;

width:40%;

font-size: 24px;

text-align: center;

background-color: #4CAF50; /\* Green background \*/

border: 5px solid green; /\* Green border \*/

color: green; /\* White text \*/

padding: 0.8% 0.8%; /\* Some padding \*/ }

.Logout{

margin-top:16px;

margin-left:auto;

margin-right:auto;

width:40%;

font-size: 24px;

text-align: center;

background-color: #4CAF50; /\* Green background \*/

border: 5px solid green; /\* Green border \*/

color: green; /\* White text \*/

padding: 0.8% 0.8%; /\* Some padding \*/ }

</style>

</head>

<body>

<form method="post">

<div class="boxed">

<input type = "Submit" name = "choice" value = "Checking and Savings account">

</div>

<div class="Profile">

<input type = "Submit" name = "choice" value = "Profile">

</div >

<div class="Transfer">

<input type = "Submit" name = "choice" value = "Transfer Funds">

</div >

<div class="Withdraw">

<input type = "Submit" name = "choice" value = "Withdraw">

</div>

<div class="Deposit">

<input type = "Submit" name = "choice" value = "Deposit">

</div>

<div class="Statement">

<input type = "Submit" name = "choice" value = "Monthly Statements">

</div>

<div class="Logout">

<input type = "Submit" name = "choice" value = "Logout">

</div>

</form>

</body>

</html>

* + 1. **Customer page 2**

<!DOCTYPE html>

<html>

<head>

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.0/jquery.min.js"></script>

<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>

<style>

body {background-color: #DB7093;}

h1 {color: black;}

.error{

margin: 10px 52px 10px 0;

margin-top:130px;

text-align: center;

padding-left:500px;

font-size: 22px;

font-weight: bold

}

.check{

padding-left:500px;

text-align: center;

font-size: 22px;

font-weight: bold

}

.save{

padding-left:500px;

text-align: center;

font-size: 22px;

font-weight: bold

}

</style>

</head>

<body>

<!-- <div class='container'> -->

<!-- <h1>Login Page</h1>

<br>

<form action="" method="post">

<label for="username"> Username </label>

<input type='text' placeholder="Username" name="username" value="{{ request.form.username }}">

<label for="password"> Password </label>

<input type='password' placeholder="Password" name="password" value="{{ request.form.password }}">

<br>

<input type='submit' value="Login" >

</form> -->

{% if error %}

<p class="error"><strong>Error!</strong> {{ error }}</p>

{% endif %}

{% if choice == "Checking and Savings account" %}

<p class="check account"><strong>Checking Account</strong>{{ response\_dict['chk\_acct'] }}</p>

<p class="check bal"><strong>Checking Balance</strong> {{ response\_dict['chk\_bal'] }}</p>

<p class="save account"><strong>Saving Account</strong> {{ response\_dict['sav\_acct'] }}</p>

<p class="save bal"><strong>Saving Balance</strong> {{ response\_dict['sav\_bal'] }}</p>

{% endif %}

{% if choice == "Profile" %}

<p class="Print"><strong> Customer ID </strong> {{ response\_dict['customer\_id'] }}</p>

<p class="Print"><strong> First Name </strong>{{ response\_dict['first\_name'] }}</p>

<p class="Print"><strong> Last Name </strong> {{ response\_dict['last\_name'] }} </p>

<p class="Print"><strong> Checking Account </strong>{{ response\_dict['chk\_acct'] }}</p>

<p class="Print"><strong> Checking Account Balance </strong>{{ response\_dict['chk\_bal'] }}</p>

<p class="Print"><strong> Savings Account </strong>{{ response\_dict['sav\_acct'] }}</p>

<p class="Print"><strong> Savings Balance </strong>{{ response\_dict['sav\_bal'] }}</p>

<p class="Print"><strong> Date of Birth </strong>{{ response\_dict['DOB'] }}</p>

<p class="Print"><strong> Phone Number </strong>{{ response\_dict['phone'] }}</p>

<p class="Print"><strong> Address </strong>{{ response\_dict['address'] }}</p>

<p class="Print"><strong> Mail ID </strong>{{ response\_dict['email'] }}</p>

{% endif %}

{% if choice == "Transfer Funds" %}

<form action="" method="post">

<label for="Bank Name"> Bank Name </label>

<input type='text' placeholder="Bank Name" name="bankname" value="{{ request.form.bankname }}">

<label for="Checking/Saving "> Checking/Saving </label>

<input type='text' placeholder="Checking/Saving" name="checksave" value="{{ request.form.checksave }}">

<label for="To Bank Account Number"> To Bank Account Number </label>

<input type='text' placeholder="To Bank Account Number" name="tobankaccountnumber" value="{{ request.form.tobankaccountnumber }}">

<label for="From Bank Account Number"> From Bank Account Number </label>

<input type='text' placeholder="From Bank Account Number" name="frombankaccountnumber" value="{{ request.form.frombankaccountnumber }}">

<br>

<label for="Amount"> Amount </label>

<input type='text' placeholder="Amount" name="Amount" value="{{ request.form.Amount }}">

<br>

<input type='submit' value="Send Money" >

</form>

<p class="Print"><strong> Customer ID </strong> {{ response\_dict['customer\_id'] }}</p>

<p class="Print"><strong> Checking Account </strong>{{ response\_dict['chk\_acct'] }}</p>

<p class="Print"><strong> Checking Account Balance </strong> {{ response\_dict['chk\_bal'] }} </p>

<p class="Print"><strong> Saving Account </strong>{{ response\_dict['sav\_acct'] }}</p>

<p class="Print"><strong> Saving Account Balance </strong>{{ response\_dict['sav\_bal'] }}</p>

{% endif %}

{% if choice == "Withdrawal" %}

<p class="error"><strong>Notice:</strong> {{ error }}</p>

{% endif %}

{% if choice == "Deposit" %}

<p class="error"><strong>Notice:</strong> {{ error }}</p>

{% endif %}

{% if choice == "Monthly Statements" %}

<form action="" method="post">

<label for="Customer\_ID"> Customer\_ID </label>

<input type='text' placeholder="Customer\_ID" name="Customer\_ID" value="{{ request.form.Customer\_ID }}">

<br>

<input type='submit' value="Create" >

</form>

{% endif %}

</body>

<!-- </div> -->

<!-- <head>

<form method="get">

<body>

<script type="text/javascipt">

document.write("<h1>Hello World!</h1><p>Have a nice day!</p>");

var choice = {{ choice }} ;

var response\_dict = {{ response\_dict }};

document.write('Hello World!');

if (choice == "Checking and Savings account")

{

var status = response\_dict['status'];

if status == 'SUCCESS'{

var account\_number = {{ response\_dict['chk\_acct'] }};

var balance = {{ response\_dict['chk\_bal'] }};

var account\_number\_sav = {{ response\_dict['sav\_acct'] }};

var balance\_sav = {{ response\_dict['sav\_bal'] }};

var customer\_id = {{ response\_dict['customer\_id'] }};

document.write("Customer ID " + customer\_id.toString());

document.write("Checking Account ");

document.write("Account " + account\_number.toString());

document.write("Balance " + "$" + balance.toString());

document.write("Savings Account ");

document.write("Account " + account\_number\_sav.toString());

document.write("Balance " + "$" + balance\_sav.toString());}

}

</script>

</body>

</form>

</head>

<body>

</body> -->

</html>

* + 1. **Teller page 1**

<!DOCTYPE html>

<html>

<head>

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.0/jquery.min.js"></script>

<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>

<style>

.Access{

margin-top:100px;

margin-left:auto;

margin-right:auto;

width:400px;

text-align: center;

background-color: #4CAF50; /\* Green background \*/

border: 10px solid green; /\* Green border \*/

color: green; /\* White text \*/

padding: 10px 100px; /\* Some padding \*/

}

.Update{

margin-top:100px;

margin-left:auto;

margin-right:auto;

width:400px;

text-align: center;

background-color: #4CAF50; /\* Green background \*/

border: 10px solid green; /\* Green border \*/

color: green; /\* White text \*/

padding: 10px 100px; /\* Some padding \*/

}

.Create{

margin-top:16px;

margin-left:auto;

margin-right:auto;

width:400px;

text-align: center;

background-color: #4CAF50; /\* Green background \*/

border: 10px solid green; /\* Green border \*/

color: green; /\* White text \*/

padding: 10px 100px; /\* Some padding \*/ }

.Delete{

margin-top:16px;

margin-left:auto;

margin-right:auto;

width:400px;

text-align: center;

background-color: #4CAF50; /\* Green background \*/

border: 10px solid green; /\* Green border \*/

color: green; /\* White text \*/

padding: 10px 100px; /\* Some padding \*/ }

.Logout{

margin-top:16px;

margin-left:auto;

margin-right:auto;

width:400px;

text-align: center;

background-color: #4CAF50; /\* Green background \*/

border: 10px solid green; /\* Green border \*/

color: green; /\* White text \*/

padding: 10px 100px; /\* Some padding \*/ }

</style>

</head>

<body>

<form method="post">

<div class="Access">

<input type = "Submit" name = "choice" value = "Access Customer Account">

</div>

<div class="Update">

<input type = "Submit" name = "choice" value = "Update Customer Account">

</div>

<div class="Create">

<input type = "Submit" name = "choice" value = "Create Customer Account">

</div >

<div class="Delete">

<input type = "Submit" name = "choice" value = "Delete Customer Account">

</div >

<div class="Logout">

<input type = "Submit" name = "choice" value = "Logout">

</div>

</form>

</body>

</html>

* + 1. **Teller page 2**

<!DOCTYPE html>

<html>

<head>

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.0/jquery.min.js"></script>

<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>

<style>

body {background-color: #DB7093;}

h1 {color: black;}

.error{

margin: 10px 52px 10px 0;

margin-top:130px;

text-align: center;

padding-left:500px;

font-size: 22px;

font-weight: bold

}

</style>

</head>

<body>

{% if error %}

<p class="error"><strong>Error!</strong> {{ error }}</p>

{% endif %}

{% if choice == "Access Customer Account" %}

<form action="" method="post">

<label for="customer ID"> Customer ID </label>

<input type='text' placeholder="Customer ID" name="Customer\_ID" value="{{ request.form.Customer\_ID }}">

</form>

<p class="Customer ID"><strong>Customer ID</strong>{{ response\_dict['customer\_id'] }}</p>

<p class="check account"><strong>Checking Account </strong> {{ response\_dict['chk\_acct'] }}</p>

<p class="check bal"><strong>Checking Balance</strong> {{ response\_dict['chk\_bal'] }}</p>

<p class="save account"><strong>Saving Account</strong> {{ response\_dict['sav\_acct'] }}</p>

<p class="save bal"><strong>Saving Balance</strong> {{ response\_dict['sav\_bal'] }}</p>

<p class="Transaction List"><strong>Transactions of Customer</strong> {{ transaction\_dict['trns\_list'] }}</p>

{% endif %}

{% if choice == "Update Customer Account" %}

<form action="" method="post">

<label for="Checking/Saving "> Checking/Saving </label>

<input type='text' placeholder="Checking/Saving" name="checksave" value="{{ request.form.checksave }}">

<label for="To Bank Account Number"> To Bank Account Number </label>

<input type='text' placeholder="To Bank Account Number" name="tobankaccountnumber" value="{{ request.form.tobankaccountnumber }}">

<label for="Withdraw/Deposit"> Withdraw/Deposit </label>

<input type='text' placeholder="Withdraw/Deposit" name="withdep" value="{{ request.form.withdep }}">

<label for="Customer\_ID"> Customer\_ID </label>

<input type='text' placeholder="Customer\_ID" name="Customer\_ID" value="{{ request.form.Customer\_ID }}">

<br>

<label for="Amount"> Amount </label>

<input type='text' placeholder="Amount" name="Amount" value="{{ request.form.Amount }}">

<br>

<input type='submit' value="Send Money" >

</form>

<p class="Print"><strong> Customer ID </strong> {{ response\_dict['customer\_id'] }}</p>

<p class="Print"><strong> Checking Account </strong>{{ response\_dict['chk\_acct'] }}</p>

<p class="Print"><strong> Checking Account Balance </strong> {{ response\_dict['chk\_bal'] }} </p>

<p class="Print"><strong> Saving Account </strong>{{ response\_dict['sav\_acct'] }}</p>

<p class="Print"><strong> Saving Account Balance </strong>{{ response\_dict['sav\_bal'] }}</p>

{% endif %}

{% if choice == "Create Customer Account" %}

<div class='container'>

<h1>Login Page</h1>

<br>

<form action="" method="post">

<label for="customer ID"> Customer ID </label>

<input type='text' placeholder="Customer ID" name="Customer\_ID" value="{{ request.form.Customer\_ID }}">

<label for="username"> Username </label>

<input type='text' placeholder="Username" name="username" value="{{ request.form.username }}">

<label for="password"> Password </label>

<input type='password' placeholder="Password" name="password" value="{{ request.form.password }}">

<br>

</form>

<h1>Account Record</h1>

<br>

<form action="" method="post">

<label for="Checking Balance"> Checking Balance </label>

<input type='text' placeholder="Checking Balance" name="checkbal" value="{{ request.form.checkbal }}">

<label for="Saving Balance"> Saving Balance </label>

<input type='text' placeholder="Saving Balance" name="savbal" value="{{ request.form.savbal }}">

<label for="Checking Account"> Checking Account </label>

<input type='text' placeholder="Checking Account" name="checkacct" value="{{ request.form.checkacct }}">

<label for="Saving Account"> Saving Account </label>

<input type='text' placeholder="Saving Account" name="savcct" value="{{ request.form.savacct }}">

<br>

</form>

<h1>Customer Details</h1>

<form action="" method="post">

<label for="Last Name"> Last Name </label>

<input type='text' placeholder="Last Name" name="Last\_name" value="{{ request.form.Last\_name }}">

<br>

<label for="Last Name"> First Name </label>

<input type='text' placeholder="First Name" name="First\_name" value="{{ request.form.First\_name }}">

<br>

<label for="Date Of Birth"> Date Of Birth </label>

<input type='text' placeholder="Date Of Birth" name="dob" value="{{ request.form.dob }}">

<br>

<label for="Email"> Email </label>

<input type='text' placeholder="Email" name="Email" value="{{ request.form.Email }}">

<br>

<label for="Phone number"> Phone number </label>

<input type='text' placeholder="Phone number" name="phnum" value="{{ request.form.phnum }}">

<br>

<label for="Apt number">Apt number</label>

<input type='text' placeholder="Apt number" name="Aptnumber" value="{{ request.form.Aptnumber}}">

<br>

<label for="Street num"> Street num </label>

<input type='text' placeholder="Street num" name="stnum" value="{{ request.form.stnum }}">

<br>

<label for="City"> City </label>

<input type='text' placeholder="City" name="City" value="{{ request.form.City }}">

<br>

<label for="State"> State </label>

<input type='text' placeholder="State" name="State" value="{{ request.form.State }}">

<br>

<label for="Country"> Country </label>

<input type='text' placeholder="Country" name="Country" value="{{ request.form.Country }}">

<br>

<label for="Zipcode"> Zipcode </label>

<input type='text' placeholder="Zipcode" name="Zipcode" value="{{ request.form.Zipcode }}">

<br>

<label for="Gender"> Gender </label>

<input type='text' placeholder="Gender" name="Gender" value="{{ request.form.Gender }}">

<br>

<input type='submit' value="Create" >

<br>

</form>

</div>

{% endif %}

{% if choice == "Delete Customer Account" %}

<form action="" method="post">

<label for="Customer\_ID"> Customer\_ID </label>

<input type='text' placeholder="Customer\_ID" name="Customer\_ID" value="{{ request.form.Customer\_ID }}">

<br>

<input type='submit' value="Delete" >

</form>

{% endif %}

</body>

</html>

* + 1. **Administrator page 1**

<!DOCTYPE html>

<html>

<head>

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.0/jquery.min.js"></script>

<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>

<style>

.ViewTellers{

margin-top:100px;

margin-left:auto;

margin-right:auto;

width:400px;

text-align: center;

background-color: #4CAF50; /\* Green background \*/

border: 10px solid green; /\* Green border \*/

color: green; /\* White text \*/

padding: 10px 100px; /\* Some padding \*/

}

.ViewTellInfo{

margin-top:16px;

margin-left:auto;

margin-right:auto;

width:400px;

text-align: center;

background-color: #4CAF50; /\* Green background \*/

border: 10px solid green; /\* Green border \*/

color: green; /\* White text \*/

padding: 10px 100px; /\* Some padding \*/

}

.ViewTran{

margin-top:16px;

margin-left:auto;

margin-right:auto;

width:400px;

text-align: center;

background-color: #4CAF50; /\* Green background \*/

border: 10px solid green; /\* Green border \*/

color: green; /\* White text \*/

padding: 10px 100px; /\* Some padding \*/

}

.Create{

margin-top:16px;

margin-left:auto;

margin-right:auto;

width:400px;

text-align: center;

background-color: #4CAF50; /\* Green background \*/

border: 10px solid green; /\* Green border \*/

color: green; /\* White text \*/

padding: 10px 100px; /\* Some padding \*/

}

.Delete{

margin-top:16px;

margin-left:auto;

margin-right:auto;

width:400px;

text-align: center;

background-color: #4CAF50; /\* Green background \*/

border: 10px solid green; /\* Green border \*/

color: green; /\* White text \*/

padding: 10px 100px; /\* Some padding \*/

}

.Logout{

margin-top:16px;

margin-left:auto;

margin-right:auto;

width:400px;

text-align: center;

background-color: #4CAF50; /\* Green background \*/

border: 10px solid green; /\* Green border \*/

color: green; /\* White text \*/

padding: 10px 100px; /\* Some padding \*/

}

</style>

</head>

<body>

<form method="post">

<div class="ViewTellers">

<input type = "Submit" name = "choice" value = "View Tellers">

</div>

<div class="ViewTellInfo">

<input type = "Submit" name = "choice" value = "View Teller information">

</div >

<div class="ViewTran">

<input type = "Submit" name = "choice" value = "View Transactions">

</div >

<div class="Create">

<input type = "Submit" name = "choice" value = "Create Teller record">

</div>

<div class="Delete">

<input type = "Submit" name = "choice" value = "Delete Teller record">

</div>

<div class="Logout">

<input type = "Submit" name = "choice" value = "Logout">

</div>

</form>

</body>

</html>

* + 1. **Administrator page 2**

<!DOCTYPE html>

<html>

<head>

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.0/jquery.min.js"></script>

<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>

<style>

body {background-color: #DB7093;}

h1 {color: black;}

.error{

margin: 10px 52px 10px 0;

margin-top:130px;

text-align: center;

padding-left:500px;

font-size: 22px;

font-weight: bold

}

</style>

</head>

<body>

{% if error %}

<p class="error"><strong>Error!</strong> {{ error }}</p>

{% endif %}

{% if choice == "View Tellers" %}

<p class="Teller Name "><strong><font size='24'>Teller Name Teller ID</font></strong></p>

<br>

<p class="Teller Info"><font size='18'>{{ response\_dict['teller1'] }}</font></p>

{% endif %}

{% if choice == "View Teller information" %}

{% endif %}

{% if choice == "View Transactions" %}

{% endif %}

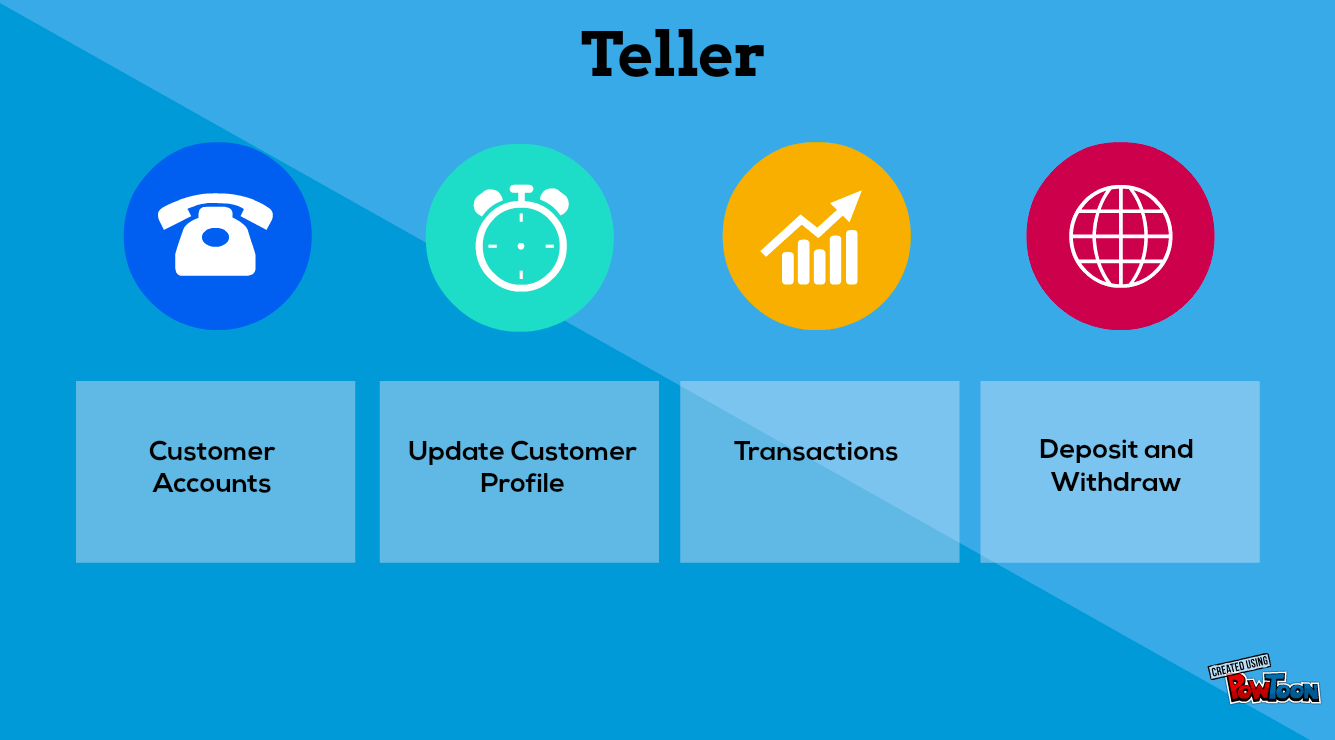
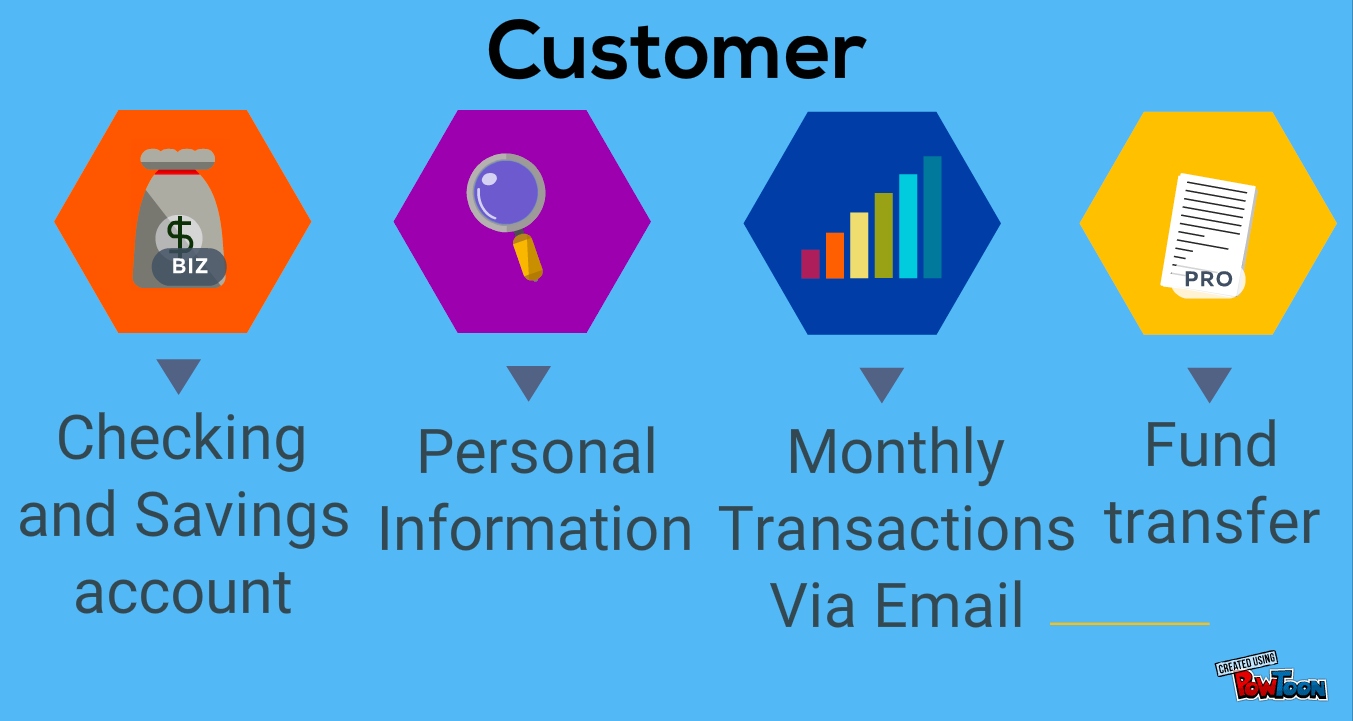
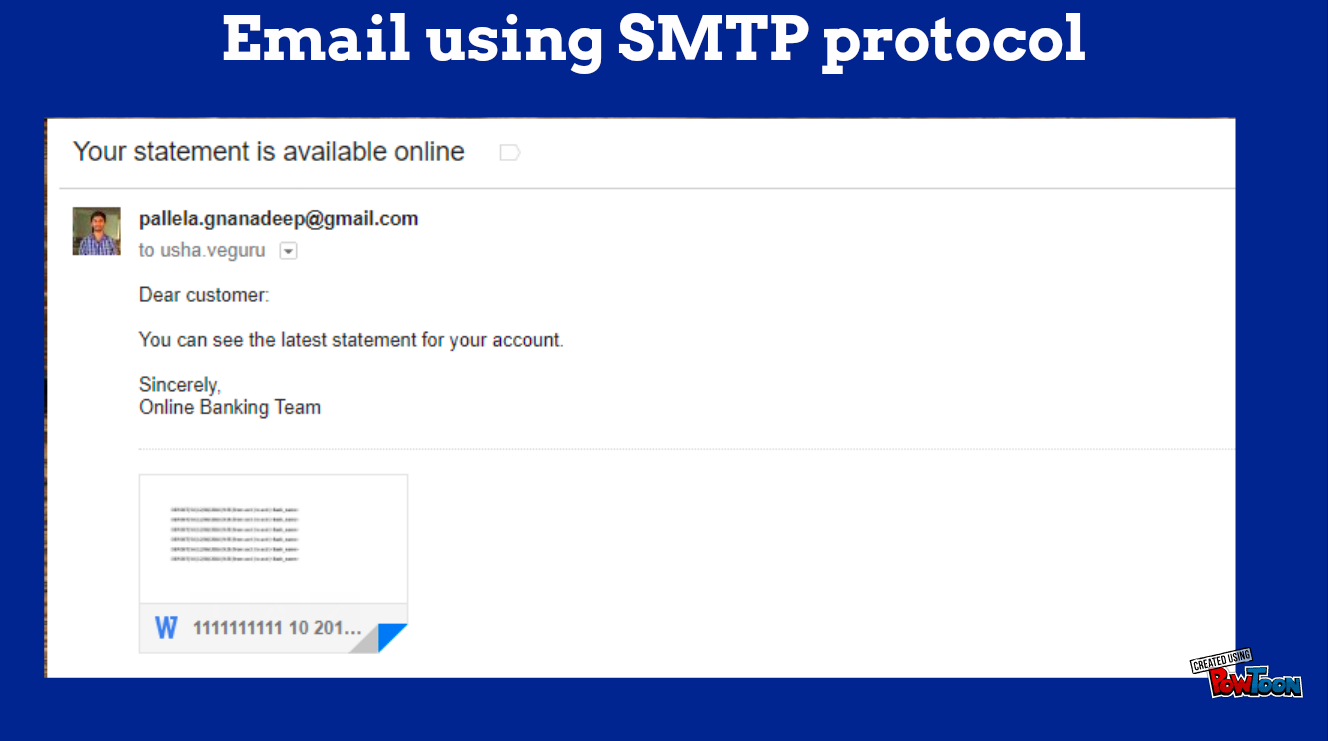
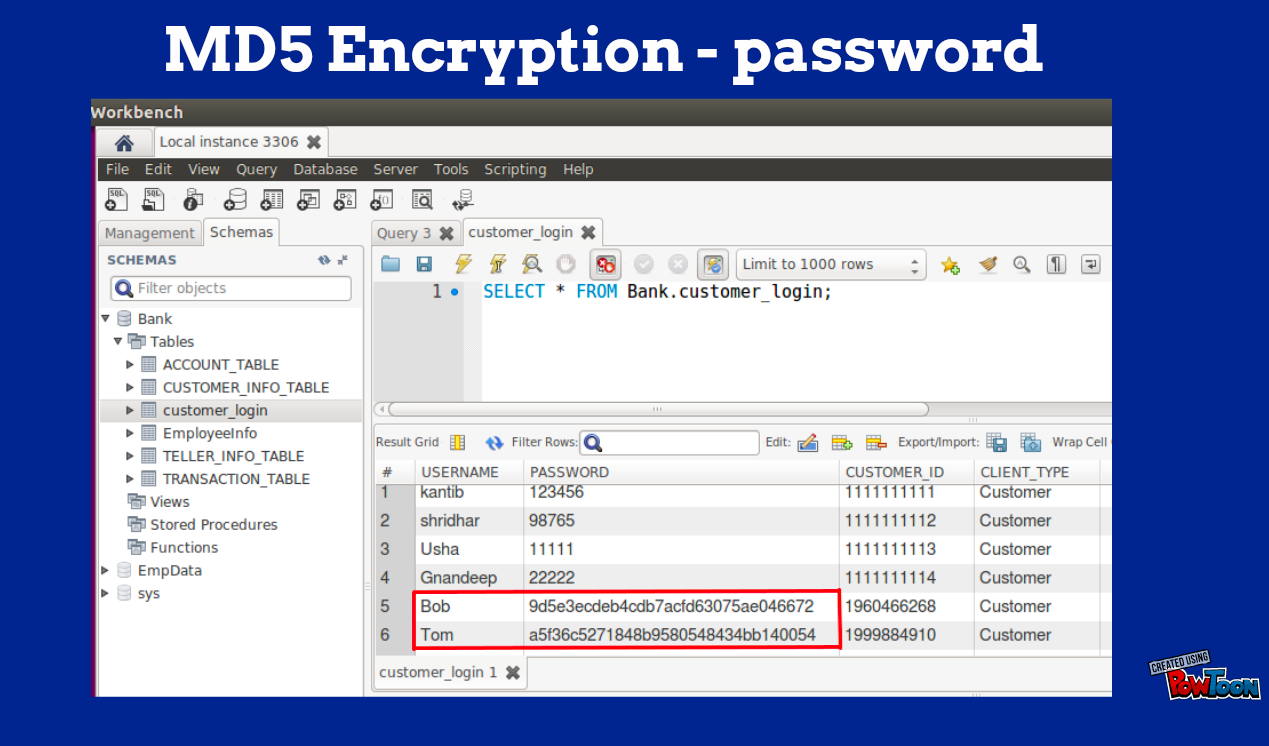
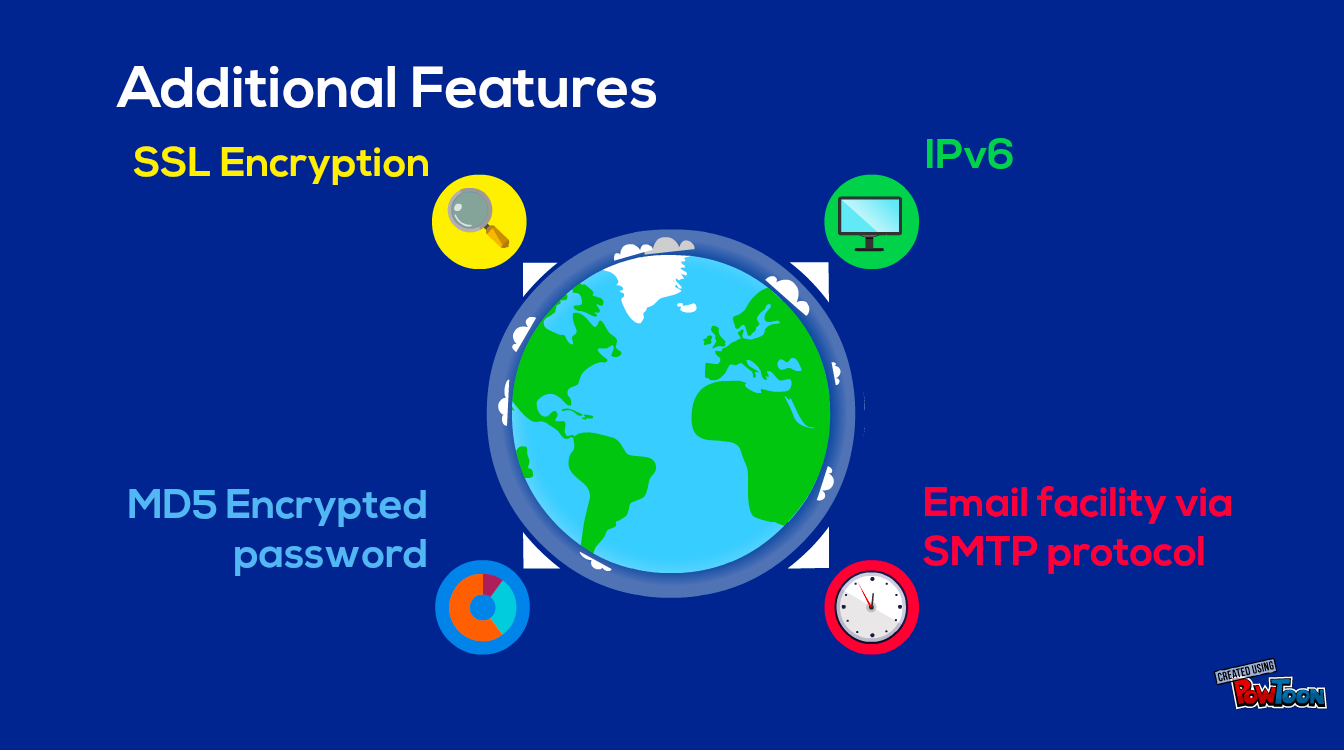
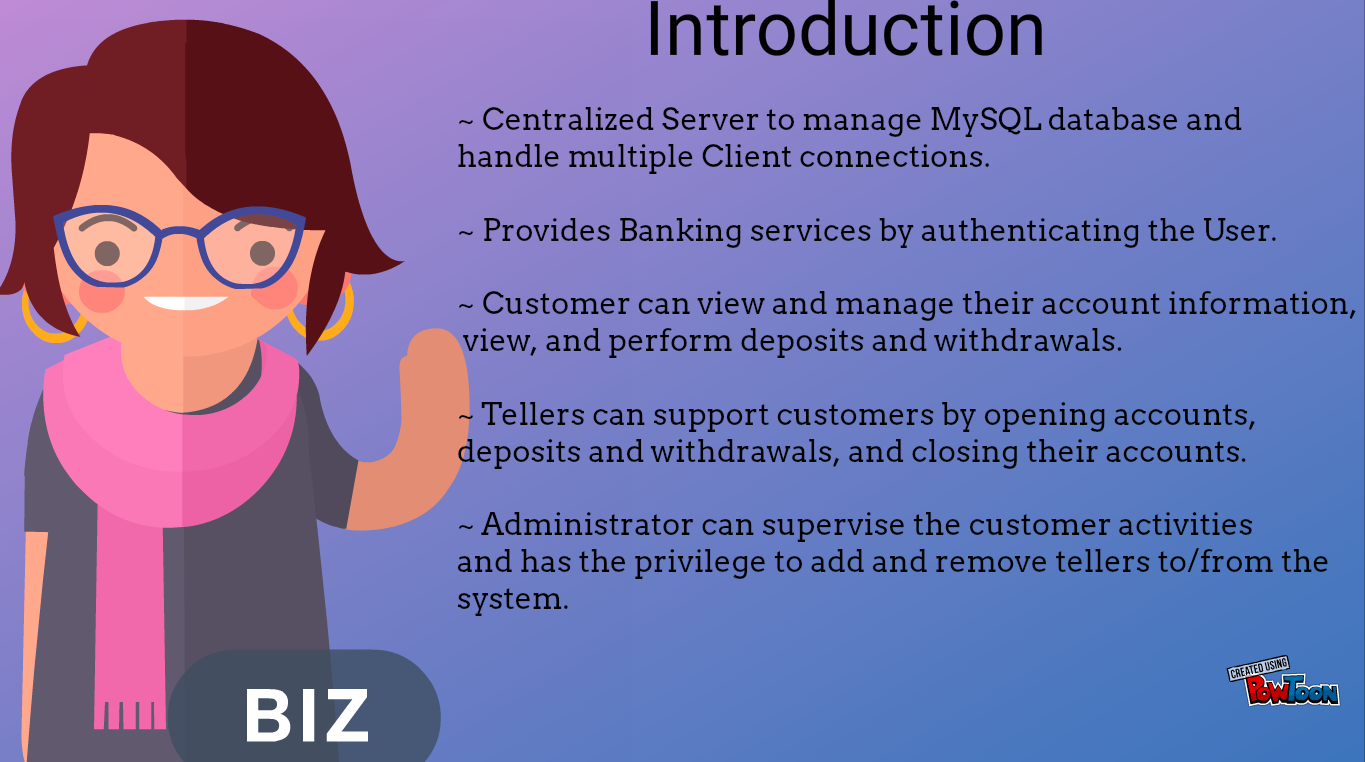
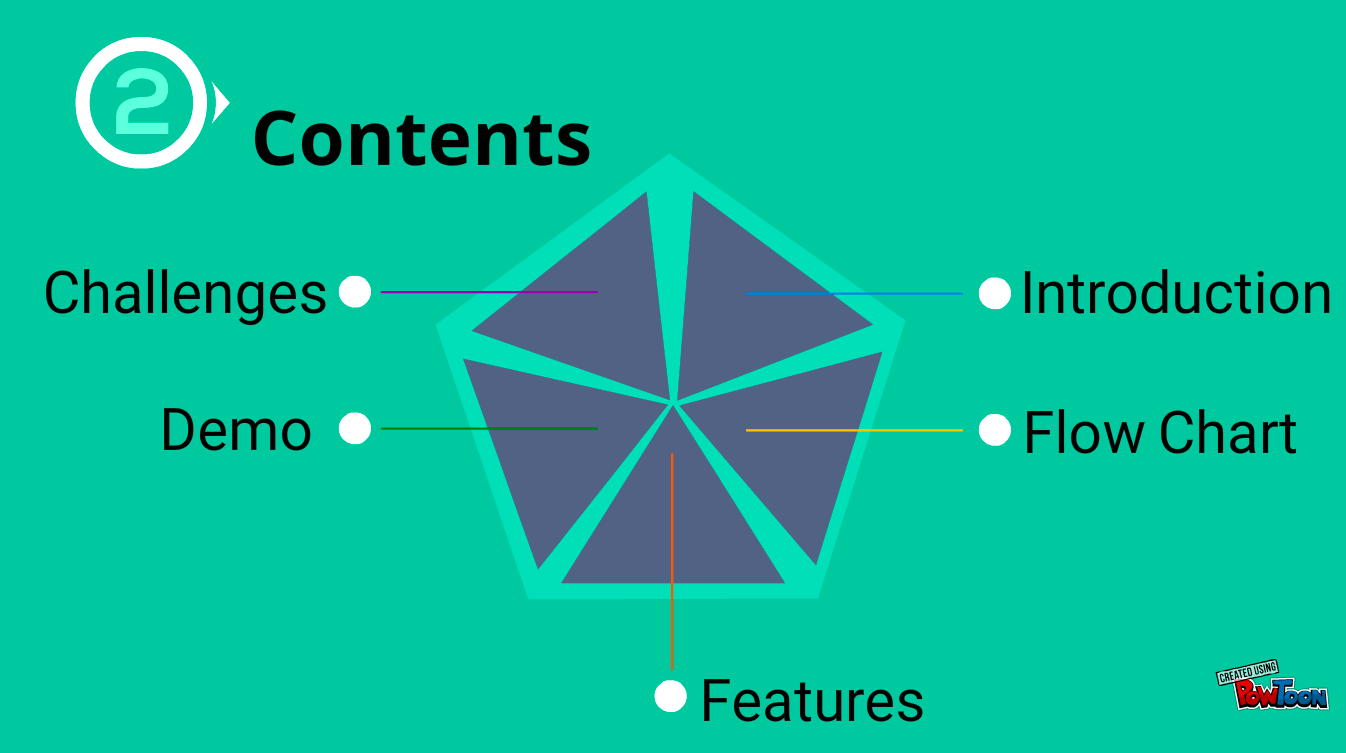
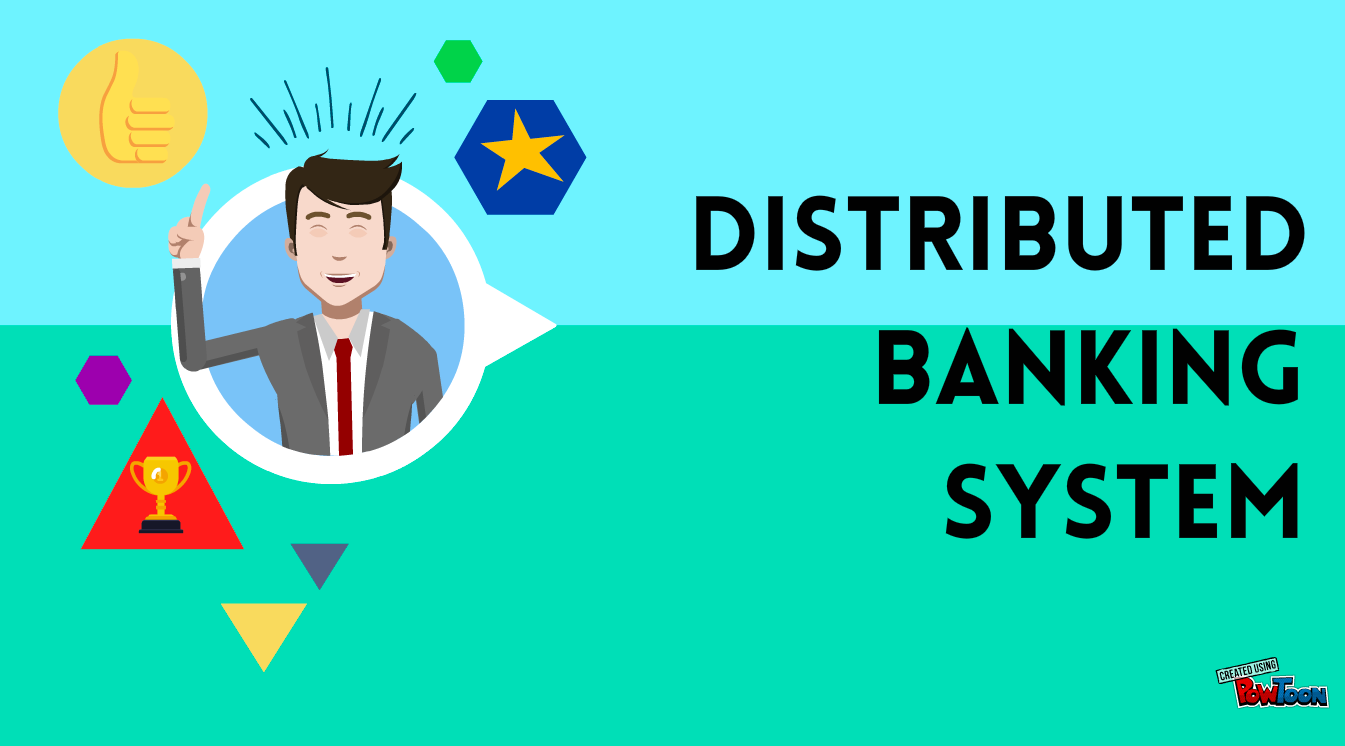
{% if choice == "Delete Teller record" %}

{% endif %}

</body>

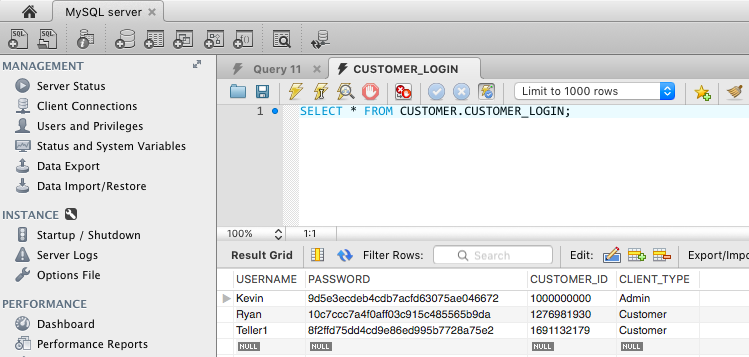
</html>

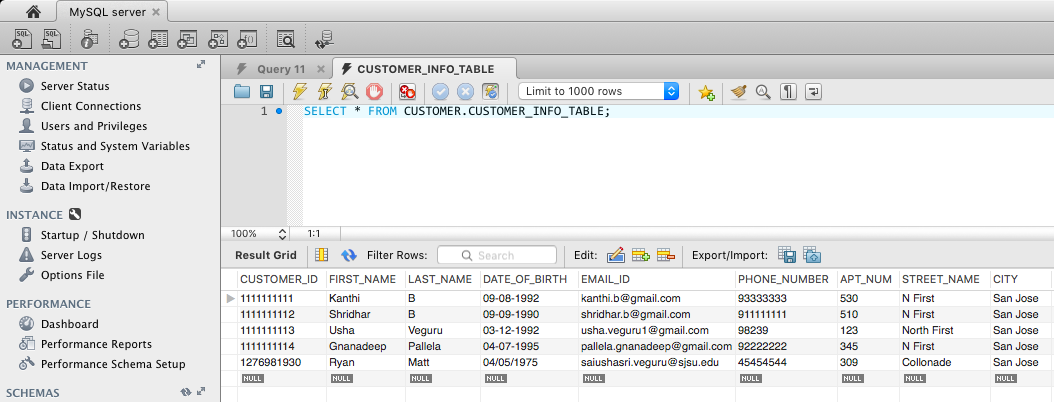
1. **Presentation Slides**

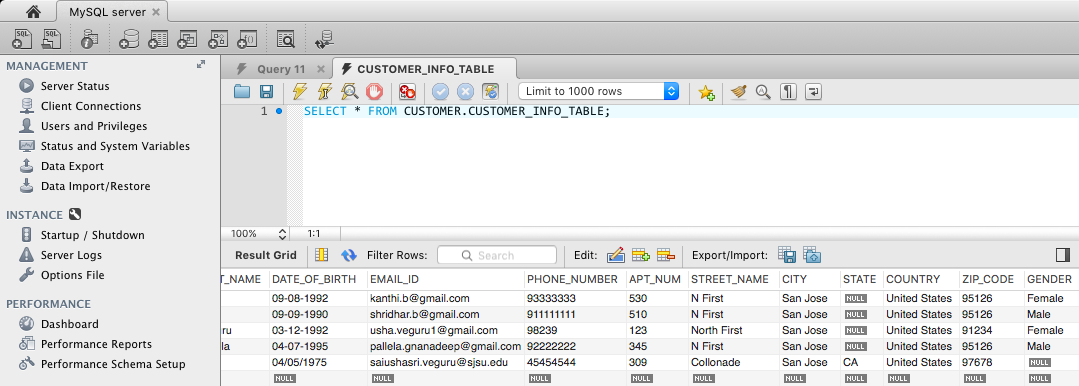
****

1. **Database - Screenshots**

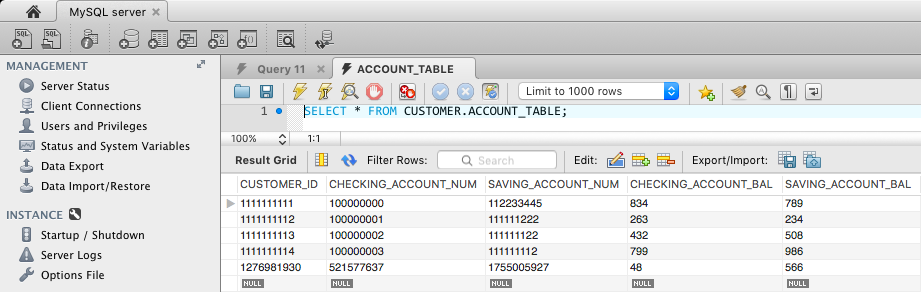
**Table 1: Customer Login**

****

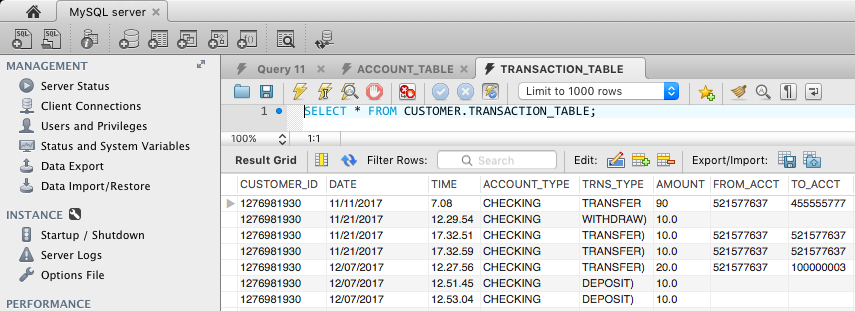
**Table 2: Customer Info Table**

****

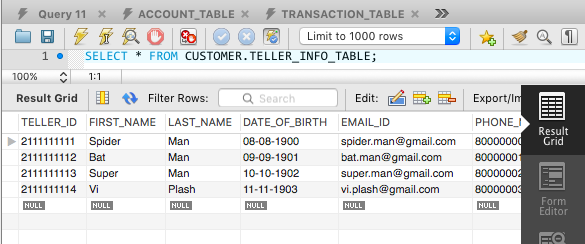
**Table 3: Account Table**

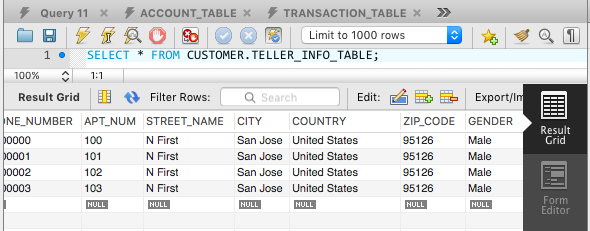
****

**Table 4: Transaction Table**

****

**Table 5: Teller Info Table**

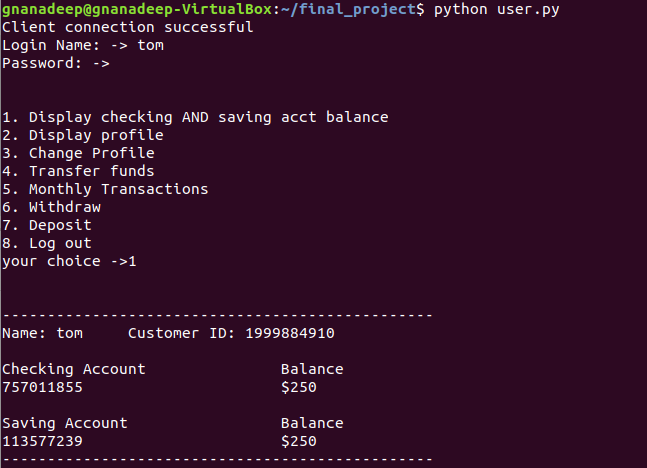
****

****

1. **Results - Screenshots**
   1. **Terminal**
2. **Customer**

**Case 1: Customer Login**

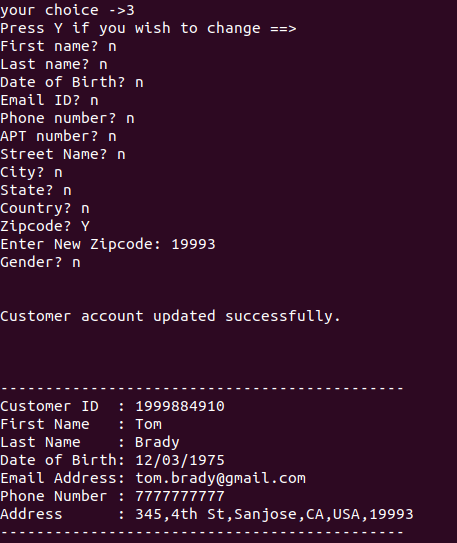
**Displaying checking and savings account balance**

****

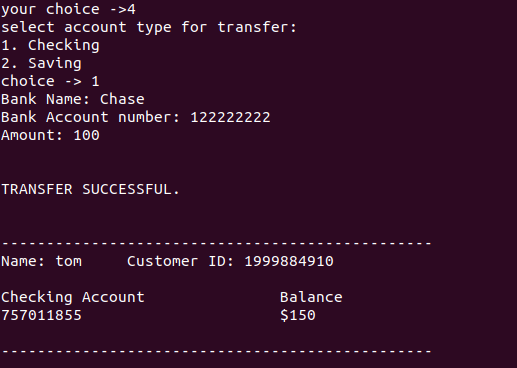
**Case 2: Display Profile**

****

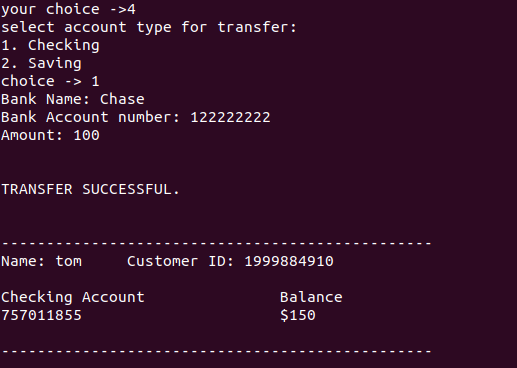
**Case 3: Update profile**

****

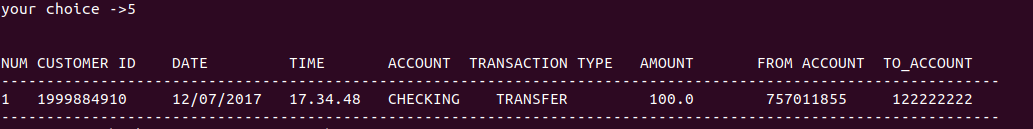
**Case 4: Transfer Funds - Checking Account**

****

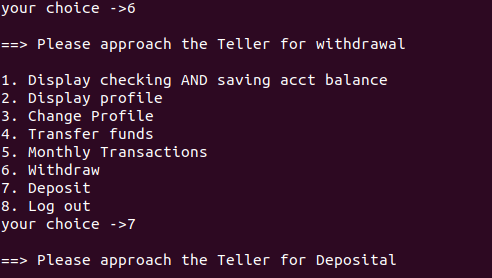
**Case 4: Transfer Funds - Checking Account**

****

**Case 5: Transaction Details**

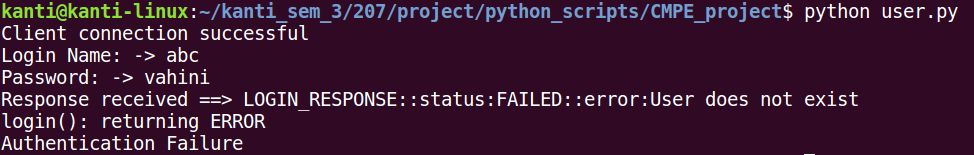
****

**Case 6: Withdraw and Deposit**

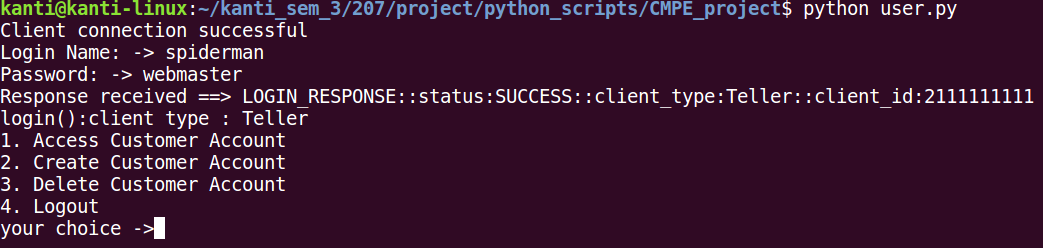
****

**2. Teller**

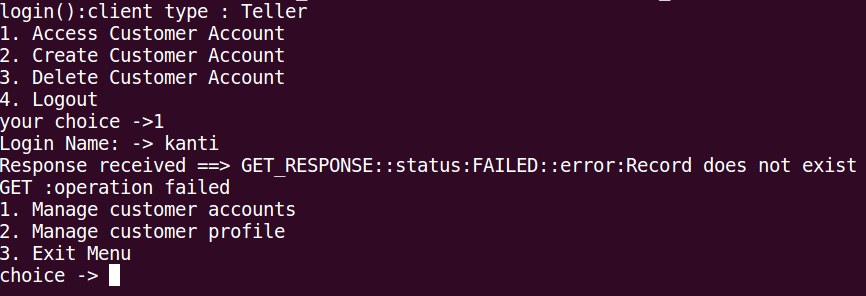
**Case 1: Un-registered Teller login failure**



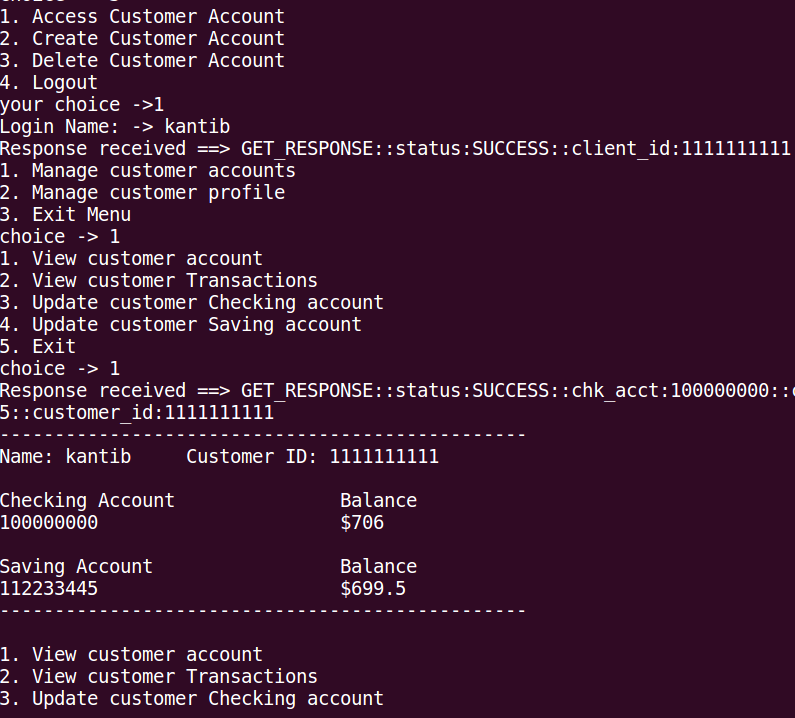
**Case 2: Valid Teller login**



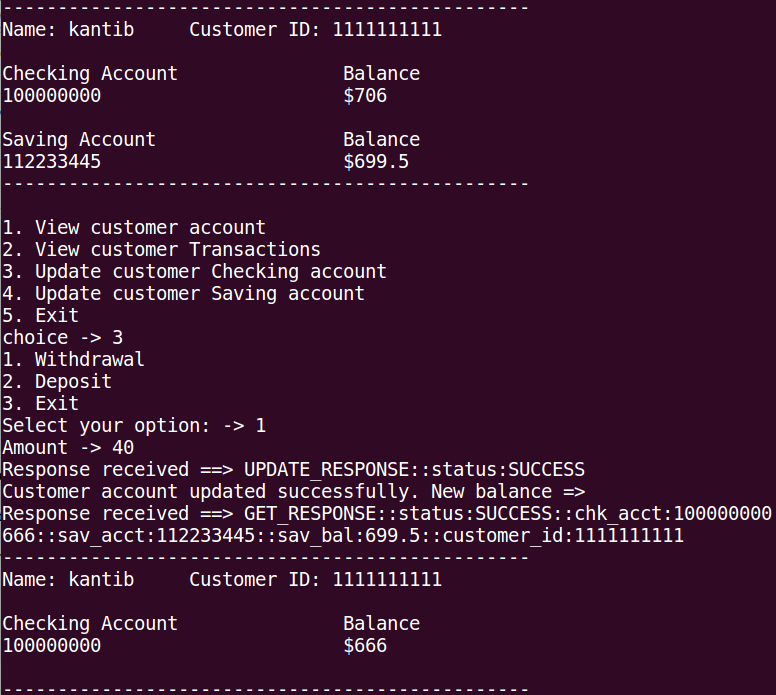
**Case 3: Invalid customer login name given to access customer accounts**



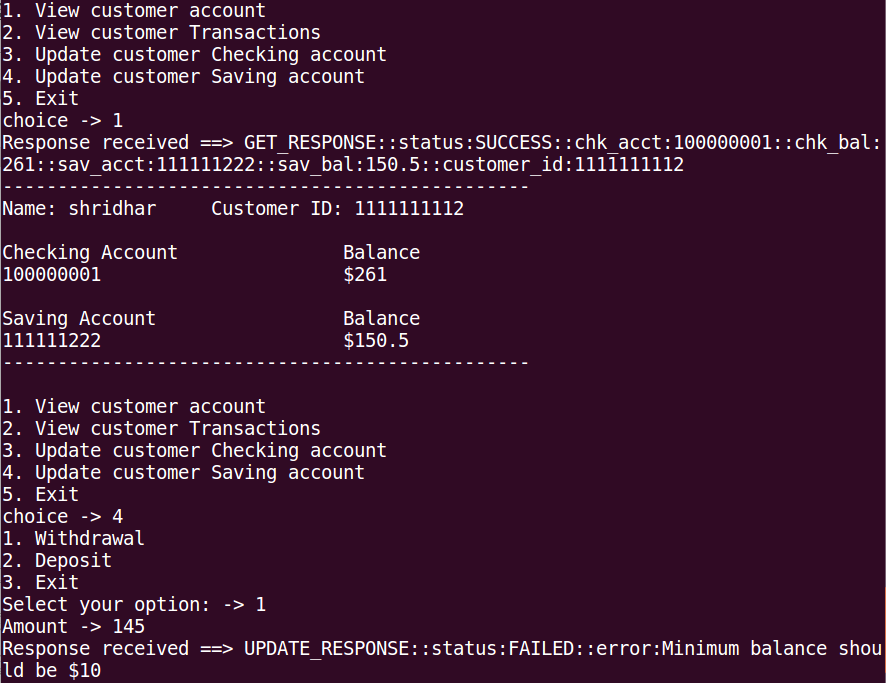
**Case 4: Valid customer login given to access customer accounts**



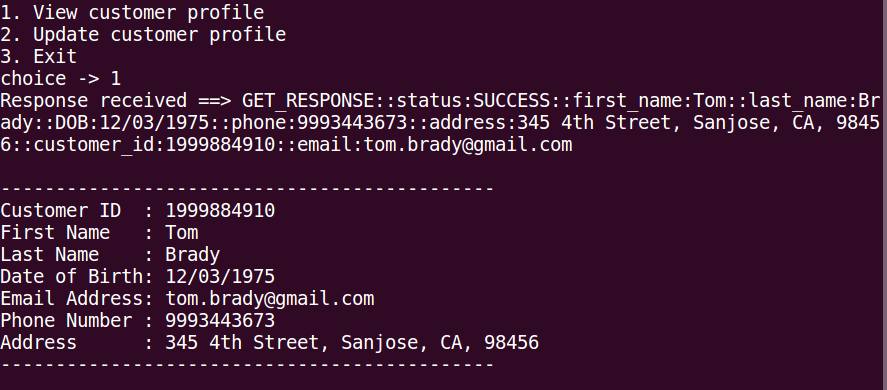
**Case 5: Customer withdrawal from checking account**



**Case 6: Not permitting the withdrawal if balance lowers than minimum balance of $10**



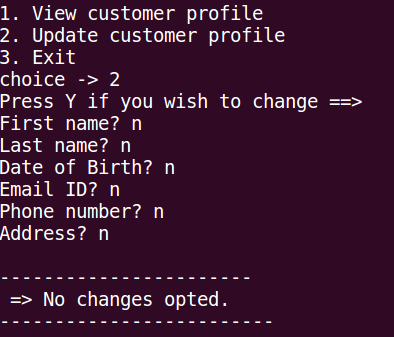
**Case 7: View Customer Information**



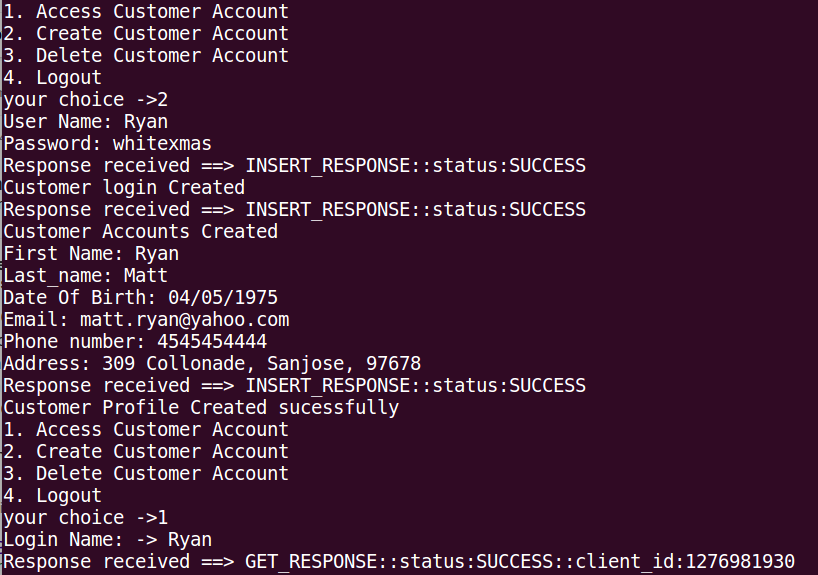
**Case 8: Update Customer Information**

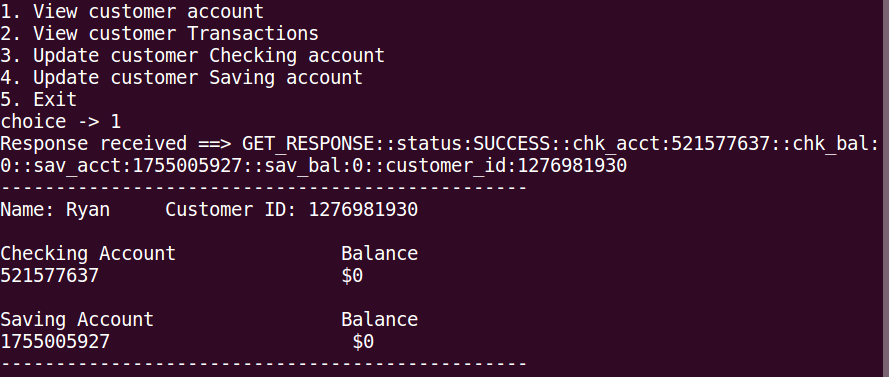


**Case 9: Error case for “change customer profile”**



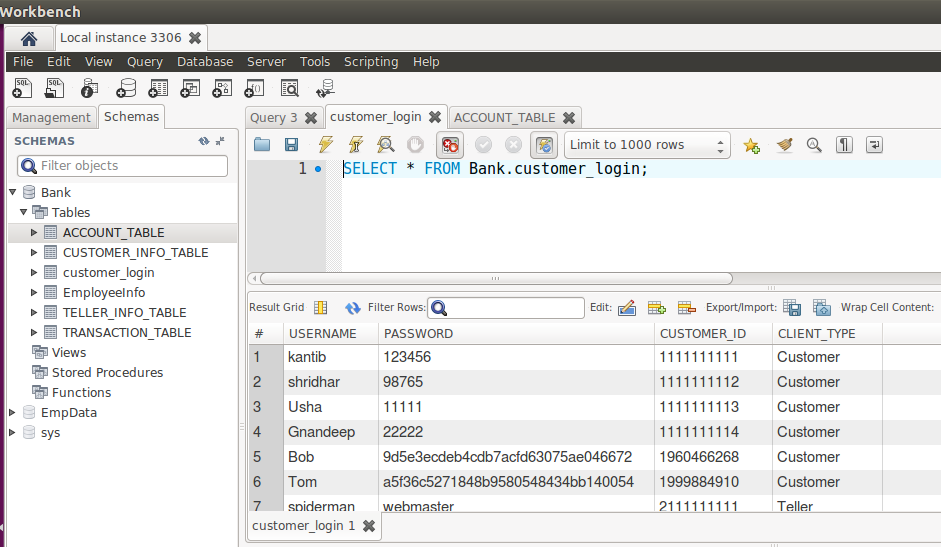
**Case 10: Create new Customer login and account records**

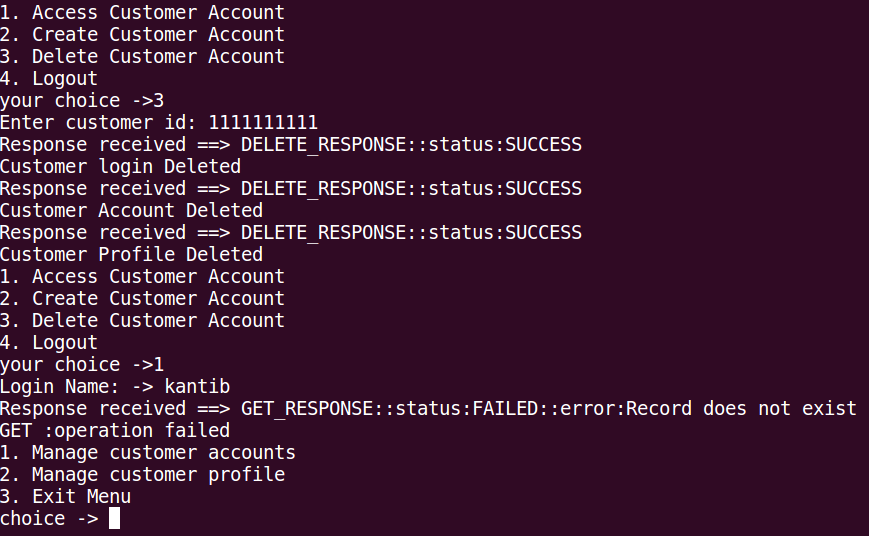




**Case 11: Delete existing customer**

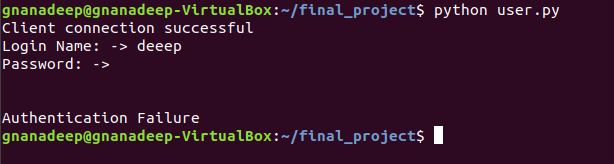
Existing customer records before deleting the Customer record:





**3. Administrator**

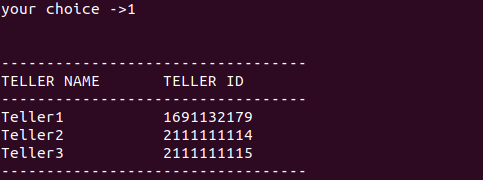
**Case 1: Login Failure - Wrong credentials**



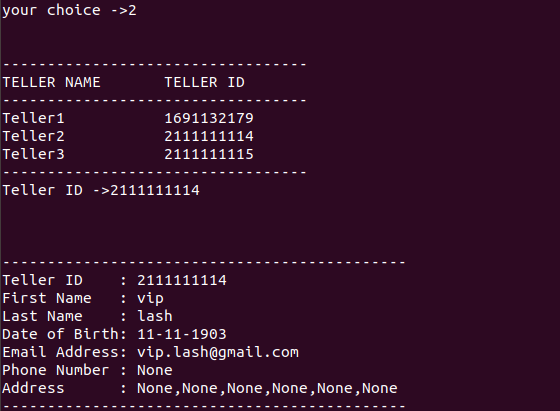
**Case 2: Login Successful - correct credentials**



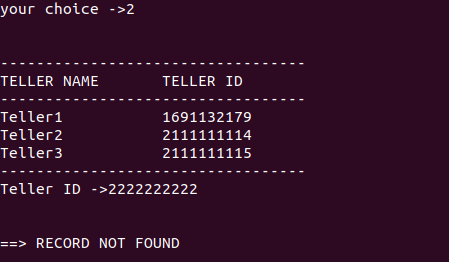
**Case 3: View Tellers**



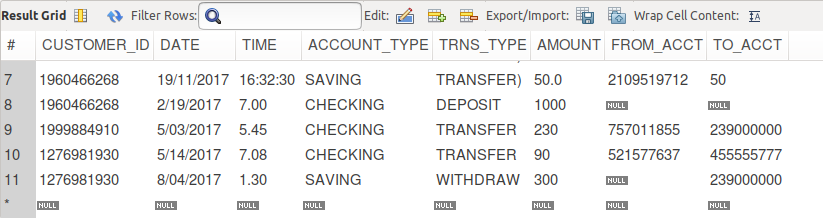
**Case 4: View teller information - Record in the database**



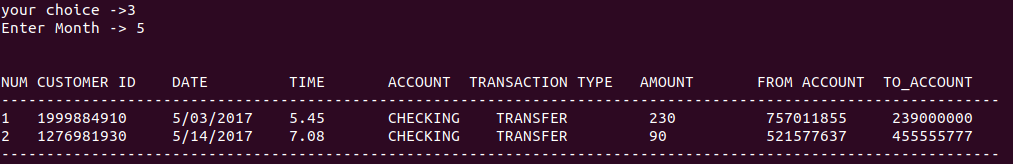
**Case 5: View teller information - Record not in the database**



**Case 6: Transaction Table**



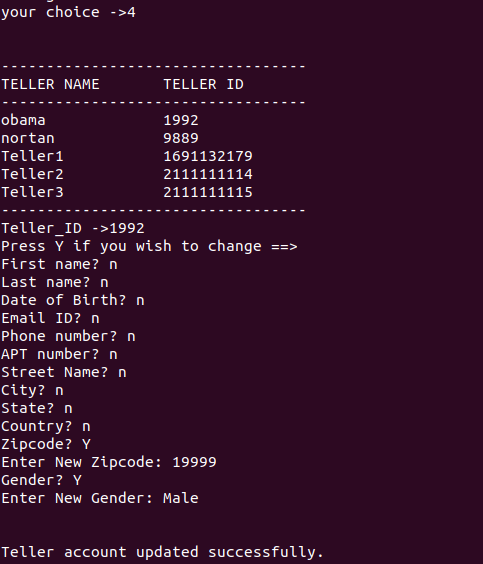
**Case 7: Monthly transactions**



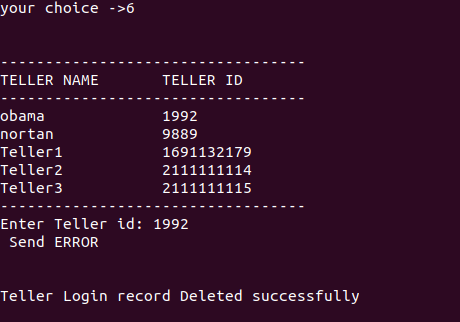
**Case 8: Create Teller account**



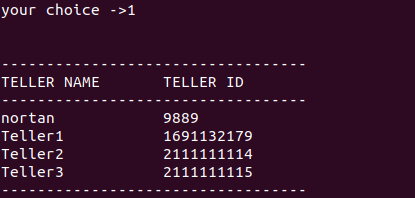
**Case 9: Update Teller Information**



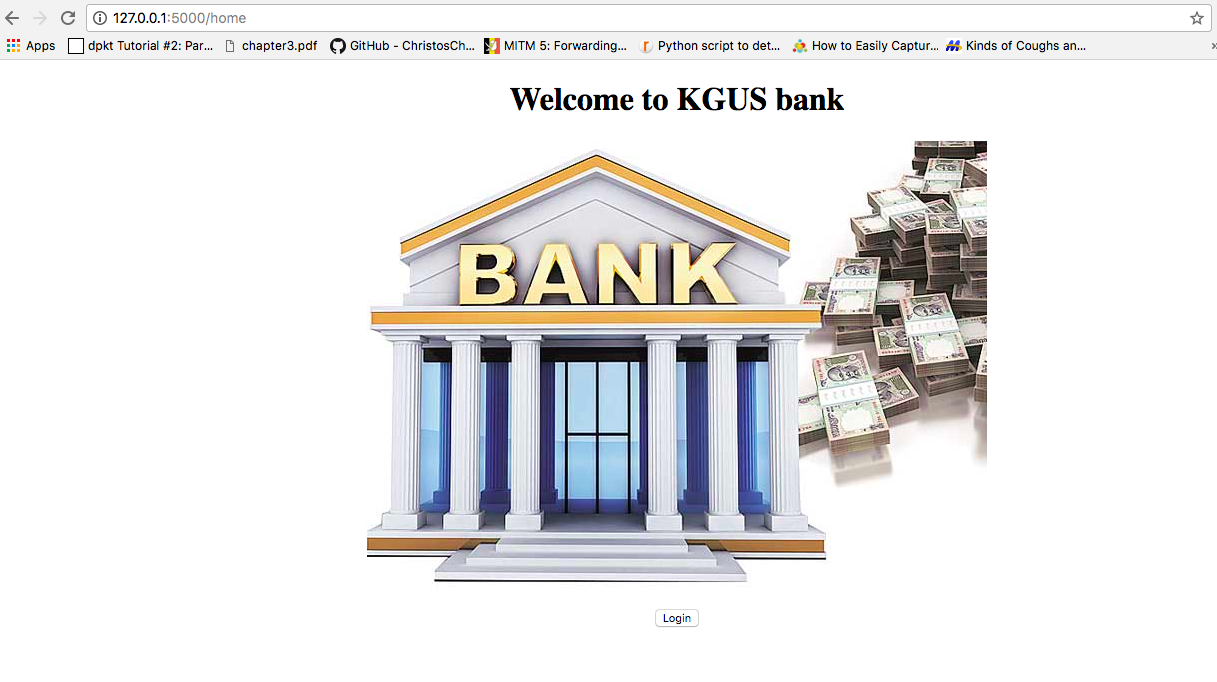
**Case 10: Delete Teller Information**



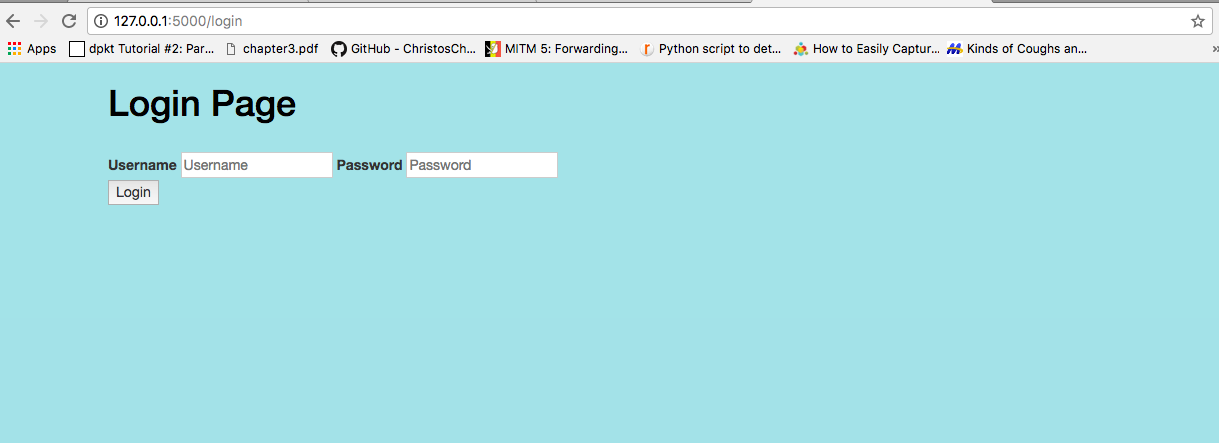
**View tellers after deleting teller**



* 1. **GUI**
* **Home page**

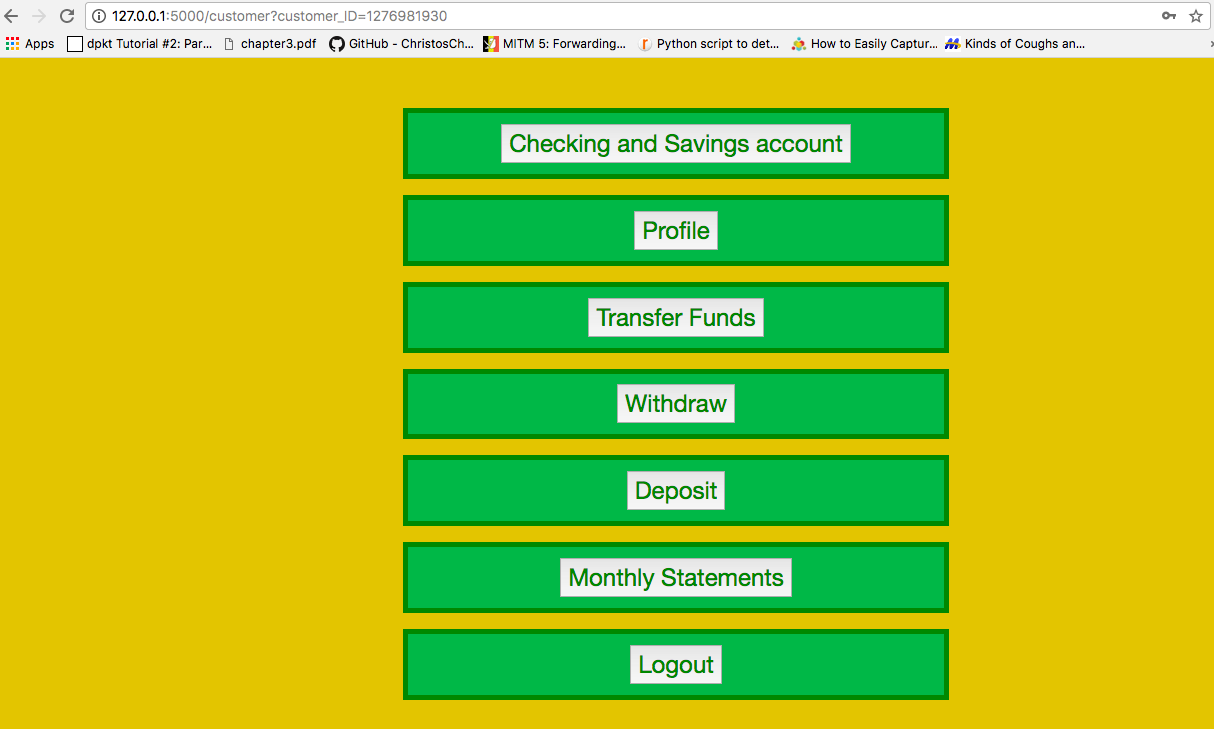
****

* **Login Page**

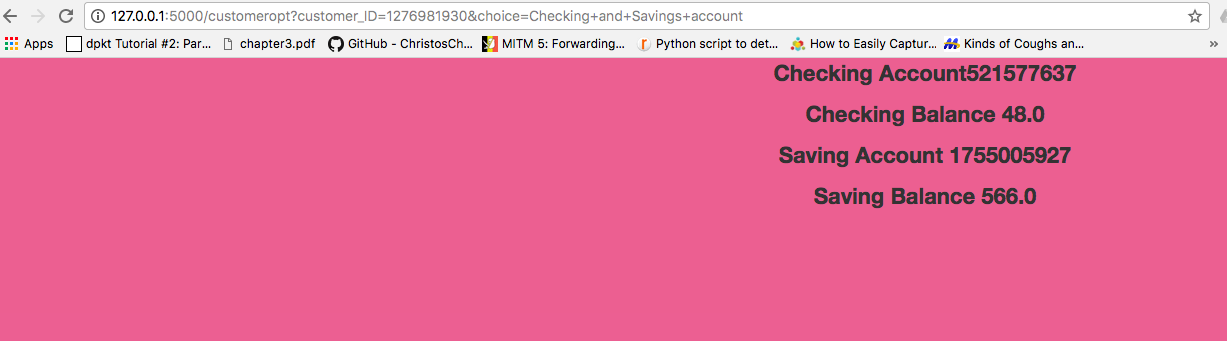
****

**Customer**

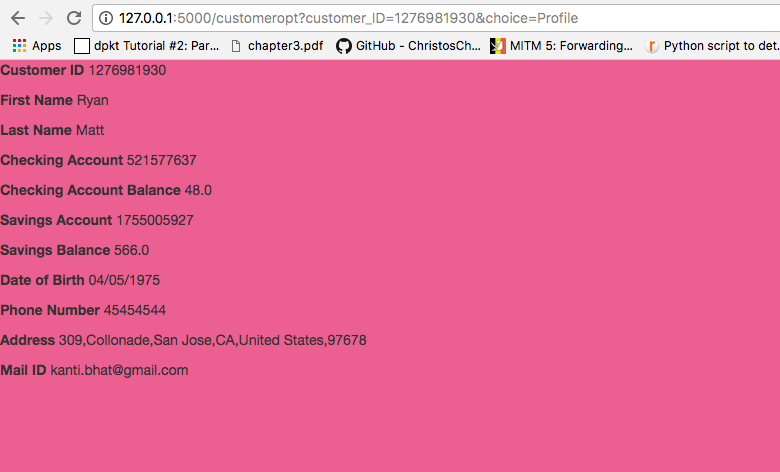
* Customer Options page

****

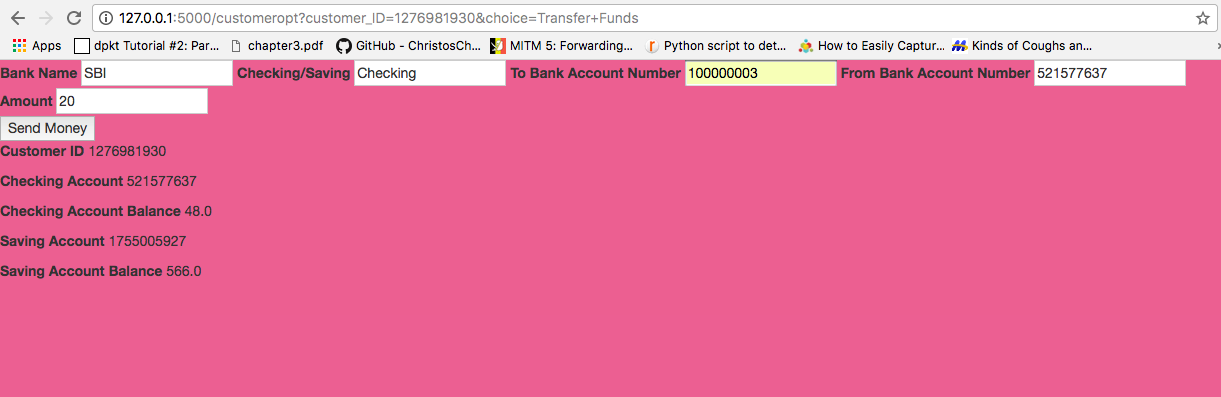
* Checking and Savings balance

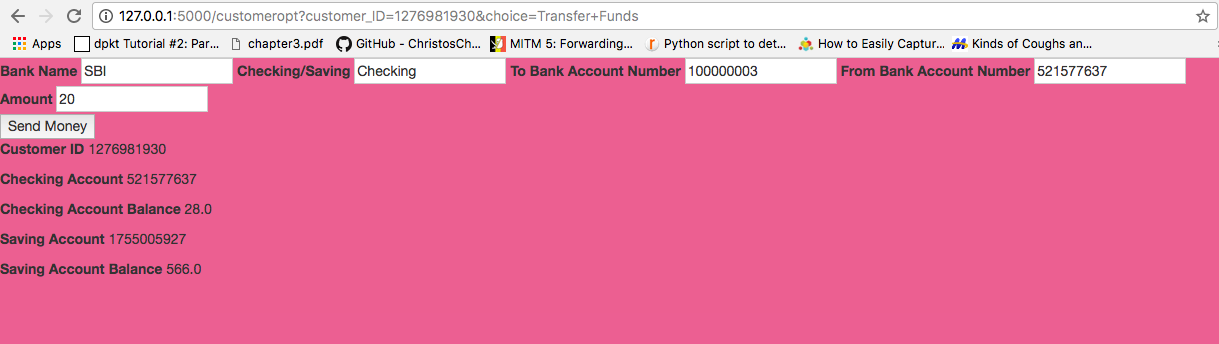
****

* View Customer profile

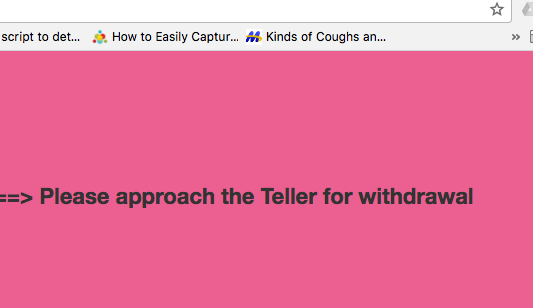
****

* Customer Transfer funds

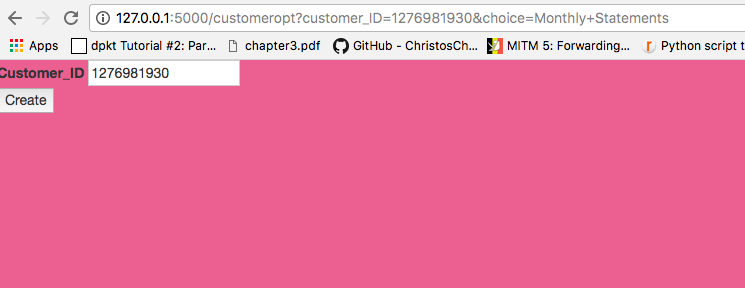




* Withdrawal

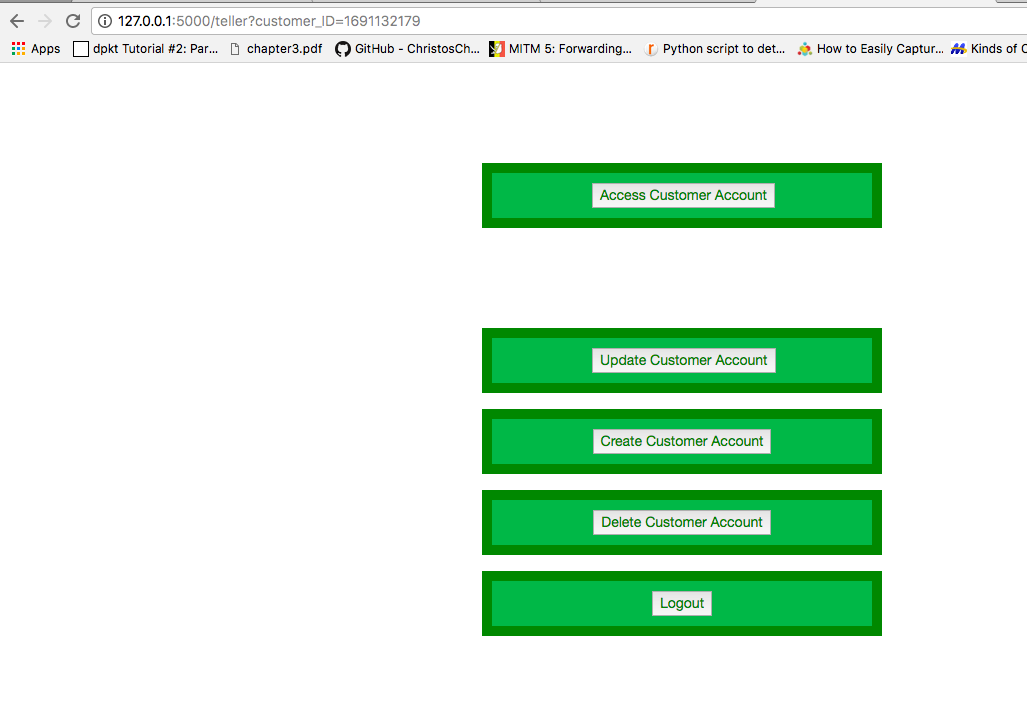


* Monthly Statements

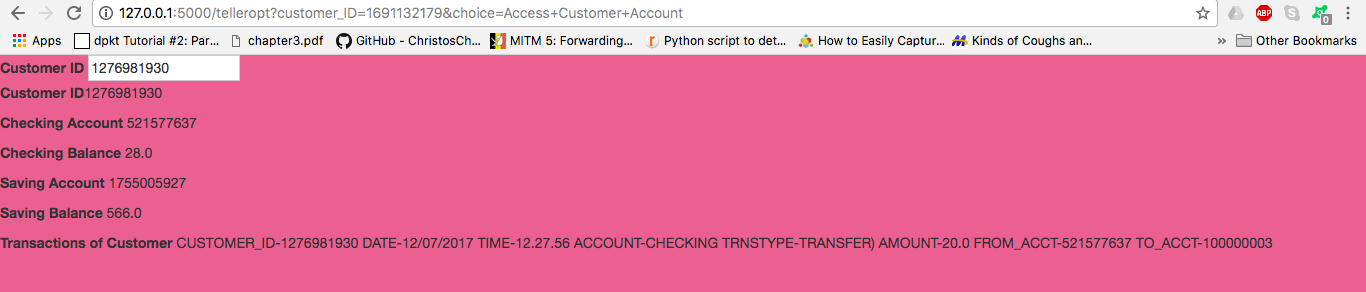


**Teller**

* Teller options

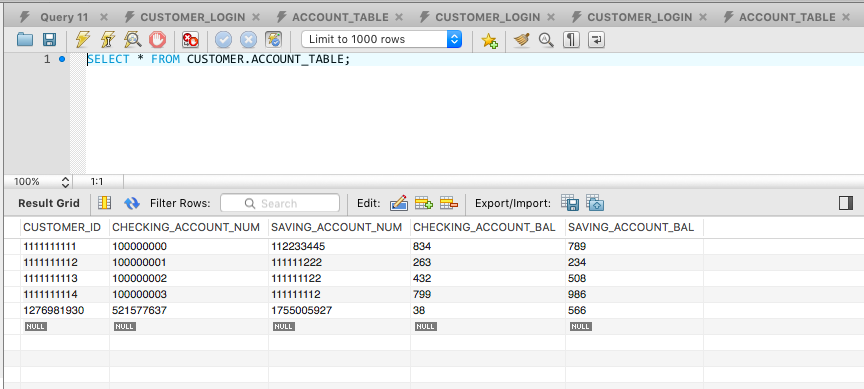


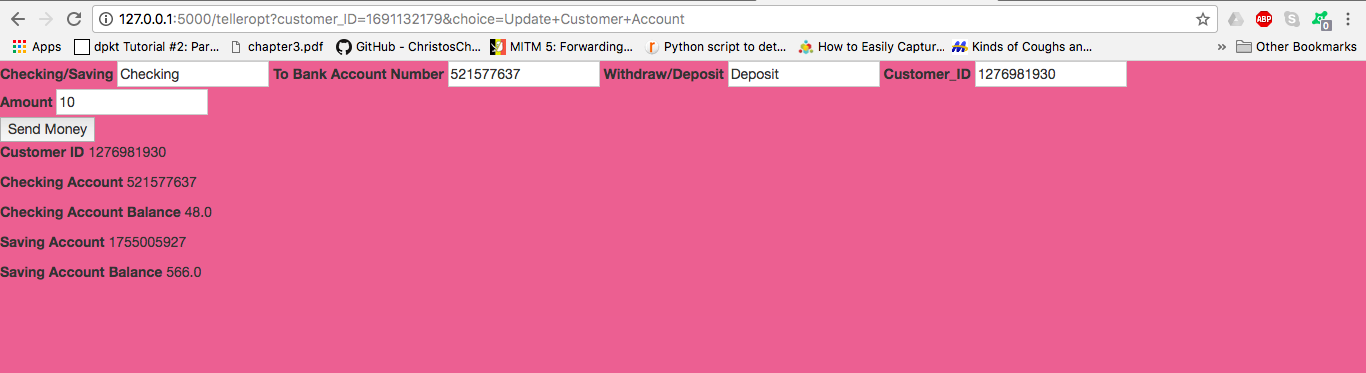
* Access customer account

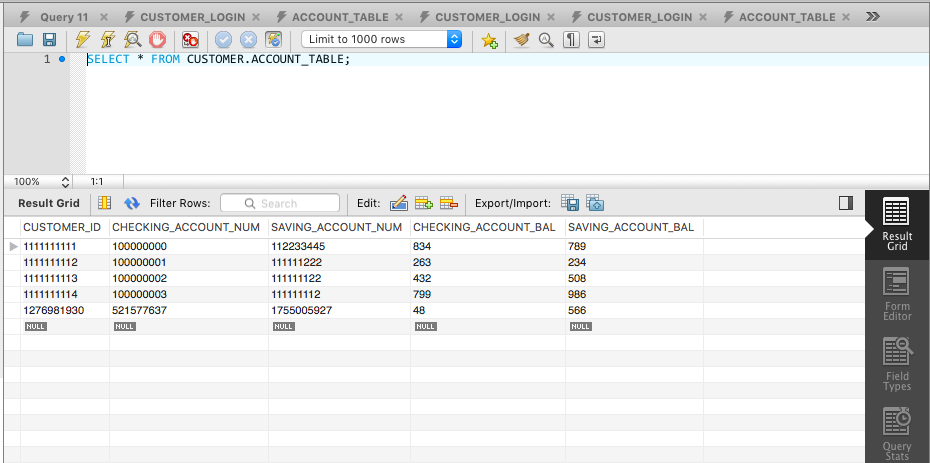


* Update customer account

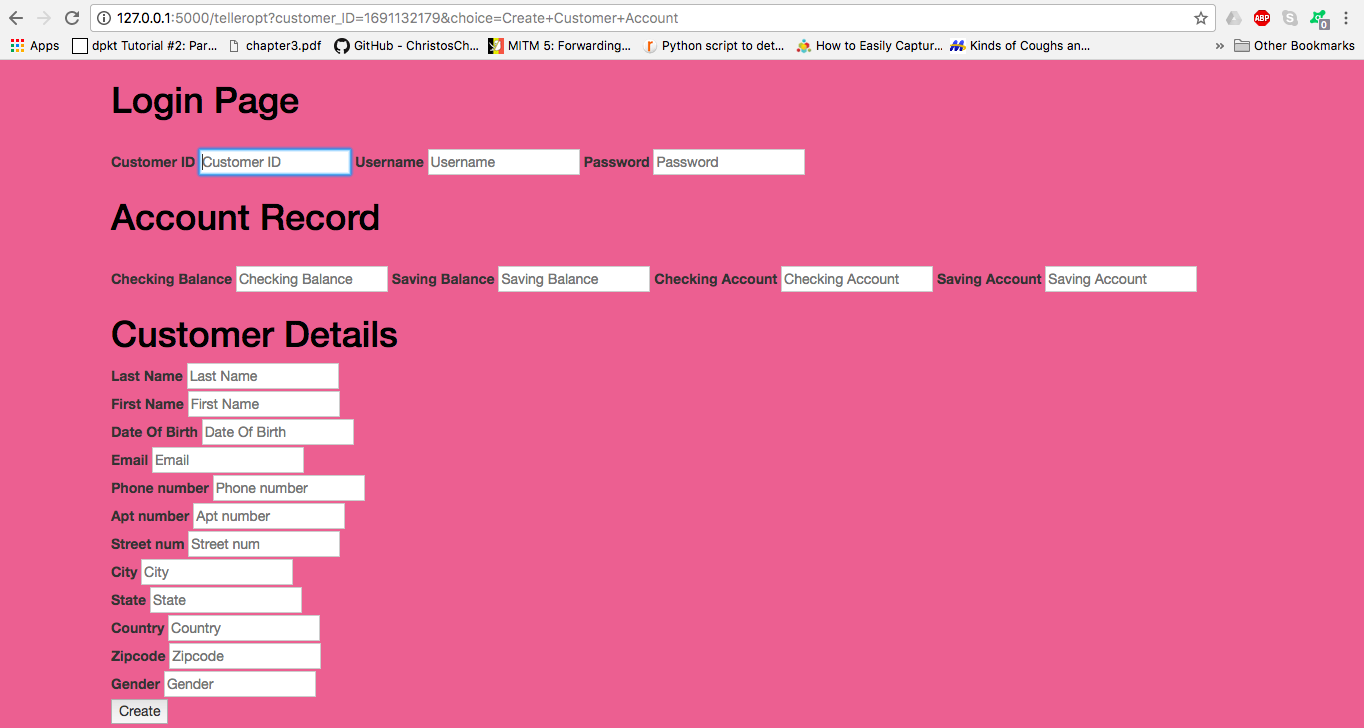




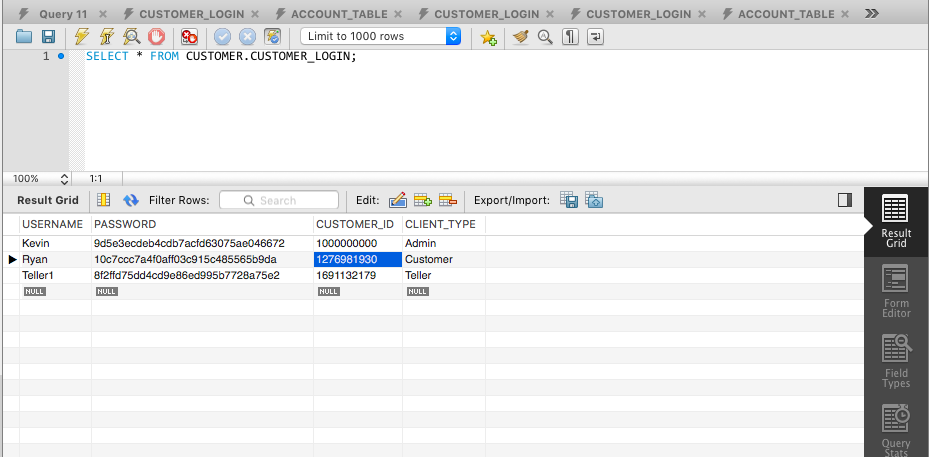


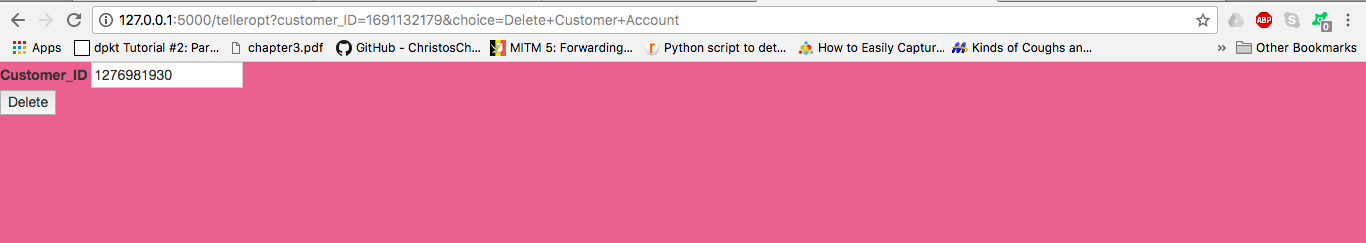


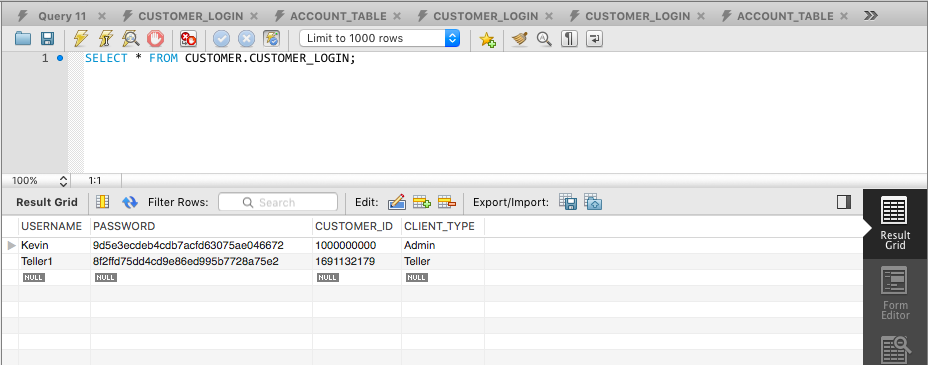
* Create customer details



* Delete Customer account

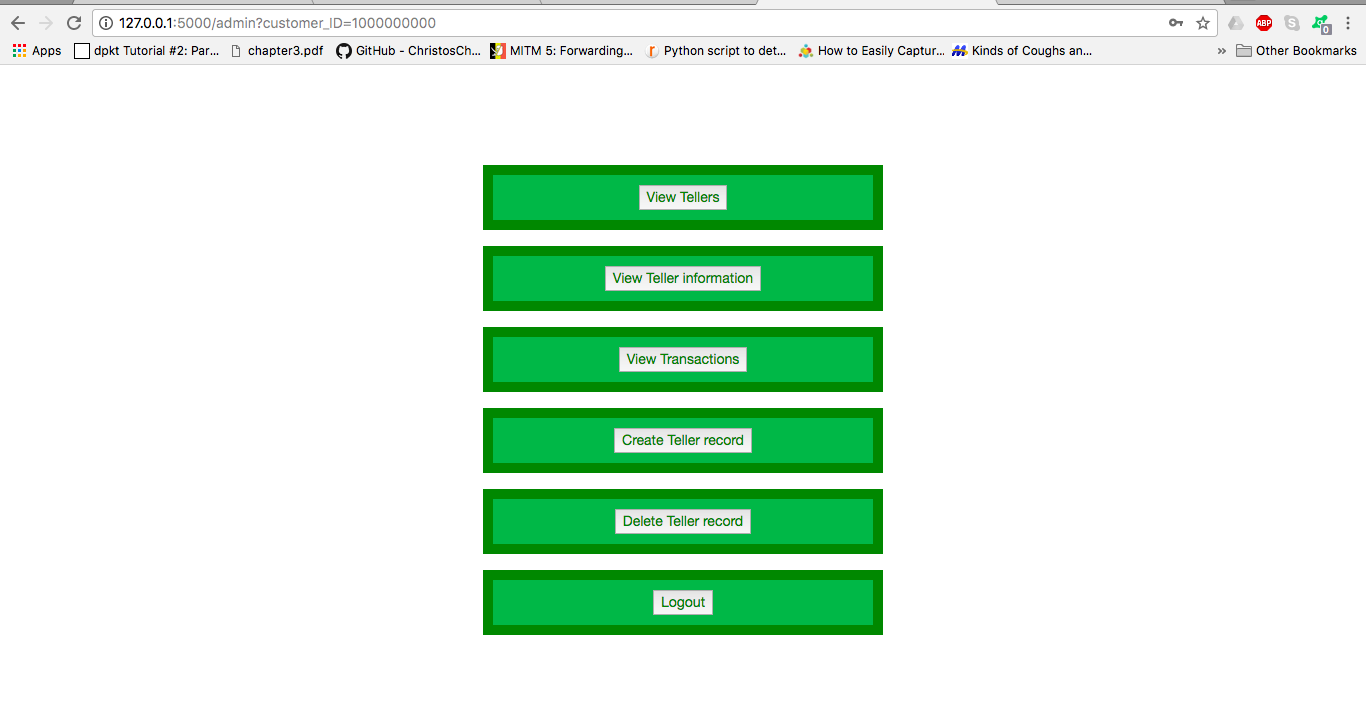


****

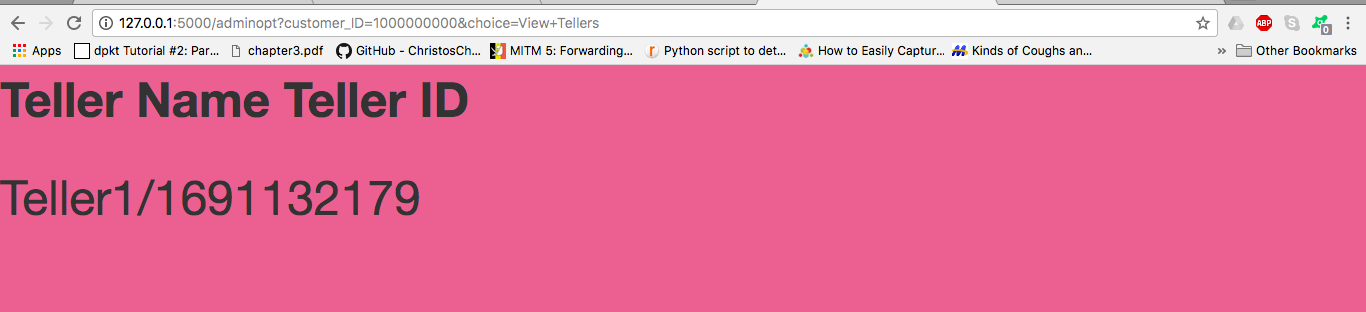
****

**Admin**

* Admin options

****

* Teller info

****

1. [↑](#footnote-ref-1)