NAVEED SABIR

Student No. 2224755

Contents

[**INTRODUCTION** 3](#_Toc125046858)

[**REQUIREMENTS AND MODELING** 4](#_Toc125046859)

[USER AND SYSTEM REQUIREMENTS 4](#_Toc125046860)

[FUNCTIONAL, NON-FUNCTIONAL AND DOMAIN REQUIREMENTS 5](#_Toc125046861)

[UML DIAGARAMS 6](#_Toc125046862)

[PSEUDOCODE 12](#_Toc125046863)

[**SOFTWARE ARTEFACT** 15](#_Toc125046864)

[DEVELOPMENT OF A PROTOTYPE APPLICATION 15](#_Toc125046865)

[ISSUES FACED AND DEALT WITH 15](#_Toc125046866)

[CREATION OF DATABASE 16](#_Toc125046867)

[**MENU FUNCTION** 17](#_Toc125046868)

[**CREATION OF TABLES** 23](#_Toc125046869)

[**USER CRUD FUNCTIONS** 26](#_Toc125046870)

[**CUSTOMER CRUD FUNCTIONS** 28](#_Toc125046871)

[**ACCOUNT CRUD FUNCTIONS** 30](#_Toc125046872)

[**TRANSACTION CRUD FUNCTIONS** 33](#_Toc125046873)

[**ADDRESS CRUD FUNCTIONS** 37](#_Toc125046874)

[**TESTING OF APPLICATION** 40](#_Toc125046875)

[**REFERENCES** 41](#_Toc125046876)

[BIBLIOGRAPHY 42](#_Toc125046877)

[Figure 1 Use Case diagram 6](#_Toc125146179)

[Figure 2 Class Diagram 7](#_Toc125146180)

[Figure 3 Sequence Diagram 9](#_Toc125146181)

[Figure 4 Activity Diagram 11](#_Toc125146182)

[Figure 5 Hierarchy diagram 11](#_Toc125146183)

[Figure 6 create .db 16](#_Toc125146184)

[Figure 7 Error Handling 21](#_Toc125146185)

[Figure 8 Error handling in Menu 22](#_Toc125146186)

[Figure 9 Table1 & Table 2 23](#_Toc125146187)

[Figure 10 Table 3 & Table 4 24](#_Toc125146188)

[Figure 11 Table 5 25](#_Toc125146189)

[Figure 12 Testing 41](#_Toc125146190)

# **INTRODUCTION**

OVERVIEW

Due to the limited internet infrastructure and technological accessibility in some regions, development of a solution that can run on a Raspberry Pi 400 has been requested, which will be included in the "Welcome Pack" for each participating entrepreneur and help them manage their customers' credit accounts. Because Python 3 and the embedded SQLite 3 database both offer respectable performance individually on low-powered devices, this developed application will have a Command Line Interface (CLI) for the User Interface (UI) for simplicity. Python 3 will be used for the development of the persistence layer.

WHAT TO EXPECT

The report goes into depth and provides description and mentions a step-by-step execution of the above scenario and application.

Multiple requirements aimed to be met, accurate modelling of the scenario with diagrams expected, effective software development techniques expected, along with software testing.

# **REQUIREMENTS AND MODELING**

## USER AND SYSTEM REQUIREMENTS

USER REQUIREMENTS

* Create, view and update customers’ details

After the customer (entrepreneur) has full access to the system as an Admin, they get to add new values (customers) in the system, they can read information about that value, and they are also allowed to update or change customer details.

* Record new transactions relating to individual customers

*When a transaction occurs, the admins are able to store it for safety purposes which means they are able to view every transaction.*

* Mark accounts as ‘closed’ using status and date fields

*Closing accounts due to inactivity based on dates.*

* View a customers' transactions for a specific date range

*Ability to view transactions.*

* View all transactions for a specific date range

*Ability to view transactions.*

SYSTEM REQUIREMENTS

* A Structured programming approach is expected with the appropriate use of sequence, selection and iteration constructs.
* Functions that do specific tasks should be written with an aim of algorithm optimisation and code reuse.
* Techniques to ensure that the application is robust and resilient should be employed including data validation and exception handling techniques.
* Optimised SQL queries featuring Joins and Aggregation should be used where possible rather than complex programmatic solutions which may impede speed and efficiency
* Formatting features or libraries such as Tabulate or Pandas data frames should be used to enhance User Experience (UX) by formatting data output in tabular format.

## FUNCTIONAL, NON-FUNCTIONAL AND DOMAIN REQUIREMENTS

FUNCTIONAL REQUIREMENTS

**OBJECTIVES**

* Manage customers in the system
* Manage all transactions in the system
* Authentication for security purposes

**FEATURES**

* Add new customers to the system
* View customer details
* Update customer details
* Record new transactions
* De-activate customer accounts
* View transactions of any date range
* Authentication

NON-FUNCTIONAL REQUIREMENTS

Non-functional requirements are functional requirements but considering the following:

* **Security**

Protecting personal customer details and passwords by using an authentication system that encrypts data.

* **Reliability**

Forming a reliable system that does not cause problems or does not have bugs.

* **Performance**

The ability for the system to work fast and not slow. A slow system could also jeopardize and affect negatively the security of the system.

* **Maintainability**

Maintaining the system and keeping it up to date by adding new potential features and options that help the user in various tasks accurately.

* **Scalability**

The ability of a system to grow and not get potentially damaged at the same time. In this case, add more customers, process more transactions faster and processing more data than the initial mode it was created on.

* **Usability**

How difficult it will be for a customer to use and access all the features, that requires a user- friendly environment which with time will help the customer understand how the system works. It is also defined by how many tasks can a user complete in a certain amount of time.

## UML DIAGARAMS

The **Use Case** diagram describes the context and the important requirements that form a system. In the diagram below its described how the developer (Administrator) designs the system for the client (Entrepreneur) and it shows how the client has access to all the requirements by connecting the actor with the requirement.

Diagram

Description automatically generated

Figure Use Case diagram

The **Class** Diagram below describes the object – oriented side of a system and outlines the relationships and how they connect to one another so the system or subsystem can operate effectively. The diagram shows how for this project, the tables; Customer, Account, Transaction, Address & User are connected along with the attributes that attach to each one separately.

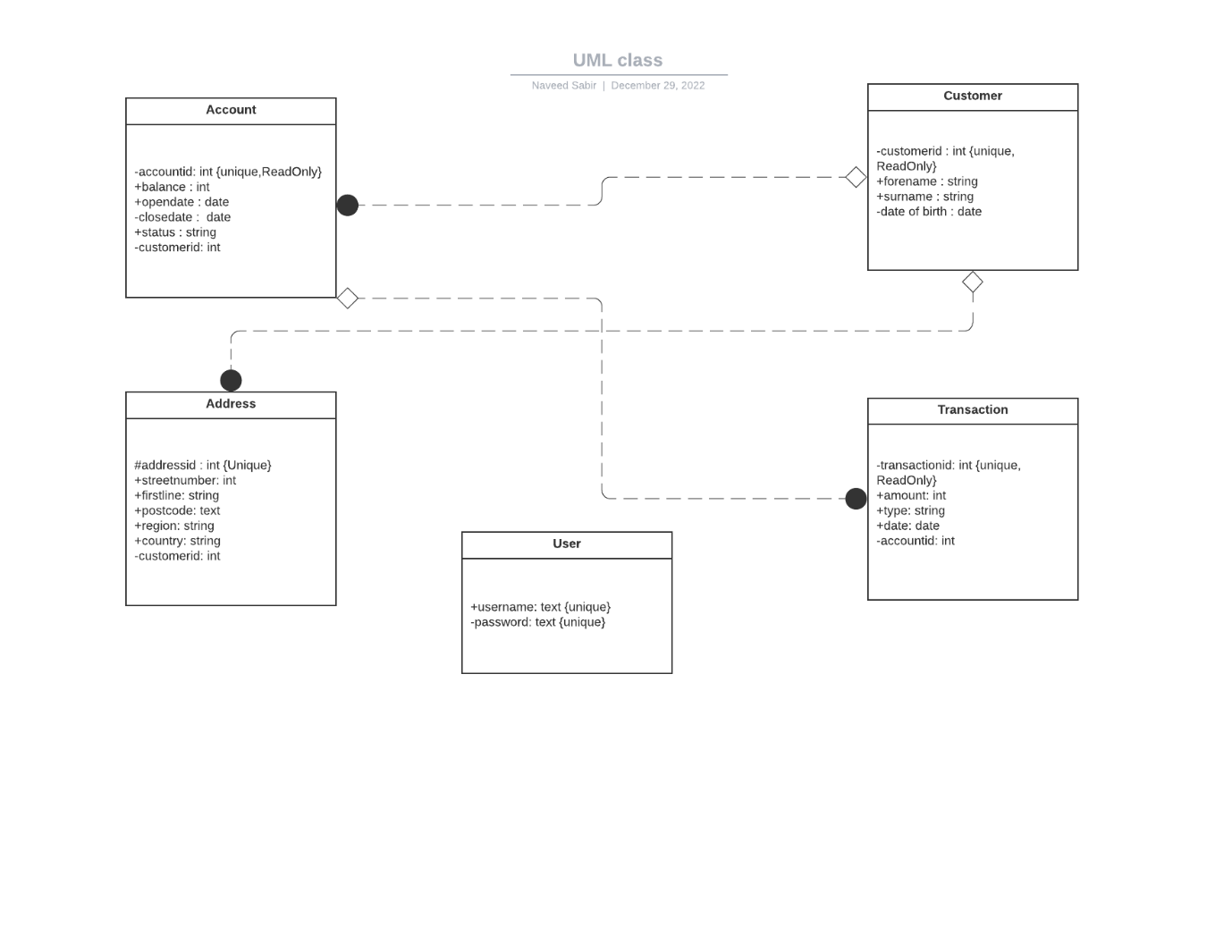


Figure Class Diagram

The **sequence diagram** describes how the objects or the functions of the system interact with one another in a sequential way. The diagram describes a group of projects and their lifeline as to how long each task lasts for and the duration behind it.

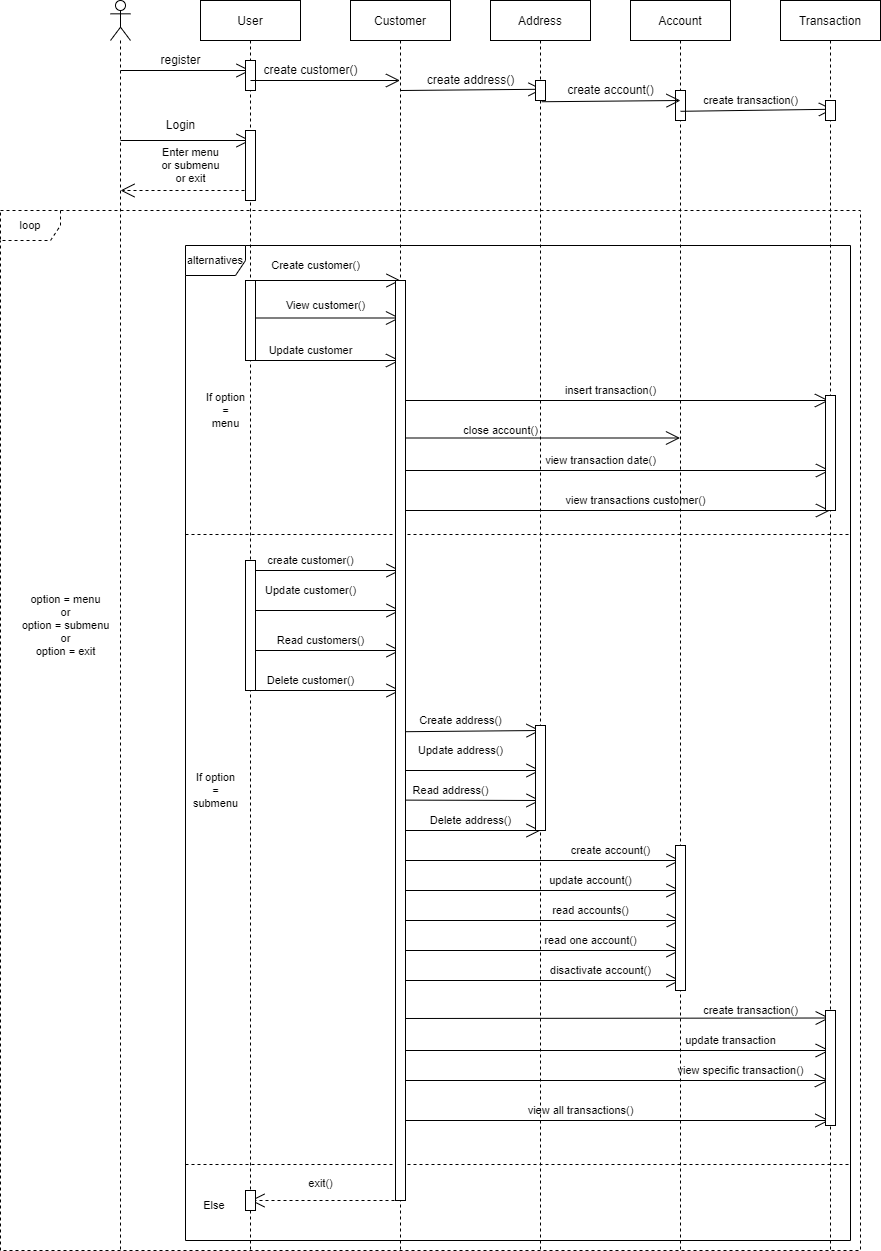


Figure Sequence Diagram

The **Activity diagram** describes the implementation of system process in depth and how each function or class works individually and in what order. It describes the most complex progressions of a system as well as the workflow of the system.

The diagram below shows all the conditions and the workflow of the system along with that activities take place and in what order. It is clear how each condition is affected by each action separately.

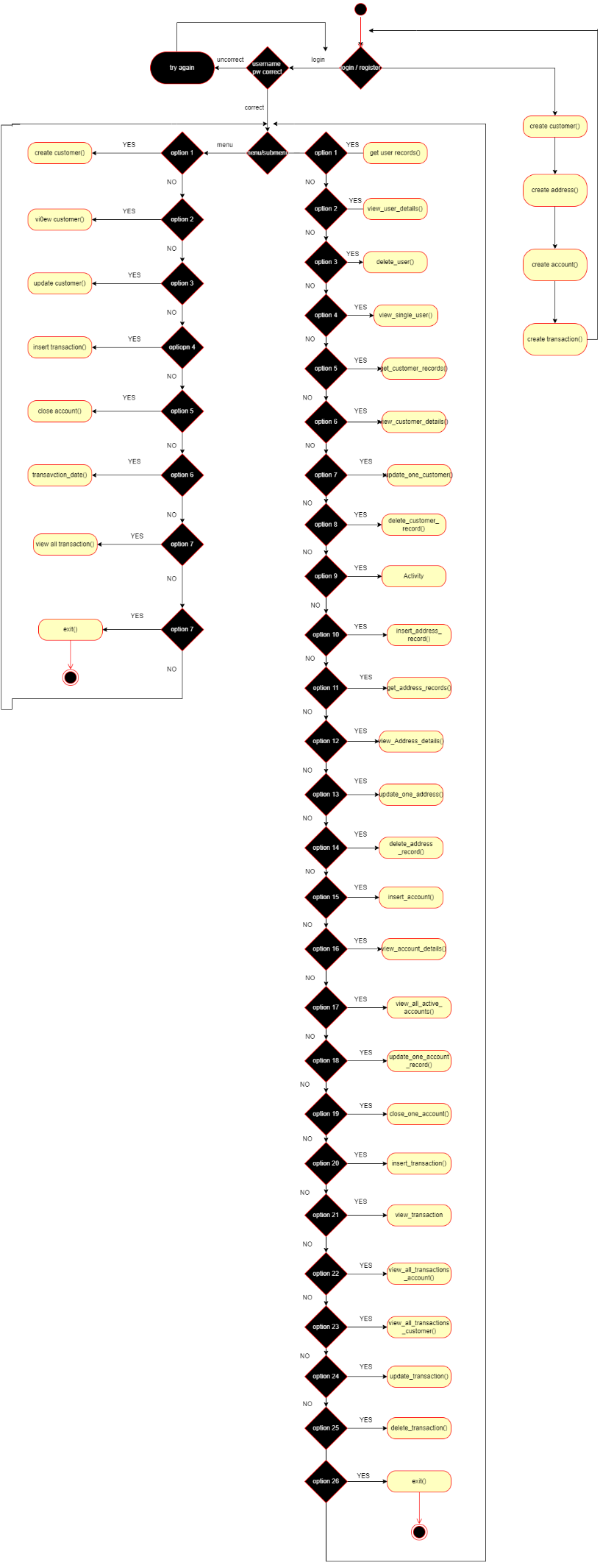


Figure Activity Diagram

The **hierarchy diagram** is structure based diagram used to present and describe the classes, attributes, functions or modules of a system and the relationship between them.

The diagram below outlines all the functions and attributes of the system from top to bottom and shows how the User needs to login or register in order to have access to the CRUDs of the system that are listed below. (Diagram was made prior to coding which means there are changes applied to the actual system that are not actively updated with the diagram. However it shows accurately how the system is ran and the logic behind the creation of it.

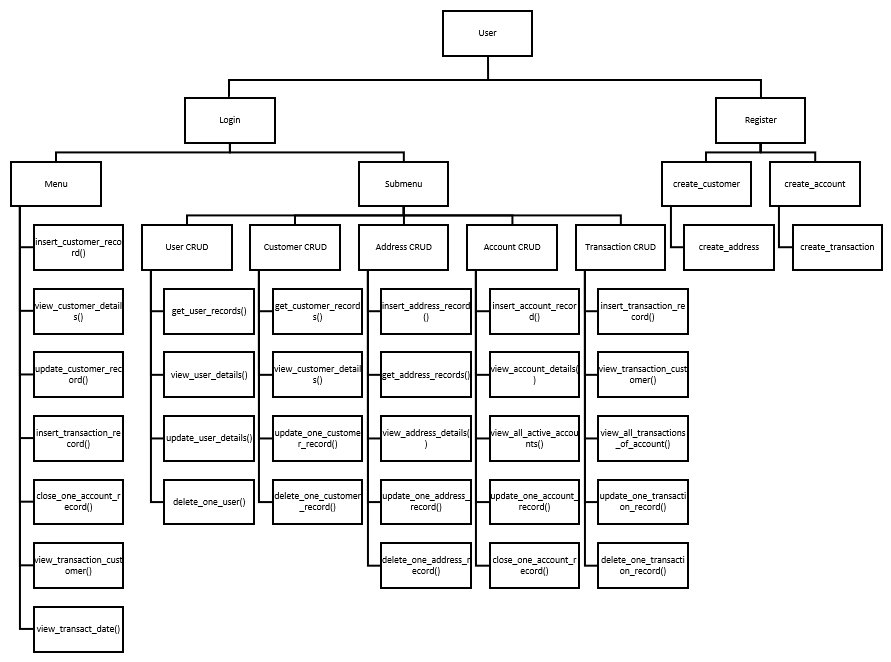


Figure Hierarchy diagram

## PSEUDOCODE

Main

Call create user table function

Call create customer table function

Call create account table function

Call create address table function

Call create transact table function

Add loop and add main menu & sub menu inside

Add login/register function in main menu

Register and input information

Login and choose main menu or sub menu

Add Continue\_check to continue

**Step1** Create Database function

Add python command to create .db file

**Step2** Create Tables function

Add all tables with primary and foreign keys and cur.execute

**Step3** Create Menu function

Add main menu for basic requirements and sub menu for CRUDs

Call each function to the relevant if satement

Add correct input handling and user friendly messages in each step

**Step4** Create USER table CRUD functions

Add Create user function with secure password

Add values to the user table

Add Read all users function and print

Read all values from user table

Add Read one user function and print output

Read one specific value in user table

Add Update user function and print user record

Change one user record in user table

Add Delete user function and print user record

Delete one user record from user table

**Step5** Create ADDRESS table CRUD functions

Add Create customer address function

Add values to the address table

Add Read customer address function

SELECT the customer and JOIN the address to read all addresses

Add Read one address record function

SELECT the customer and JOIN the address to read a specific address

Add Update customer address function

UPDATE the address and set the new values to update a specific address record

Add Delete customer address function

DELETE the value the value from the address table

**Step6** Create ACCOUNT table CRUD functions

Add Create account function

Add values to the account table

Add Read one account function

SELECT one specific value in user table

Add Read all open account functions

SELECT and print all account with ACTIVE status

Add Update account function

UPDATE details and SET new values

Add Delete account function

UPDATE status to INACTIVE and add date

**Step7** Create TRANSACT table CRUD functions

Add Create Transaction function

Add input values to create the transaction

Add Read one account transaction function

SELECT account to read one transaction from

Add Read all transactions function

SELECT all transactions from account table

Add Read all transaction by date function

SELECT all transactions from TRANSACT table when there is date range

Add Update transaction function

DELETE transaction and insert new transaction to update details

Add Delete transaction function

UPDATE transaction type to debit and lower balance to 0

**Step8** Create Login and Register functions

Input username and password

Hide password for security

Add password authentication function

Compare input password to password in user table

Add hash password function

Store hash password after imporing bcrypt library

# **SOFTWARE ARTEFACT**

## DEVELOPMENT OF A PROTOTYPE APPLICATION

### ISSUES FACED AND DEALT WITH

All of the issues bellow where faced while developing the system, no screenshots taken during the process of the errors.

Login / Authentication error (crashing after false login details) FIXED

Error handling in menus (wrong input messages) FIXED

Error handling in submenu (wrong input messages) FIXED

Transaction queries (INNER JOIN usage) FIXED

Address queries (INNER JOIN usage) FIXED

Syntax error while using colour extensions (print commands) FIXED

No such column error in queries FIXED

Rowid usage where not needed in queries FIXED

Replacement index 1 out of range for positional args tuple error in queries FIXED

Syntax error no such column in SQL FIXED

Syntax error “,” missing between SELECT queries FIXED

Output information from database aligned NOT FIXED

### CREATION OF DATABASE

import sqlite3

import bcrypt

import getpass

from prettytable import prettytable

conn=sqlite3.connect('swe4207.db')

cur=conn.cursor()

def check\_table\_exist(table\_name:str):

    cur.execute (""" SELECT count(name) FROM sqlite\_master WHERE type='table'

    AND name= ?;""",(table\_name,))

    count\_of\_tables=cur.fetchone()[0]

    conn.commit()

    if count\_of\_tables==1:

        return True

    else:

        return False

After database is created

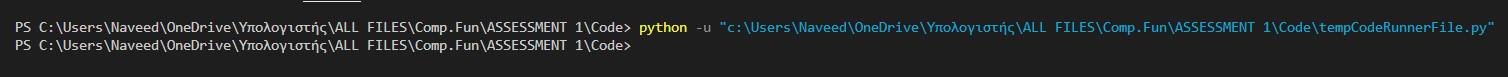


Figure create .db

The screenshot above shows the path in which the .db file is stored and will be used every time its launched through vscode.

### **MENU FUNCTION**

The code below shows the *menu* function with the main menu that consists of the requirements, the sub menu consists of the sub options such as inserting or deleting customers and lastly it includes the initial menu that allows the user to have access to the other 2 menus through the options provided.

The section also includes multiple error handling variables and techniques to help improve the execution of the program, along with several *if* statements that help form the function.

def menu\_options():

    #MENU OPTIONS

    continue\_choice = "Y"

    while True:

        print("""

        1. Enter Main Menu

        2. Enter Sub Menu

        3. Exit

        """)

        menu\_choice = input("Please enter option between 1 and 3: ")

        menu\_options1 = ["1","2","3"]

        while menu\_choice in menu\_options1:

            if menu\_choice not in menu\_options1:

                menu\_choice = ("Error! You must chooce an option between 1 and 3: ")

            elif menu\_choice == "1":

                print("""

0.Create customer details.

1. View customers details.

2. Update customers details.

3. Record new transactions relating to individual customers.

4. Mark accounts as closed using status and date fields.

5. View a customers transactions for a specific date range.

6. View all transactions for a specific date range.

7. View sub menu.

8. Exit

""")

                mainmenu\_choice = input("Please enter option between 0 and 7: ")

                mainmenu\_options = ["0","1","2","3","4","5","6","7"]

                while mainmenu\_choice in mainmenu\_options:

                    if mainmenu\_choice not in mainmenu\_options:

                        mainmenu\_choice = input("Error! You must choose an option between 0 and 7: ")

                    elif mainmenu\_choice == "0":

                        insert\_customer\_record()

                    elif mainmenu\_choice == "1":

                        view\_customer\_details() #done

                    elif mainmenu\_choice == "2":

                        update\_one\_customer\_record() #done

                    elif mainmenu\_choice== "3":

                        insert\_transaction\_record() #done

                    elif mainmenu\_choice=="4":

                        close\_one\_account\_record()

                    elif mainmenu\_choice=="5":

                        view\_transaction\_customer()

                    elif mainmenu\_choice == "6":

                        view\_transact\_date()

                    elif mainmenu\_choice == "7":

                        menu\_choice = "2"

                    else:

                        print("Good bye")

                        conn.close()

                        exit()

                    if mainmenu\_choice != "7":

                        continue\_choice = input("Would you like to exit (Y/N): ").upper()

                    if continue\_choice == "Y":

                        break

                    else:

                        mainmenu\_choice = input("Please enter option between 0 and 7: ")

            elif menu\_choice == "2":

                print("""

\*\*\*\*\*\*USER SETTINGS\*\*\*\*\*\*

1. Read all users

2. Read one user

3. Update user details

4. Delete user

\*\*\*\*\*\*CUSTOMER SETTINGS\*\*\*\*\*\*

5. Read Customers

6. Read one Customer

7. Update one Customer

8. Delete one Cucstomer

\*\*\*\*\*\*ADDRESS SETTINGS\*\*\*\*\*\*

9. Read Customer Address

10. Read one Customer Address

11. Update Address

12. Delete Address

\*\*\*\*\*\*ACCOUNT SETTINGS\*\*\*\*\*\*

13. Read one Account

14. Read all open Active Accounts

15. Update Account details

\*\*\*\*\*\*TRANSACTION SETTINGS\*\*\*\*\*\*

16. Read one transaction

17. Read all transactions from an account

18. Update transaction

19. Delete transaction

20. View main menu

""")

                submenu\_choice = input("Please enter option between 1 and 20: ")

                submenu\_options2 = ["1","2","3","4","5","6","7","8","9","10","11","12","13","14","15","16","17","18","19"]

                while submenu\_choice in submenu\_options2:

                    if submenu\_choice not in submenu\_options2:

                        submenu\_choice = input("Error! You must choose an option between 1 and 20: ")

                    if submenu\_choice == "1":

                        get\_user\_records()

                    elif submenu\_choice == "2":

                        view\_user\_details()

                    elif submenu\_choice == "3":

                        update\_user\_details()

                    elif submenu\_choice == "4":

                        delete\_one\_user()

                    elif submenu\_choice == "5":

                        get\_customer\_records()

                    elif submenu\_choice == "6":

                        view\_customer\_details()

                    elif submenu\_choice == "7":

                        update\_one\_customer\_record()

                    elif submenu\_choice =="8":

                        delete\_one\_customer\_record()

                    elif submenu\_choice == "9":

                        get\_address\_records()

                    elif submenu\_choice == "10":

                        view\_address\_details()

                    elif submenu\_choice == "11":

                        update\_one\_address\_record()

                    elif submenu\_choice == "12":

                        delete\_one\_address\_record()

                    elif submenu\_choice == "13":

                        view\_account\_details()

                    elif submenu\_choice == "14":

                        view\_all\_active\_accounts()

                    elif submenu\_choice == "15":

                        update\_one\_account\_record()

                    elif submenu\_choice == "16":

                        view\_transaction\_details()

                    elif submenu\_choice == "17":

                        view\_transact\_of\_account()

                    elif submenu\_choice == "18":

                        update\_one\_transaction\_record()

                    elif submenu\_choice == "19":

                        delete\_one\_transaction\_record()

                    elif submenu\_choice == "20":

                        menu\_choice = "1"

                    else:

                        continue\_choice == 'Y'

                        continue\_choice = input("Would you like to exit (Y/N): ").upper()

                    if continue\_choice == "Y":

                        break

            elif menu\_choice == "3":

                 continue\_choice = input("Would you like to exit (Y/N): ").upper()

                 if continue\_choice == "Y":

                    print("Good bye")

                    exit()

            else:

                print("""

Please try again.

1. Enter Main Menu

2. Enter Sub Menu

3. Exit""")

            menu\_choice = input("Please enter option between 1 and 3: ")

        conn.close()

create\_user\_table()

create\_customer\_table()

create\_address\_table()

create\_account\_table()

create\_tansaction\_table()

while True:

    option = input("""

1.LOGIN.

2.REGISTER.

Selected option """)

    options = ["1", "2"]

    while option in options:

        if option not in options:

            print("Please try again by choosing 1 to LOGIN or 2 to REGISTER")

            option = input("""

1.LOGIN.

2.REGISTER.

Please select option 1 or 2: """)

        elif option == "1":

            check\_password()

            menu\_options()

        elif option == "2":

            create\_user\_account()

            newclient()

            menu\_options()

        else:

            exit()

    option = input("""

PLEASE TRY AGAIN

1.LOGIN.

2.REGISTER.

Selected option """)

**Error Handling**

Wrong inputs unable to break the program.

Text

Description automatically generated

Figure Error Handling

Text

Description automatically generated

Figure Error handling in Menu

### **CREATION OF TABLES**

Creation of tables so values that enter later on with functions can be stored and modified.

Tables 1 and 2.

Text

Description automatically generated

Figure Table1 & Table 2

Table 3 and 4.

Text

Description automatically generated

Figure Table 3 & Table 4

Table 5

Text

Description automatically generated

Figure Table 5

### **USER CRUD FUNCTIONS**

Table 1

Insert use data table receives data and stored in the user table.

def insert\_user\_data(username:str, password:str):

    #accepts user data as args and inserts into user table

    cur = conn.cursor()

    cur.execute("INSERT INTO user VALUES (?, ?)" , (username, password))

    conn.commit()

    return

Table 2

Update user details function allows modification of values from the user table.

def update\_user\_details():

    #Update user details in user table

    password = ("Enter new password: ")

    username = ("Enter username of account: ")

    cur.execute("UPDATE user SET password = ? WHERE rowid = ?",(password, username,))

Table 3

Insert user record function adds more users to the table by asking for values.

def insert\_user\_record():

    #insert record in table

    #collects data entered by user and sends to insert\_user\_data function

    username = input("Please enter username: ")

    hash\_password.password = input("Enter password: ")

    # send values as args to insert\_user\_data()

    insert\_customer\_data(username, hash\_password.password)

    return

Table 4

View user details function prints values of a specific user based on the username.

def view\_user\_details():

#output one user record

    cur = conn.cursor()

    username = input("Enter the username of user: ")

    print ("{:<20}".format("username"))

    for row in cur.execute("SELECT \* FROM user WHERE username = ?", (username,)):

        print("{:<20}".format(row[0]))

    return

Table 5

Get user records function prints all records that are inside the user table.

def get\_user\_records():

    #output all customer records

    cur = conn.cursor()

    print ("{:<3} {:<20} {:<16}".format("ID","username","password")

    )

    for row in cur.execute("SELECT rowid, \* FROM user"):

        print("{:<3} {:<20}".format(row[0], row[1]))

    return

Table 6

Delete one user function prints the full list of users that exist in the user table and then gives the option to delete a user and prints changes afterwards.

def delete\_one\_user():

    #delete one user from database

    cur = conn.cursor()

    get\_user\_records() # shows all records before delete process

    username = input("Enter user to delete: ")

    cur.execute("DELETE from user WHERE username = ?", (username,))

    conn.commit()

    get\_user\_records() # shows all records after delete process

### **CUSTOMER CRUD FUNCTIONS**

Table 1

Insert customer data function accepts data and stores it inside the customer table so it can later on be modified or used.

def insert\_customer\_data(customerid:int,forename:str, surname:str, dob:str):

    #accepts customer data as args and inserts into customer table

    cur = conn.cursor()

    cur.execute("INSERT or IGNORE INTO customer VALUES (?, ?, ?, ?)" , (customerid,forename,surname,dob))

    conn.commit()

    return

Table 2

Get customer records function prints all the records that are stored inside the customer table.

def get\_customer\_records():

    #output all customer records

    cur = conn.cursor()

    print ("{:<10} {:<12} {:<12} {:<10}".format("customerid","forename","surname", "dob") )

    for row in cur.execute("SELECT \* FROM customer"):

        print("{:<10} {:<12} {:<12} {:<10}"

        .format(row[0],row[1],row[2],row[3])

        )

    return

Table 3

View customer details function prints a customer record after the user enters the inputs needed.

def view\_customer\_details():

#output one customer record

    cur = conn.cursor()

    customerid = input("Enter the ID of customer: ")

    print ("{:<10} {:<12} {:<12} {:<10}".format("customerid","forename","surname", "dob"))

    for row in cur.execute("SELECT \* FROM customer WHERE customerid = ?", customerid):

        print("{:<10} {:<12} {:<12} {:<10}"

        .format(row[0],row[1],row[2],row[3])

        )

    return

Table 4

Update one customer record function allows the user to update and modify information from the customer table and then print the records so the outcome can be visible.

def update\_one\_customer\_record():

    #Change customer record

    get\_customer\_records() # shows all customers before update

    customerid = input("Enter customer's id to update details of: ")

    forename = input("Enter updated forename: ")

    surname = input("Enter updated surname: ")

    dob = input("Enter updated dob: ")

    cur.execute("UPDATE customer SET forename = ?, surname = ?, dob = ? WHERE customerid = ?", (forename,surname, dob, customerid))

    get\_customer\_records() # shows all records after update

Table 5

Insert\_customer\_record function allows the user to enter more customers to the customer table.

def insert\_customer\_record():

    #insert record in table

    customerid = input("Please enter customer id: ")

    forename = input("Please enter forename: ")

    surname = input("Please enter surname: ")

    dob = input ("Please enter date of birth: ")

    # send values as args to insert\_student\_data()

    insert\_customer\_data(customerid,forename,surname,dob)

    return

Table 6

Delete\_one\_customer\_record function allows user to delete one customer from the customer table.

def delete\_one\_customer\_record():

    #delete one customer from database

    cur = conn.cursor()

    get\_customer\_records() # shows all records before delete process

    customerid = input("Enter customer to delete: ")

    cur.execute("DELETE from customer WHERE customerid = ?", (customerid,))

    conn.commit()

    get\_customer\_records() # shows all records after delete process

### **ACCOUNT CRUD FUNCTIONS**

Table 1

Insert\_account\_data function allows the account values to be stored into the table so the account can be created.

def insert\_account\_data(accountid:int, balance:int, opendate:str, closedate:str, status:str, customerid:int):

    #accepts account data as args and inserts into account table

    cur = conn.cursor()

    cur.execute("INSERT INTO account VALUES (?, ?, ?, ?, ?, ?)" , (accountid, balance, opendate, closedate, status, customerid))

    conn.commit()

    return

Table 2

Get\_account\_records function prints all the records stored inside the account table.

def get\_account\_records():

    #output all account records

    cur = conn.cursor()

    print ("{:<10} {:<12} {:<20} {:<20} {:<10} {:<10}"

    .format("accountid","balance","opendate", "closedate", "status", "customerid")

    )

    for row in cur.execute("SELECT \* FROM account"):

        print("{:<10} {:<12} {:<12} {:<10} {:<10} {:<10}"

        .format(row[0],row[1],row[2],row[3],row[4],row[5])

        )

    return

Table 4

View\_account\_details function prints all the records stored in the account table and outputs all the values stored in a specific account.

def view\_account\_details():

#output all account records

    cur = conn.cursor()

    accountid = input("Enter the ID of account: ")

    print ("{:<10} {:<12} {:<12} {:<10} {:<10} {:<10}"

    .format("accountid","balance","opendate", "closedate", "status", "customerid"))

    for row in cur.execute("SELECT \* FROM account WHERE accountid = ?", accountid):

        print("{:<10} {:<12} {:<12} {:<10} {:<10} {:<10}"

        .format(row[0],row[1],row[2],row[3],row[4],row[5]))

    return

Table 5

View\_all\_active\_accounts function prints out all the ACTIVE accounts in the account table. The ACTIVE accounts are the ones that are open and running.

def view\_all\_active\_accounts():

    #output all active accounts

    cur = conn.cursor()

    status = "ACTIVE".upper()

    print ("{:<10} {:<12} {:<12} {:<10} {:<10} {:<10}".format("accountid","balance","opendate", "closedate", "status", "customerid"))

    for row in cur.execute("SELECT \* FROM account WHERE status = 'ACTIVE'"):

        print("{:<10} {:<12} {:<12} {:<10} {:<10} {:<10}"

        .format(row[0],row[1],row[2],row[3],row[4],row[5]))

    return

Table 5

Insert\_account\_record function allows creation of a new account and stores it as a record in the account table.

def insert\_account\_record():

    accountid = input("Please enter account id: ")

    balance = input("Please enter balance: ")

    opendate = input("Please enter open date: ")

    closedate = input ("Please enter close date: ")

    status= input ("Please enter status: ")

    customerid = input("Please enter customer id: ")

    # send values as args to insert\_account\_data()

    insert\_account\_data(accountid, balance, opendate, closedate, status, customerid)

Table 6

Update\_one\_account\_record function allows modification of values in a specific account and updates the changes after the user is done changing a record and stores everything into the account table.

def update\_one\_account\_record():

    #Change account record

    get\_account\_records() # shows all accounts before update

    accountid = input("Enter account id for update: ")

    balance = input("Enter balance for update: ")

    opendate = input("Enter opendate for update: ")

    closedate = input("Enter closedate for update: ")

    status = input("Enter status for update: ")

    customerid= input("Enter custome id for udpate: ")

    cur.execute("""UPDATE OR REPLACE account

    SET accountid = ?, balance = ?, opendate = ?, closedate = ?, status = ?, customerid = ?""",(accountid, balance, opendate, closedate, status, customerid))

    get\_customer\_records() # shows all records after update

Table 7

Close\_one\_account\_record function closed an account by updating the account status to ‘INACTIVE’ instead of completely deleting the account. (keeping logs/history purposes)

def close\_one\_account\_record():

    # Close one account from database

    cur = conn.cursor()

    get\_account\_records() # shows all records before starting process

    accountid = input("Enter account id to set inactive: ")

    closedate = input("Enter account close date: ")

    cur.execute("""UPDATE account SET status = 'INACTIVE', closedate = ? WHERE accountid = ?""",(closedate, accountid))

    conn.commit()

### **TRANSACTION CRUD FUNCTIONS**

Table 1

Insert\_transaction\_data function stores data into the transact table.

def insert\_transaction\_data(transactionid:int, amount:int, type:str, date:str, accountid:int):

    #accepts transaction data as args and inserts into transaction table

    cur = conn.cursor()

    cur.execute("INSERT INTO transact VALUES (?, ?, ?, ?, ?)", (transactionid, amount, type, date, accountid,))

    conn.commit()

    return

Table 2

Get\_transaction\_records function prints all records stored in transact table.

def get\_transaction\_records():

    #output all transaction records

    cur = conn.cursor()

    print ("{:<2} {:<10} {:<5} {:<10} {:<10} {:<10}".format("ID","transactionid, amount, type, date, accountid"))

    for row in cur.execute("SELECT \* FROM transact"):

        print('{:<10} {:<5} {:<10} {:<10} {:<10}'

        .format(row[0],row[1],row[2],row[3],row[4])

        )

    return

Table 3

View\_transact\_of\_account function prints all transaction made by a specific account.

def view\_transact\_of\_account(): #new

    #output transactions of an account

    cur = conn.cursor()

    accountid = input("Enter account ID to print transactions: ")

    #print ("{:<2} {:<5} {:<10} {:<6} {:<2}".format("accountid, amount, type, date, transactionid"))

    print ("{:<10}".format("accountid"),end=" ")

    print ("{:<5}".format("amount"),end=" ")

    print ("{:<10}".format("type"),end=" ")

    print ("{:<6}".format("date"),end=" ")

    print ("{:<10}".format("transactionid"))

    for row in cur.execute("""SELECT t.transactid, t.amount, t.date, t.accountid

                        FROM transact AS t INNER JOIN account AS a ON t.accountid = a.accountid

                        WHERE a.status = 'ACTIVE' AND t.accountid = ?""",(accountid,)):

        print("{:<10} {:<5} {:<10} {:<10} {:<10}".format(row[0],row[1],row[2],row[3],row[4]))

    return

Table 4

View\_transact\_date function prints transactions based on date range entered by user input.

def view\_transact\_date(): #new

    #output transactions based on date

    cur = conn.cursor()

    accountid = input("Enter account ID to print transactions based on a date: ")

    date1 = input("Enter date: ")

    date2 = input("Enter second date: ")

    #print ("{:<2} {:<5} {:<10} {:<6} {:<2}".format("accountid, amount, type, date, transactionid"))

    print ("{:<10}".format("accountid"),end=" ")

    print ("{:<5}".format("amount"),end=" ")

    print ("{:<10}".format("type"),end=" ")

    print ("{:<6}".format("date"),end=" ")

    print ("{:<10}".format("transactionid"))

    for row in cur.execute("SELECT \* FROM transact WHERE accountid = ? AND date BETWEEN ? AND ?", (accountid, date1, date2)):

        print("{:<10} {:<5} {:<10} {:<10} {:<10}".format(row[0],row[1],row[2],row[3],row[4]))

    return

Table 5

View\_transaction\_customer function prints all trnasactions made by a specific customer based on the date range.

def view\_transaction\_customer():

#output all transact records

    cur = conn.cursor()

    customerid = input("Enter the customer ID to view transactions: ")

    date1 = input("Enter date: ")

    date2 = input("Enter second date: ")

    #print ("{:<2} {:<5} {:<10} {:<6} {:<2}".format("transactionid, amount, type, date, accountid"))

    print ("{:<10}".format("accountid"),end=" ")

    print ("{:<5}".format("amount"),end=" ")

    print ("{:<10}".format("type"),end=" ")

    print ("{:<10}".format("date"),end=" ")

    print ("{:<10}".format("accountid"),end=" ")

    print ("{:<10}".format("customerid"))

    # for row in cur.execute("SELECT \* FROM account as a  WHERE transactid = ? date BETWEEN ? AND ?", transactionid, date1, date2):

    #     print("{:<10} {:<5} {:<10} {:<6} {:<10}"

    #     .format(row[0],row[1],row[2],row[3],row[4])

    #     )

    # return

    for row in cur.execute("SELECT t.transactid, t.amount, t.type, t.date, t.accountid, a.customerid FROM transact AS t INNER JOIN account AS a ON t.accountid = a.accountid WHERE a.customerid = ?", (customerid,)):

        print("{:<10} {:<5} {:<10} {:<10} {:<10} {:<10}"

        .format(row[0],row[1],row[2],row[3],row[4], row[5])

        )

    return

Table 6

Insert\_transaction\_record function allows creation of new transactions and stores all values and data into insert\_transaction\_data table.

def insert\_transaction\_record():

    #print("""collects data entered by user and sends to insert\_transaction\_data function""")

    transactionid = input("Please enter trasnaction id: ")

    amount = input("Please enter amount: ")

    type = input("Please enter type: ")

    date = input ("Please enter date: ")

    accountid = input("Please enter account id: ")

    # send values as args to insert\_transaction\_data()

    insert\_transaction\_data(transactionid, amount, type, date, accountid)

Table 6

Update\_one\_transaction\_record function modifies and changes data in a specific transaction and prints results before and after the update.

def update\_one\_transaction\_record():

    #Change transaction record

    get\_transaction\_records() # shows all transactions before update

    transactionid = input("Enter transaction id for update: ")

    amount = input("Enter amount for update: ")

    type = input("Edit type of transaction ")

    date = input("Enter transaction date for update: ")

    accountid = input("Enter account id for update: ")

    cur.execute("""UPDATE transact

    SET transactid = ?, amount = ?, type = ?, date = ?, accountid = ?

    WHERE rowid = ?""",(transactionid, amount, type, date, accountid))

    get\_transaction\_records() # shows all records after update

Table 7

Delete\_one\_transaction\_record function deletes a transaction from the transact table and prints the changes by shows all records.

def delete\_one\_transaction\_record():

    #delete one transaction from database

    cur = conn.cursor()

    get\_transaction\_records() # shows all records before delete process

    transactionid = input("Enter transaction to delete: ")

    cur.execute("DELETE from transact WHERE rowid=?", transactionid)

    conn.commit()

    get\_transaction\_records() # shows all records after delete process

### **ADDRESS CRUD FUNCTIONS**

Table 1

Insert\_address\_data function stores all data inside the address table.

def insert\_address\_data(addressid:int, streetnumber:str, firstline:str, postcode:str, region:str, country:str, customerid:int):

    #accepts address data as args and inserts into transaction table

    cur = conn.cursor()

    cur.execute("INSERT INTO address VALUES (?, ?, ?, ?, ?, ?, ?)" , (addressid, streetnumber, firstline, postcode, region, country, customerid))

    conn.commit()

    return

Table 2

Get\_address\_records function prints all records stored in address table.

def get\_address\_records():

    #output all address records

    cur = conn.cursor()

    #print ("{:<2} {:<3} {:<20} {:<6} {:<15} {:<12} {:<2}".format("addressid, streetnumber, firstline, postcode, region, country, customerid"))

    print ("{:<10}".format("addressid"),end = " ")

    print ("{:<3}".format("streetnumber"),end = " ")

    print ("{:<20}".format("firstline"),end = " ")

    print ("{:<6}".format("postcode"),end = " ")

    print ("{:<15}".format("region"),end = " ")

    print ("{:<12}".format("country"),end = " ")

    print ("{:<10}".format("customerid"))

    for row in cur.execute("SELECT \* FROM address"):

        print("{:<10} {:<3} {:<20} {:<6} {:<15} {:<12} {:<10}"

        .format(row[0],row[1],row[2],row[3],row[4],row[5],row[6])

        )

    return

Table 4

View\_address\_details function prints all records for a specific address record stored in the address table.

def view\_address\_details():

#output one address record

    cur = conn.cursor()

    addressid = input("Enter the ID of address: ")

    #print ("{:<2} {:<3} {:<20} {:<6} {:<15} {:<12} {:<2}".format("addressid, streetnumber, firstline, postcode, region, country, customerid"))

    print ("{:<10}".format("addressid"),end = " ")

    print ("{:<3}".format("streetnumber"),end = " ")

    print ("{:<20}".format("firstline"),end = " ")

    print ("{:<6}".format("postcode"),end = " ")

    print ("{:<15}".format("region"),end = " ")

    print ("{:<12}".format("country"),end = " ")

    print ("{:<10}".format("customerid"))

    for row in cur.execute("SELECT \*  FROM address WHERE addressid = ?", (addressid,)):

        print("{:<10} {:<3} {:<20} {:<6} {:<15} {:<12} {:<10}"

        .format(row[0],row[1],row[2],row[3],row[4],row[5],row[6])

        )

    return

Table 5

Insert\_address\_record function adds new addresses to the address table, adds a new address record.

def insert\_address\_record():

    addressid = input("Please enter address id: ")

    streetnumber = input("Please enter streetnumber: ")

    firstline = input("Please enter firstline: ")

    postcode = input ("Please enter postcode: ")

    region = input ("Please enter region: ")

    country = input ("Please enter country: ")

    customerid = input ("Please enter customer id")

    # send values as args to insert\_address\_data()

    insert\_address\_data(addressid, streetnumber, firstline, postcode, region, country, customerid)

Table 6

Update\_one\_address\_record function modifies values about a specific address record stored inside the address table and prints the records to show what is updates.

def update\_one\_address\_record():

    #Change address record

    get\_address\_records() # shows all addresses before update

    addressid = input("Enter address id for update: ")

    streetnumber = input("Enter streetnumber for update: ")

    firstline = input("Enter firstline for update: ")

    postcode = input("Enter postcode for update: ")

    region = input("Enter region for update: ")

    country = input("Enter country for update: ")

    customerid = input("Enter customer id for update: ")

    cur.execute("""UPDATE OR REPLACE address

    SET addressid = ?, streetnumber = ?, firstline = ?, postcode = ?, region = ?, country = ?, customerid = ?""",(addressid, streetnumber, firstline, postcode, region, country, customerid,))

    get\_address\_records() # shows all records after update

Table 7

Delete\_one\_address\_record function erases one address record from the address table and prints all address records.

def delete\_one\_address\_record():

    #delete one address from database

    cur = conn.cursor()

    get\_address\_records() # shows all records before delete process

    addressid = input("Enter address to delete: ")

    cur.execute("DELETE from address WHERE addressid = ?", (addressid,))

    conn.commit()

    get\_address\_records() # shows all records after delete process

*NOTE: The full updated version of the above code located in py file along with the word submission*

# **TESTING OF APPLICATION**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| DATE | TEST DESCRIPTION | TEST PASS? | EXPECTED RESULT | ACTUAL RESULT | ANALYSIS | SCREENSHOTS |
| 19/01/2023 | Component / Unit Test | PASS | Successful testing of the most detailed parts of the application | Very Accurate | In depth usage of every function and module of the system separately, different inputs and outputs tested | <https://prnt.sc/Os-GqnoBcjaF>  <https://prnt.sc/NQG02uHJGb-9> |
| 19/01/2023 | Integration Test | PASS | Successful application of all units and modules of the system combined. | Accurate | Collaboration of all functions and how they function. | <https://prnt.sc/ob8YLtTLMDCV>  <https://prnt.sc/_IPfdH55tnCA> |
| 19/01/2023 | System Test | PASS | Successful application of the requirements in the system that need to be met. | Very Accurate | All requirements of the banking application met. | <https://prnt.sc/OD4T0ODCB4s0>  <https://prnt.sc/CR2ncO-WWNM_>  <https://prnt.sc/S1mao2byNFIG>  <https://prnt.sc/zxIvDjGN7guJ>  <https://prnt.sc/abXQEdAHaWWX>  <https://prnt.sc/T23dzi6FkwQQ> |
| 19/01/2023 | Acceptance Test | PASS | Quality assurance of the system that determines whether the system is usable by the requesting end. | Accurate | System properly functional and effective. In need of slight improvements in details. |  |

Figure Testing

# BIBLIOGRAPHY

*Seidl, M., Scholz, M., Huemer, C. & Kappel, G. 2015, UML @ classroom: an introduction to object-oriented modeling, Springer, Berlin.*

*Vaze, Smita, and Subhalaxmi Joshi. Computer Fundamentals and Rdbms, Global Media, 2009. ProQuest Ebook Central,* [*https://ebookcentral.proquest.com/lib/bolton/detail.action?docID=3011260*](https://ebookcentral.proquest.com/lib/bolton/detail.action?docID=3011260)*.*

*Hoffer, Jeff, et al. Modern Database Management, Global Edition, Pearson Education Limited, 2016. ProQuest Ebook Central,* [*https://ebookcentral.proquest.com/lib/bolton/detail.action?docID=5185641*](https://ebookcentral.proquest.com/lib/bolton/detail.action?docID=5185641)

*Kroenke, David. Database Concepts, Global Edition, Pearson Education Limited, 2015. ProQuest Ebook Central,* [*https://ebookcentral.proquest.com/lib/bolton/detail.action?docID=5176542*](https://ebookcentral.proquest.com/lib/bolton/detail.action?docID=5176542)*.*

*Harrington, Jan L.. Relational Database Design and Implementation : Clearly Explained, Elsevier Science & Technology, 2009. ProQuest Ebook Central,* [*https://ebookcentral.proquest.com/lib/bolton/detail.action?docID=535142*](https://ebookcentral.proquest.com/lib/bolton/detail.action?docID=535142)*.*

*Pathak, Nirupma. Database Management System, Global Media, 2007. ProQuest Ebook Central,* [*https://ebookcentral.proquest.com/lib/bolton/detail.action?docID=3011378*](https://ebookcentral.proquest.com/lib/bolton/detail.action?docID=3011378)*.*

*Teorey, Toby J., et al. Database Modeling and Design : Logical Design, Elsevier Science & Technology, 2011. ProQuest Ebook Central,* [*https://ebookcentral.proquest.com/lib/bolton/detail.action?docID=667713*](https://ebookcentral.proquest.com/lib/bolton/detail.action?docID=667713)*.*

*Sommerville, I 2016, Software Engineering, Global Edition, Pearson Education Limited, Harlow. Available from: ProQuest Ebook Central. [16 March 2020].*

*Stephens, Rod. Beginning Software Engineering, John Wiley & Sons, Incorporated, 2015. ProQuest Ebook Central,* [*https://ebookcentral.proquest.com/lib/bolton/detail.action?docID=1895174*](https://ebookcentral.proquest.com/lib/bolton/detail.action?docID=1895174)*.*

*Martin, Robert C.. Agile Software Development, Principles, Patterns, and Practices: Pearson New International Edition, Pearson Education Limited, 2013. ProQuest Ebook Central,* [*https://ebookcentral.proquest.com/lib/bolton/detail.action?docID=5174949*](https://ebookcentral.proquest.com/lib/bolton/detail.action?docID=5174949)*.*

*Homès, Bernard, and Bernard Homès. Fundamentals of Software Testing, John Wiley & Sons, Incorporated, 2012. ProQuest Ebook Central,* [*https://ebookcentral.proquest.com/lib/bolton/detail.action?docID=1120766*](https://ebookcentral.proquest.com/lib/bolton/detail.action?docID=1120766)*.*

*Sale, David. Testing Python : Applying Unit Testing, TDD, BDD and Acceptance Testing, John Wiley & Sons, Incorporated, 2014. ProQuest Ebook Central,* [*https://ebookcentral.proquest.com/lib/bolton/detail.action?docID=1729555*](https://ebookcentral.proquest.com/lib/bolton/detail.action?docID=1729555)

*Watkins, John. Testing IT : An Off-the-Shelf Software Testing Process, Cambridge University Press, 2001. ProQuest Ebook Central,* [*https://ebookcentral.proquest.com/lib/bolton/detail.action?docID=164738*](https://ebookcentral.proquest.com/lib/bolton/detail.action?docID=164738)*.*

*Chemuturi, Murali. Mastering Software Quality Assurance : Best Practices, Tools and Techniques for Software Developers, J. Ross Publishing Inc., 2010. ProQuest Ebook Central,* [*https://ebookcentral.proquest.com/lib/bolton/detail.action?docID=3319498*](https://ebookcentral.proquest.com/lib/bolton/detail.action?docID=3319498)*.*

*Karac, Itir, and Burak Turhan. "What do we (really) Know about Test-Driven Development?" IEEE Software, vol. 35, no. 4, 2018, pp. 81-85. ProQuest, https://search-proquest-com.ezproxy.bolton.ac.uk/scholarly-journals/what-do-we-really-know-about-test-driven/docview/2066517777/se-2?accountid=9653, doi:http://dx.doi.org.ezproxy.bolton.ac.uk/10.1109/MS.2018.2801554.*