VYDEHI SCHOOL OF EXCELLENCE

Affiliated to CBSE, Delhi
Vydehi campus, Whitefield, Bengaluru
Karnataka

COMPUTER SCIENCE (083)

Project on:

Banking System implemented using Python and MySQL

Year: 2024-25

Submitted to	Submitted by–Nandan Goyal
Ms. Ranjeeta Shrivastava	Class – XII **A**

VYDEHI SCHOOL OF EXCELLENCE

DEPARTMENT OF COMPUTER SCIENCE

CERTIFICATE

This is to certify that **NANDAN GOYAL** of class **XII-A** has successfully completed the project under the guidance of **MS. RANJEETA SHRIVASTAVA** during the academic year **2024-25** in partial fulfilment of **COMPUTER SCIENCE** practical examination conducted by AISSCE, New Delhi**.**

Signature of the external examiner ***Signature of the internal examiner***

External examiner no: ***Signature of the Principal***

ACKNOWLEDGMENT

I want to express my most sincere thanks and gratitude to Ms. Ranjeeta, my computer science teacher, who's valuable feedback and guidance in this project was a major contribution. Her constant motivation and advice were the foundation to this project.

Next, I'd like to thank Ms. Chhavi Garg, our principal ma'am, for extending support and coordination wherever possible which aided in the productive and successful completion of this project.

Finally, I'd like to thank everyone who helped me directly, or indirectly in the completion of this project.

INDEX

S.no	Title	Page no.
1	Introduction	1
2	Why Python?	2
3	System Specs	3
4	Aim	4
5	Some Background	5
6	Program Code	7
7	Flowchart	21
8	Sample Output	22
9	Bibliography	30

INTRODUCTION

This project is a terminal-based net-banking app, in which users can:

1. Create and log into a password protected account
2. Deposit money
3. Make transactions with other users
4. Create fixed deposits and gain interest

All user-data is stored in and retrieved from a MySQL database. For the application itself, Python is used.

One of the main features of this project in terms of development is its state machine architecture, which is discussed in detail later in this document.

WHY PYTHON?

- **Cross-platform Language**: Python can run equally well on variety of platforms – Windows, Linus/UNIX, smartphones, etc.

- **Simple and expressive syntax:** Python has a simple syntax similar to the English language. It is thus very expressive with fewer lines of code and simplicity compared to other popular languages like C++, Java etc.

- **Quick prototyping:** Python runs on an interpreter system, so the code can be executed as soon as it is written. This, along with its simplicity, means that prototyping can be very quick.

- **Multi-paradigm:** Python can be written in a procedural way, an object-oriented way or a functional way.

SYSTEM SPECS

Operating System	Windows 10
Processor	Intel Core i3 7th gen @ 2.30 Ghz
RAM	12 GB
Hard disk	SSD 233 GB, HDD 932 GB

AIM

To create a net-banking client application with terminal-based UI.

Some Background

As mentioned before, the state machine architecture of this application is a highlight of this project.

Based on what functionality the user wants to access, different kinds of processing have to be done. The idea of a state arises from this situation naturally. Based on user input, we will set a certain “state”, and based on the current state, some processing will be done. Each state can also change the current state to a different one, allowing navigation between different states.

A naïve implementation would declare constants that represent different states, and would check in an if-elif chain what state is currently set, and run code based on that, like so:

```
STATE0 = 0
STATE1 = 1
currentState = 0

while True: # main process-loop
    if currentState == STATE0:
        # STATE0'S processing

        inp = userInput()

        if inp == "change_state":
            # some user-input condition
            currentState = STATE1

        continue

    elif currentState == STATE1:
        # STATE1'S processing

        inp = userInput()

        if inp == "change_state":
            currentState = STATE0

        continue
```

However, this if-elif chain can quickly grow very large. Since the states are being set by the code itself explicitly, there shouldn't be any need to check for the state in each process-loop. Moreover, the implementation for different states cannot be separated and thus modularization cannot be achieved, which would be desirable from a code-design standpoint.

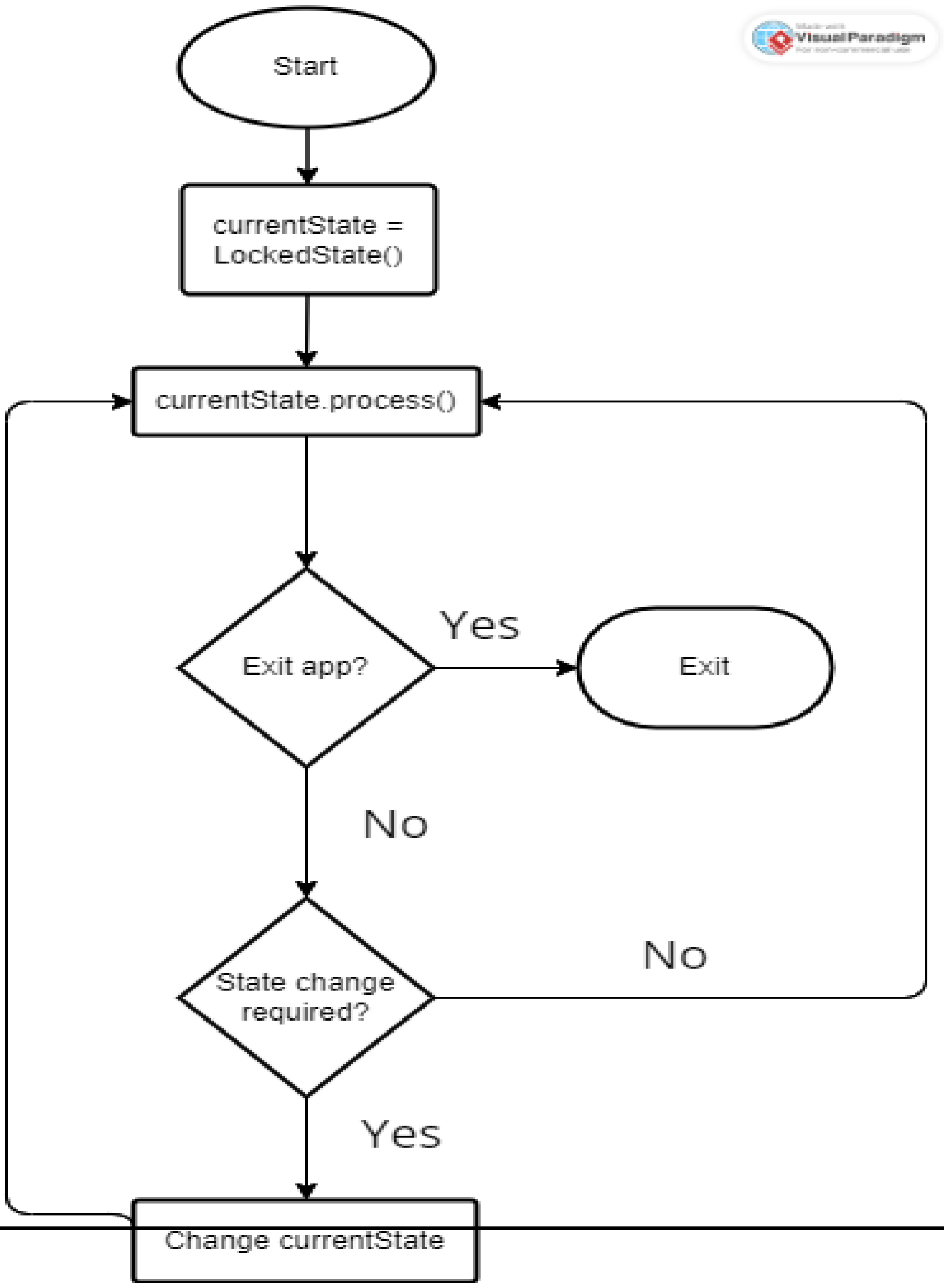
The problem is that the state in the above code is represented by an integer object, which does not contain any information about what kind of processing it needs. Thus, the current state needs to be checked and its implementation has to be provided by the main process-loop itself. However, if the state was represented by an object that itself contained information about the required processing, then we could just use that information without caring what the current state exactly is. This state-object can be a class that contains a `process()` function, which is called by the main process-loop. This eliminates the need of if-checks altogether. Also, since the class definitions can be written separately, it improves code-design by allowing modularization. This design is exemplified by the application code following this page.

Note: We have designed the program so that many in-program days pass in just a few real-life seconds, so that we can demonstrate fixed deposit interests.

The complete development history and project files can be found here :

https://github.com/satwik-krit/banking-system

The general program execution can be expressed as follows:



PROGRAM CODE

SQL queries :

```sql
DROP DATABASE IF EXISTS Bank;
CREATE DATABASE Bank;
USE Bank;

CREATE TABLE Users (
    password VARCHAR(10) NOT NULL,
    username VARCHAR(50),
    firstname VARCHAR(50) NOT NULL,
    lastname VARCHAR(50) NOT NULL,
    age INT NOT NULL,
    phone VARCHAR(15) NOT NULL,
    inactive TINYINT(1) NOT NULL DEFAULT 0,

    PRIMARY KEY(username),
    CHECK(age > 0)
);

CREATE TABLE Account (
    balance INT NOT NULL,
    created DATE NOT NULL,
    frozen TINYINT(1) NOT NULL DEFAULT 0,
    username VARCHAR(50),

    PRIMARY KEY(username),
    CHECK(balance >= 0),
    FOREIGN KEY(username) REFERENCES Users(username)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);

CREATE TABLE FixedDepo (
    fdName VARCHAR(30),
    username VARCHAR(50),
    principal INT NOT NULL,
    interest INT NOT NULL,
    creationdate DATE NOT NULL,
    timeperiod INT NOT NULL,
    maturedate DATE NOT NULL,
    withdrawn INT NOT NULL DEFAULT 0,

    PRIMARY KEY(fdName, username),
    CHECK(principal > 0),
    CHECK(interest > 0),
    CHECK(timeperiod BETWEEN 0 AND 10),
    CHECK(maturedate = DATE_ADD(creationdate, INTERVAL timeperiod
    YEAR)),
    FOREIGN KEY(username) REFERENCES Users(username)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);

CREATE TABLE Transactions (
    transID INT AUTO_INCREMENT,
    payerID VARCHAR(50) NOT NULL,
    receiverID VARCHAR(50) NOT NULL,
    transDate DATE NOT NULL,
    amount INT NOT NULL,
    comment TINYTEXT,

    PRIMARY KEY(transID),
    CHECK(amount > 0),
    FOREIGN KEY(payerID) REFERENCES Users(username)
    ON DELETE RESTRICT
    ON UPDATE CASCADE,
    FOREIGN KEY(payerID) REFERENCES Users(username)
    ON DELETE RESTRICT
    ON UPDATE CASCADE
);

CREATE TABLE Updates (
    username VARCHAR(50) NOT NULL,
    baseContent TINYTEXT NOT NULL,
    extraContent TEXT NOT NULL,
    updateDate DATE,

    FOREIGN KEY(username) REFERENCES Users(username)
);

CREATE TABLE EnvInfo (
    DBCreationDateTime TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

INSERT INTO EnvInfo
VALUES ();
```

Python code :

```python
import time

from getpass import getpass
import datetime as dt
from dateutil.relativedelta import relativedelta
import mysql.connector as sqlconn
from mysql.connector import DataError, DatabaseError, OperationalError,
NotSupportedError, IntegrityError, ProgrammingError, InternalError

try:

    currentState = None
    TIMEDELTA = 0.2
    currentDate = None

    db = sqlconn.connect(host="localhost", user="root",
         password="root", database="bank", charset="utf8")
    crsr = db.cursor(buffered=True)

    def EXIT(code=0):
        db.close()
        exit(code)

    def execute(query : str, args : tuple) -> None:
        crsr.execute(query.format(*args))

    def resultExists(result):
        if len(result):
            return True
        else:
            return False

    def getBalance(username : str) -> int:
        Q_GET_BALANCE = ("SELECT balance "
                         "FROM account "
                         "WHERE username = '{}';")

        execute(Q_GET_BALANCE, (username,))
        return crsr.fetchone()[0]

    def c_changeBalance(username : str, change : int) -> None:
        QC_CHANGE_BALANCE = ("UPDATE Account "
                             "SET balance = balance + {1} "
                             "WHERE username = '{0}'; ")

        execute(QC_CHANGE_BALANCE, (username, change))

    def userExists(username : str) -> bool:
        Q_CHECK_USERNAME = ("SELECT username "
                            "FROM Users "
                            "WHERE username = '{}';")

        execute(Q_CHECK_USERNAME, (username,))

        if len(crsr.fetchall()) != 0:
            return True
        else:
            return False

    def checkFDExists(username : str, fdName : str) -> bool:
        Q_CHECK_FD_EXISTS = ("SELECT * FROM FixedDepo "
                             "WHERE username = '{}' AND fdName = '{}';")

        execute(Q_CHECK_FD_EXISTS, (username, fdName))

        if len(crsr.fetchall()) != 0:
            return True
        else:
            return False

    def intInput(prompt : str, failMsg : str = "Invalid input.") -> int:
        while True:
            inpStr = input(prompt).strip()

            if not inpStr.isdigit():
                print(failMsg)
            else:
                return int(inpStr)

    def getUpdates(username, date=None):
        _Q_GET_UPDATES_ALL = ("SELECT baseContent, extraContent, updateDate "
                              "FROM Updates "
                              "WHERE username = '{}';")

        _Q_GET_UPDATES_DAY = ("SELECT baseContent, extraContent, updateDate "
                              "FROM Updates "
                              "WHERE username = '{}' "
                              "AND updateDate = '{}'")

        if date:
            execute(_Q_GET_UPDATES_DAY, (username, date))
            return crsr.fetchall()
        else:
            execute(_Q_GET_UPDATES_ALL, (username, ))
            return crsr.fetchall()

    def c_createUpdate(username, baseContent, extraContent="No comment", _date=None):
        _QC_CREATE_UPDATE = ("INSERT INTO Updates "
                             "VALUES "
                             "('{}', '{}', '{}', '{}')")

        if _date:
            execute(_QC_CREATE_UPDATE, (username, baseContent, extraContent, _date))
        else:
            execute(_QC_CREATE_UPDATE, (username, baseContent, extraContent, currentDate))

    def getUserInfo(username):
        _Q_GET_USER = ("SELECT firstname, lastname, age, phone, inactive "
                       "FROM Users "
                       "WHERE username = '{}' ;")

        execute(_Q_GET_USER, (username,))
        return crsr.fetchone()

    class LockedState:
        def __init__(self):
            pass

        def process(self):
            global currentState

            print("======================================================================")
            print("Enter username and password to view details or create a new account")
            print("(1) Login")
            print("(2) Create an account")
            print("(3) Quit")
            print()

            option = intInput("(Option) -> ")

            if option == 1:
                currentState = LoginState()

            elif option == 2:
                currentState = CreateAccountState()

            elif option == 3:
                EXIT()

            else:
                print()
                print("Please choose a valid option.")

    class LoginState:
        _Q_LOGIN_USER = ("SELECT username, password "
                         "FROM Users "
                         "WHERE username = '{}'; ")

        def __init__(self):
            pass

        def _login(self, username : str, password : str) -> int:
            global currentState

            execute(self._Q_LOGIN_USER, (username,))
            record = crsr.fetchone()

            if record == None:
                print("Username not found.")
                currentState = LockedState()
                return

            if record[1] != password:
                print("Incorrect password.")
                currentState = LockedState()
                return

            print("Logged in successfully.")

            currentState = UnlockedState(username)

        def process(self):
            print("=======================================")
            username = input("(Enter Username) -> ").strip()
            password = getpass("(Enter Password) -> ").strip()
            print()

            self._login(username, password)

    class CreateAccountState:
        _QC_CREATE_USER = ("INSERT INTO Users VALUES "
                           "('{}', '{}', '{}', '{}', {}, '{}', {}); ")

        _QC_CREATE_ACCOUNT = ("INSERT INTO account "
                              "VALUES "
                              "({}, '{}', {}, '{}'); ")

        def __init__(self):
            pass

        def _createNewUser(self, username : str, password : str, firstname : str,
                           lastname : str, age : int, phone : int) -> int:
            execute(self._QC_CREATE_USER, (password, username, firstname, lastname, age, phone, 0))
            execute(self._QC_CREATE_ACCOUNT, (0, str(currentDate), 0, username))
            db.commit()

        def process(self):
            global currentState

            print("========================================")
            print("(0) Create account")
            print("(1) Abort")
            print()

            option = intInput("(Option) -> ")

            if option == 0:
                print()
                username = input("(Enter NEW Username) -> ").strip()

                if userExists(username):
                    print()
                    print("Username not unique.")
                    return

                while True:
                    password = input("(Enter NEW Password) -> ").strip()
                    confirmPassword = getpass("(Enter password for confirmation) -> ").strip()

                    if password == confirmPassword:
                        break

                    print("Passwords do not match. Enter again.")

                firstname = input("(Enter first name) -> ").strip()
                lastname = input("(Enter last name) -> ").strip()
                age = intInput("(Enter age) -> ")
                phone = intInput("(Enter phone no.) -> ")
                print()

                self._createNewUser(username, password, firstname, lastname, age, phone)

                currentState = UnlockedState(username)

            elif option == 1:
                currentState = LockedState()

            else:
                print()
                print("Please choose a valid option")

    class UnlockedState:
        def __init__(self, username : str):
            self._username = username

        def process(self):
            global currentState
            global currentDate

            # print and remove updates
            balance = getBalance(self._username)
            updates = getUpdates(self._username, currentDate)

            print("===================================")
            print(currentDate)
            print(f"BALANCE: {balance}")
            if resultExists(updates):
                print("TODAY'S UPDATES:", end=" ")
                for content, _, __ in updates:
                    print(f"{content}", end=", ")
            print()
            print("(0) Logout")
            print("(1) Pay")
            print("(2) Deposit")
            print("(3) Create a fixed deposit")
            print("(4) Modify/View fixed deposits")
            print("(5) View all updates for your account")
            print()

            option = intInput("(Option) -> ")

            if option == 1:
                currentState = PayState(self._username)

            elif option == 2:
                currentState = DepositState(self._username)

            elif option == 3:
                currentState = CreateFDState(self._username)

            elif option == 0:
                currentState = LockedState()

            elif option == 4:
                currentState = ViewFDState(self._username)
```

```python

        elif option == 5:
            currentState = ViewUpdatesState(self._username)

        else:
            print()
            print("Please choose a valid option.")

class PayState:
    _QC_PAY_USER = ("INSERT INTO transactions "
                    "(payerID, receiverID, transDate, amount, comment) "
                    "VALUES "
                    "('{}', '{}', '{}', {}, '{}'); ")

    _Q_GETUSERPASSWORD = ("SELECT password "
                          "FROM Users "
                          "WHERE username = '{}'; ")

    def __init__(self, username : str):
        self._username = username

    def _pay(self, receiverName : str, amount : float, comment: str) -> int:
        global currentState

        global currentState
        balance = getBalance(self._username)

        if receiverName == self._username:
            print("You cannot pay yourself.")
            return

        if not userExists(receiverName):
            print("This receiver does not exist.")
            return

        if amount == 0:
            print("Enter a valid amount to pay.")
            return

        if amount > balance:
            print("You do not have sufficient balance.")
            return

        inpPwd = getpass("(Enter password to proceed with payment) -> ")
        execute(self._Q_GETUSERPASSWORD, (self._username, ))
        userPwd = crsr.fetchone()[0]

        if inpPwd != userPwd:
            print("Incorrect password, aborting payment.")
            return

        c_changeBalance(self._username, -amount)
        execute(self._QC_PAY_USER, (self._username, receiverName, str(currentDate), amount, comment))
        c_changeBalance(receiverName, amount)

        recFirstName = getUserInfo(receiverName)[0]
        userFirstName = getUserInfo(self._username)[0]
```

```python
            c_createUpdate(receiverName, f"{userFirstName} paid {amount}", f"{comment}")
            c_createUpdate(self._username, f"Paid {amount} to {recFirstName}", f"{comment}")

            db.commit()

            print("Transaction made successfully.")

        def process(self):
            global currentState

            print("===========================")
            print("(0) Pay to another user")
            print("(1) Abort")
            print()

            option = intInput("(Option) -> ")

            if option == 0:
                print()
                receiverName = input("(Enter username of receiver) -> ").strip()
                amount = intInput("(Enter amount to pay) -> ")
                comment =  input("Enter comment (optional)) -> ").strip()
                print()

                if not comment:
                    comment = "No comment"

                self._pay(receiverName, amount, comment)

            elif option == 1:
                currentState = UnlockedState(self._username)

            else:
                print()
                print("Please choose a valid option.")

    class DepositState:
        def __init__(self, username : str):
            self._username = username

        def _deposit(self, amount : int) -> None:
            c_changeBalance(self._username, amount)
            c_createUpdate(self._username, f"Deposit {amount}")
            db.commit()

        def process(self):
            global currentState

            print("======================================================================")
            amount = intInput("(Enter amount to deposit (cash to digital money)) -> ")
            self._deposit(amount)

            currentState = UnlockedState(self._username)

```

```python
class CreateFDState:
    _QC_CREATE_FD = ("INSERT INTO FixedDepo "
                     "(fdName, username, principal, interest, creationdate, timeperiod, maturedate) "
                     "VALUES('{}', '{}', {}, {}, '{}', {}, '{}');")

    def __init__(self, username : str):
        self._username = username

    def _createFD(self, name : str, amount : int, period : int) -> None:
        if checkFDExists(self._username, name):
            print("FD with this name already exists")
            return

        if getBalance(self._username) < amount:
            print("You do not have sufficient balance.")
            return

        c_changeBalance(self._username, -amount)
        execute(self._QC_CREATE_FD, (name, self._username, amount, 2, str(currentDate), period,
                currentDate + relativedelta(years=period)))
        c_createUpdate(self._username, f"Create {name} FD")
        db.commit()
        print("FD created successfully.")

    def process(self):
        global currentState

        print("======================")
        print("(0) Create new FD")
        print("(1) Return")
        print()

        option = intInput("(Option) -> ")

        if option == 0:
            print()
            name = input("(Enter FD name) -> ")
            amount = intInput("(Enter amount) -> ")
            period = intInput("(Enter time period in years (under 10)) -> ")
            print()

            self._createFD(name, amount, period)

        elif option == 1:
            currentState = UnlockedState(self._username)

        else:
            print()
            print("Please choose a valid option.")

class ViewFDState:
    _Q_GET_FD_DETAILS = ("SELECT * FROM FixedDepo "
                         "WHERE username = '{}' AND fdName = '{}';")
    _QC_WITHDRAW_FD = ("UPDATE FixedDepo "
                       "SET withdrawn = 1 "
                       "WHERE username = '{}' AND fdName = '{}'; ")
    _Q_GET_ALL_FDS = ("SELECT fdName FROM FixedDepo "
                      "WHERE username = '{}'; ")

    def __init__(self, username : str):
        self._username = username

    def _getFDComputedDetails(self, record : tuple):
            passedTimeDelta = relativedelta(currentDate, record[4])
            yearsPassed = int(passedTimeDelta.years + (passedTimeDelta.months / 12) + (passedTimeDelta.days / 365.25))
            matured = False if yearsPassed < record[5] else True
            value = (record[2] * record[3] * (record[5] if matured else yearsPassed) / 100) + record[2]

            return (yearsPassed, matured, value)

    def _printFD(self, fdName : str) -> None:
        if not checkFDExists(self._username, fdName):
            print("FD with this name does not exist.")
            return

        execute(self._Q_GET_FD_DETAILS, (self._username, fdName))
        record = crsr.fetchone()
        computedDetails = self._getFDComputedDetails(record)

        print(f"Principal : {record[2]}")
        print(f"Interest : {record[3]}")
        print(f"Created : {record[4]}")
        print(f"Total time period (years) : {record[5]}")
        print(f"Time passed (years) : {computedDetails[0]}")
        print(f"Current value : {computedDetails[2]}")
        print(f"Mature date : {record[6]}")
        print(f"Matured? : {'Yes' if computedDetails[1] else 'No'}")
        print(f"Widthdrawn? : {'Yes' if record[7] else 'No'}")

    def _withdrawFD(self, fdName : str) -> None:
        if not checkFDExists(self._username, fdName):
            print("FD with this name does not exist.")
            return

        execute(self._Q_GET_FD_DETAILS, (self._username, fdName))
        record = crsr.fetchone()

        if record[7]:
            print("You have already withdrawn this FD.")
            return

        computedDetails = self._getFDComputedDetails(record)
        execute(self._QC_WITHDRAW_FD, (self._username, fdName))
        c_changeBalance(self._username, computedDetails[2])
        c_createUpdate(self._username, f"Withdrew amount {computedDetails[2]} from FD {fdName}.")

```

```python
            db.commit()

            print(f"Withdrew amount {computedDetails[2]} from FD {fdName}.")

    def process(self):
        global currentState

        # display FDs

        print("=============================")
        print("(0) Show all FDs")
        print("(1) View details of a particular FD")
        print("(2) Withdraw an FD")
        print("(3) Return")
        print()

        option = intInput("(Option) -> ")

        if option == 0:
            execute(self._Q_GET_ALL_FDS, (self._username,))
            fdNames = crsr.fetchall()

            if not resultExists(fdNames):
                print("You don't have any FDs yet.")
                return

            for fdName in fdNames:
                print(fdName[0])

        elif option == 1:
            print()
            fdName = input("(Enter FD name) -> ").strip()
            print()
            self._printFD(fdName)

        elif option == 2:
            print()
            fdName = input("(Enter FD name) -> ").strip()
            print()
            self._withdrawFD(fdName)

        elif option == 3:
            currentState = UnlockedState(self._username)

        else:
            print()
            print("Please choose a valid option.")

class ViewUpdatesState:
    def __init__(self, username):
        self._username = username

    def _displayUpdates(self, updates):
        if not resultExists(updates) :
            print("You have no updates for the requested query.")
            return

        # sort updates from most recent to last
        updates.sort(key = lambda x: x[2])
        for index, update in enumerate(updates):
            baseContent, extraContent, updateDate = update
            print()
            print(f"({index}): {baseContent}")
            print(f"Date: {updateDate}")
            print(f"Comment: {extraContent}")

    def process(self):
        global currentState

        print("=============================")
        print("(0) View all updates")
        print("(1) View all updates for a day")
        print("(2) Return")
        print()

        option = intInput("(Option) -> ")

        if option == 0:
            updates = getUpdates(self._username)
            self._displayUpdates(updates)

        elif option == 1:
            inp = input("(Required date, in YYYY-MM-DD format) -> ")

            try:
                date = dt.date.fromisoformat(inp)
                updates = getUpdates(self._username, date)
                self._displayUpdates(updates)

            except ValueError:
                print("Invalid date.")

        elif option == 2:
            currentState = UnlockedState(self._username)

        else:
            print("Please choose a valid option.")

if __name__ == '__main__':
    currentState = LockedState()

    _Q_GETDBCREATIONDATETIME = ("SELECT DBCreationDateTime "
                                "FROM EnvInfo ;")

    # Get the date and time when we created the database
    execute(_Q_GETDBCREATIONDATETIME, ())

    creationDateTime = crsr.fetchone()[0]
    creationTime = creationDateTime.timestamp()
    creationDate = creationDateTime.date()

    previousTime = creationTime
    currentDate = creationDate

    while True:
        currentTime = time.time()
        elapsedDays = (currentTime - previousTime) // TIMEDELTA
        currentDate += dt.timedelta(days=elapsedDays)

```

```python
            currentState.process()
            previousTime = currentTime

except (DataError, DatabaseError, OperationalError, NotSupportedError,
        IntegrityError, ProgrammingError, InternalError) as e:
    print("DB Error!", e)

except KeyboardInterrupt:
    EXIT(0)

except Exception as e:
    print("ERROR: ", e)
    EXIT(1)
```

SAMPLE

OUTPUT

```
Enter username and password to view details or create a new account
(1) Login
(2) Create an account
(3) Quit

(Option) -> 2
========================================
(0) Create account
(1) Abort

(Option) -> 0

(Enter NEW Username) -> ng
(Enter NEW Password) -> ng@2007
(Enter password for confirmation) ->
(Enter first name) -> nandan
(Enter last name) -> goyal
(Enter age) -> 19
(Enter phone no.) -> 9837461850

===================================
2025-01-01
BALANCE: 0

(0) Logout
(1) Pay
(2) Deposit
(3) Create a fixed deposit
(4) Modify/View fixed deposits
(5) View all updates for your account

(Option) -> 2
=====================================================================
(Enter amount to deposit (cash to digital money)) -> 5000
===================================
2025-01-10
BALANCE: 5000

(0) Logout
(1) Pay
(2) Deposit
(3) Create a fixed deposit
(4) Modify/View fixed deposits
(5) View all updates for your account

(Option) -> 3
======================
(0) Create new FD
(1) Return

(Option) -> 0

(Enter FD name) -> fd1
(Enter amount) -> 1000
(Enter time period in years (under 10)) -> 5

FD created successfully.
```

```
======================

(0) Create new FD
(1) Return

(Option) -> 1
===================================
2025-01-22
BALANCE: 4000

(0) Logout
(1) Pay
(2) Deposit
(3) Create a fixed deposit
(4) Modify/View fixed deposits
(5) View all updates for your account

(Option) -> 4
=============================
(0) Show all FDs
(1) View details of a particular FD
(2) Withdraw an FD
(3) Return

(Option) -> 0
fd1
=============================
(0) Show all FDs
(1) View details of a particular FD
(2) Withdraw an FD
(3) Return

(Option) -> 1

(Enter FD name) -> fd1

Principal : 1000
Interest : 2
Created : 2025-01-17
Total time period (years) : 5
Time passed (years) : 0
Current value : 1000.0
Mature date : 2030-01-17
Matured? : No
Widthdrawn? : No
=============================
(0) Show all FDs
(1) View details of a particular FD
(2) Withdraw an FD
(3) Return

(Option) -> 3
===================================
2025-02-02
BALANCE: 4000

(0) Logout
(1) Pay
(2) Deposit
(3) Create a fixed deposit
(4) Modify/View fixed deposits
(5) View all updates for your account

(Option) -> 5
=============================
(0) View all updates
(1) View all updates for a day
(2) Return

(Option) -> 0

(0): Deposit 5000
Date: 2025-01-05
Comment: No comment

(1): Create fd1 FD
Date: 2025-01-17
Comment: No comment
=============================
(0) View all updates
(1) View all updates for a day
(2) Return

(Option) -> 2
===================================
2025-02-10
BALANCE: 4000

(0) Logout
(1) Pay
(2) Deposit
(3) Create a fixed deposit
(4) Modify/View fixed deposits
(5) View all updates for your account

(Option) -> 0
======================================================================
Enter username and password to view details or create a new account
(1) Login
(2) Create an account
(3) Quit

(Option) -> 2
========================================
(0) Create account
(1) Abort

(Option) -> sg
Invalid input.
(Option) -> 0


(Enter NEW Username) -> sg
(Enter NEW Password) -> sg@2007
```

```
(Enter password for confirmation) ->
(Enter first name) -> satwik
(Enter last name) -> gupta
(Enter age) -> 18
(Enter phone no.) -> 9487468553

===================================
2025-02-27
BALANCE: 0

(0) Logout
(1) Pay
(2) Deposit
(3) Create a fixed deposit
(4) Modify/View fixed deposits
(5) View all updates for your account

(Option) -> 2
======================================================================
(Enter amount to deposit (cash to digital money)) -> 10000
===================================
2025-03-01
BALANCE: 10000

(0) Logout
(1) Pay
(2) Deposit
(3) Create a fixed deposit
(4) Modify/View fixed deposits
(5) View all updates for your account

(Option) -> 1
===========================
(0) Pay to another user
(1) Abort

(Option) -> 0

(Enter username of receiver) -> ng
(Enter amount to pay) -> 2000
Enter comment (optional)) -> first payment

(Enter password to proceed with payment) ->
Transaction made successfully.
===========================
(0) Pay to another user
(1) Abort

(Option) -> 1
===================================
2025-03-12
BALANCE: 8000

(0) Logout
(1) Pay
(2) Deposit
(3) Create a fixed deposit
(4) Modify/View fixed deposits
(5) View all updates for your account

(Option) -> 5
```

```
=============================
(0) View all updates
(1) View all updates for a day
(2) Return

(Option) -> 0

(0): Deposit 10000
Date: 2025-02-28
Comment: No comment

(1): Paid 2000 to nandan
Date: 2025-03-03
Comment: first payment
=============================
(0) View all updates
(1) View all updates for a day
(2) Return

(Option) -> 2
===================================
2025-03-20
BALANCE: 8000

(0) Logout
(1) Pay
(2) Deposit
(3) Create a fixed deposit
(4) Modify/View fixed deposits
(5) View all updates for your account

(Option) -> 0
======================================================================
Enter username and password to view details or create a new account
(1) Login
(2) Create an account
(3) Quit

(Option) -> 1
=======================================
(Enter Username) -> ng
(Enter Password) ->

Logged in successfully.
===================================
2025-03-22
BALANCE: 6000

(0) Logout
(1) Pay
(2) Deposit
(3) Create a fixed deposit
(4) Modify/View fixed deposits
(5) View all updates for your account

(Option) -> 5
=============================
(0) View all updates
(1) View all updates for a day
(2) Return
```

```
(Option) -> 0

(0): Deposit 5000
Date: 2025-01-05
Comment: No comment

(1): Create fd1 FD
Date: 2025-01-17
Comment: No comment

(2): satwik paid 2000
Date: 2025-03-03
Comment: first payment
=============================
(0) View all updates
(1) View all updates for a day
(2) Return

(Option) -> 2
===================================
2025-04-01
BALANCE: 6000

(0) Logout
(1) Pay
(2) Deposit
(3) Create a fixed deposit
(4) Modify/View fixed deposits
(5) View all updates for your account

(Option) -> 5
=============================
(0) View all updates
(1) View all updates for a day
(2) Return

(Option) -> 0

(0): Deposit 5000
Date: 2025-01-05
Comment: No comment

(1): Create fd1 FD
Date: 2025-01-17
Comment: No comment

(2): satwik paid 2000
Date: 2025-03-03
=============================
(0) View all updates
(1) View all updates for a day
(2) Return

(Option) -> 2
===================================
2040-04-19
BALANCE: 6000

(0) Logout
(1) Pay
(2) Deposit
(3) Create a fixed deposit
(4) Modify/View fixed deposits
(5) View all updates for your account

(Option) -> 4
=============================
(0) Show all FDs
(1) View details of a particular FD
(2) Withdraw an FD
(3) Return

(Option) -> 1

(Enter FD name) -> fd1

Principal : 1000
Interest : 2
Created : 2025-01-17
Total time period (years) : 5
Time passed (years) : 15
Current value : 1100.0
Mature date : 2030-01-17
Matured? : Yes
Widthdrawn? : No
=============================
(0) Show all FDs
(1) View details of a particular FD
(2) Withdraw an FD
(3) Return

(Option) -> 2

(Enter FD name) -> fd1

Withdrew amount 1100.0 from FD fd1.
=============================
(0) Show all FDs
(1) View details of a particular FD
(2) Withdraw an FD
(3) Return

(Option) -> 3
===================================
2040-08-22
BALANCE: 7100

(0) Logout
(1) Pay
(2) Deposit
(3) Create a fixed deposit
(4) Modify/View fixed deposits
(5) View all updates for your account
```

BIBLIOGRAPHY

- Sumita Arora Textbook Computer Science with Python Class XII
- Various online resources