**Python Coding Challenge**

Name – Priyansh Rusia

**SQL code –**

CREATE DATABASE InsuranceManagementDB;

USE InsuranceManagementDB;

CREATE TABLE Users (

userId INT PRIMARY KEY identity(101,1),

username NVARCHAR(50) NOT NULL,

password NVARCHAR(50) NOT NULL,

role NVARCHAR(50) NOT NULL

);

CREATE TABLE Clients (

clientId INT PRIMARY KEY identity(201,1),

clientName NVARCHAR(100) NOT NULL,

contactInfo NVARCHAR(100) NOT NULL,

policy NVARCHAR(100) NOT NULL

);

CREATE TABLE Policies (

policyId INT PRIMARY KEY IDENTITY(1,1),

policyName NVARCHAR(100) NOT NULL,

policyDescription NVARCHAR(255) NOT NULL

);

CREATE TABLE Claims (

claimId INT PRIMARY KEY identity(301,1),

claimNumber NVARCHAR(100) NOT NULL,

dateFiled DATE NOT NULL,

claimAmount DECIMAL(10, 2) NOT NULL,

status NVARCHAR(50) NOT NULL,

clientId INT,

policy NVARCHAR(100),

FOREIGN KEY (clientId) REFERENCES Clients(clientId)

);

CREATE TABLE Payments (

paymentId INT PRIMARY KEY identity(401,1),

paymentDate DATE NOT NULL,

paymentAmount DECIMAL(10, 2) NOT NULL,

clientId INT,

FOREIGN KEY (clientId) REFERENCES Clients(clientId)

);

INSERT INTO Users (username, password, role) VALUES

('john\_doe', 'securepass123', 'admin'),

('jane\_smith', 'password123', 'user'),

('michael\_brown', 'abc12345', 'user'),

('alice\_jones', 'qwerty987', 'admin'),

('chris\_wilson', 'mypassword', 'user');

INSERT INTO Clients (clientName, contactInfo, policy) VALUES

('Acme Corp', 'acme@corporate.com', 'Premium Health Policy'),

('Tech Solutions', 'info@techsol.com', 'Comprehensive IT Policy'),

('Bright Future Inc', 'contact@bfuture.com', 'Life Insurance Plan'),

('Green Planet', 'support@greenplanet.com', 'Environmental Liability Policy'),

('Oceanic Ventures', 'hello@oceanic.com', 'Marine Insurance Policy');

INSERT INTO Claims (claimNumber, dateFiled, claimAmount, status, clientId, policy) VALUES

('CLM001', '2024-10-01', 5000.00, 'Pending', 201, 'Premium Health Policy'),

('CLM002', '2024-09-15', 10000.00, 'Approved', 202, 'Comprehensive IT Policy'),

('CLM003', '2024-08-20', 2500.00, 'Rejected', 203, 'Life Insurance Plan'),

('CLM004', '2024-10-05', 8000.00, 'Pending', 204, 'Environmental Liability Policy'),

('CLM005', '2024-07-10', 15000.00, 'Approved', 205, 'Marine Insurance Policy');

INSERT INTO Payments (paymentDate, paymentAmount, clientId) VALUES

('2024-10-01', 3000.00, 201),

('2024-09-20', 7500.00, 202),

('2024-08-25', 2000.00, 203),

('2024-10-10', 6000.00, 204),

('2024-07-15', 12000.00, 205);

INSERT INTO Policies (policyName, policyDescription) VALUES

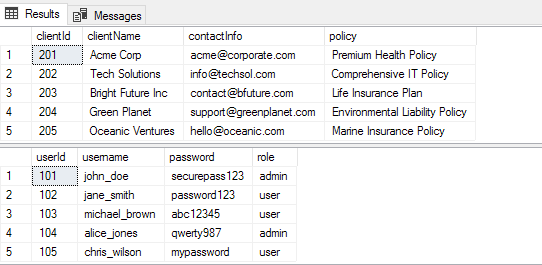
('Premium Health Policy', 'Provides extensive medical coverage for employees.'),

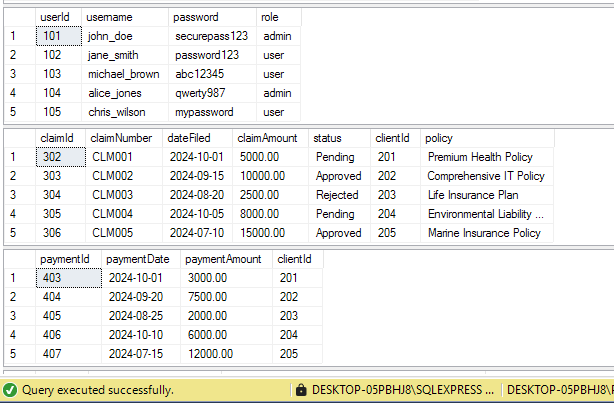
('Comprehensive IT Policy', 'Covers hardware and software failures.'),

('Life Insurance Plan', 'Ensures financial security for family after death.'),

('Environmental Liability Policy', 'Covers environmental hazards and damages.'),

('Marine Insurance Policy', 'Provides coverage for cargo and ships.');





**Python Source Code**

**Claim.py**

class Claim:

def \_\_init\_\_(self, claim\_id=None, claim\_number=None, date\_filed=None,

claim\_amount=None, status=None, policy=None, client=None):

self.\_\_claim\_id = claim\_id

self.\_\_claim\_number = claim\_number

self.\_\_date\_filed = date\_filed

self.\_\_claim\_amount = claim\_amount

self.\_\_status = status

self.\_\_policy = policy

self.\_\_client = client

# Getters

def get\_claim\_id(self):

return self.\_\_claim\_id

def get\_claim\_number(self):

return self.\_\_claim\_number

def get\_date\_filed(self):

return self.\_\_date\_filed

def get\_claim\_amount(self):

return self.\_\_claim\_amount

def get\_status(self):

return self.\_\_status

def get\_policy(self):

return self.\_\_policy

def get\_client(self):

return self.\_\_client

# Setters

def set\_claim\_id(self, claim\_id):

self.\_\_claim\_id = claim\_id

def set\_claim\_number(self, claim\_number):

self.\_\_claim\_number = claim\_number

def set\_date\_filed(self, date\_filed):

self.\_\_date\_filed = date\_filed

def set\_claim\_amount(self, claim\_amount):

self.\_\_claim\_amount = claim\_amount

def set\_status(self, status):

self.\_\_status = status

def set\_policy(self, policy):

self.\_\_policy = policy

def set\_client(self, client):

self.\_\_client = client

def \_\_str\_\_(self):

return (f"Claim(claim\_id={self.\_\_claim\_id}, claim\_number='{self.\_\_claim\_number}', "

f"date\_filed='{self.\_\_date\_filed}', claim\_amount={self.\_\_claim\_amount}, "

f"status='{self.\_\_status}', policy='{self.\_\_policy}', client='{self.\_\_client}')")

**Client.py**

class Client:

    def \_\_init\_\_(self, client\_id=None, client\_name=None, contact\_info=None, policy=None):

        self.\_\_client\_id = client\_id

        self.\_\_client\_name = client\_name

        self.\_\_contact\_info = contact\_info

        self.\_\_policy = policy

    # Getters

    def get\_client\_id(self):

        return self.\_\_client\_id

    def get\_client\_name(self):

        return self.\_\_client\_name

    def get\_contact\_info(self):

        return self.\_\_contact\_info

    def get\_policy(self):

        return self.\_\_policy

    # Setters

    def set\_client\_id(self, client\_id):

        self.\_\_client\_id = client\_id

    def set\_client\_name(self, client\_name):

        self.\_\_client\_name = client\_name

    def set\_contact\_info(self, contact\_info):

        self.\_\_contact\_info = contact\_info

    def set\_policy(self, policy):

        self.\_\_policy = policy

    def \_\_str\_\_(self):

        return (f"Client(client\_id={self.\_\_client\_id}, client\_name='{self.\_\_client\_name}', "

                f"contact\_info='{self.\_\_contact\_info}', policy='{self.\_\_policy}')")

**Policy.py**

class Policy:

    def \_\_init\_\_(self, policyId=None, policyName=None, policyDescription=None):

        self.\_\_policyId = policyId

        self.\_\_policyName = policyName

        self.\_\_policyDescription = policyDescription

    def get\_policyId(self):

        return self.\_\_policyId

    def get\_policyName(self):

        return self.\_\_policyName

    def get\_policyDescription(self):

        return self.\_\_policyDescription

    def set\_policyId(self, policyId):

        self.\_\_policyId = policyId

    def set\_policyName(self, policyName):

        self.\_\_policyName = policyName

    def set\_policyDescription(self, policyDescription):

        self.\_\_policyDescription = policyDescription

    def \_\_str\_\_(self):

        return f"Policy (policyId={self.\_\_policyId}, policyName={self.\_\_policyName}, policyDescription={self.\_\_policyDescription})"

**Payment.py**

class Payment:

    def \_\_init\_\_(self, payment\_id=None, payment\_date=None, payment\_amount=None, client=None):

        self.\_\_payment\_id = payment\_id

        self.\_\_payment\_date = payment\_date

        self.\_\_payment\_amount = payment\_amount

        self.\_\_client = client

    # Getters

    def get\_payment\_id(self):

        return self.\_\_payment\_id

    def get\_payment\_date(self):

        return self.\_\_payment\_date

    def get\_payment\_amount(self):

        return self.\_\_payment\_amount

    def get\_client(self):

        return self.\_\_client

    # Setters

    def set\_payment\_id(self, payment\_id):

        self.\_\_payment\_id = payment\_id

    def set\_payment\_date(self, payment\_date):

        self.\_\_payment\_date = payment\_date

    def set\_payment\_amount(self, payment\_amount):

        self.\_\_payment\_amount = payment\_amount

    def set\_client(self, client):

        self.\_\_client = client

    # String representation (Equivalent to Java's toString())

    def \_\_str\_\_(self):

        return (f"Payment(payment\_id={self.\_\_payment\_id}, payment\_date='{self.\_\_payment\_date}', "

                f"payment\_amount={self.\_\_payment\_amount}, client='{self.\_\_client}')")

**User.py**

class User:

    def \_\_init\_\_(self, user\_id=None, username=None, password=None, role=None):

        self.\_\_user\_id = user\_id

        self.\_\_username = username

        self.\_\_password = password

        self.\_\_role = role

    # Getters

    def get\_user\_id(self):

        return self.\_\_user\_id

    def get\_username(self):

        return self.\_\_username

    def get\_password(self):

        return self.\_\_password

    def get\_role(self):

        return self.\_\_role

    # Setters

    def set\_user\_id(self, user\_id):

        self.\_\_user\_id = user\_id

    def set\_username(self, username):

        self.\_\_username = username

    def set\_password(self, password):

        self.\_\_password = password

    def set\_role(self, role):

        self.\_\_role = role

    def \_\_str\_\_(self):

        return (f"User(user\_id={self.\_\_user\_id}, username='{self.\_\_username}', "

                f"password='{self.\_\_password}', role='{self.\_\_role}')")

**Exception Code-**

class PolicyNotFoundException(Exception):

    def \_\_init\_\_(self, message):

        super().\_\_init\_\_(message)

**DAO code**

**ClientService.py**

import pyodbc

from entity.Client import Client

from util.DBConnect import DBConnUtil

class ClientServiceImpl:

    def \_\_init\_\_(self):

        self.conn = DBConnUtil.get\_connection()

    def \_\_del\_\_(self):

        """Ensures the database connection is closed when the object is destroyed."""

        if self.conn:

            self.conn.close()

            print("Database connection closed.")

    def createClient(self, client):

        cursor = self.conn.cursor()

        query = "INSERT INTO Clients (clientName, contactInfo, policy) VALUES (?, ?, ?)"

        cursor.execute(query,client.get\_clientName(), client.get\_contactInfo(), client.get\_policy())

        self.conn.commit()

        return True

    def getClient(self, clientId):

        cursor = self.conn.cursor()

        query = "SELECT \* FROM Clients WHERE clientId = ?"

        cursor.execute(query, clientId)

        result = cursor.fetchone()

        if result:

            return Client(clientId=result.clientId, clientName=result.clientName, contactInfo=result.contactInfo, policy=result.policy)

        else:

            return None

    def getAllClients(self):

        cursor = self.conn.cursor()

        query = "SELECT \* FROM Clients"

        cursor.execute(query)

        clients = []

        for row in cursor.fetchall():

            client = Client(clientId=row.clientId, clientName=row.clientName, contactInfo=row.contactInfo, policy=row.policy)

            clients.append(client)

        return clients

    def updateClient(self, client):

        cursor = self.conn.cursor()

        query = "UPDATE Clients SET clientName = ?, contactInfo = ?, policy = ? WHERE clientId = ?"

        cursor.execute(query, client.get\_clientName(), client.get\_contactInfo(), client.get\_policy(), client.get\_clientId())

        self.conn.commit()

        return True

    def deleteClient(self, clientId):

        cursor = self.conn.cursor()

        query = "DELETE FROM Clients WHERE clientId = ?"

        cursor.execute(query, clientId)

        self.conn.commit()

        return True

**Claim Service**

import pyodbc

from entity.Claim import Claim

from util.DBConnect import DBConnUtil

class ClaimServiceImpl:

    def \_\_init\_\_(self):

        self.conn = DBConnUtil.get\_connection()

    def \_\_del\_\_(self):

        """Ensures the database connection is closed when the object is destroyed."""

        if self.conn:

            self.conn.close()

            print("Database connection closed.")

    def createClaim(self, claim):

        cursor = self.conn.cursor()

        query = "INSERT INTO Claims (claimNumber, dateFiled, claimAmount, status, clientId, policy) VALUES (?, ?, ?, ?, ?, ?)"

        cursor.execute(query, claim.get\_claimNumber(), claim.get\_dateFiled(), claim.get\_claimAmount(), claim.get\_status(), claim.get\_client(), claim.get\_policy())

        self.conn.commit()

        return True

    def getClaim(self, claimId):

        cursor = self.conn.cursor()

        query = "SELECT \* FROM Claims WHERE claimId = ?"

        cursor.execute(query, claimId)

        result = cursor.fetchone()

        if result:

            return Claim(claimId=result.claimId, claimNumber=result.claimNumber, dateFiled=result.dateFiled, claimAmount=result.claimAmount, status=result.status, clientId=result.clientId, policy=result.policy)

        else:

            return None

    def getAllClaims(self):

        cursor = self.conn.cursor()

        query = "SELECT \* FROM Claims"

        cursor.execute(query)

        claims = []

        for row in cursor.fetchall():

            claim = Claim(claimId=row.claimId, claimNumber=row.claimNumber, dateFiled=row.dateFiled, claimAmount=row.claimAmount, status=row.status, clientId=row.clientId, policy=row.policy)

            claims.append(claim)

        return claims

    def updateClaim(self, claim):

        cursor = self.conn.cursor()

        query = "UPDATE Claims SET claimNumber = ?, dateFiled = ?, claimAmount = ?, status = ?, clientId = ?, policy = ? WHERE claimId = ?"

        cursor.execute(query, claim.get\_claimNumber(), claim.get\_dateFiled(), claim.get\_claimAmount(), claim.get\_status(), claim.get\_clientId(), claim.get\_policy(), claim.get\_claimId())

        self.conn.commit()

        return True

    def deleteClaim(self, claimId):

        cursor = self.conn.cursor()

        query = "DELETE FROM Claims WHERE claimId = ?"

        cursor.execute(query, claimId)

        self.conn.commit()

        return True

**IPolicyService.py**

from abc import ABC, abstractmethod

class IPolicyService(ABC):

    @abstractmethod

    def createPolicy(self, policy):

        pass

    @abstractmethod

    def getPolicy(self, policyId):

        pass

    @abstractmethod

    def getAllPolicies(self):

        pass

    @abstractmethod

    def updatePolicy(self, policy):

        pass

    @abstractmethod

    def deletePolicy(self, policyId):

        pass

**PaymentService**

import pyodbc

from entity.Payment import Payment

from util.DBConnect import DBConnUtil

class PaymentServiceImpl:

    def \_\_init\_\_(self):

        self.conn = DBConnUtil.get\_connection()

    def \_\_del\_\_(self):

        """Ensures the database connection is closed when the object is destroyed."""

        if self.conn:

            self.conn.close()

            print("Database connection closed.")

    def createPayment(self, payment):

        cursor = self.conn.cursor()

        query = "INSERT INTO Payments (paymentDate, paymentAmount, clientId) VALUES (?, ?, ?)"

        cursor.execute(query, payment.get\_paymentDate(), payment.get\_paymentAmount(), payment.get\_client())

        self.conn.commit()

        return True

    def getPayment(self, paymentId):

        cursor = self.conn.cursor()

        query = "SELECT \* FROM Payments WHERE paymentId = ?"

        cursor.execute(query, paymentId)

        result = cursor.fetchone()

        if result:

            return Payment(paymentId=result.paymentId, paymentDate=result.paymentDate, paymentAmount=result.paymentAmount, client=result.clientId)

        else:

            return None

    def getAllPayments(self):

        cursor = self.conn.cursor()

        query = "SELECT \* FROM Payments"

        cursor.execute(query)

        payments = []

        for row in cursor.fetchall():

            payment = Payment(paymentId=row.paymentId, paymentDate=row.paymentDate, paymentAmount=row.paymentAmount, client=row.clientId)

            payments.append(payment)

        return payments

    def updatePayment(self, payment):

        cursor = self.conn.cursor()

        query = "UPDATE Payments SET paymentDate = ?, paymentAmount = ?, clientId = ? WHERE paymentId = ?"

        cursor.execute(query, payment.get\_paymentDate(), payment.get\_paymentAmount(), payment.get\_client(), payment.get\_paymentId())

        self.conn.commit()

        return True

    def deletePayment(self, paymentId):

        cursor = self.conn.cursor()

        query = "DELETE FROM Payments WHERE paymentId = ?"

        cursor.execute(query, paymentId)

        self.conn.commit()

        return True

**PolicyService.py**

import pyodbc

from dao.IPolicyService import IPolicyService

from entity.Policy import Policy

from exceptions.PolicyNotFoundException import PolicyNotFoundException

from util.DBConnect import DBConnUtil

class PolicyServiceImpl(IPolicyService):

    def \_\_init\_\_(self):

        self.conn = DBConnUtil.get\_connection()

    def \_\_del\_\_(self):

        """Ensures the database connection is closed when the object is destroyed."""

        if self.conn:

            self.conn.close()

            print("Database connection closed.")

    def createPolicy(self, policy):

        cursor = self.conn.cursor()

        query = "INSERT INTO Policies (policyName, policyDescription) VALUES (?, ?)"

        cursor.execute(query, policy.get\_policyName(), policy.get\_policyDescription())

        self.conn.commit()

        return True

    def getPolicy(self, policyId):

        cursor = self.conn.cursor()

        query = "SELECT \* FROM Policies WHERE policyId = ?"

        cursor.execute(query, policyId)

        result = cursor.fetchone()

        if result:

            return Policy(policyId=result.policyId, policyName=result.policyName, policyDescription=result.policyDescription)

        else:

            raise PolicyNotFoundException(f"Policy with ID {policyId} not found.")

    def getAllPolicies(self):

        cursor = self.conn.cursor()

        query = "SELECT \* FROM Policies"

        cursor.execute(query)

        policies = []

        for row in cursor.fetchall():

            policy = Policy(policyId=row.policyId, policyName=row.policyName, policyDescription=row.policyDescription)

            policies.append(policy)

        return policies

    def updatePolicy(self, policy):

        cursor = self.conn.cursor()

        query = "UPDATE Policies SET policyName = ?, policyDescription = ? WHERE policyId = ?"

        cursor.execute(query, policy.get\_policyName(), policy.get\_policyDescription(), policy.get\_policyId())

        self.conn.commit()

        return True

    def deletePolicy(self, policyId):

        cursor = self.conn.cursor()

        query = "DELETE FROM Policies WHERE policyId = ?"

        cursor.execute(query, policyId)

        self.conn.commit()

        return True

**DBConnect.py**

import pyodbc

class DBConnUtil:

    @staticmethod

    def get\_connection():

        connection\_string = (

            "DRIVER={ODBC Driver 17 for SQL Server};"

            "SERVER=DESKTOP-05PBHJ8\\SQLEXPRESS;"

            "DATABASE=InsuranceManagementDB;"

            "Trusted\_Connection=yes;"

        )

        try:

            conn = pyodbc.connect(connection\_string)

            print("Database connection established successfully.")

            return conn

        except pyodbc.InterfaceError as ie:

            print(f"Interface error: {ie}")

        except pyodbc.OperationalError as oe:

            print(f"Operational error: {oe}")

        except pyodbc.DatabaseError as de:

            print(f"Database error: {de}")

        except pyodbc.Error as e:

            print(f"Database connection failed: {e}")

        except Exception as ex:

            print(f"An unexpected error occurred: {ex}")

        finally:

            print("Attempted to connect to the database.")

**Main.py**

import sys

import os

path\_to\_src = os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), '..'))

sys.path.append(path\_to\_src)

from dao.PolicyServiceImpl import PolicyServiceImpl

from dao.ClientServiceImpl import ClientServiceImpl

from dao.ClaimServiceImpl import ClaimServiceImpl

from dao.UserServiceImpl import UserServiceImpl

from dao.PaymentServiceImpl import PaymentServiceImpl

from entity.Policy import Policy

from entity.Client import Client

from entity.Claim import Claim

from entity.User import User

from entity.Payment import Payment

from exceptions.PolicyNotFoundException import PolicyNotFoundException

if \_\_name\_\_ == "\_\_main\_\_":

    # Create instances of service classes

    policy\_service = PolicyServiceImpl()

    client\_service = ClientServiceImpl()

    claim\_service = ClaimServiceImpl()

    user\_service = UserServiceImpl()

    payment\_service = PaymentServiceImpl()

    while True:

        print("\nInsurance Management System")

        print("1. Create Client")

        print("2. Create Policy")

        print("3. Get Policy")

        print("4. Get All Policies")

        print("5. Update Policy")

        print("6. Delete Policy")

        print("7. Create User")

        print("8. Get User")

        print("9. Get All Users")

        print("10. Update User")

        print("11. Delete User")

        print("12. Create Claim")

        print("13. Get Claim")

        print("14. Get All Claims")

        print("15. Create Payment")

        print("16. Get Payment")

        print("17. Exit")

        choice = input("Enter your choice: ")

        if choice == '1':  # Create Client

            clientName = input("Enter client name: ")

            contactInfo = input("Enter contact info: ")

            policy = input("Enter policy: ")

            client = Client(clientName=clientName, contactInfo=contactInfo, policy=policy)

            client\_service.createClient(client)

            print("Client created successfully!")

        elif choice == '2':  # Create Policy

            policyName = input("Enter policy name: ")

            policyDescription = input("Enter policy description: ")

            policy = Policy(policyName=policyName, policyDescription=policyDescription)

            policy\_service.createPolicy(policy)

            print("Policy created successfully!")

        elif choice == '3':  # Get Policy

            policyId = int(input("Enter policy ID: "))

            try:

                policy = policy\_service.getPolicy(policyId)

                print(policy)

            except PolicyNotFoundException as e:

                print(e)

        elif choice == '4':  # Get All Policies

            policies = policy\_service.getAllPolicies()

            for policy in policies:

                print(policy)

        elif choice == '5':  # Update Policy

            policyId = int(input("Enter policy ID: "))

            policyName = input("Enter new policy name: ")

            policyDescription = input("Enter new policy description: ")

            policy = Policy(policyId=policyId, policyName=policyName, policyDescription=policyDescription)

            policy\_service.updatePolicy(policy)

            print("Policy updated successfully!")

        elif choice == '6':  # Delete Policy

            policyId = int(input("Enter policy ID: "))

            policy\_service.deletePolicy(policyId)

            print("Policy deleted successfully!")

        elif choice == '7':  # Create User

            username = input("Enter username: ")

            password = input("Enter password: ")

            role = input("Enter role (admin/user): ")

            user = User(username=username, password=password, role=role)

            user\_service.createUser(user)

            print("User created successfully!")

        elif choice == '8':  # Get User

            userId = int(input("Enter user ID: "))

            user = user\_service.getUser(userId)

            if user:

                print(user)

            else:

                print("User not found.")

        elif choice == '9':  # Get All Users

            users = user\_service.getAllUsers()

            for user in users:

                print(user)

        elif choice == '10':  # Update User

            userId = int(input("Enter user ID: "))

            username = input("Enter new username: ")

            password = input("Enter new password: ")

            role = input("Enter new role (admin/user): ")

            user = User(userId=userId, username=username, password=password, role=role)

            user\_service.updateUser(user)

            print("User updated successfully!")

        elif choice == '11':  # Delete User

            userId = int(input("Enter user ID: "))

            user\_service.deleteUser(userId)

            print("User deleted successfully!")

        elif choice == '12':  # Create Claim

            claimNumber = input("Enter claim number: ")

            dateFiled = input("Enter date filed (YYYY-MM-DD): ")

            claimAmount = float(input("Enter claim amount: "))

            status = input("Enter status: ")

            policy = input("Enter associated policy: ")

            clientId = input("Enter associated client: ")

            claim = Claim(claimNumber=claimNumber, dateFiled=dateFiled, claimAmount=claimAmount, status=status,

                          policy=policy, clientId=clientId)

            claim\_service.createClaim(claim)

            print("Claim created successfully!")

        elif choice == '13':  # Get Claim

            claimId = int(input("Enter claim ID: "))

            claim = claim\_service.getClaim(claimId)

            if claim:

                print(claim)

            else:

                print("Claim not found.")

        elif choice == '14':  # Get All Claims

            claims = claim\_service.getAllClaims()

            for claim in claims:

                print(claim)

        elif choice == '15':  # Create Payment

            paymentDate = input("Enter payment date (YYYY-MM-DD): ")

            paymentAmount = float(input("Enter payment amount: "))

            client = input("Enter associated client: ")

            payment = Payment(paymentDate=paymentDate, paymentAmount=paymentAmount, client=client)

            payment\_service.createPayment(payment)

            print("Payment created successfully!")

        elif choice == '16':  # Get Payment

            paymentId = int(input("Enter payment ID: "))

            payment = payment\_service.getPayment(paymentId)

            if payment:

                print(payment)

            else:

                print("Payment not found.")

        elif choice == '17':  # Exit

            print("Exiting...")

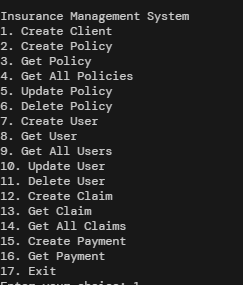
            break

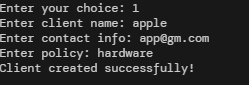
        else:

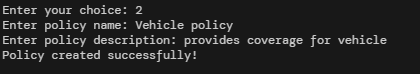
            print("Invalid choice! Please try again.")

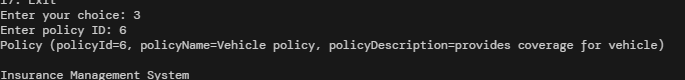
**OUTPUT**

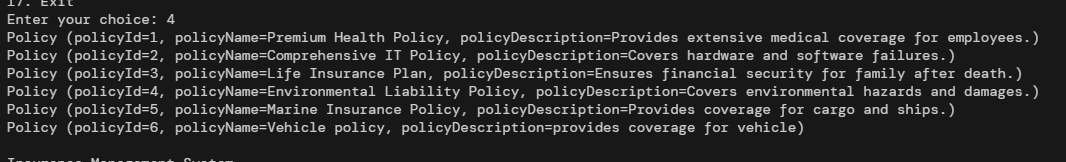
**Menu**

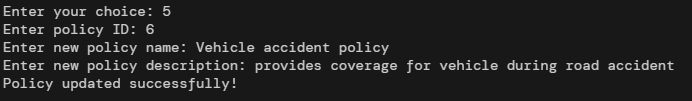
****

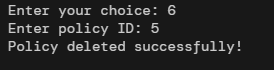
****

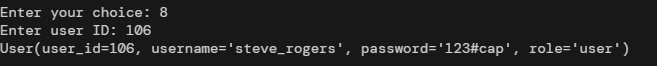
****

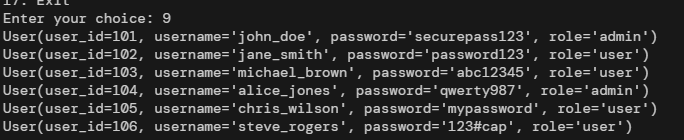
****

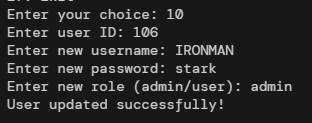
****

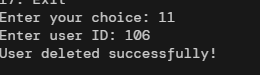
****

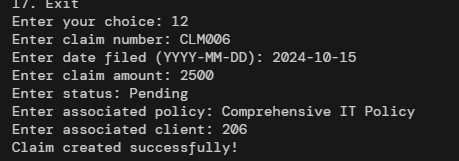
****

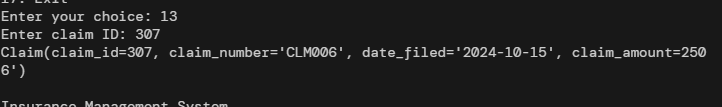
****

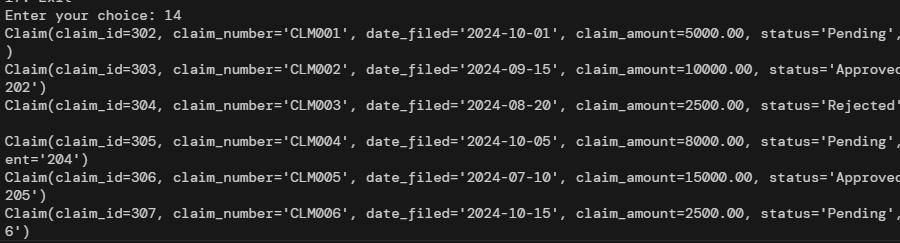
****

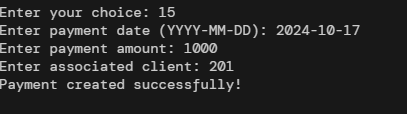
****

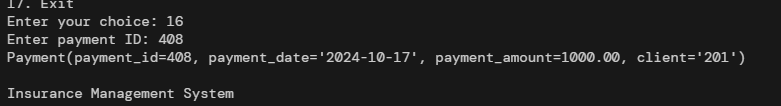
****

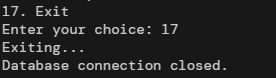
****

****

****

****

****

****