

INSURANCE MANAGEMENT SYSTEM

- 1 Define `User` class with the following confidential attributes: a. userId; b. username; c. password; d. role;

```
Creation_user.py > User > connect_to_database
1 import mysql.connector
2
3 class User:
4     def __init__(self, userId=None, username=None, password=None, role=None):
5         self.__userId = userId
6         self.__username = username
7         self.__password = password
8         self.__role = role
9
10    # Getters and setters
11
12    def get_userId(self):
13        return self.__userId
14
15    def set_userId(self, userId):
16        self.__userId = userId
17
18    def get_username(self):
19        return self.__username
20
21    def set_username(self, username):
22        self.__username = username
23
24    def get_password(self):
25        return self.__password
26
27    def set_password(self, password):
28        self.__password = password
29
30    def get_role(self):
31        return self.__role
32
33    def set_role(self, role):
34        self.__role = role
35
36    # Database operations
37
38    @staticmethod
39    def connect_to_database():
40        try:
41            return mysql.connector.connect(
42                host="localhost",
43                port="3306",
44                user="root",
45                password="Mghv@1725",
46                database="insurance"
47            )
48        except mysql.connector.Error as e:
49            print(f"Error connecting to MySQL database: {e}")
50            return None
51
52    def save_to_database(self):
53        try:
54            connection = User.connect_to_database()
55            if connection:
56                cursor = connection.cursor()
57                sql = "INSERT INTO User (userId, username, password, role) VALUES (%s, %s, %s, %s)"
58                values = (self.__userId, self.__username, self.__password, self.__role)
59                cursor.execute(sql, values)
60                connection.commit()
61                print("User saved to database successfully.")
62                cursor.close()
63            connection.close()
```

```
Creation_user.py 7 User 7 connect_to_database
3 class User:
52 def save_to_database(self):
62 cursor.close()
63 connection.close()
64 else:
65 print("Failed to connect to the database.")
66 except mysql.connector.error as e:
67 print("Error saving user to database: {e}")
68
69 def __str__(self):
70 return f"User(userId={self.__userId}, username={self.__username}, password={self.__password}, role={self.__role})"
71
72 id = int(input("Enter your Id : "))
73 name = input("Enter your name : ")
74 pas = input("Enter your password : ")
75 role = input("Enter your role : ")
76 user1 = User(userId=id, username=name, password=pas, role=role)
77 user1.save_to_database()
78

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

ModuleNotFoundError: No module named 'Service'
PS C:\Users\Balaji\Desktop\hexa\New folder\ass 1\Insurance & C:\Users\Balaji\AppData\Local\Microsoft\WindowsApps\python3.12.exe "C:\Users\Balaji\Desktop\hexa\New folder\ass 1\Insurance\Creation_user.py"
Enter your Id : 123
Enter your name : PG
Enter your password : pgh@1725
Enter your role : user
User saved to database successfully.
PS C:\Users\Balaji\Desktop\hexa\New folder\ass 1\Insurance >
```

2. Define `Client` class with the following confidential attributes: a. clientId; b. clientName; c. contactInfo; d. policy; //Represents the policy associated with the client

```

Creation_client.py > Client > connect_to_database
1 import mysql.connector
2
3 class Client:
4     def __init__(self, clientId=None, clientName=None, contactInfo=None, policy=None):
5         self.__clientId = clientId
6         self.__clientName = clientName
7         self.__contactInfo = contactInfo
8         self.__policy = policy
9
10    def get_clientId(self):
11        return self.__clientId
12
13    def set_clientId(self, clientId):
14        self.__clientId = clientId
15
16    def get_clientName(self):
17        return self.__clientName
18
19    def set_clientName(self, clientName):
20        self.__clientName = clientName
21
22    def get_contactInfo(self):
23        return self.__contactInfo
24
25    def set_contactInfo(self, contactInfo):
26        self.__contactInfo = contactInfo
27
28    def get_policy(self):
29        return self.__policy
30
31    def set_policy(self, policy):
32        self.__policy = policy
33
34    @staticmethod
35    def connect_to_database():
36        try:
37            return mysql.connector.connect(
38                host="localhost",
39                port="3306",
40                user="root",
41                password="Mghv@1725",
42                database="insurance"
43            )
44        except mysql.connector.Error as e:
45            print(f"Error connecting to MySQL database: {e}")
46            return None
47
48    def save_to_database(self):
49        try:
50            connection = Client.connect_to_database()
51            if connection:
52                cursor = connection.cursor()
53                sql = "INSERT INTO Client (clientId, clientName, contactInfo, policy) VALUES (%s, %s, %s, %s)"

```

```
Creation_client.py > Client > connect_to_database
3 class Client:
4     def save_to_database(self):
5         cursor = connection.cursor()
6         sql = "INSERT INTO Client (clientId, clientName, contactInfo, policy) VALUES (%s, %s, %s, %s)"
7         values = (self.__clientId, self.__clientName, self.__contactInfo, self.__policy)
8         cursor.execute(sql, values)
9         connection.commit()
10        print("Client saved to database successfully.")
11        cursor.close()
12    connection.close()
13    else:
14        print("Failed to connect to the database.")
15    except mysql.connector.Error as e:
16        print(f"Error saving client to database: {e}")
17
18    def __str__(self):
19        return f"Client(clientId={self.__clientId}, clientName={self.__clientName}, contactInfo={self.__contactInfo}, policy={self.__policy})"
20
21    clientId = int(input("Enter Client ID: "))
22    clientName = input("Enter Client Name: ")
23    contactInfo = input("Enter Contact Info: ")
24    policy = input("Enter Policy: ")
25    client1 = Client(clientId=clientId, clientName=clientName, contactInfo=contactInfo, policy=policy)
26    client1.save_to_database()
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\balaji\Desktop\New folder\ass 1\Insurance> & "C:\Users\balaji\AppData\Local\Microsoft\WindowsApps\python3.12.exe" "C:\Users\balaji\Desktop\New folder\ass 1\Insurance\Creation_client.py"
Enter Client ID: 123
Enter Client Name: MURUGA
Enter Contact Info: 8867888996
Enter Policy: 156
Client saved to database successfully.
PS C:\Users\balaji\Desktop\New folder\ass 1\Insurance>
```

3. Define `Claim` class with the following confidential attributes: a. claimId; b. claimNumber; c. dateFiled; d. claimAmount; e. status; f. policy; //Represents the policy associated with the claim g. client; // Represents the client associated with the claim

```

Creation_Claim.py > Claim > connect_to_database
1 import mysql.connector
2
3 class Claim:
4     def __init__(self, claimId=None, claimNumber=None, dateFiled=None, claimAmount=None, status=None, policy=None, client=None):
5         self.__claimId = claimId
6         self.__claimNumber = claimNumber
7         self.__dateFiled = dateFiled
8         self.__claimAmount = claimAmount
9         self.__status = status
10        self.__policy = policy
11        self.__client = client
12
13
14        def get_claimId(self):
15            return self.__claimId
16
17        def set_claimId(self, claimId):
18            self.__claimId = claimId
19
20        def get_claimNumber(self):
21            return self.__claimNumber
22
23        def set_claimNumber(self, claimNumber):
24            self.__claimNumber = claimNumber
25
26        def get_dateFiled(self):
27            return self.__dateFiled
28
29        def set_dateFiled(self, dateFiled):
30            self.__dateFiled = dateFiled
31
32        def get_claimAmount(self):
33            return self.__claimAmount
34
35        def set_claimAmount(self, claimAmount):
36            self.__claimAmount = claimAmount
37
38        def get_status(self):
39            return self.__status
40
41        def set_status(self, status):
42            self.__status = status
43
44        def get_policy(self):
45            return self.__policy
46
47        def set_policy(self, policy):
48            self.__policy = policy
49
50        def get_client(self):
51            return self.__client
52

```

```

Creation_Claim.py > Claim > connect_to_database
3 class Claim:
52
53     def set_client(self, client):
54         self.__client = client
55
56
57     @staticmethod
58     def connect_to_database():
59         try:
60             return mysql.connector.connect(
61                 host="localhost",
62                 port="3306",
63                 user="root",
64                 password="hghv@1725",
65                 database="insurance"
66             )
67         except mysql.connector.Error as e:
68             print(f"Error connecting to MySQL database: {e}")
69             return None
70
71     def save_to_database(self):
72         try:
73             connection = Claim.connect_to_database()
74             if connection:
75                 cursor = connection.cursor()
76                 sql = "INSERT INTO Claim (claimId, claimNumber, dateFiled, claimAmount, status, policy, client) VALUES (%s, %s, %s, %s, %s, %s, %s)"
77                 values = (self.__claimId, self.__claimNumber, self.__dateFiled, self.__claimAmount, self.__status, self.__policy, self.__client)
78                 cursor.execute(sql, values)
79                 connection.commit()
80                 print("Claim saved to database successfully.")
81                 cursor.close()
82                 connection.close()
83             else:
84                 print("Failed to connect to the database.")
85         except mysql.connector.Error as e:
86             print(f"Error saving claim to database: {e}")
87
88     def __str__(self):
89         return f"Claim(claimId={self.__claimId}, claimNumber={self.__claimNumber}, dateFiled={self.__dateFiled}, claimAmount={self.__claimAmount}, status={self.__status}, policy={self.__policy}, client={self.__client})"
90
91 claimId = input("Enter Claim ID: ")
92 claimNumber = input("Enter Claim Number: ")
93 dateFiled = input("Enter Date Filed (YYYY-MM-DD): ")
94 claimAmount = float(input("Enter Claim Amount: "))
95 status = input("Enter Status: ")
96 policy = input("Enter Policy: ")
97 client = input("Enter Client: ")
98
99 claim1 = Claim(claimId=claimId, claimNumber=claimNumber, dateFiled=dateFiled, claimAmount=claimAmount, status=status, policy=policy, client=client)
100
101 claim1.save_to_database()
102

```

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Balaji\Desktop\hexa\new folder\ass 1\Insurance> & C:\Users\Balaji\AppData\Local\Microsoft\WindowsApps\python8.12.exe "C:\Users\Balaji\Desktop\hexa\new folder\ass 1\Insurance\Creation_Claim.py"
Enter Claim ID: 123
Enter Claim Number: 8667368996
Enter Date Filed (YYYY-MM-DD): 2024-05-01
Enter Claim Amount: 1000
Enter Status: claimed
Enter Policy: 123
Enter Client: 1
Claim saved to database successfully.
PS C:\Users\Balaji\Desktop\hexa\new folder\ass 1\Insurance>

```

4.. Define `Claim` class with the following confidential attributes: a. paymentId; b. paymentDate; c. paymentAmount; d. client; // Represents the client associated with the payment

```
Creation_payment.py > Payment
1  import mysql.connector
2
3  class Payment:
4      def __init__(self, paymentId=None, paymentDate=None, paymentAmount=None, client=None):
5          self.__paymentId = paymentId
6          self.__paymentDate = paymentDate
7          self.__paymentAmount = paymentAmount
8          self.__client = client
9
10
11     def get_paymentId(self):
12         return self.__paymentId
13
14     def set_paymentId(self, paymentId):
15         self.__paymentId = paymentId
16
17     def get_paymentDate(self):
18         return self.__paymentDate
19
20     def set_paymentDate(self, paymentDate):
21         self.__paymentDate = paymentDate
22
23     def get_paymentAmount(self):
24         return self.__paymentAmount
25
26     def set_paymentAmount(self, paymentAmount):
27         self.__paymentAmount = paymentAmount
28
29     def get_client(self):
30         return self.__client
31
32     def set_client(self, client):
33         self.__client = client
34
35     @staticmethod
36     def connect_to_database():
37         try:
38             return mysql.connector.connect(
39                 host="localhost",
40                 port="3306",
41                 user="root",
42                 password="Mghv@1725",
43                 database="insurance"
44             )
45         except mysql.connector.Error as e:
46             print(f"Error connecting to MySQL database: {e}")
47             return None
48
49     def save_to_database(self):
50         try:
51             connection = Payment.connect_to_database()
52             if connection:
```

```
Creation_payment.py > Payment
3 class Payment:
36     def connect_to_database():
46         print(f"Error connecting to MySQL database: {e}")
47         return None
48
49     def save_to_database(self):
50         try:
51             connection = Payment.connect_to_database()
52             if connection:
53                 cursor = connection.cursor()
54                 sql = "INSERT INTO Payment (paymentId, paymentDate, paymentAmount, client) VALUES (%s, %s, %s, %s)"
55                 values = (self.__paymentId, self.__paymentDate, self.__paymentAmount, self.__client)
56                 cursor.execute(sql, values)
57                 connection.commit()
58                 print("Payment saved to database successfully.")
59                 cursor.close()
60                 connection.close()
61             else:
62                 print("Failed to connect to the database.")
63         except mysql.connector.Error as e:
64             print(f"Error saving payment to database: {e}")
65
66     def __str__(self):
67         return f"Payment(paymentId={self.__paymentId}, paymentDate={self.__paymentDate}, paymentAmount={self.__paymentAmount}, client={self.__client})"
68
69 paymentId = int(input("Enter Payment ID: "))
70 paymentDate = input("Enter Payment Date (YYYY-MM-DD): ")
71 paymentAmount = float(input("Enter Payment Amount: "))
72 client = input("Enter Client: ")
73
74 payment1 = Payment(paymentId=paymentId, paymentDate=paymentDate, paymentAmount=paymentAmount, client=client)
75
76 payment1.save_to_database()
77
78
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Balaji\Desktop\hexa\New folder\ass 1\Insurance> & C:\Users\Balaji\AppData\Local\Microsoft\WindowsApps\python3.12.exe "c:\Users\Balaji\Desktop\hexa\New folder\ass 1\Insurance\Creation_payment.py"
Enter Payment ID: 125
Enter Payment Date (YYYY-MM-DD): 2024-05-01
Enter Payment Amount: 1000
Enter Client: 1
Payment saved to database successfully.
PS C:\Users\Balaji\Desktop\hexa\New folder\ass 1\Insurance>
```

5. Define IPolicyService interface/abstract class with following methods to interact with database Keep the interfaces and implementation classes in package dao
- a. createPolicy() I. parameters: Policy Object II. return type: boolean

```
User.py > ...
1  from mysql.connector import connect, Error
2  from typing import List
3
4  class Policy:
5      def __init__(self, policy_id, policy_name, Coverage_Amount, start_date, end_date):
6          self.policy_id = policy_id
7          self.policy_name = policy_name
8          self.Coverage_Amount = Coverage_Amount
9          self.start_date = start_date
10         self.end_date = end_date
11
12
13     class PolicyServiceImpl:
14         def __init__(self):
15             try:
16                 self.conn = connect(
17                     host="localhost",
18                     port="3306",
19                     user="root",
20                     password="Mghv@1725",
21                     database="insurance"
22                 )
23                 self.cursor = self.conn.cursor()
24                 print("Connected to MySQL server")
25             except Error as e:
26                 print(f"Error connecting to MySQL server: {e}")
27
28         def create_policy(self, policy) -> bool:
29             try:
30                 sql = "INSERT INTO policy (policyid, policyname, Coverageamount, startdate, enddate) VALUES (%s, %s, %s, %s, %s)"
31                 values = (policy.policy_id, policy.policy_name, policy.Coverage_Amount, policy.start_date, policy.end_date)
32                 self.cursor.execute(sql, values)
33                 self.conn.commit()
34                 return True
35             except Error as e:
36                 print(f"Error creating policy: {e}")
37                 self.conn.rollback()
38                 return False
39
40         def get_policy(self, policy_id) -> Policy:
41             sql = "SELECT * FROM policy WHERE policyid = %s"
42             self.cursor.execute(sql, (policy_id,))
43             result = self.cursor.fetchone()
44             if result:
45                 return Policy(*result)
46             else:
47                 return None
48
49         def get_all_policies(self) -> List[Policy]:
50             sql = "SELECT * FROM policy"
51             self.cursor.execute(sql)
52             result = self.cursor.fetchall()
53
54 PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
55
56 PS C:\Users\Balaji\Desktop\hexa\New folder\ass 1\Insurance> & C:\Users\Balaji\AppData\Local\Microsoft\WindowsApps\python3.12.exe "c:\Users\Balaji\Desktop\hexa\New folder\ass 1\Insurance\User.py"
57 Connected to MySQL server
58 1 - Create policy
59 2 - Read Policy
60 3 - Update Policy
61 4 - Delete Policy
62 5 - Get all policies detail
63 Enter your choice : 1
64 Enter policy id : 145
65 Enter policy name : Hv Health policy
66 Enter coverage amount : 150000
67 PS C:\Users\Balaji\Desktop\hexa\New folder\ass 1\Insurance> |
```

b. getPolicy() I. parameters: policyId II. return type: Policy Object


```
User.py > ...
1 from mysql.connector import connect, Error
2 from typing import List
3
4 class Policy:
5     def __init__(self, policy_id, policy_name, Coverage_Amount, start_date, end_date):
6         self.policy_id = policy_id
7         self.policy_name = policy_name
8         self.Coverage_Amount = Coverage_Amount
9         self.start_date = start_date
10        self.end_date = end_date
11
12
13 class PolicyServiceImpl:
14     def __init__(self):
15         try:
16             self.conn = connect(
17                 host="localhost",
18                 port="3306",
19                 user="root",
20                 password="Mghv@1725",
21                 database="insurance"
22             )
23             self.cursor = self.conn.cursor()
24             print("Connected to MySQL server")
25         except Error as e:
26             print(f"Error connecting to MySQL server: {e}")
27
28     def create_policy(self, policy) -> bool:
29         try:
30             sql = "INSERT INTO policy (policyid, polycname, Coverageamount, startdate, enddate) VALUES (%s, %s, %s, %s, %s)"
31             values = (policy.policy_id, policy.policy_name, policy.Coverage_Amount, policy.start_date, policy.end_date)
32             self.cursor.execute(sql, values)
33             self.conn.commit()
34             return True
35         except Error as e:
36             print(f"Error creating policy: {e}")
37             self.conn.rollback()
38             return False
39
40     def get_policy(self, policy_id) -> Policy:
41         sql = "SELECT * FROM policy WHERE policyid = %s"
42         self.cursor.execute(sql, (policy_id,))
43         result = self.cursor.fetchone()
44         if result:
45             return Policy(*result)
46         else:
47             return None
48
49     def get_all_policies(self) -> List[Policy]:
50         sql = "SELECT * FROM policy"
51         self.cursor.execute(sql)
52         result = self.cursor.fetchall()
53
54 PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
55
56 PS C:\Users\Balaji\Desktop\hexa\New folder\ass 1\Insurance> & C:\Users\Balaji\AppData\Local\Microsoft\WindowsApps\python3.12.exe "c:/Users/Balaji/Desktop/hexa/New folder/ass 1/Insurance/User.py"
Connected to MySQL server
1 - Create policy
2 - Read Policy
3 - Update Policy
4 - Delete Policy
5 - Get all policies detail
Enter your choice : 2
Enter policy id : 145
{'policy_id': 145, 'policy_name': 'Hv Health policy', 'Coverage_Amount': Decimal('150000.00'), 'start_date': datetime.date(2024, 1, 1), 'end_date': datetime.date(2025, 1, 1)}
PS C:\Users\Balaji\Desktop\hexa\New folder\ass 1\Insurance>
```

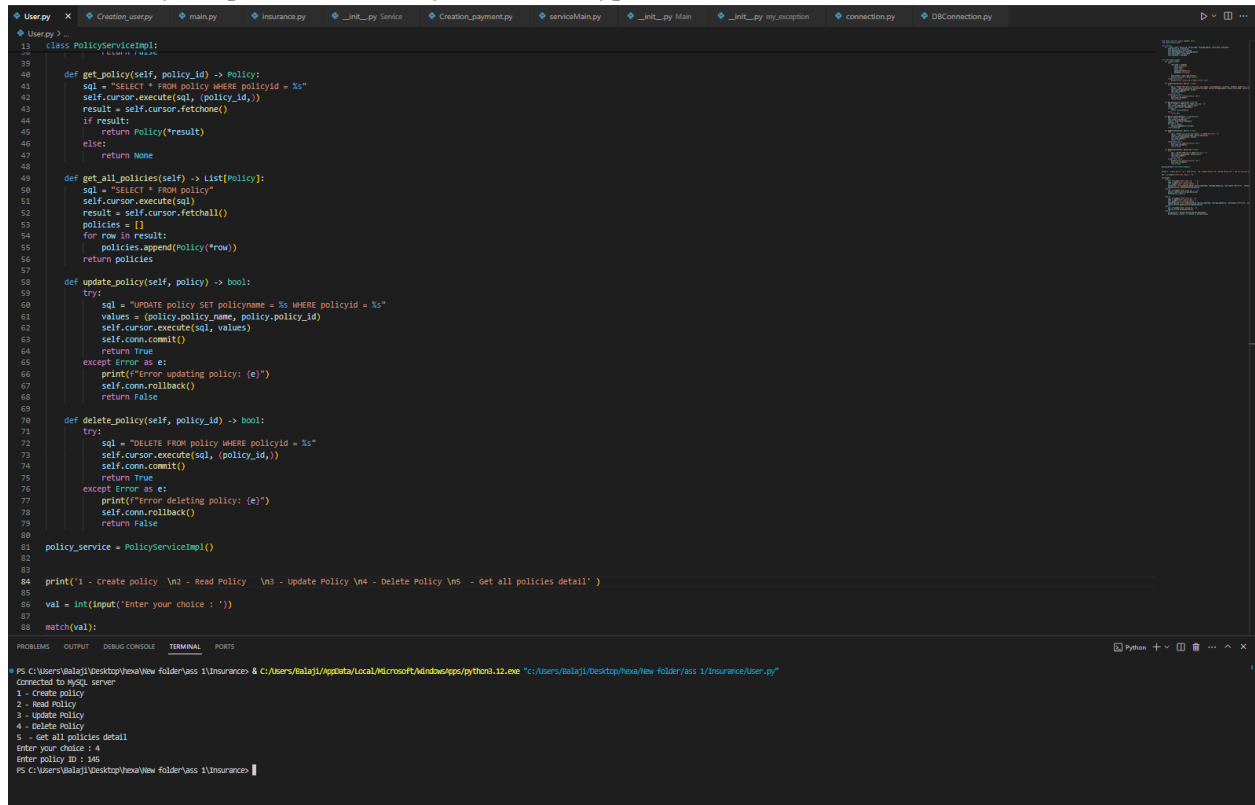
c.getAllPolicies() I. parameters: none II. return type: Collection of Policy Objects

```
User.py x  Creation_user.py  main.py  Insurance.py  __init__.py Service  Creation_payment.py  serviceMain.py  __init__.py Main  __init__.py my_exception  connection.py  DBConnection.py
13 class PolicyServiceImpl:
14     def __init__(self):
15         self.cursor = None
16         self.conn = None
17
18     def get_policy(self, policy_id) -> Policy:
19         sql = "SELECT * FROM policy WHERE policyid = %s"
20         self.cursor.execute(sql, (policy_id,))
21         result = self.cursor.fetchone()
22         if result:
23             return Policy(*result)
24         else:
25             return None
26
27     def get_all_policies(self) -> List[Policy]:
28         sql = "SELECT * FROM policy"
29         self.cursor.execute(sql)
30         result = self.cursor.fetchall()
31         policies = []
32         for row in result:
33             policies.append(Policy(*row))
34         return policies
35
36     def update_policy(self, policy) -> bool:
37         try:
38             sql = "UPDATE policy SET policyname = %s WHERE policyid = %s"
39             values = (policy.policy_name, policy.policy_id)
40             self.cursor.execute(sql, values)
41             self.conn.commit()
42             return True
43         except Error as e:
44             print("Error updating policy: (e)")
45             self.conn.rollback()
46             return False
47
48     def delete_policy(self, policy_id) -> bool:
49         try:
50             sql = "DELETE FROM policy WHERE policyid = %s"
51             self.cursor.execute(sql, (policy_id,))
52             self.conn.commit()
53             return True
54         except Error as e:
55             print("Error deleting policy: (e)")
56             self.conn.rollback()
57             return False
58
59 policy_service = PolicyServiceImpl()
60
61 print('1 - Create policy \n2 - Read Policy \n3 - update Policy \n4 - Delete Policy \n5 - get all policies detail')
62
63 val = int(input('Enter your choice : '))
64
65 match(val):
66     case 1:
67         policy_name = input('Enter policy name: ')
68         policy_id = input('Enter policy id: ')
69         policy_service.get_policy(policy_id)
70     case 2:
71         policy_name = input('Enter policy name: ')
72         policy_id = input('Enter policy id: ')
73         policy_service.get_policy(policy_id)
74     case 3:
75         policy_name = input('Enter policy name: ')
76         policy_id = input('Enter policy id: ')
77         policy_service.update_policy(policy_id)
78     case 4:
79         policy_id = input('Enter policy id: ')
80         policy_service.delete_policy(policy_id)
81     case 5:
82         policy_service.get_all_policies()
83
84 # Output:
85 1 - Create policy
86 2 - Read Policy
87 3 - update Policy
88 4 - Delete Policy
89 5 - get all policies detail
90 Enter your choice : 5
91 [{"policy_id": 2, "policy_name": "Accident policy", "coverage_amount": Decimal("200000.00"), "start_date": datetime.date(2024, 1, 1), "end_date": datetime.date(2025, 1, 1)}, {"policy_id": 54, "policy_name": "star life", "coverage_amount": Decimal("2100000.00"), "start_date": datetime.date(2024, 1, 1), "end_date": datetime.date(2025, 1, 1)}, {"policy_id": 146, "policy_name": "W health policy", "coverage_amount": Decimal("150000.00"), "start_date": datetime.date(2024, 1, 1), "end_date": datetime.date(2025, 1, 1)}, {"policy_id": 256, "policy_name": "W insurance", "coverage_amount": Decimal("150000.00"), "start_date": datetime.date(2024, 1, 1), "end_date": datetime.date(2025, 1, 1)}, {"policy_id": 305, "policy_name": "mur unsure", "coverage_amount": Decimal("15000.00"), "start_date": datetime.date(2024, 1, 1), "end_date": datetime.date(2025, 1, 1)}]
```

d.updatePolicy() I. parameters: Policy Object II. return type: Boolean

```
User.py x  Creation_user.py  main.py  Insurance.py  __init__.py Service  Creation_payment.py  serviceMain.py  __init__.py Main  __init__.py my_exception  connection.py  DBConnection.py
13 class PolicyServiceImpl:
14     def __init__(self):
15         self.cursor = None
16         self.conn = None
17
18     def get_policy(self, policy_id) -> Policy:
19         sql = "SELECT * FROM policy WHERE policyid = %s"
20         self.cursor.execute(sql, (policy_id,))
21         result = self.cursor.fetchone()
22         if result:
23             return Policy(*result)
24         else:
25             return None
26
27     def get_all_policies(self) -> List[Policy]:
28         sql = "SELECT * FROM policy"
29         self.cursor.execute(sql)
30         result = self.cursor.fetchall()
31         policies = []
32         for row in result:
33             policies.append(Policy(*row))
34         return policies
35
36     def update_policy(self, policy) -> bool:
37         try:
38             sql = "UPDATE policy SET policyname = %s WHERE policyid = %s"
39             values = (policy.policy_name, policy.policy_id)
40             self.cursor.execute(sql, values)
41             self.conn.commit()
42             return True
43         except Error as e:
44             print("Error updating policy: (e)")
45             self.conn.rollback()
46             return False
47
48     def delete_policy(self, policy_id) -> bool:
49         try:
50             sql = "DELETE FROM policy WHERE policyid = %s"
51             self.cursor.execute(sql, (policy_id,))
52             self.conn.commit()
53             return True
54         except Error as e:
55             print("Error deleting policy: (e)")
56             self.conn.rollback()
57             return False
58
59 policy_service = PolicyServiceImpl()
60
61 print('1 - Create policy \n2 - Read Policy \n3 - update Policy \n4 - Delete Policy \n5 - get all policies detail')
62
63 val = int(input('Enter your choice : '))
64
65 match(val):
66     case 1:
67         policy_name = input('Enter policy name: ')
68         policy_id = input('Enter policy id: ')
69         policy_service.get_policy(policy_id)
70     case 2:
71         policy_name = input('Enter policy name: ')
72         policy_id = input('Enter policy id: ')
73         policy_service.get_policy(policy_id)
74     case 3:
75         policy_name = input('Enter policy name: ')
76         policy_id = input('Enter policy id: ')
77         policy_service.update_policy(policy_id)
78     case 4:
79         policy_id = input('Enter policy id: ')
80         policy_service.delete_policy(policy_id)
81     case 5:
82         policy_service.get_all_policies()
83
84 # Output:
85 1 - Create policy
86 2 - Read Policy
87 3 - update Policy
88 4 - Delete Policy
89 5 - get all policies detail
90 Enter your choice : 3
91 Enter policy id : 146
92 Enter policy name : W health policy
93 Enter coverage amount : 140000
94 PS C:\Users\Balaji\Desktop\new folder> python3.12.exe "C:\Users\Balaji\Desktop\new folder\ss 1\Insurance\User.py"
```

e. deletePolicy() I. parameters: PolicyId II. return type: Boolean



```
12 class PolicyServiceImpl:
13     def get_policy(self, policy_id) -> Policy:
14         sql = "SELECT * FROM policy WHERE policyid = %s"
15         self.cursor.execute(sql, (policy_id,))
16         result = self.cursor.fetchone()
17         if result:
18             return Policy(*result)
19         else:
20             return None
21
22     def get_all_policies(self) -> List[Policy]:
23         sql = "SELECT * FROM policy"
24         self.cursor.execute(sql)
25         result = self.cursor.fetchall()
26         policies = []
27         for row in result:
28             policies.append(Policy(*row))
29         return policies
30
31     def update_policy(self, policy) -> bool:
32         try:
33             sql = "UPDATE policy SET policyname = %s WHERE policyid = %s"
34             values = (policy.policy_name, policy.policy_id)
35             self.cursor.execute(sql, values)
36             self.conn.commit()
37             return True
38         except Error as e:
39             print("Error updating policy: (e)")
40             self.conn.rollback()
41             return False
42
43     def delete_policy(self, policy_id) -> bool:
44         try:
45             sql = "DELETE FROM policy WHERE policyid = %s"
46             self.cursor.execute(sql, (policy_id,))
47             self.conn.commit()
48             return True
49         except Error as e:
50             print("Error deleting policy: (e)")
51             self.conn.rollback()
52             return False
53
54 policy_service = PolicyServiceImpl()
55
56 print('1 - Create policy \n2 - Read Policy \n3 - Update Policy \n4 - Delete Policy \n5 - get all policies detail' )
57
58 val = int(input("Enter your choice : "))
59
60 match(val):
61     case 1:
62         # Create policy
63     case 2:
64         # Read Policy
65     case 3:
66         # Update Policy
67     case 4:
68         # Delete Policy
69     case 5:
70         # get all policies detail
71
72         enter your choice : 4
73         enter policy ID : 145
74
75 PS C:\Users\balaji\Desktop\new folder\ass 1\insurance>
```

6. Define InsuranceServiceImpl class and implement all the methods InsuranceServiceImpl

```

Service > insurance.py > main
1 import mysql.connector
2 from typing import List
3
4 class Insurance:
5     def __init__(self, insurance_id, insurance_name):
6         self.insurance_id = insurance_id
7         self.insurance_name = insurance_name
8
9 class InsuranceServiceImpl:
10     def __init__(self, host, port, user, password, database):
11         self.conn = mysql.connector.connect(
12             host=host,
13             port=port,
14             user=user,
15             password=password,
16             database=database
17         )
18         self.cursor = self.conn.cursor()
19
20     def create_insurance(self, insurance) -> bool:
21         try:
22             sql = "INSERT INTO Insurance (insurance_id, insurance_name) VALUES (%s, %s)"
23             values = (insurance.insurance_id, insurance.insurance_name)
24             self.cursor.execute(sql, values)
25             self.conn.commit()
26             print("added")
27             return True
28         except mysql.connector.Error as e:
29             print(f"Error creating insurance: {e}")
30             self.conn.rollback()
31             return False
32
33     def get_insurance(self, insurance_id) -> Insurance:
34         sql = "SELECT * FROM Insurance WHERE insurance_id = %s"
35         self.cursor.execute(sql, (insurance_id,))
36         result = self.cursor.fetchone()
37         if result:
38             return Insurance(result[0], result[1])
39         else:
40             return None
41
42     def get_all_insurances(self) -> List[Insurance]:
43         sql = "SELECT * FROM Insurance"
44         self.cursor.execute(sql)
45         result = self.cursor.fetchall()
46         insurances = []
47         for row in result:
48             insurances.append(Insurance(row[0], row[1]))
49         return insurances
50
51     def update_insurance(self, insurance) -> bool:
52         try:
53             sql = "UPDATE Insurance SET insurance_name = %s WHERE insurance_id = %s"
54             values = (insurance.insurance_name, insurance.insurance_id)
55             self.cursor.execute(sql, values)
56             self.conn.commit()
57             print("updated")
58             return True
59         except mysql.connector.Error as e:
60             print(f"Error updating insurance: {e}")
61             self.conn.rollback()
62             return False
63
64     def delete_insurance(self, insurance_id) -> bool:
65         try:
66             sql = "DELETE FROM Insurance WHERE insurance_id = %s"
67             self.cursor.execute(sql, (insurance_id,))
68             self.conn.commit()
69             print("deleted")
70             return True
71         except mysql.connector.Error as e:
72             print(f"Error deleting insurance: {e}")
73             self.conn.rollback()
74             return False
75
76     def __str__(self):
77         return f"InsuranceServiceImpl(host={self.host}, port={self.port}, user={self.user}, password={self.password}, database={self.database})"
78
79 if __name__ == "__main__":
80     host = input("Enter host: ")
81     port = int(input("Enter port: "))
82     user = input("Enter user: ")
83     password = input("Enter password: ")
84     database = input("Enter database: ")
85     service = InsuranceServiceImpl(host, port, user, password, database)
86     menu = """
87 Insurance Service Menu:
88 1. Create Insurance
89 2. Get Insurance
90 3. Get All Insurances
91 4. Update Insurance
92 5. Delete Insurance
93 6. Exit
94 Enter your choice (1-6): """
95     while True:
96         choice = input(menu)
97         if choice == "1":
98             insurance_id = int(input("Enter Insurance ID: "))
99             insurance_name = input("Enter Insurance Name: ")
100             new_insurance = Insurance(insurance_id, insurance_name)
101             service.create_insurance(new_insurance)
102         elif choice == "2":
103             insurance_id = int(input("Enter Insurance ID to get: "))
104             insurance = service.get_insurance(insurance_id)
105             if insurance:
106                 print(insurance.__dict__)
107             else:
108                 print("Insurance not found.")
109         elif choice == "3":
110             all_insurances = service.get_all_insurances()
111             print([insurance.__dict__ for insurance in all_insurances])
112         elif choice == "4":
113             insurance_id = int(input("Enter Insurance ID to update: "))
114             insurance_name = input("Enter updated Insurance Name: ")
115             updated_insurance = Insurance(insurance_id, insurance_name)
116             service.update_insurance(updated_insurance)
117         elif choice == "5":
118             insurance_id = int(input("Enter Insurance ID to delete: "))
119             service.delete_insurance(insurance_id)
120         elif choice == "6":
121             print("Exiting program. Goodbye!")
122             break
123         else:
124             print("Invalid choice. Please enter a number from 1 to 6.")

```

```

Service > insurance.py > main
73 def main():
74     insurance_id = int(input("Enter Insurance ID: "))
75     insurance_name = input("Enter Insurance Name: ")
76     new_insurance = Insurance(insurance_id, insurance_name)
77     insurance_service.create_insurance(new_insurance)
78
79     if choice == "2":
80         insurance_id = int(input("Enter Insurance ID to get: "))
81         insurance = insurance_service.get_insurance(insurance_id)
82         if insurance:
83             print(insurance.__dict__)
84         else:
85             print("Insurance not found.")
86     elif choice == "3":
87         all_insurances = insurance_service.get_all_insurances()
88         print([insurance.__dict__ for insurance in all_insurances])
89     elif choice == "4":
90         insurance_id = int(input("Enter Insurance ID to update: "))
91         insurance_name = input("Enter updated Insurance Name: ")
92         updated_insurance = Insurance(insurance_id, insurance_name)
93         insurance_service.update_insurance(updated_insurance)
94     elif choice == "5":
95         insurance_id = int(input("Enter Insurance ID to delete: "))
96         insurance_service.delete_insurance(insurance_id)
97     elif choice == "6":
98         print("Exiting program. Goodbye!")
99         break
100     else:
101         print("Invalid choice. Please enter a number from 1 to 6.")
102
103 if __name__ == "__main__":
104     main()

```

7. Create a utility class DBConnection in a package util with a static variable connection of Type Connection and a static method getConnection() which returns connection. Connection properties supplied in the connection string should be read from a property file. Create a utility class PropertyUtil which contains a static method named getPropertyString() which reads a property file containing connection details like hostname, dbname, username, password, port number and returns a connection string.

```

connection.py > ...
1
2 from util.DBConnection import DBConnection
3 connection = DBConnection.getConnection()
4
5 connection.close()

```

```

util > DBConnection.py > DBConnection > getConnection
4 class DBConnection:
7     @staticmethod
8     def getConnection():
9         if DBConnection.connection is None:
10            properties = PropertyUtil.getPropertyString('connection.properties')
11            print(properties)
12            try:
13                DBConnection.connection = mysql.connector.connect(
14                    host=properties['hostname'],
15                    user=properties['username'],
16                    password=properties['password'],
17                    database=properties['dbname'],
18                    port=properties['port']
19                )
20                print("Database connected!")
21            except mysql.connector.Error as e:
22                print("Error connecting to database:", e)
23            return DBConnection.connection
24

```

```

util > property.py > PropertyUtil > getPropertyString
1 class PropertyUtil:
2     @staticmethod
3     def getPropertyString(file_path):
4         properties = {}
5         with open(r'C:\Users\Balaji\Desktop\hexa\New folder\ass 1\Insurance\util\connection_properties.txt', 'r') as file:
6
7             for line in file:
8                 if '=' in line:
9                     key, value = line.strip().split('=')
10                    properties[key.strip()] = value.strip()
11            return properties
12

```

```

PS C:\Users\Balaji\Desktop\hexa\New folder\ass 1\Insurance> & C:\Users\Balaji\AppData\Local\Microsoft\WindowsApps\python3.12.exe "c:/Users/Balaji/Desktop/hexa/New folder/ass 1/Insurance/connection.py"
{'hostname': 'localhost', 'dbname': 'insurance', 'username': 'root', 'password': 'Mgh@1725', 'port': '3386'}
Database connected!
PS C:\Users\Balaji\Desktop\hexa\New folder\ass 1\Insurance>

```

8. Create the exceptions in package myexceptions Define the following custom exceptions and throw them in methods whenever needed. Handle all the exceptions in main method, 1.
PolicyNotFoundException :throw this exception when user enters an invalid patient number which

doesn't exist in db

```
my_exception > exception.py > ...
1
2 class PolicyNotFoundException(Exception):
3     def __init__(self, policy_id):
4         super().__init__(f"Policy with ID {policy_id} not found.")
5         self.policy_id = policy_id
6
7
```

```
> __pycache__
v Main
  > __init__.py
  > serviceMain.py
v my_exception
  > __pycache__
  > __init__.py
  > exception.py
```

9. Create class named MainModule with main method in package mainmod. Trigger all the methods in service implementation class.

```
> __pycache__
v Main
  > __init__.py
  > serviceMain.py
v my_exception
  > __pycache__
  > __init__.py
  > exception.py
```

```
Main > serviceMain.py > ...
1  from Service.insurance import InsuranceServiceImpl
2
3  class Insurance:
4      def __init__(self, insurance_id, insurance_name):
5          self.insurance_id = insurance_id
6          self.insurance_name = insurance_name
7
8  class MainModule:
9      @staticmethod
10     def main():
11         policy_service = InsuranceServiceImpl()
12         while True:
13             print("\nInsurance Service Menu:")
14             print("1. Create Insurance")
15             print("2. Get Insurance")
16             print("3. Get All Insurances")
17             print("4. Update Insurance")
18             print("5. Delete Insurance")
19             print("6. Exit")
20
21             choice = input("Enter your choice (1-6): ")
22
23             if choice == "1":
24                 insurance_id = int(input("Enter Insurance ID: "))
25                 insurance_name = input("Enter Insurance Name: ")
26                 new_insurance = Insurance(insurance_id, insurance_name)
27                 policy_service.create_insurance(new_insurance)
28             elif choice == "2":
29                 insurance_id = int(input("Enter Insurance ID to get: "))
30                 insurance = policy_service.get_insurance(insurance_id)
31                 if insurance:
32                     print(insurance.__dict__)
33                 else:
34                     print("Insurance not found.")
35             elif choice == "3":
36                 all_insurances = policy_service.get_all_insurances()
37                 print([insurance.__dict__ for insurance in all_insurances])
38             elif choice == "4":
39                 insurance_id = int(input("Enter Insurance ID to update: "))
40                 insurance_name = input("Enter Updated Insurance Name: ")
41                 updated_insurance = Insurance(insurance_id, insurance_name)
42                 policy_service.update_insurance(updated_insurance)
43             elif choice == "5":
44                 insurance_id = int(input("Enter Insurance ID to delete: "))
45                 policy_service.delete_insurance(insurance_id)
46             elif choice == "6":
47                 print("Exiting program. Goodbye!")
48                 break
49             else:
50                 print("Invalid choice. Please enter a number from 1 to 6.")
51
52 if name == " main ":
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

6. Exit

Enter your choice (1-6): 2

Enter Insurance ID to get: 145

Insurance not found.

Insurance Service Menu:

1. Create Insurance
2. Get Insurance
3. Get All Insurances
4. Update Insurance
5. Delete Insurance
6. Exit

Enter your choice (1-6):

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

5. Delete Insurance

6. Exit

Enter your choice (1-6): 3

[{'insurance_id': 123, 'insurance_name': 'mg hw insurance'}]

Insurance Service Menu:

1. Create Insurance
2. Get Insurance
3. Get All Insurances
4. Update Insurance
5. Delete Insurance
6. Exit

Enter your choice (1-6):