**Coding Challenge: Insurance Management System**

Name : RISHATHA SHAHIN B

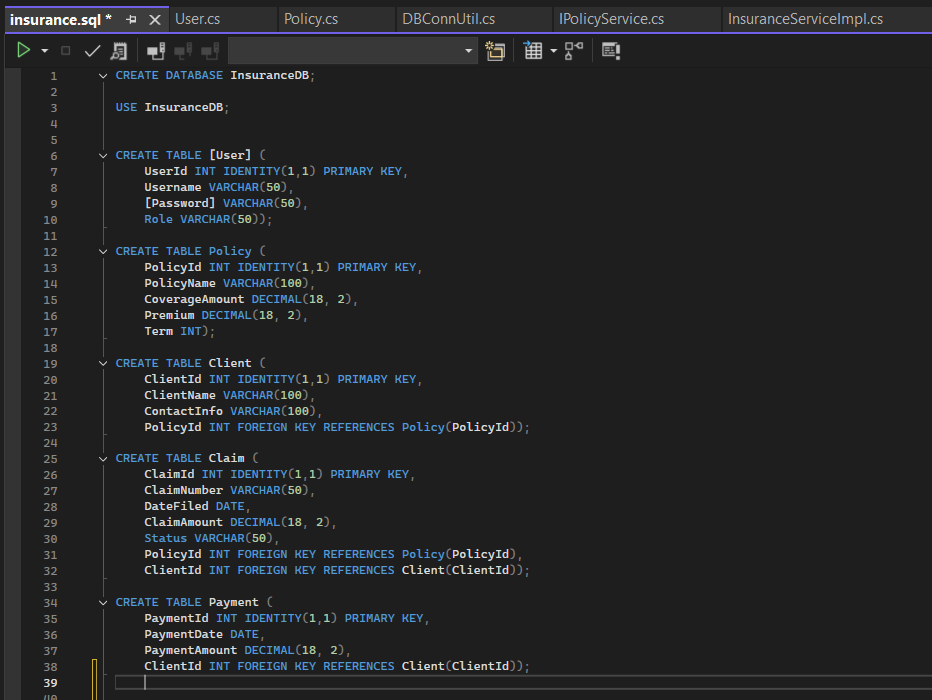
Superset ID : 5294964

Email : rishatha2004@gmail.com

Batch : C# Batch 4

Problem Statement:

Create SQL Schema from the following classes class, use the class attributes for table column names.



1.Create the following model/entity classes within package entity with variables declared private, constructors(default and parametrized,getters,setters and toString())

2. Implement the following for all model classes. Write default constructors and overload the constructor with parameters, getters and setters, method to print all the member variables and values.

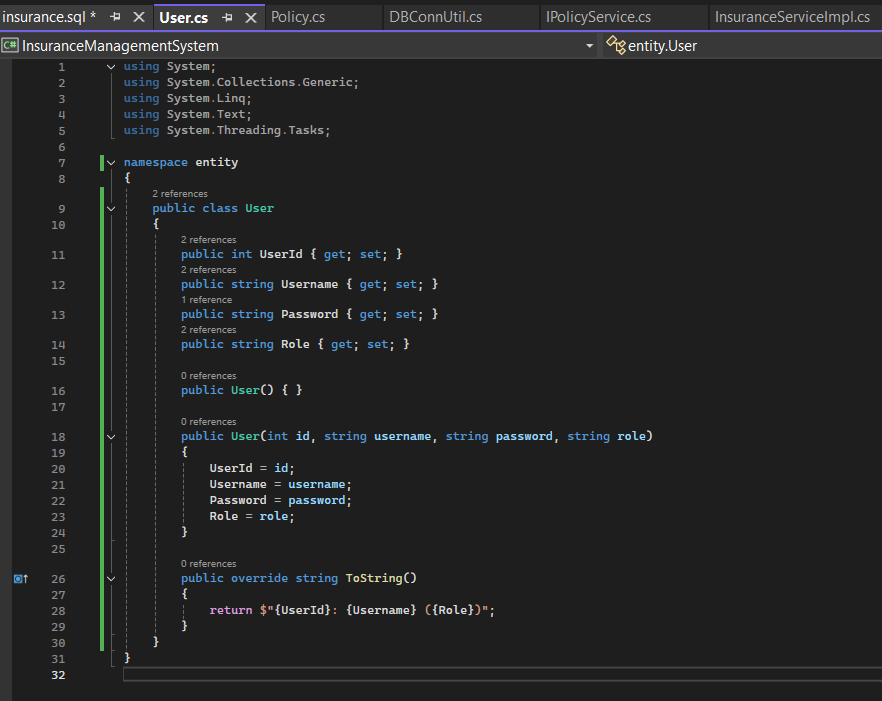
Define `User ` class with the following confidential attributes:

a. userId;

b. username;

c. password;

d. role;



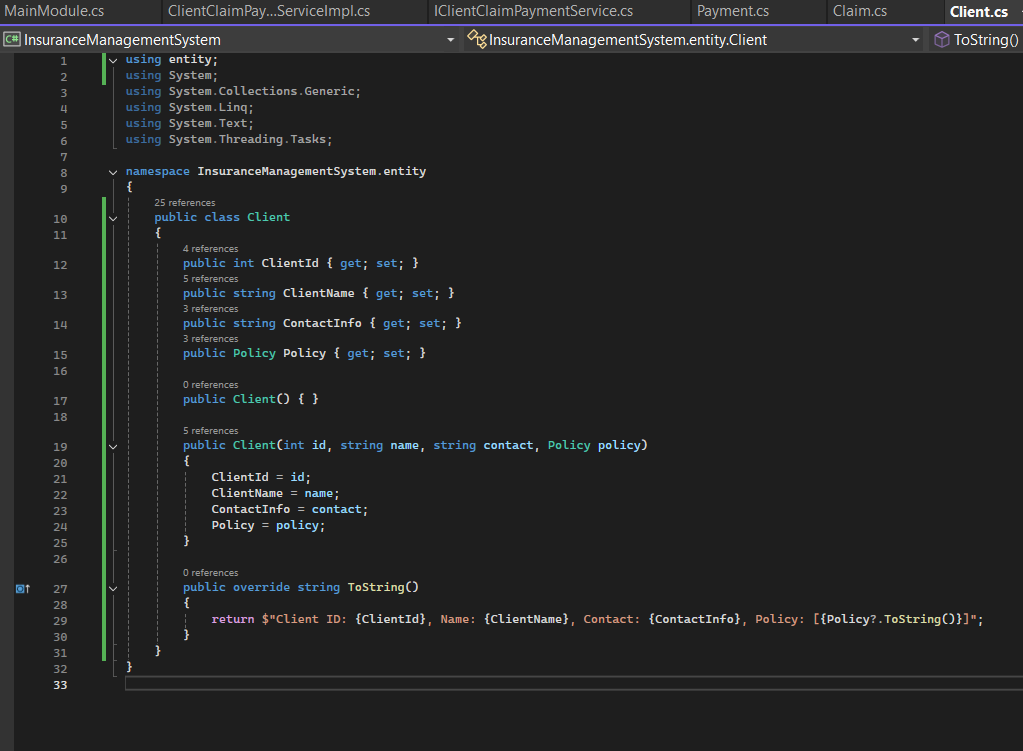
Define ` Client ` class with the following confidential attributes:

a. clientId;

b. clientName;

c. contactInfo;

d. policy;



Define ` Claim ` class with the following confidential attributes:

a. claimId;

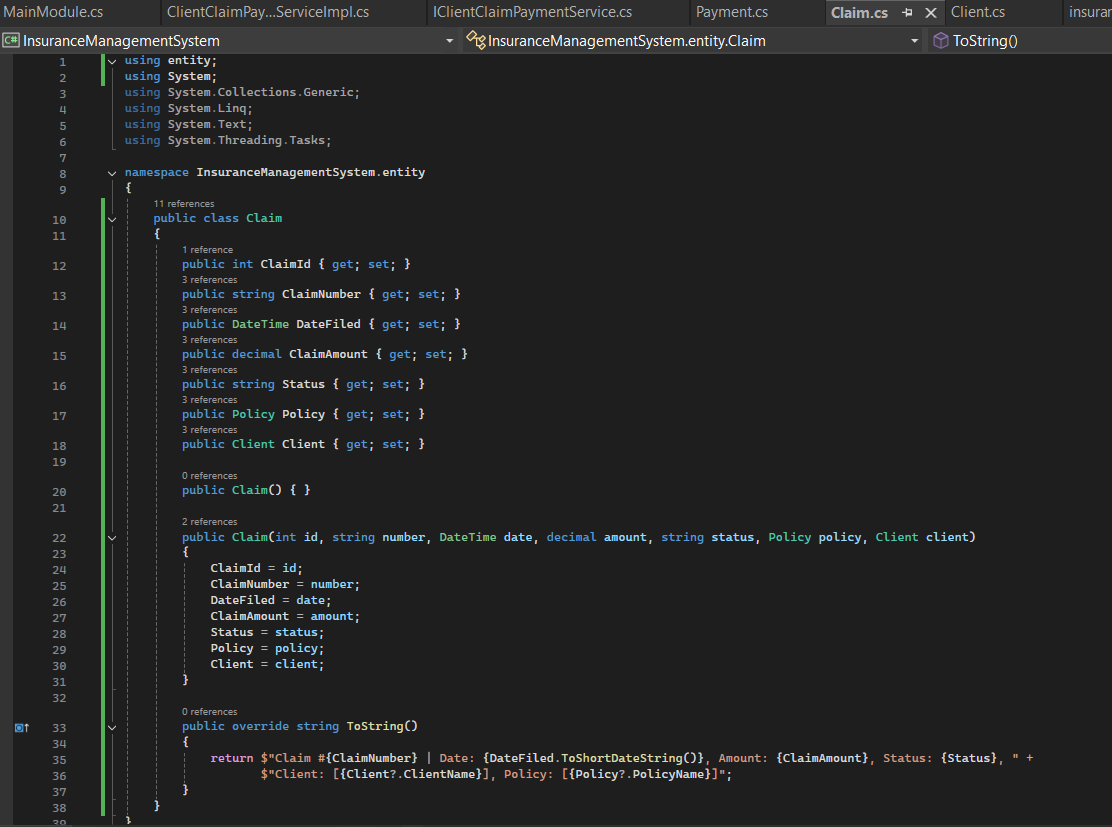
b. claimNumber;

c. dateFiled;

d. claimAmount;

e. status;

f. policy;



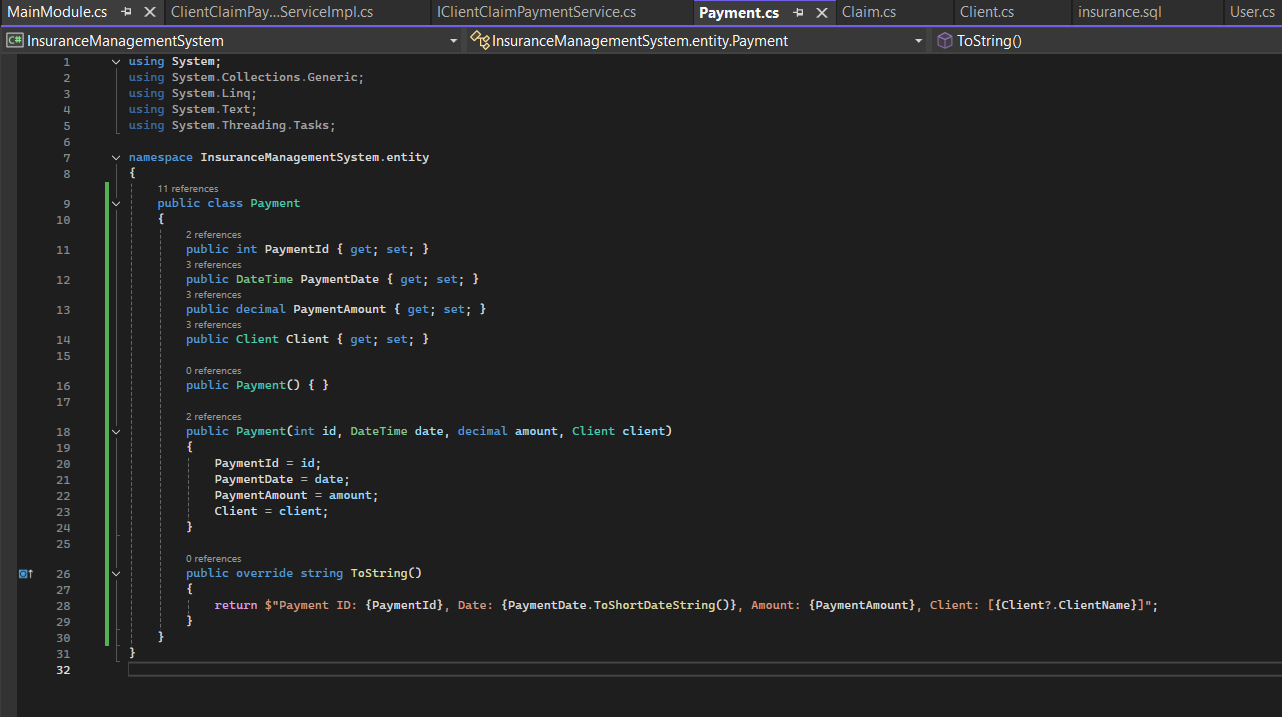
Define `payment ` class with the following confidential attributes:

a. paymentId;

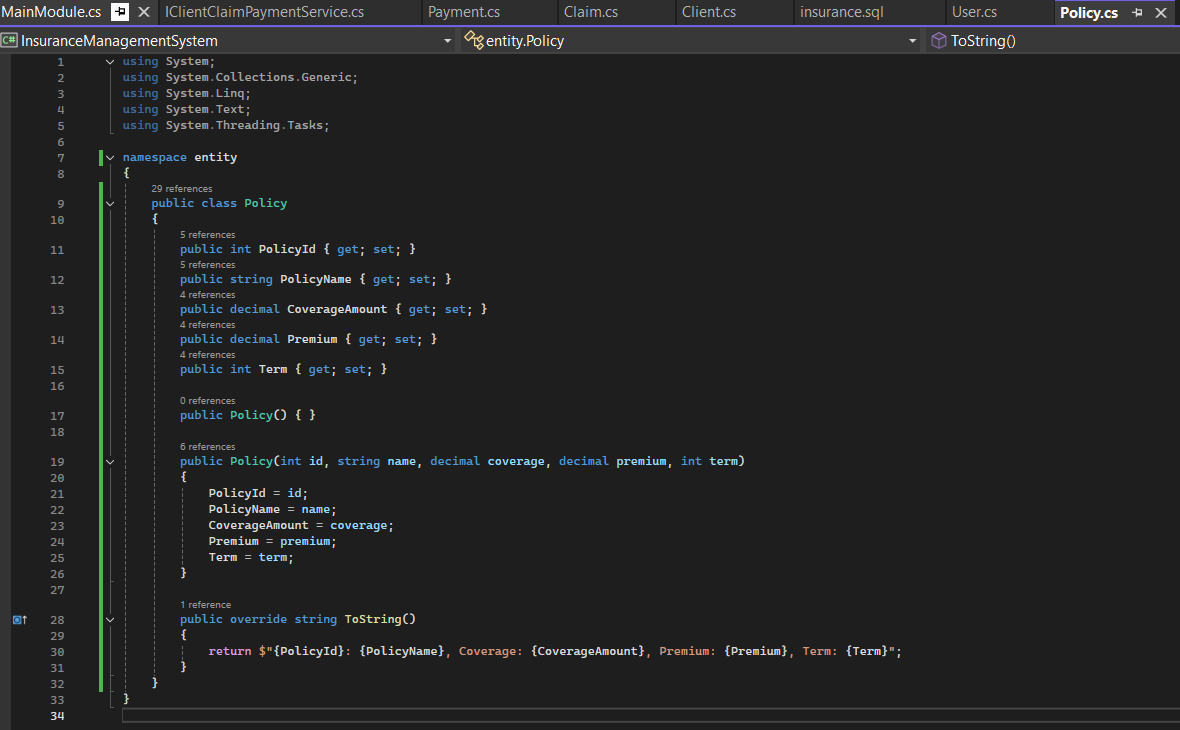
b. paymentDate;

c. paymentAmount;

d. client;

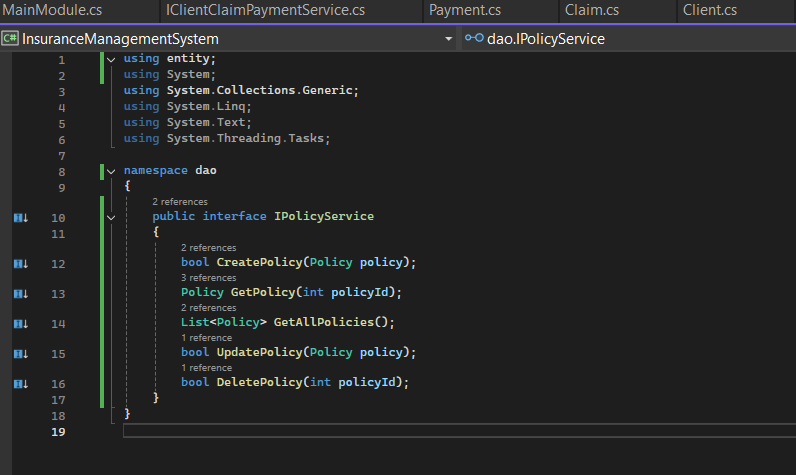


Define Class Policy

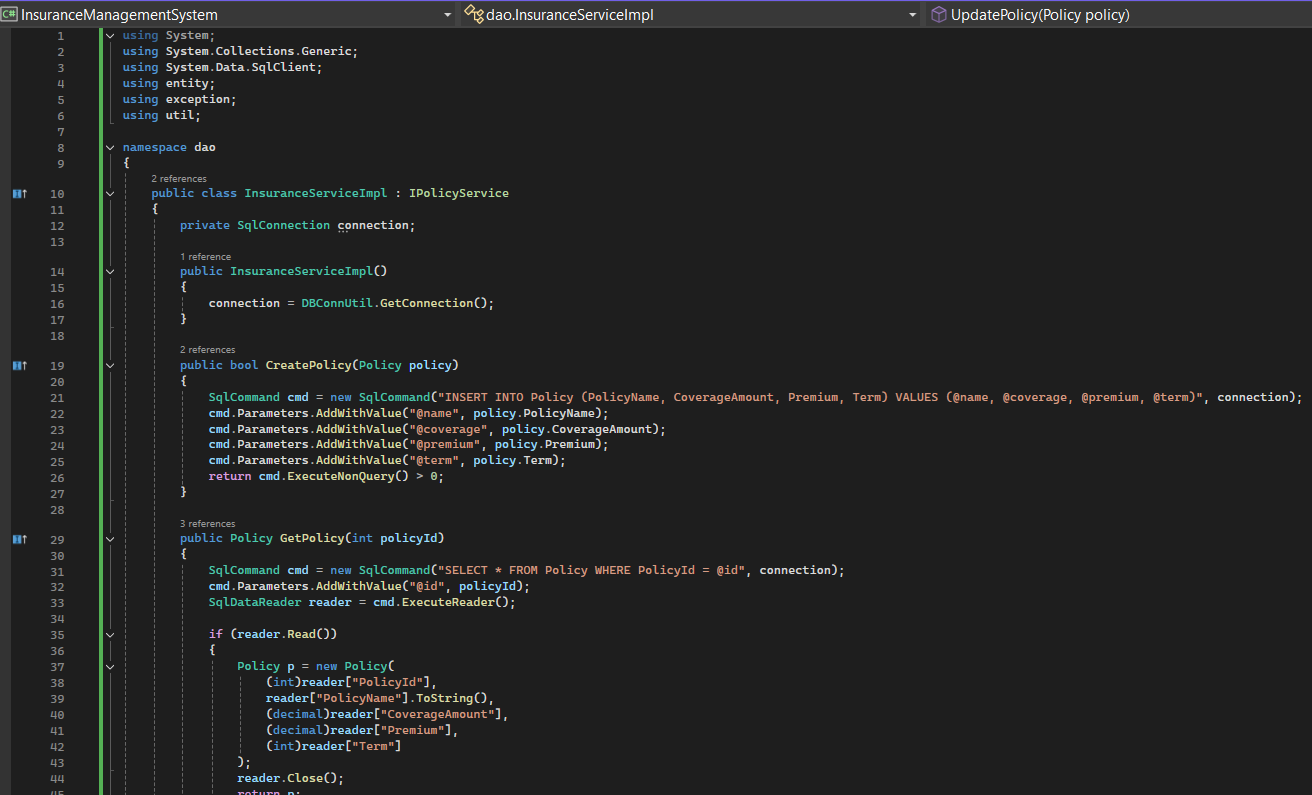


3. Define IPolicyService interface/abstract class with following methods to interact with database Keep the interfaces and implementation classes in package dao

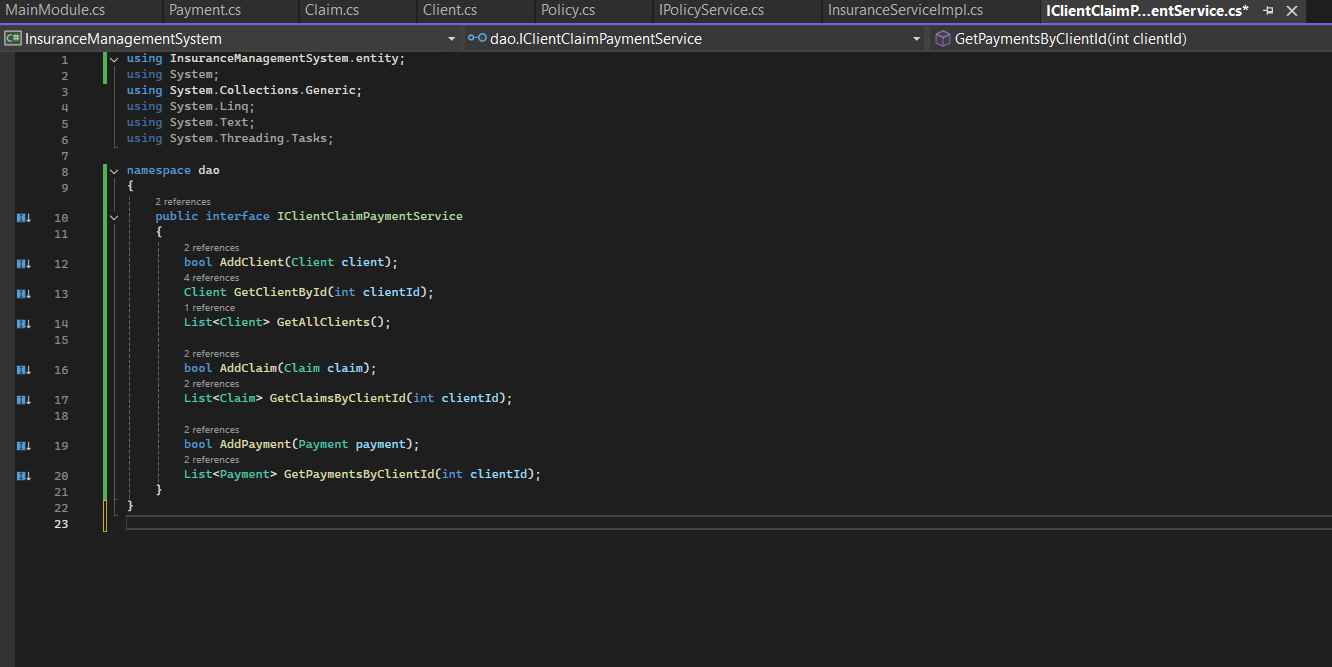
IPolicyService



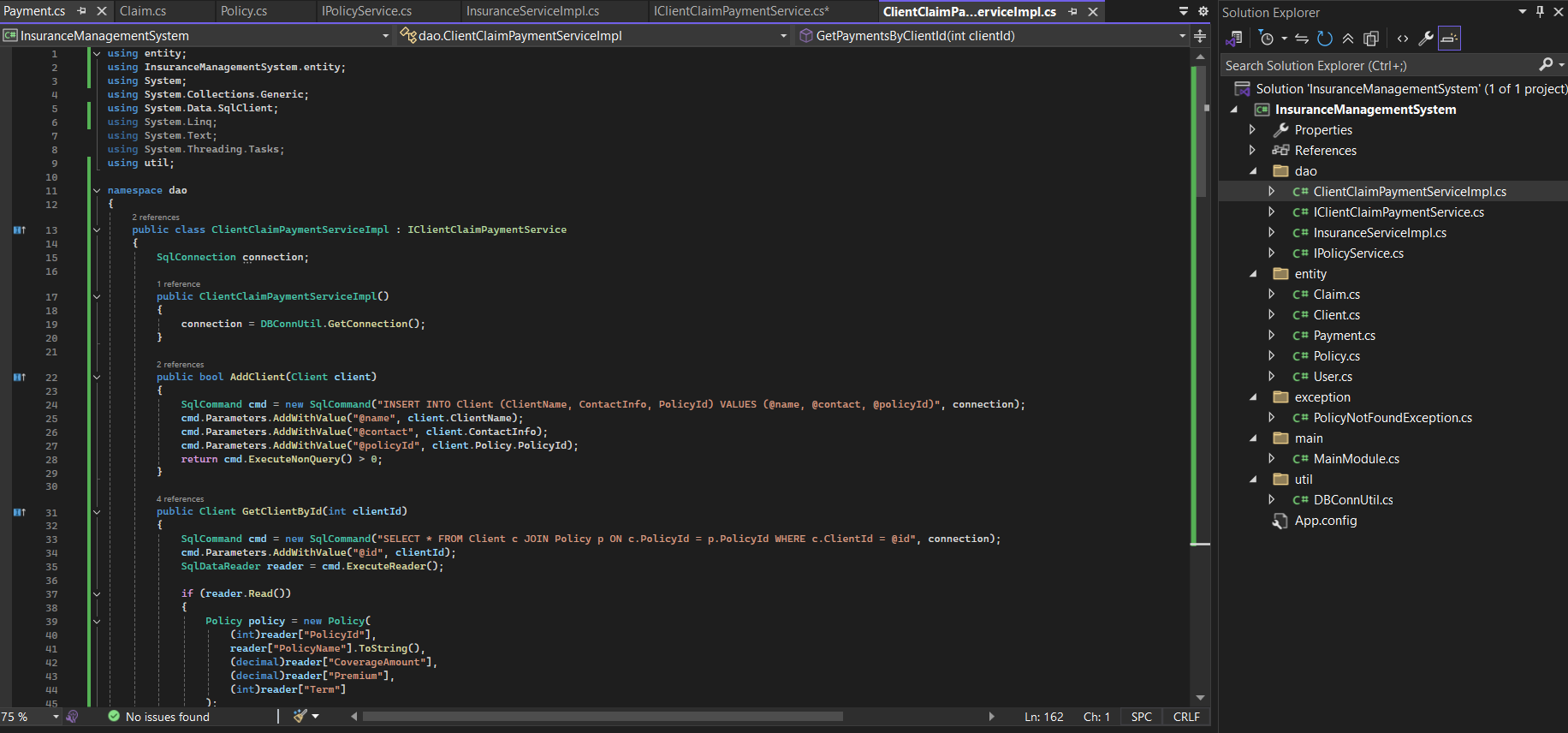
InsuranceServiceImpl

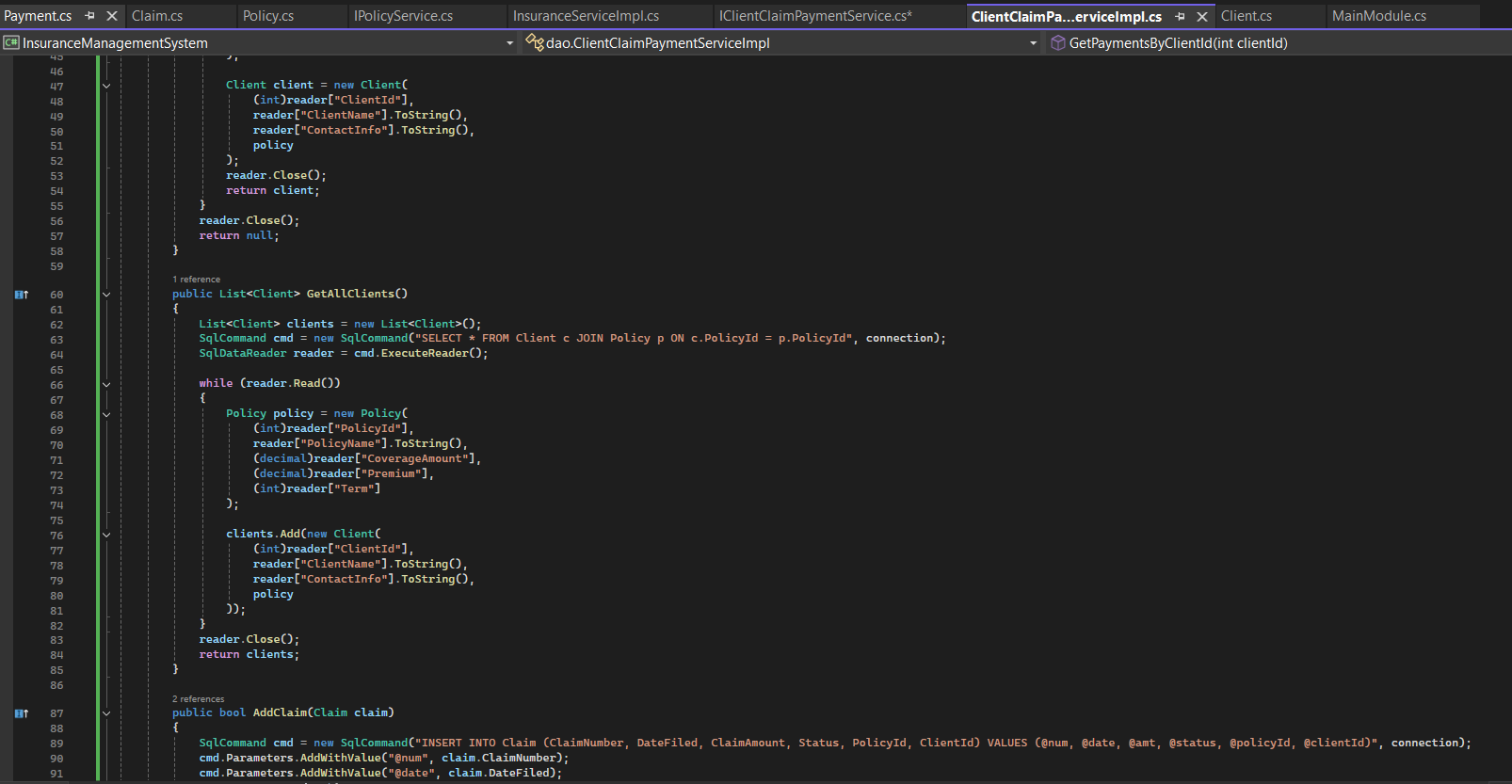


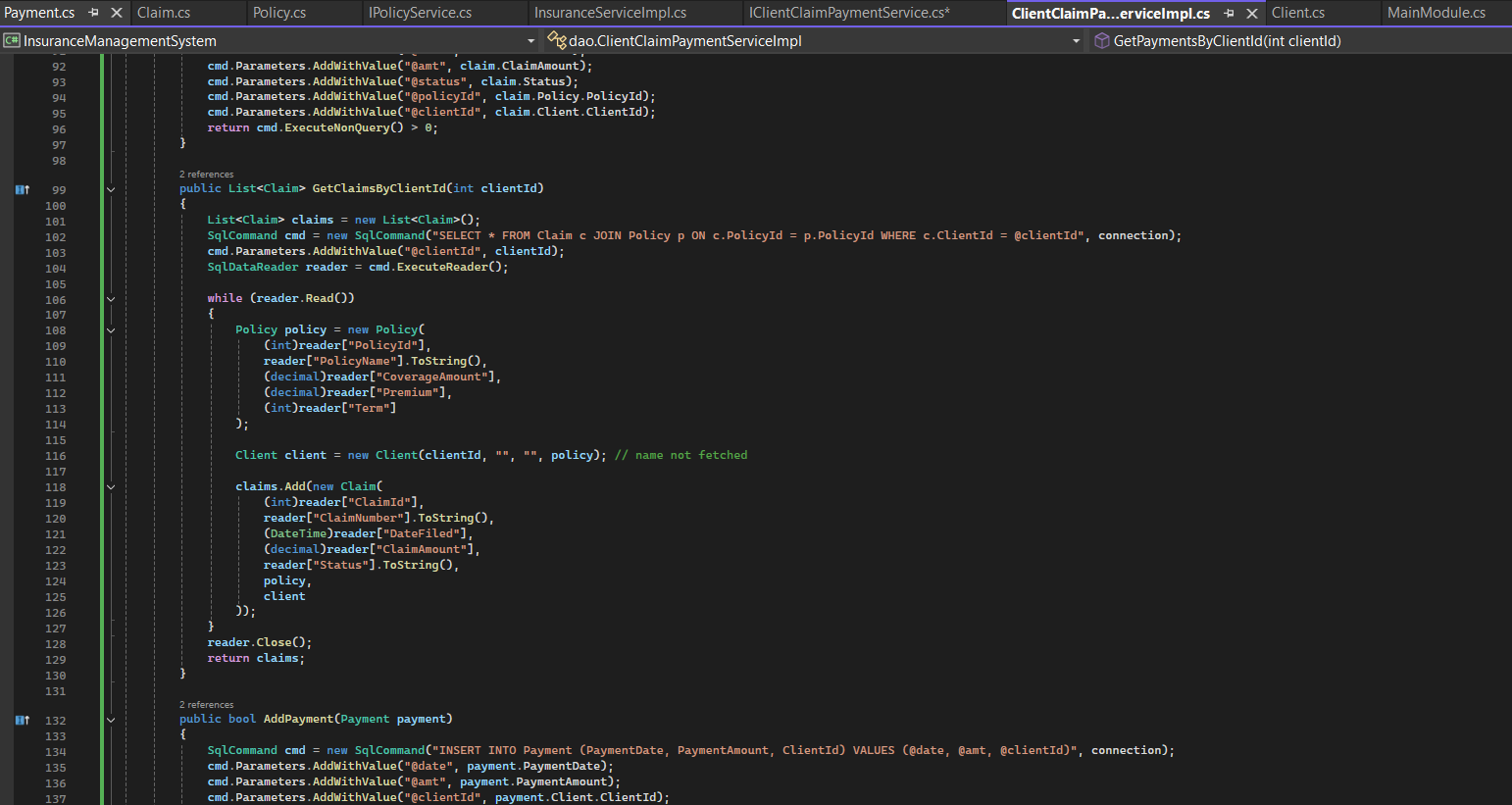
IClientClaimPaymentService

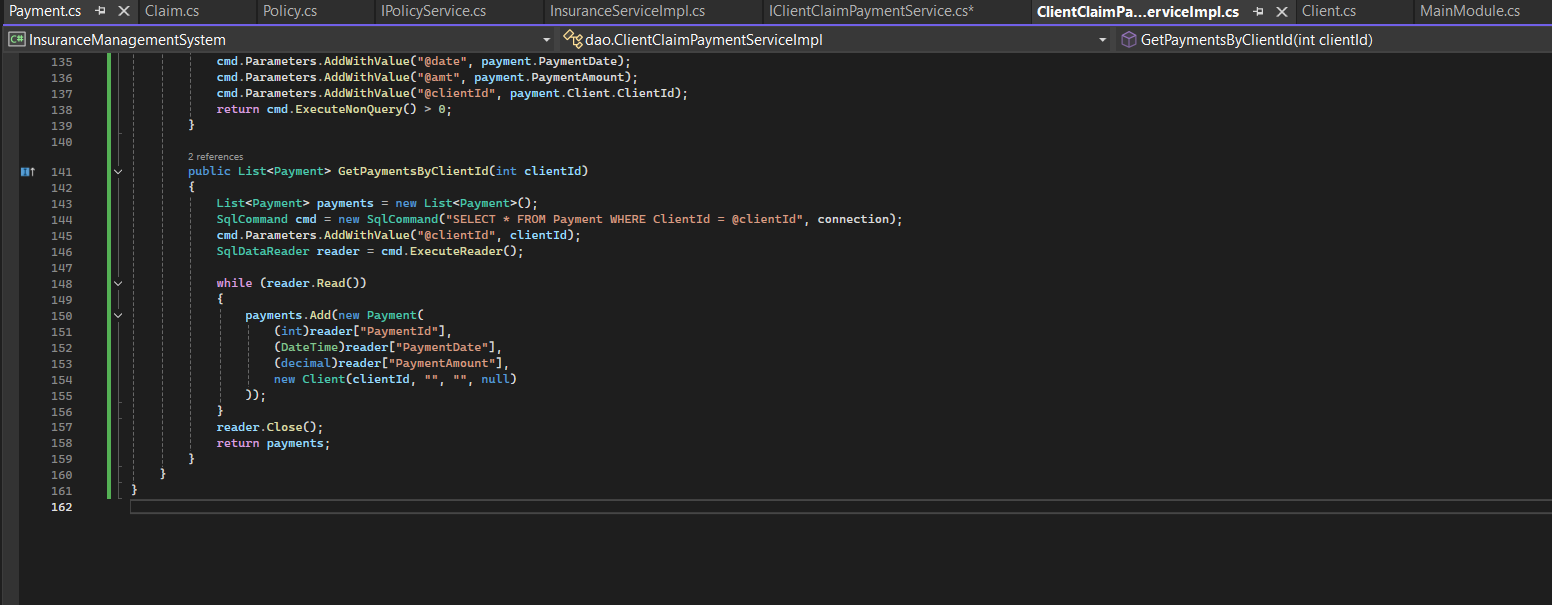


ClientClaimPaymentServiceImpl

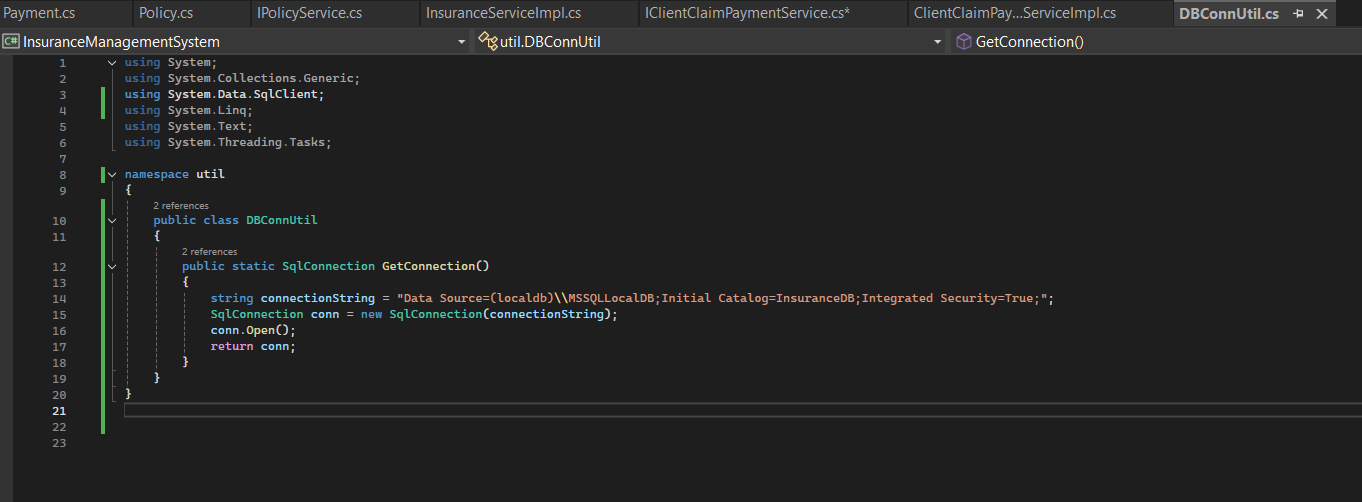


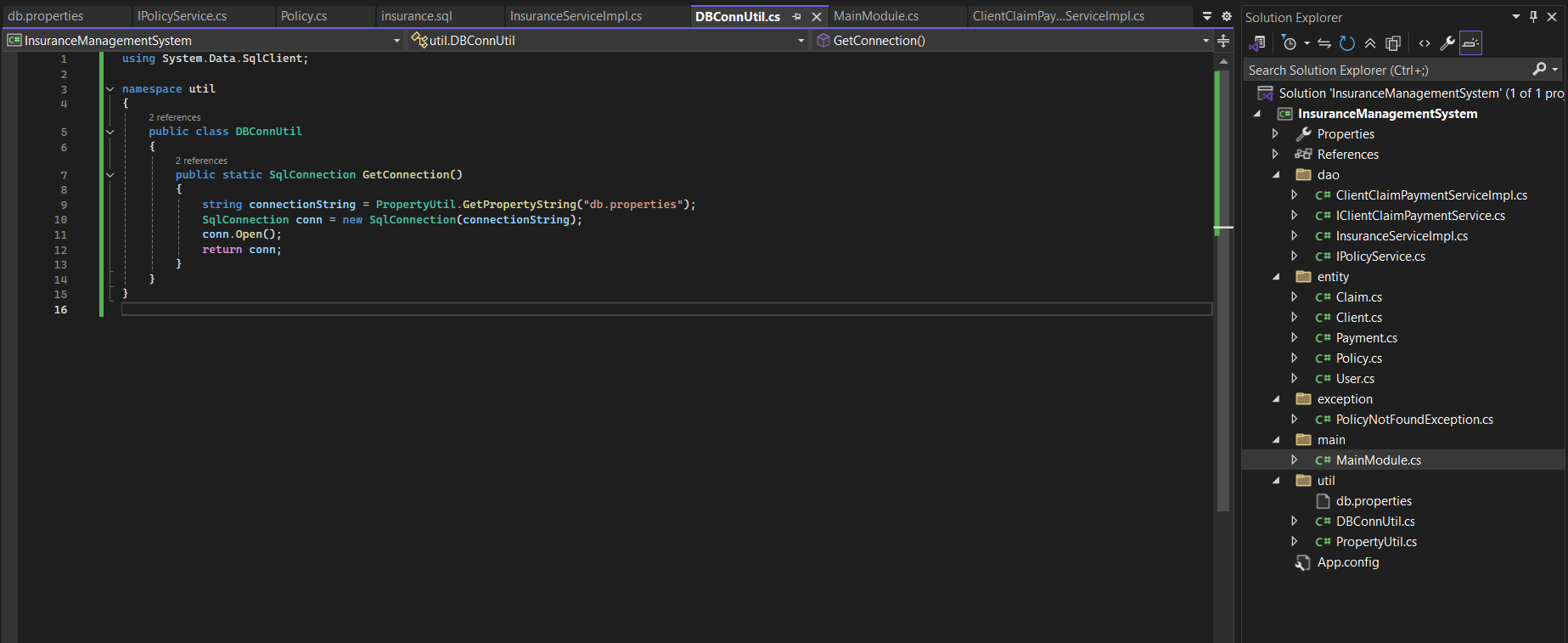


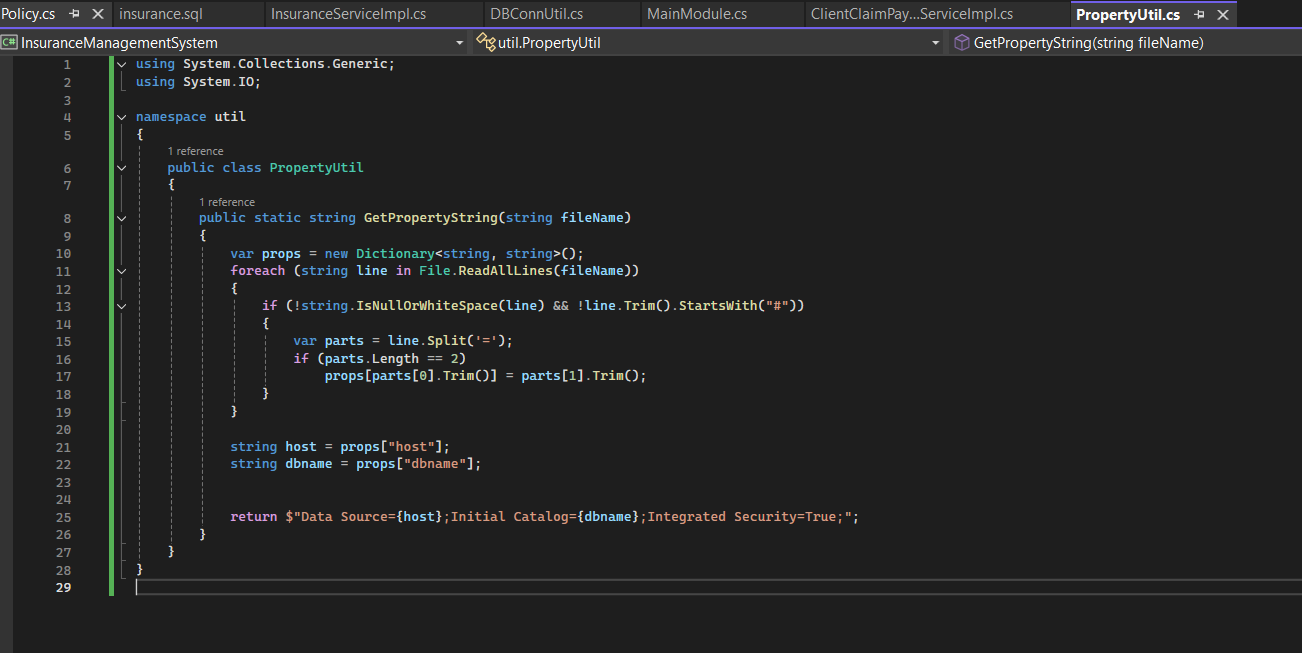




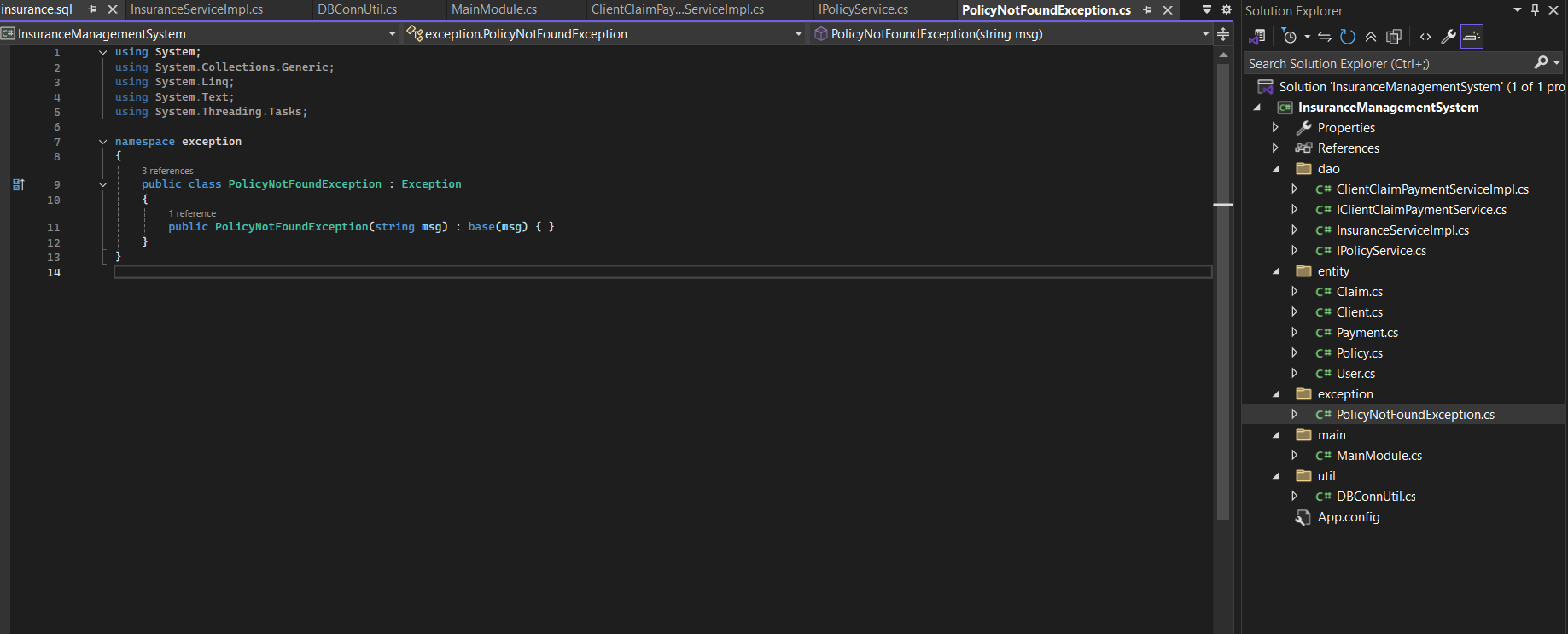
7. Create a utility class DBConnection in a package util with a static variable connection of Type Connection and a static method getConnection() which returns connection.



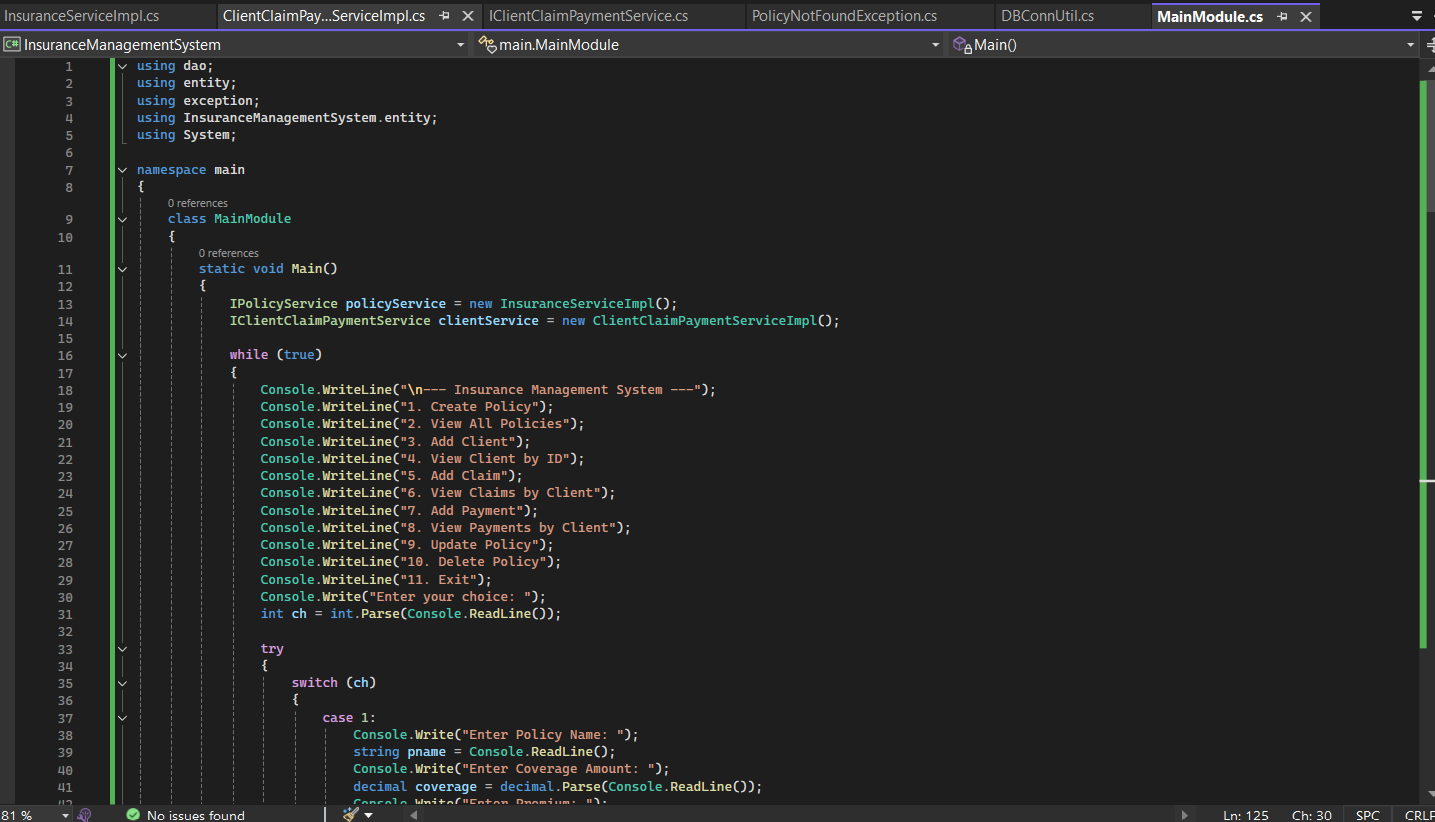


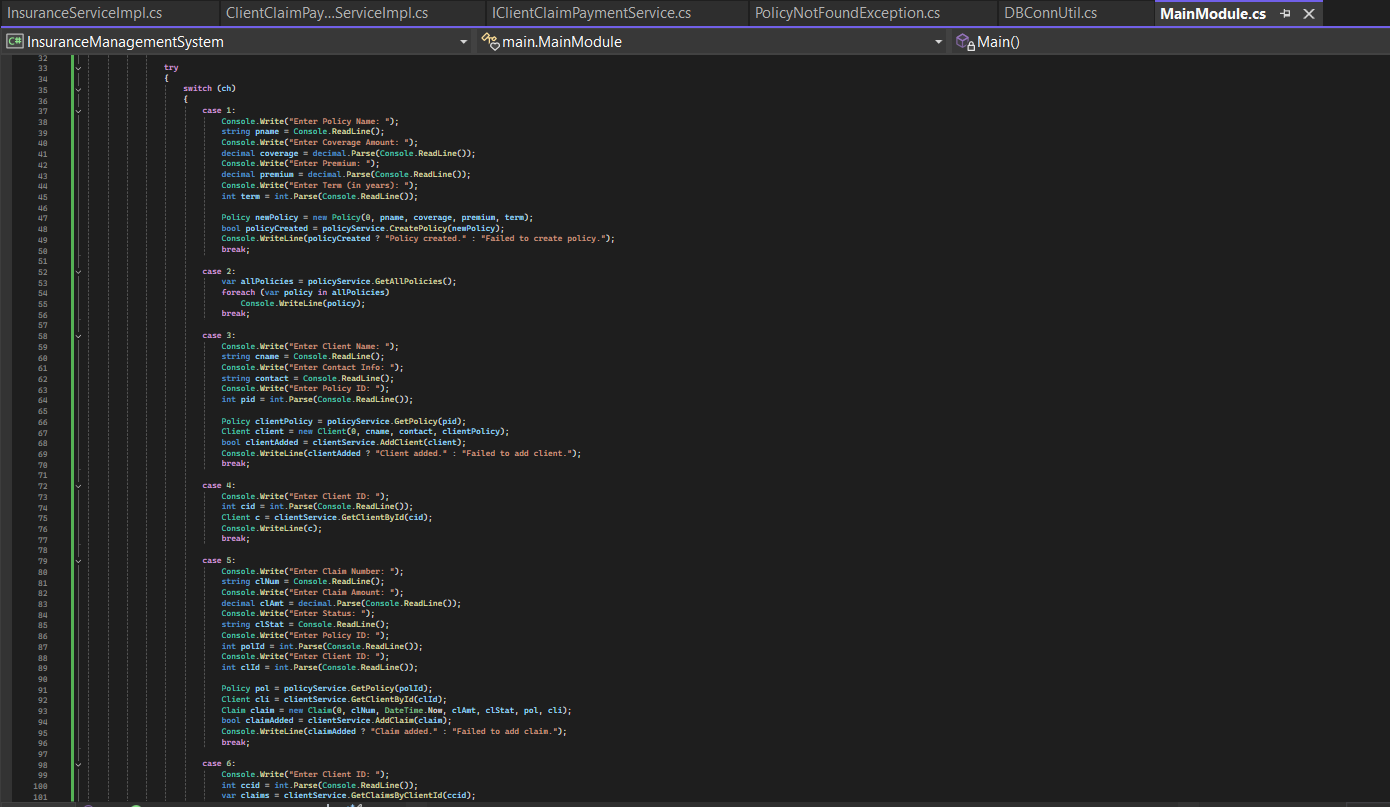


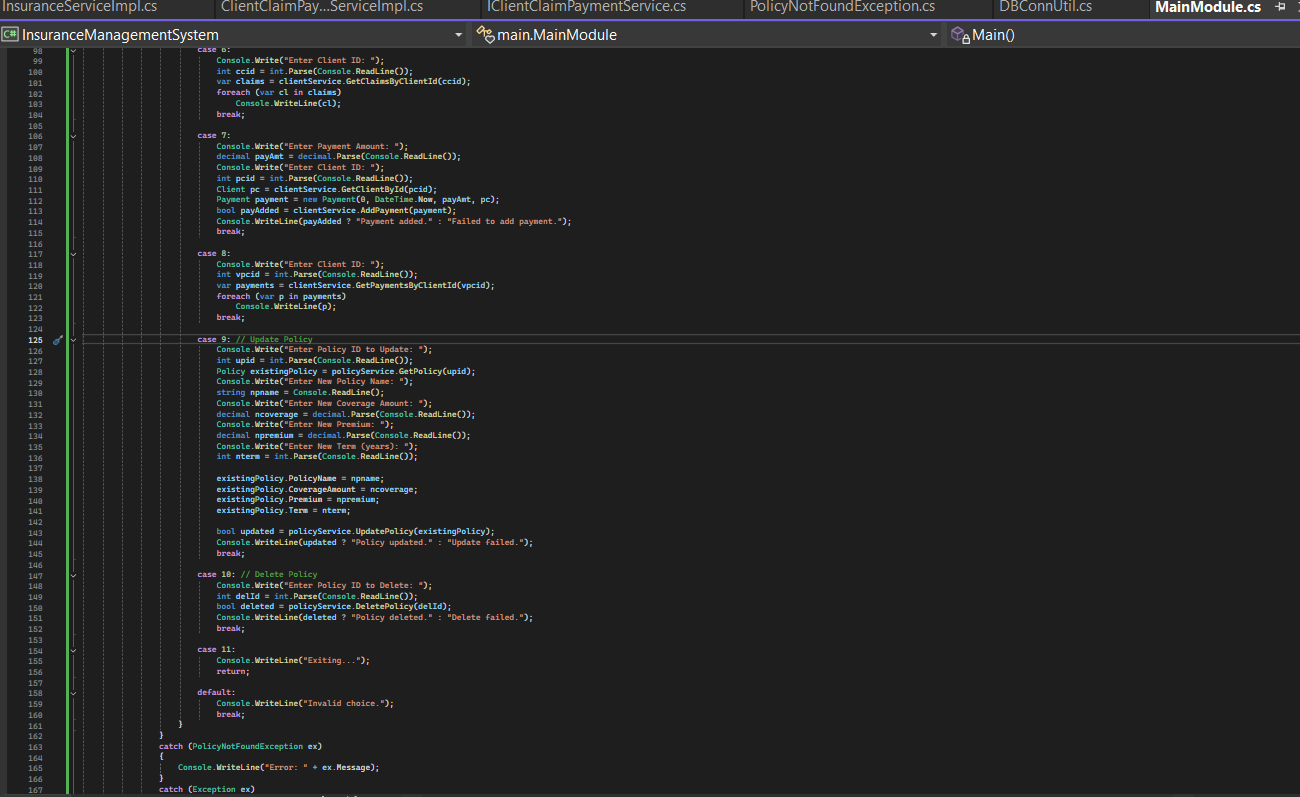
8. Create the exceptions in package myexceptions Define the following custom exceptions and throw them in methods whenever needed. Handle all the exceptions in main method, 1. PolicyNotFoundException :throw this exception when user enters an invalid patient number which doesn’t exist in db

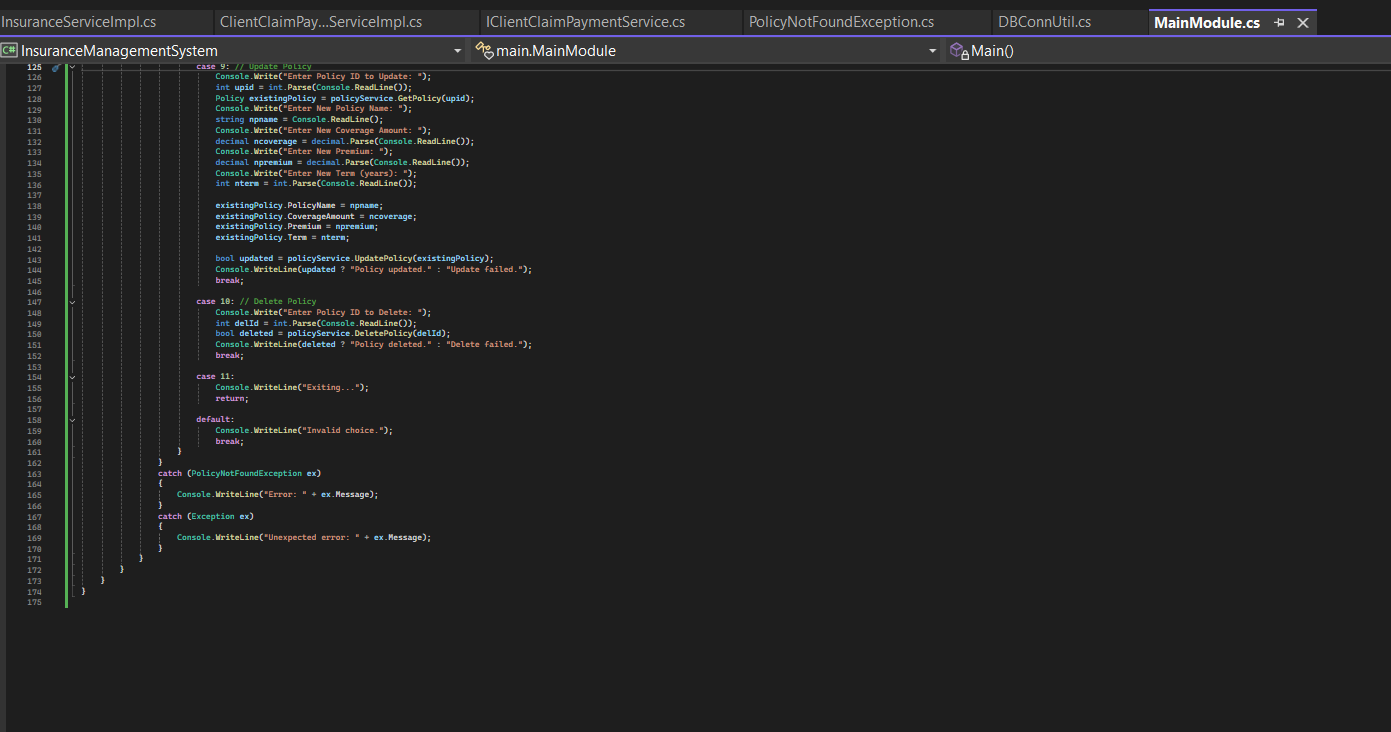


9. Create class named MainModule with main method in package mainmod. Trigger all the methods in service implementation class.

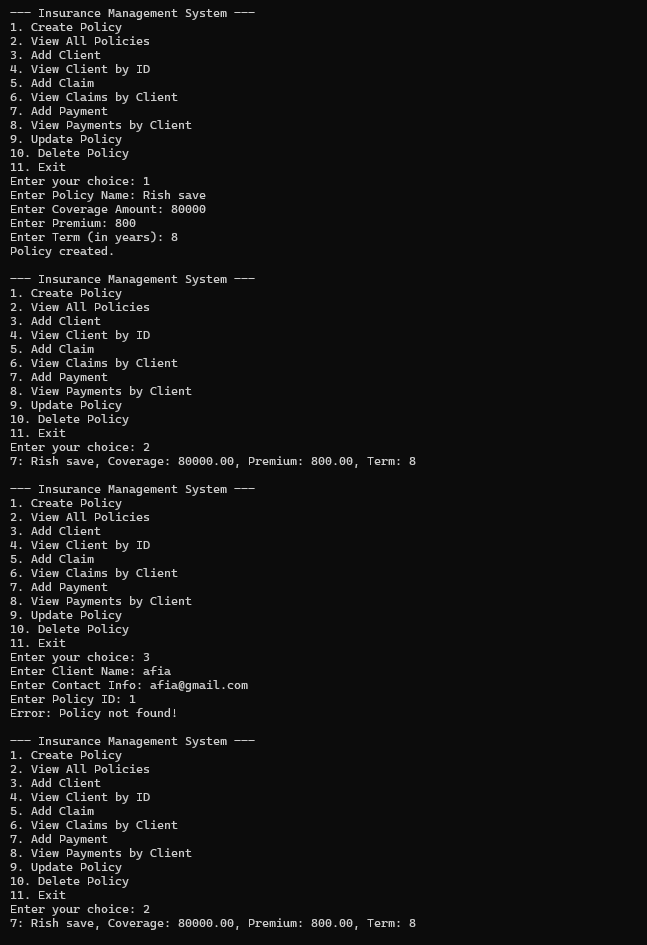








Output:



Coding:

Entity- Class user

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace entity

{

public class User

{

public int UserId { get; set; }

public string Username { get; set; }

public string Password { get; set; }

public string Role { get; set; }

public User() { }

public User(int id, string username, string password, string role)

{

UserId = id;

Username = username;

Password = password;

Role = role;

}

public override string ToString()

{

return $"{UserId}: {Username} ({Role})";

}

}

}

**Entity- Class Policy**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace entity

{

public class Policy

{

public int PolicyId { get; set; }

public string PolicyName { get; set; }

public decimal CoverageAmount { get; set; }

public decimal Premium { get; set; }

public int Term { get; set; }

public Policy() { }

public Policy(int id, string name, decimal coverage, decimal premium, int term)

{

PolicyId = id;

PolicyName = name;

CoverageAmount = coverage;

Premium = premium;

Term = term;

}

public override string ToString()

{

return $"{PolicyId}: {PolicyName}, Coverage: {CoverageAmount}, Premium: {Premium}, Term: {Term}";

}

}

}

**Entity- Class Payment**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace InsuranceManagementSystem.entity

{

public class Payment

{

public int PaymentId { get; set; }

public DateTime PaymentDate { get; set; }

public decimal PaymentAmount { get; set; }

public Client Client { get; set; }

public Payment() { }

public Payment(int id, DateTime date, decimal amount, Client client)

{

PaymentId = id;

PaymentDate = date;

PaymentAmount = amount;

Client = client;

}

public override string ToString()

{

return $"Payment ID: {PaymentId}, Date: {PaymentDate.ToShortDateString()}, Amount: {PaymentAmount}, Client: [{Client?.ClientName}]";

}

}

}

**Entity Class- Client**

using entity;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace InsuranceManagementSystem.entity

{

public class Client

{

public int ClientId { get; set; }

public string ClientName { get; set; }

public string ContactInfo { get; set; }

public Policy Policy { get; set; }

public Client() { }

public Client(int id, string name, string contact, Policy policy)

{

ClientId = id;

ClientName = name;

ContactInfo = contact;

Policy = policy;

}

public override string ToString()

{

return $"Client ID: {ClientId}, Name: {ClientName}, Contact: {ContactInfo}, Policy: [{Policy?.ToString()}]";

}

}

}

**Entity Class- Claim**

using entity;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace InsuranceManagementSystem.entity

{

public class Claim

{

public int ClaimId { get; set; }

public string ClaimNumber { get; set; }

public DateTime DateFiled { get; set; }

public decimal ClaimAmount { get; set; }

public string Status { get; set; }

public Policy Policy { get; set; }

public Client Client { get; set; }

public Claim() { }

public Claim(int id, string number, DateTime date, decimal amount, string status, Policy policy, Client client)

{

ClaimId = id;

ClaimNumber = number;

DateFiled = date;

ClaimAmount = amount;

Status = status;

Policy = policy;

Client = client;

}

public override string ToString()

{

return $"Claim #{ClaimNumber} | Date: {DateFiled.ToShortDateString()}, Amount: {ClaimAmount}, Status: {Status}, " +

$"Client: [{Client?.ClientName}], Policy: [{Policy?.PolicyName}]";

}

}

}

**Dao- IPolicyService**

using entity;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace dao

{

public interface IPolicyService

{

bool CreatePolicy(Policy policy);

Policy GetPolicy(int policyId);

List<Policy> GetAllPolicies();

bool UpdatePolicy(Policy policy);

bool DeletePolicy(int policyId);

}

}

**Dao- InsuranceServiceImpl**

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using entity;

using exception;

using util;

namespace dao

{

public class InsuranceServiceImpl : IPolicyService

{

private SqlConnection connection;

public InsuranceServiceImpl()

{

connection = DBConnUtil.GetConnection();

}

public bool CreatePolicy(Policy policy)

{

SqlCommand cmd = new SqlCommand("INSERT INTO Policy (PolicyName, CoverageAmount, Premium, Term) VALUES (@name, @coverage, @premium, @term)", connection);

cmd.Parameters.AddWithValue("@name", policy.PolicyName);

cmd.Parameters.AddWithValue("@coverage", policy.CoverageAmount);

cmd.Parameters.AddWithValue("@premium", policy.Premium);

cmd.Parameters.AddWithValue("@term", policy.Term);

return cmd.ExecuteNonQuery() > 0;

}

public Policy GetPolicy(int policyId)

{

SqlCommand cmd = new SqlCommand("SELECT \* FROM Policy WHERE PolicyId = @id", connection);

cmd.Parameters.AddWithValue("@id", policyId);

SqlDataReader reader = cmd.ExecuteReader();

if (reader.Read())

{

Policy p = new Policy(

(int)reader["PolicyId"],

reader["PolicyName"].ToString(),

(decimal)reader["CoverageAmount"],

(decimal)reader["Premium"],

(int)reader["Term"]

);

reader.Close();

return p;

}

reader.Close();

throw new PolicyNotFoundException("Policy not found!");

}

public List<Policy> GetAllPolicies()

{

List<Policy> list = new List<Policy>();

SqlCommand cmd = new SqlCommand("SELECT \* FROM Policy", connection);

SqlDataReader reader = cmd.ExecuteReader();

while (reader.Read())

{

list.Add(new Policy(

(int)reader["PolicyId"],

reader["PolicyName"].ToString(),

(decimal)reader["CoverageAmount"],

(decimal)reader["Premium"],

(int)reader["Term"]

));

}

reader.Close();

return list;

}

public bool UpdatePolicy(Policy policy)

{

SqlCommand cmd = new SqlCommand("UPDATE Policy SET PolicyName=@name, CoverageAmount=@coverage, Premium=@premium, Term=@term WHERE PolicyId=@id", connection);

cmd.Parameters.AddWithValue("@name", policy.PolicyName);

cmd.Parameters.AddWithValue("@coverage", policy.CoverageAmount);

cmd.Parameters.AddWithValue("@premium", policy.Premium);

cmd.Parameters.AddWithValue("@term", policy.Term);

cmd.Parameters.AddWithValue("@id", policy.PolicyId);

return cmd.ExecuteNonQuery() > 0;

}

public bool DeletePolicy(int policyId)

{

SqlCommand cmd = new SqlCommand("DELETE FROM Policy WHERE PolicyId = @id", connection);

cmd.Parameters.AddWithValue("@id", policyId);

return cmd.ExecuteNonQuery() > 0;

}

}

}

**Dao- IClientClaimPaymentService**

using InsuranceManagementSystem.entity;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace dao

{

public interface IClientClaimPaymentService

{

bool AddClient(Client client);

Client GetClientById(int clientId);

List<Client> GetAllClients();

bool AddClaim(Claim claim);

List<Claim> GetClaimsByClientId(int clientId);

bool AddPayment(Payment payment);

List<Payment> GetPaymentsByClientId(int clientId);

}

}

**Dao- ClientClaimPaymentServiceImpl**

using entity;

using InsuranceManagementSystem.entity;

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using util;

namespace dao

{

public class ClientClaimPaymentServiceImpl : IClientClaimPaymentService

{

SqlConnection connection;

public ClientClaimPaymentServiceImpl()

{

connection = DBConnUtil.GetConnection();

}

public bool AddClient(Client client)

{

SqlCommand cmd = new SqlCommand("INSERT INTO Client (ClientName, ContactInfo, PolicyId) VALUES (@name, @contact, @policyId)", connection);

cmd.Parameters.AddWithValue("@name", client.ClientName);

cmd.Parameters.AddWithValue("@contact", client.ContactInfo);

cmd.Parameters.AddWithValue("@policyId", client.Policy.PolicyId);

return cmd.ExecuteNonQuery() > 0;

}

public Client GetClientById(int clientId)

{

SqlCommand cmd = new SqlCommand("SELECT \* FROM Client c JOIN Policy p ON c.PolicyId = p.PolicyId WHERE c.ClientId = @id", connection);

cmd.Parameters.AddWithValue("@id", clientId);

SqlDataReader reader = cmd.ExecuteReader();

if (reader.Read())

{

Policy policy = new Policy(

(int)reader["PolicyId"],

reader["PolicyName"].ToString(),

(decimal)reader["CoverageAmount"],

(decimal)reader["Premium"],

(int)reader["Term"]

);

Client client = new Client(

(int)reader["ClientId"],

reader["ClientName"].ToString(),

reader["ContactInfo"].ToString(),

policy

);

reader.Close();

return client;

}

reader.Close();

return null;

}

public List<Client> GetAllClients()

{

List<Client> clients = new List<Client>();

SqlCommand cmd = new SqlCommand("SELECT \* FROM Client c JOIN Policy p ON c.PolicyId = p.PolicyId", connection);

SqlDataReader reader = cmd.ExecuteReader();

while (reader.Read())

{

Policy policy = new Policy(

(int)reader["PolicyId"],

reader["PolicyName"].ToString(),

(decimal)reader["CoverageAmount"],

(decimal)reader["Premium"],

(int)reader["Term"]

);

clients.Add(new Client(

(int)reader["ClientId"],

reader["ClientName"].ToString(),

reader["ContactInfo"].ToString(),

policy

));

}

reader.Close();

return clients;

}

public bool AddClaim(Claim claim)

{

SqlCommand cmd = new SqlCommand("INSERT INTO Claim (ClaimNumber, DateFiled, ClaimAmount, Status, PolicyId, ClientId) VALUES (@num, @date, @amt, @status, @policyId, @clientId)", connection);

cmd.Parameters.AddWithValue("@num", claim.ClaimNumber);

cmd.Parameters.AddWithValue("@date", claim.DateFiled);

cmd.Parameters.AddWithValue("@amt", claim.ClaimAmount);

cmd.Parameters.AddWithValue("@status", claim.Status);

cmd.Parameters.AddWithValue("@policyId", claim.Policy.PolicyId);

cmd.Parameters.AddWithValue("@clientId", claim.Client.ClientId);

return cmd.ExecuteNonQuery() > 0;

}

public List<Claim> GetClaimsByClientId(int clientId)

{

List<Claim> claims = new List<Claim>();

SqlCommand cmd = new SqlCommand("SELECT \* FROM Claim c JOIN Policy p ON c.PolicyId = p.PolicyId WHERE c.ClientId = @clientId", connection);

cmd.Parameters.AddWithValue("@clientId", clientId);

SqlDataReader reader = cmd.ExecuteReader();

while (reader.Read())

{

Policy policy = new Policy(

(int)reader["PolicyId"],

reader["PolicyName"].ToString(),

(decimal)reader["CoverageAmount"],

(decimal)reader["Premium"],

(int)reader["Term"]

);

Client client = new Client(clientId, "", "", policy); // name not fetched

claims.Add(new Claim(

(int)reader["ClaimId"],

reader["ClaimNumber"].ToString(),

(DateTime)reader["DateFiled"],

(decimal)reader["ClaimAmount"],

reader["Status"].ToString(),

policy,

client

));

}

reader.Close();

return claims;

}

public bool AddPayment(Payment payment)

{

SqlCommand cmd = new SqlCommand("INSERT INTO Payment (PaymentDate, PaymentAmount, ClientId) VALUES (@date, @amt, @clientId)", connection);

cmd.Parameters.AddWithValue("@date", payment.PaymentDate);

cmd.Parameters.AddWithValue("@amt", payment.PaymentAmount);

cmd.Parameters.AddWithValue("@clientId", payment.Client.ClientId);

return cmd.ExecuteNonQuery() > 0;

}

public List<Payment> GetPaymentsByClientId(int clientId)

{

List<Payment> payments = new List<Payment>();

SqlCommand cmd = new SqlCommand("SELECT \* FROM Payment WHERE ClientId = @clientId", connection);

cmd.Parameters.AddWithValue("@clientId", clientId);

SqlDataReader reader = cmd.ExecuteReader();

while (reader.Read())

{

payments.Add(new Payment(

(int)reader["PaymentId"],

(DateTime)reader["PaymentDate"],

(decimal)reader["PaymentAmount"],

new Client(clientId, "", "", null)

));

}

reader.Close();

return payments;

}

}

}

**Exception**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace exception

{

public class PolicyNotFoundException : Exception

{

public PolicyNotFoundException(string msg) : base(msg) { }

}

}

**Util-Dbconnection**

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace util

{

public class DBConnUtil

{

public static SqlConnection GetConnection()

{

string connectionString = "Data Source=(localdb)\\MSSQLLocalDB;Initial Catalog=InsuranceDB;Integrated Security=True;";

SqlConnection conn = new SqlConnection(connectionString);

conn.Open();

return conn;

}

}

}

**Main Module**

using dao;

using entity;

using exception;

using InsuranceManagementSystem.entity;

using System;

namespace main

{

class MainModule

{

static void Main()

{

IPolicyService policyService = new InsuranceServiceImpl();

IClientClaimPaymentService clientService = new ClientClaimPaymentServiceImpl();

while (true)

{

Console.WriteLine("\n--- Insurance Management System ---");

Console.WriteLine("1. Create Policy");

Console.WriteLine("2. View All Policies");

Console.WriteLine("3. Add Client");

Console.WriteLine("4. View Client by ID");

Console.WriteLine("5. Add Claim");

Console.WriteLine("6. View Claims by Client");

Console.WriteLine("7. Add Payment");

Console.WriteLine("8. View Payments by Client");

Console.WriteLine("9. Update Policy");

Console.WriteLine("10. Delete Policy");

Console.WriteLine("11. Exit");

Console.Write("Enter your choice: ");

int ch = int.Parse(Console.ReadLine());

try

{

switch (ch)

{

case 1:

Console.Write("Enter Policy Name: ");

string pname = Console.ReadLine();

Console.Write("Enter Coverage Amount: ");

decimal coverage = decimal.Parse(Console.ReadLine());

Console.Write("Enter Premium: ");

decimal premium = decimal.Parse(Console.ReadLine());

Console.Write("Enter Term (in years): ");

int term = int.Parse(Console.ReadLine());

Policy newPolicy = new Policy(0, pname, coverage, premium, term);

bool policyCreated = policyService.CreatePolicy(newPolicy);

Console.WriteLine(policyCreated ? "Policy created." : "Failed to create policy.");

break;

case 2:

var allPolicies = policyService.GetAllPolicies();

foreach (var policy in allPolicies)

Console.WriteLine(policy);

break;

case 3:

Console.Write("Enter Client Name: ");

string cname = Console.ReadLine();

Console.Write("Enter Contact Info: ");

string contact = Console.ReadLine();

Console.Write("Enter Policy ID: ");

int pid = int.Parse(Console.ReadLine());

Policy clientPolicy = policyService.GetPolicy(pid);

Client client = new Client(0, cname, contact, clientPolicy);

bool clientAdded = clientService.AddClient(client);

Console.WriteLine(clientAdded ? "Client added." : "Failed to add client.");

break;

case 4:

Console.Write("Enter Client ID: ");

int cid = int.Parse(Console.ReadLine());

Client c = clientService.GetClientById(cid);

Console.WriteLine(c);

break;

case 5:

Console.Write("Enter Claim Number: ");

string clNum = Console.ReadLine();

Console.Write("Enter Claim Amount: ");

decimal clAmt = decimal.Parse(Console.ReadLine());

Console.Write("Enter Status: ");

string clStat = Console.ReadLine();

Console.Write("Enter Policy ID: ");

int polId = int.Parse(Console.ReadLine());

Console.Write("Enter Client ID: ");

int clId = int.Parse(Console.ReadLine());

Policy pol = policyService.GetPolicy(polId);

Client cli = clientService.GetClientById(clId);

Claim claim = new Claim(0, clNum, DateTime.Now, clAmt, clStat, pol, cli);

bool claimAdded = clientService.AddClaim(claim);

Console.WriteLine(claimAdded ? "Claim added." : "Failed to add claim.");

break;

case 6:

Console.Write("Enter Client ID: ");

int ccid = int.Parse(Console.ReadLine());

var claims = clientService.GetClaimsByClientId(ccid);

foreach (var cl in claims)

Console.WriteLine(cl);

break;

case 7:

Console.Write("Enter Payment Amount: ");

decimal payAmt = decimal.Parse(Console.ReadLine());

Console.Write("Enter Client ID: ");

int pcid = int.Parse(Console.ReadLine());

Client pc = clientService.GetClientById(pcid);

Payment payment = new Payment(0, DateTime.Now, payAmt, pc);

bool payAdded = clientService.AddPayment(payment);

Console.WriteLine(payAdded ? "Payment added." : "Failed to add payment.");

break;

case 8:

Console.Write("Enter Client ID: ");

int vpcid = int.Parse(Console.ReadLine());

var payments = clientService.GetPaymentsByClientId(vpcid);

foreach (var p in payments)

Console.WriteLine(p);

break;

case 9: // Update Policy

Console.Write("Enter Policy ID to Update: ");

int upid = int.Parse(Console.ReadLine());

Policy existingPolicy = policyService.GetPolicy(upid);

Console.Write("Enter New Policy Name: ");

string npname = Console.ReadLine();

Console.Write("Enter New Coverage Amount: ");

decimal ncoverage = decimal.Parse(Console.ReadLine());

Console.Write("Enter New Premium: ");

decimal npremium = decimal.Parse(Console.ReadLine());

Console.Write("Enter New Term (years): ");

int nterm = int.Parse(Console.ReadLine());

existingPolicy.PolicyName = npname;

existingPolicy.CoverageAmount = ncoverage;

existingPolicy.Premium = npremium;

existingPolicy.Term = nterm;

bool updated = policyService.UpdatePolicy(existingPolicy);

Console.WriteLine(updated ? "Policy updated." : "Update failed.");

break;

case 10: // Delete Policy

Console.Write("Enter Policy ID to Delete: ");

int delId = int.Parse(Console.ReadLine());

bool deleted = policyService.DeletePolicy(delId);

Console.WriteLine(deleted ? "Policy deleted." : "Delete failed.");

break;

case 11:

Console.WriteLine("Exiting...");

return;

default:

Console.WriteLine("Invalid choice.");

break;

}

}

catch (PolicyNotFoundException ex)

{

Console.WriteLine("Error: " + ex.Message);

}

catch (Exception ex)

{

Console.WriteLine("Unexpected error: " + ex.Message);

}

}

}

}

}