**Coding Challenge C# and SQL Server**

**Insurance Management System**

**Introduction:**

The Insurance Management System is a console-based application developed using C# (.NET Core) and Microsoft SQL Server. It is designed to manage core insurance functionalities like Policy Creation, Retrieval, Update, and Deletion. This system follows Object-Oriented Programming (OOP) principles and a layered architecture for better code organization and scalability.

**Technologies Used:**

* C# (.NET Core Console Application)
* Microsoft SQL Server
* Microsoft.Data.SqlClient (for DB connection)
* NUnit (for unit testing)
* Visual Studio 2022

**Project Structure:**

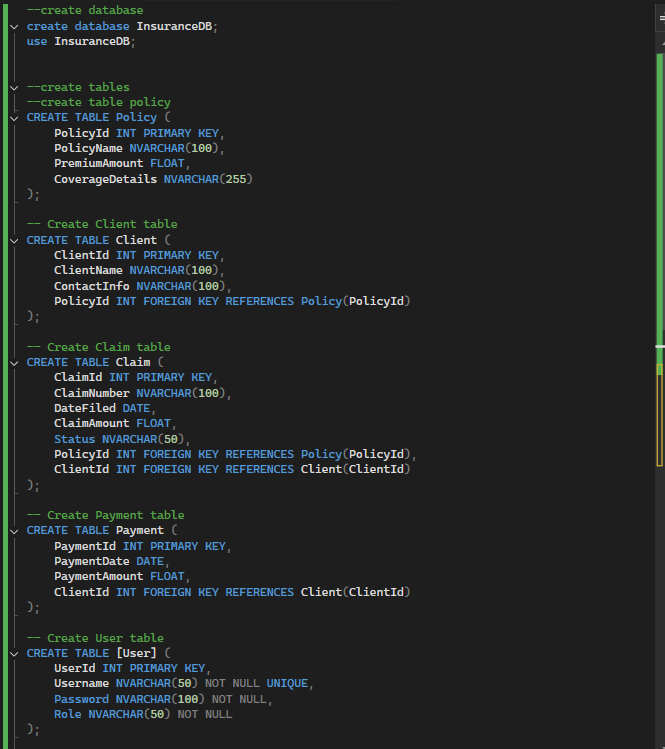
The project is organized into the following folders/packages:

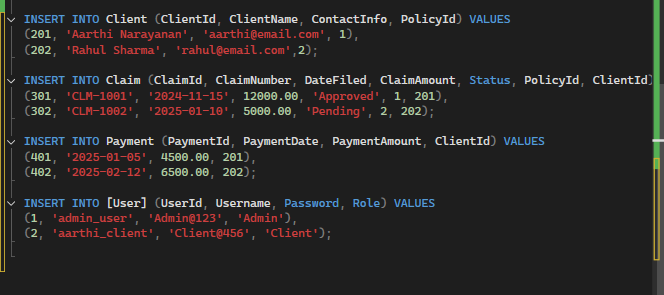
* **entity** – Contains data models: **Policy, Client, Claim, Payment, User**
* **dao** – Interface **IPolicyService** and its implementation **PolicyServiceImpl**
* **exceptions** – Custom exception: **PolicyNotFoundException**
* **util** – Utility classes for DB connection:
  + **DBPropertyUtil.cs** – Reads connection details from file
  + **DBConnUtil.cs** – Establishes SQL connection
* **mainmod** – Contains **MainModule.cs** for menu-driven user interaction
* **Program**.**cs –** used to call the **MainModule.cs**

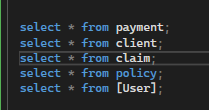
**Implementation Steps:**

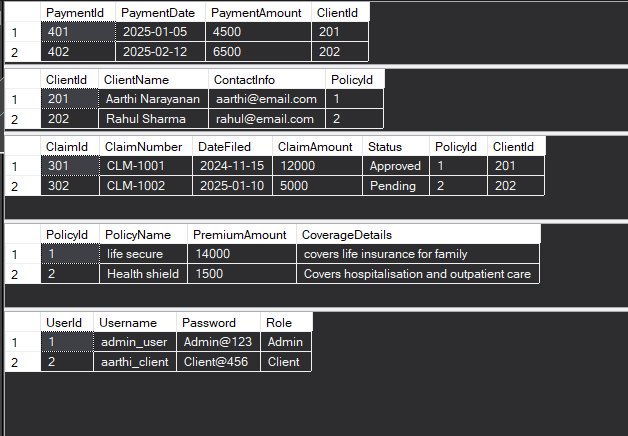
1. **Created entity classes** with private variables, constructors, getters, setters, and **ToString().**
2. **Designed SQL tables** using appropriate foreign keys and constraints.
3. Implemented **CRUD operations for Policy** via IPolicyService and PolicyServiceImpl.
4. Used **custom exception** **(PolicyNotFoundException)** to handle invalid operations.
5. **Connected the app to SQL Server** using utility classes (**DBPropertyUtil, DBConnUtil).**
6. Built a **menu-driven UI** in MainModule.cs to interact with the system.
7. Wrote **NUnit test cases** to validate service logic and ensure reliable functionality.

**SQL Queries:**

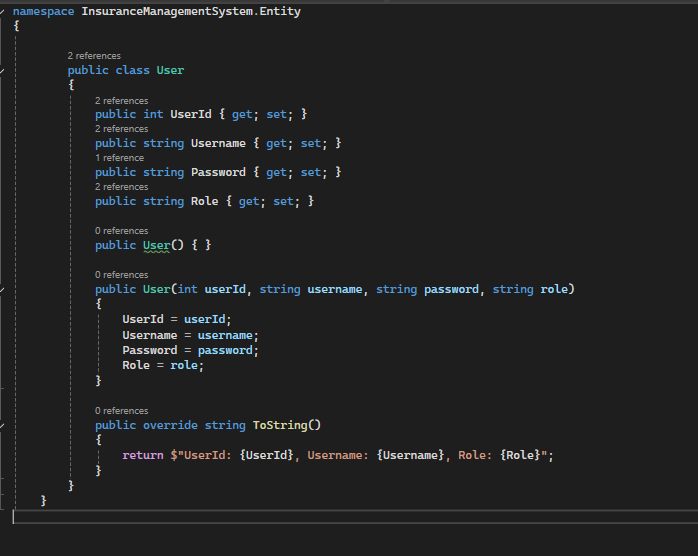




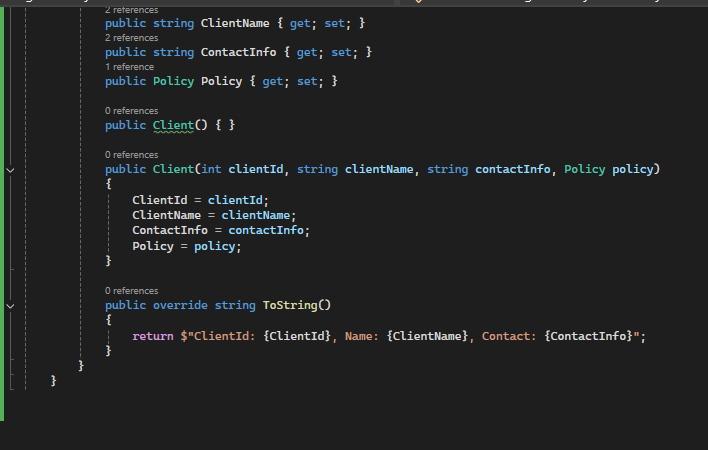




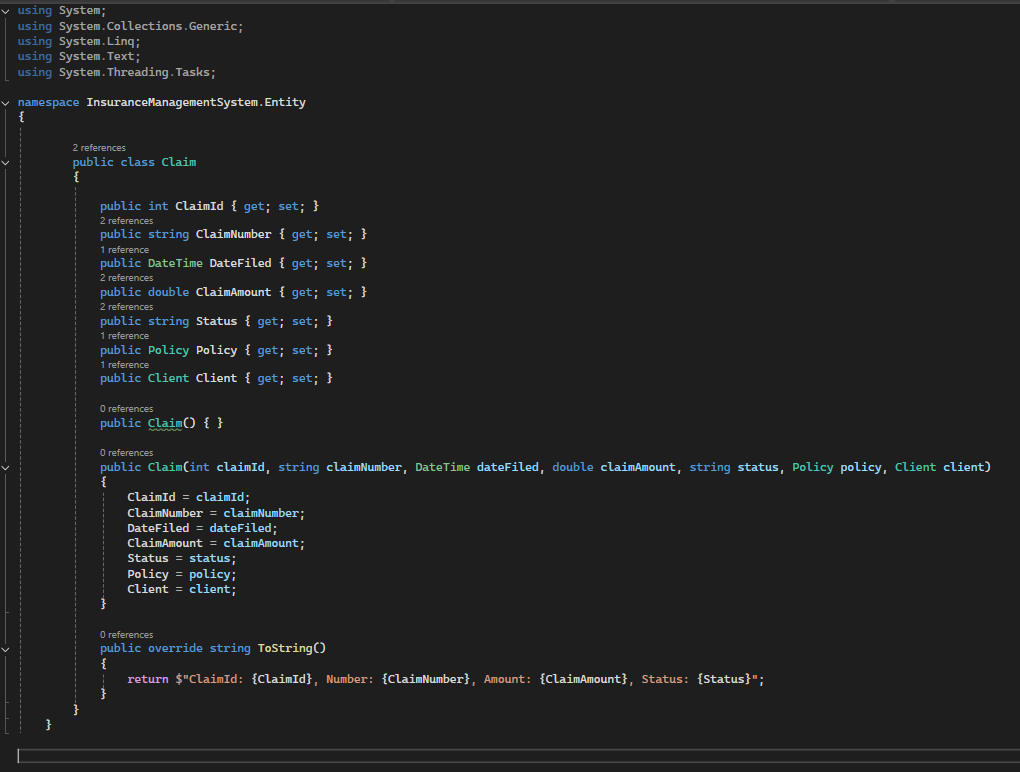
**1. Entity🡪User.cs**



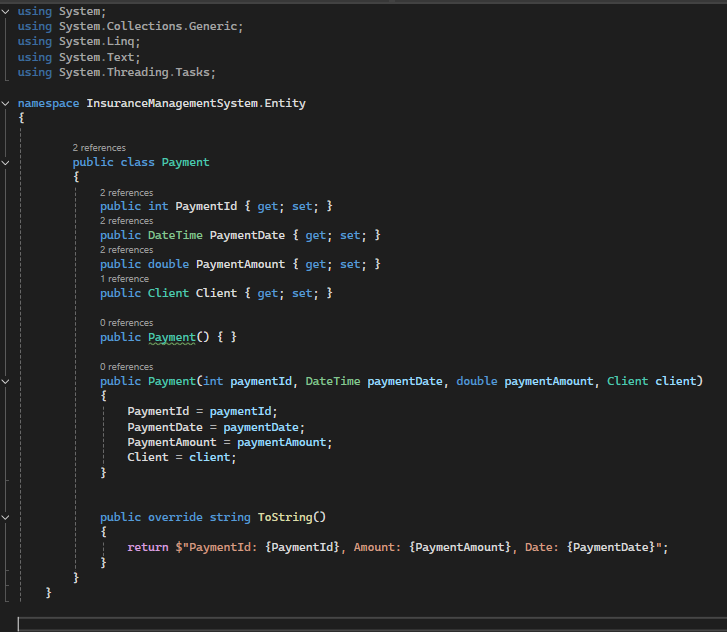
**2.Entity🡪Client.cs**



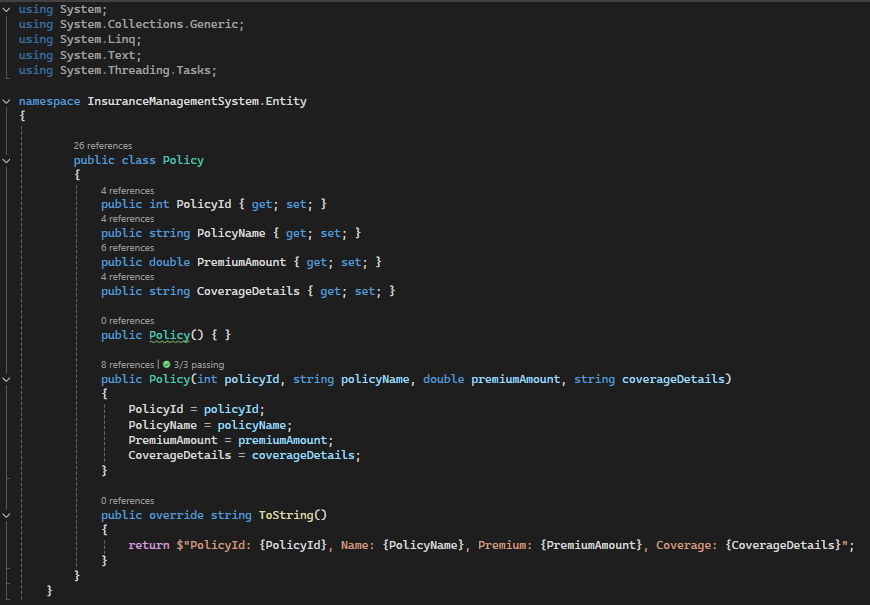
**3. Entity🡪Claim.cs**



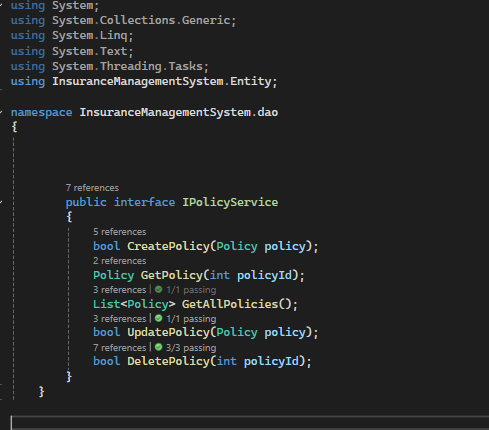
**4. Entity🡪payment.cs**



**5. Entity🡪Policy.cs**



**6. dao🡪IPolicyService.cs**



**7. dao🡪PolicyServiceImpl.cs**

using InsuranceManagementSystem.Entity;

using InsuranceManagementSystem.Exceptions;

using InsuranceManagementSystem.util;

using Microsoft.Data.SqlClient;

using System;

using System.Collections.Generic;

namespace InsuranceManagementSystem.dao

{

public class PolicyServiceImpl : IPolicyService

{

public bool CreatePolicy(Policy policy)

{

bool isInserted = false;

try

{

using (SqlConnection conn = DBConnUtil.GetConnection())

{

string query = "INSERT INTO Policy (PolicyId, PolicyName, PremiumAmount, CoverageDetails) " +

"VALUES (@id, @name, @premium, @coverage)";

SqlCommand cmd = new SqlCommand(query, conn);

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

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

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

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

int rows = cmd.ExecuteNonQuery();

isInserted = rows > 0;

}

}

catch (Exception ex)

{

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

}

return isInserted;

}

public Policy GetPolicy(int policyId)

{

Policy policy = null;

try

{

using (SqlConnection conn = DBConnUtil.GetConnection())

{

string query = "SELECT \* FROM Policy WHERE PolicyId = @id";

SqlCommand cmd = new SqlCommand(query, conn);

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

SqlDataReader reader = cmd.ExecuteReader();

if (reader.Read())

{

policy = new Policy(

reader.GetInt32(0),

reader.GetString(1),

reader.GetDouble(2),

reader.GetString(3)

);

}

else

{

throw new PolicyNotFoundException($"Policy with ID {policyId} not found.");

}

}

}

catch (PolicyNotFoundException)

{

throw;

}

catch (Exception ex)

{

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

}

return policy;

}

public List<Policy> GetAllPolicies()

{

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

try

{

using (SqlConnection conn = DBConnUtil.GetConnection())

{

string query = "SELECT \* FROM Policy";

SqlCommand cmd = new SqlCommand(query, conn);

SqlDataReader reader = cmd.ExecuteReader();

while (reader.Read())

{

Policy policy = new Policy(

reader.GetInt32(0),

reader.GetString(1),

reader.GetDouble(2),

reader.GetString(3)

);

policies.Add(policy);

}

}

}

catch (Exception ex)

{

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

}

return policies;

}

public bool UpdatePolicy(Policy policy)

{

bool isUpdated = false;

try

{

using (SqlConnection conn = DBConnUtil.GetConnection())

{

string query = "UPDATE Policy SET PolicyName = @name, PremiumAmount = @premium, " +

"CoverageDetails = @coverage WHERE PolicyId = @id";

SqlCommand cmd = new SqlCommand(query, conn);

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

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

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

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

int rows = cmd.ExecuteNonQuery();

isUpdated = rows > 0;

}

}

catch (Exception ex)

{

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

}

return isUpdated;

}

public bool DeletePolicy(int policyId)

{

bool isDeleted = false;

try

{

using (SqlConnection conn = DBConnUtil.GetConnection())

{

string query = "DELETE FROM Policy WHERE PolicyId = @id";

SqlCommand cmd = new SqlCommand(query, conn);

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

int rows = cmd.ExecuteNonQuery();

if (rows == 0)

{

throw new PolicyNotFoundException($"Policy with ID {policyId} does not exist.");

}

isDeleted = true;

}

}

catch (PolicyNotFoundException)

{

throw;

}

catch (Exception ex)

{

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

}

return isDeleted;

}

}

}

**8. Util🡪DBPropertyUtil.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace InsuranceManagementSystem.util

{

public class DBPropertyUtil

{

public static Dictionary<string, string> LoadProperties(string filePath)

{

Dictionary<string, string> properties = new Dictionary<string, string>();

try

{

if (!File.Exists(filePath))

{

Console.WriteLine(" Property file not found: " + filePath);

return properties;

}

foreach (var line in File.ReadAllLines(filePath))

{

if (!string.IsNullOrWhiteSpace(line) && line.Contains("="))

{

var tokens = line.Split('=', 2);

string key = tokens[0].Trim();

string value = tokens[1].Trim();

properties[key] = value;

}

}

}

catch (Exception ex)

{

Console.WriteLine("Error reading DB properties file: " + ex.Message);

}

return properties;

}

public static string GetConnectionString(string filePath)

{

var props = LoadProperties(filePath);

if (!props.ContainsKey("server") || !props.ContainsKey("database"))

{

Console.WriteLine("Missing required DB config keys: 'server' or 'database'");

return null;

}

// Windows Authentication (no username/password)

return $"Server={props["server"]};Database={props["database"]};Integrated Security=True;TrustServerCertificate=True;";

}

}

}

**9. Util🡪DBConnUtil.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Microsoft.Data.SqlClient;

namespace InsuranceManagementSystem.util

{

public class DBConnUtil

{

private static SqlConnection connection;

public static SqlConnection GetConnection()

{

if (connection == null || connection.State == System.Data.ConnectionState.Closed)

{

try

{

string filePath = "appsettings.properties"; // make sure this file is in output directory

string connectionString = DBPropertyUtil.GetConnectionString(filePath);

if (string.IsNullOrEmpty(connectionString))

{

Console.WriteLine("Invalid or missing connection string.");

return null;

}

connection = new SqlConnection(connectionString);

connection.Open();

Console.WriteLine("Database connection established.");

}

catch (Exception ex)

{

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

}

}

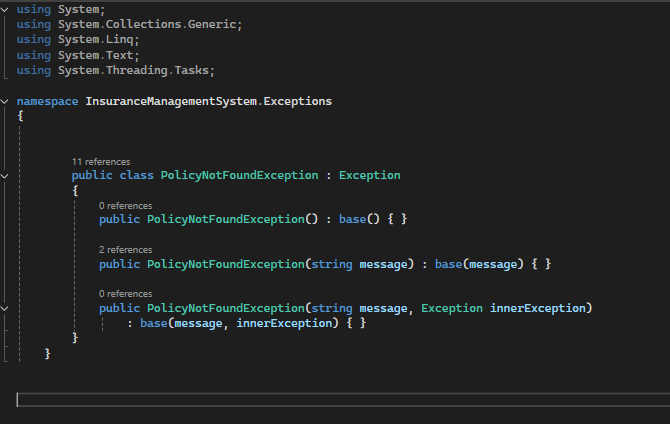
return connection;

}

}

}

**10. Exceptions🡪PolicyNotFoundException.cs**



**11. Mainmod🡪MainModule.cs**

using InsuranceManagementSystem.dao;

using InsuranceManagementSystem.Entity;

using InsuranceManagementSystem.Exceptions;

using InsuranceManagementSystem.util;

using System;

using System.Collections.Generic;

namespace InsuranceManagementSystem.mainmod

{

public class MainModule

{

public static void Start()

{

IPolicyService policyService = new PolicyServiceImpl();

int choice;

do

{

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

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

Console.WriteLine("2. Get Policy by ID");

Console.WriteLine("3. Get All Policies");

Console.WriteLine("4. Update Policy");

Console.WriteLine("5. Delete Policy");

Console.WriteLine("0. Exit");

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

bool valid = int.TryParse(Console.ReadLine(), out choice);

if (!valid) choice = -1;

switch (choice)

{

case 1: CreatePolicy(policyService); break;

case 2: GetPolicy(policyService); break;

case 3: GetAllPolicies(policyService); break;

case 4: UpdatePolicy(policyService); break;

case 5: DeletePolicy(policyService); break;

case 0: Console.WriteLine("Exiting application..."); break;

default: Console.WriteLine("Invalid choice! Try again."); break;

}

} while (choice != 0);

}

static void CreatePolicy(IPolicyService service)

{

try

{

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

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

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

string name = Console.ReadLine();

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

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

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

string coverage = Console.ReadLine();

Policy policy = new Policy(id, name, premium, coverage);

bool result = service.CreatePolicy(policy);

Console.WriteLine(result ? "Policy created." : "Creation failed.");

}

catch (Exception ex)

{

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

}

}

static void GetPolicy(IPolicyService service)

{

try

{

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

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

Policy policy = service.GetPolicy(id);

Console.WriteLine(policy);

}

catch (PolicyNotFoundException ex)

{

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

}

}

static void GetAllPolicies(IPolicyService service)

{

List<Policy> policies = service.GetAllPolicies();

if (policies.Count == 0)

{

Console.WriteLine("No policies found.");

return;

}

foreach (var p in policies)

{

Console.WriteLine(p);

}

}

static void UpdatePolicy(IPolicyService service)

{

try

{

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

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

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

string name = Console.ReadLine();

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

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

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

string coverage = Console.ReadLine();

Policy updated = new Policy(id, name, premium, coverage);

bool result = service.UpdatePolicy(updated);

Console.WriteLine(result ? "Updated successfully." : " Update failed.");

}

catch (Exception ex)

{

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

}

}

static void DeletePolicy(IPolicyService service)

{

try

{

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

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

bool result = service.DeletePolicy(id);

Console.WriteLine(result ? " Deleted successfully." : "Deletion failed.");

}

catch (PolicyNotFoundException ex)

{

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

}

}

}

}

**12. appsettings.Properties**

server=(localdb)\MSSQLLocalDB

database=InsuranceDB

port=1433

**13. Program.cs**

using InsuranceManagementSystem.mainmod;

namespace InsuranceManagementSystem

{

public class Program

{

public static void Main(string[] args)

{

MainModule.Start(); // Calls menu-driven logic

}

}

}

**14. PolicyServiceImplTests.cs**

using InsuranceManagementSystem.dao;

using InsuranceManagementSystem.Entity;

using InsuranceManagementSystem.Exceptions;

using NUnit.Framework;

using System.Collections.Generic;

namespace InsuranceManagementSystem.Tests

{

[TestFixture]

public class PolicyServiceImplTests

{

private IPolicyService \_service;

[SetUp]

public void Setup()

{

\_service = new PolicyServiceImpl();

// Ensure a clean test policy exists before every test

try { \_service.DeletePolicy(999); } catch { }

var testPolicy = new Policy(999, "Test Policy", 1200.50, "Unit test coverage");

\_service.CreatePolicy(testPolicy);

}

[TearDown]

public void Cleanup()

{

// Clean up test data after each test

try { \_service.DeletePolicy(999); } catch { }

}

[Test]

public void CreatePolicy\_ShouldReturnTrue\_WhenValidPolicyIsProvided()

{

var newPolicy = new Policy(998, "Another Policy", 1500.00, "Another coverage");

bool result = \_service.CreatePolicy(newPolicy);

Assert.IsTrue(result);

// Clean up

\_service.DeletePolicy(998);

}

[Test]

public void GetPolicy\_ShouldReturnCorrectPolicy\_WhenPolicyExists()

{

var policy = \_service.GetPolicy(999);

Assert.IsNotNull(policy);

Assert.AreEqual("Test Policy", policy.PolicyName);

Assert.AreEqual(1200.50, policy.PremiumAmount);

}

[Test]

public void GetPolicy\_ShouldThrowException\_WhenPolicyDoesNotExist()

{

Assert.Throws<PolicyNotFoundException>(() => \_service.GetPolicy(8888));

}

[Test]

public void UpdatePolicy\_ShouldReturnTrue\_WhenPolicyIsUpdated()

{

var updatedPolicy = new Policy(999, "Updated Policy", 2000.00, "Updated coverage");

bool result = \_service.UpdatePolicy(updatedPolicy);

Assert.IsTrue(result);

var policy = \_service.GetPolicy(999);

Assert.AreEqual("Updated Policy", policy.PolicyName);

Assert.AreEqual(2000.00, policy.PremiumAmount);

}

[Test]

public void GetAllPolicies\_ShouldReturnAtLeastOnePolicy()

{

var allPolicies = \_service.GetAllPolicies();

Assert.IsNotNull(allPolicies);

Assert.IsTrue(allPolicies.Count > 0);

}

[Test]

public void DeletePolicy\_ShouldReturnTrue\_WhenPolicyExists()

{

var tempPolicy = new Policy(997, "ToDelete", 1000, "To be deleted");

\_service.CreatePolicy(tempPolicy);

bool result = \_service.DeletePolicy(997);

Assert.IsTrue(result);

}

[Test]

public void DeletePolicy\_ShouldThrowException\_WhenPolicyDoesNotExist()

{

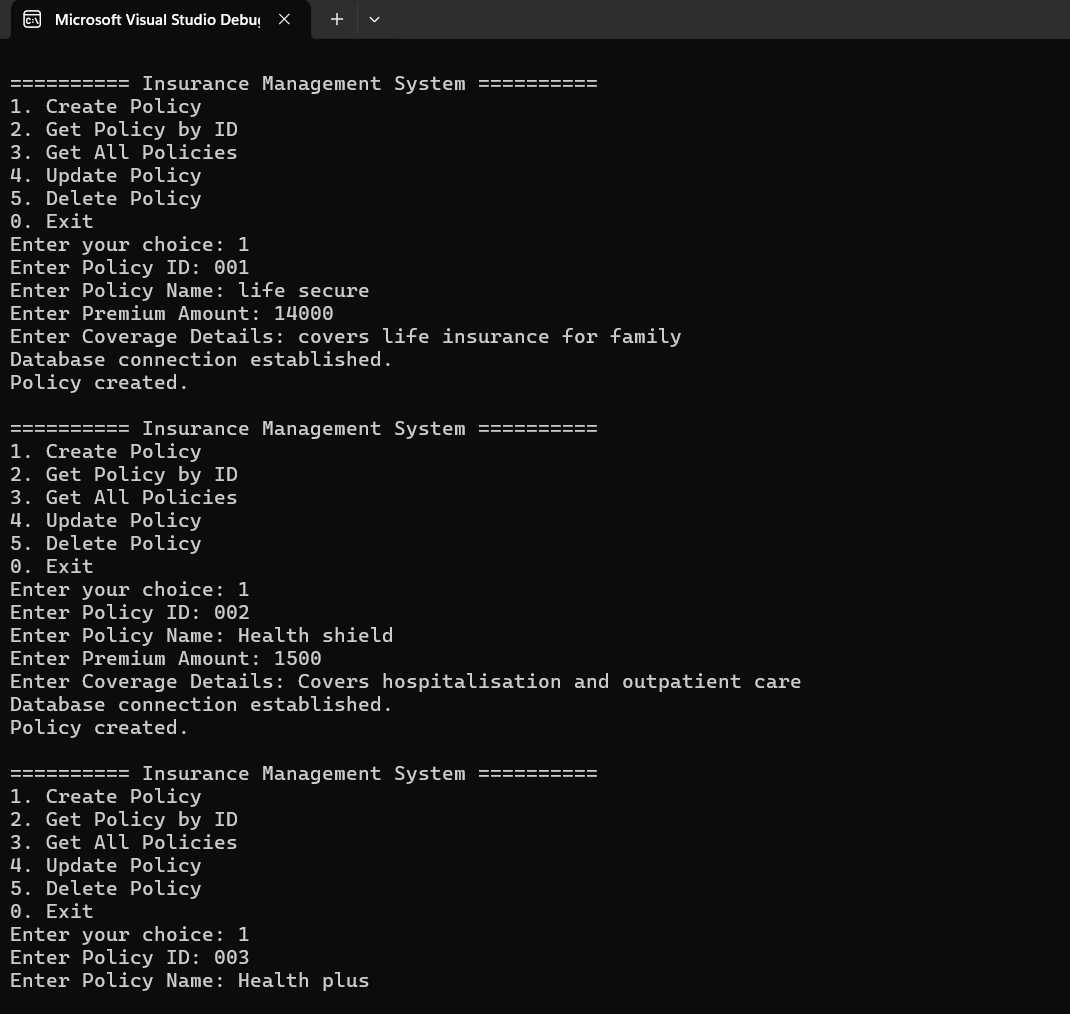
Assert.Throws<PolicyNotFoundException>(() => \_service.DeletePolicy(8888));

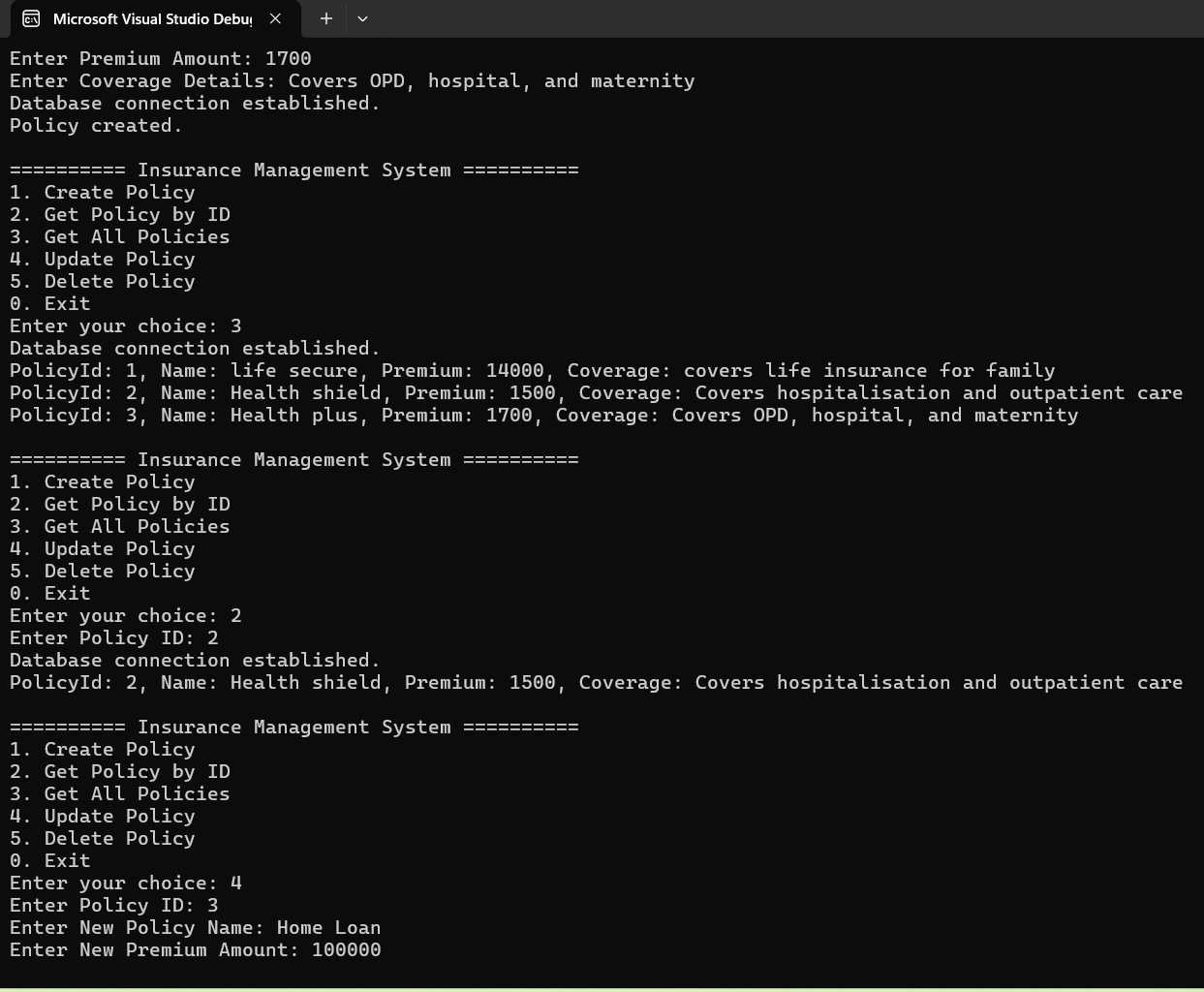
}

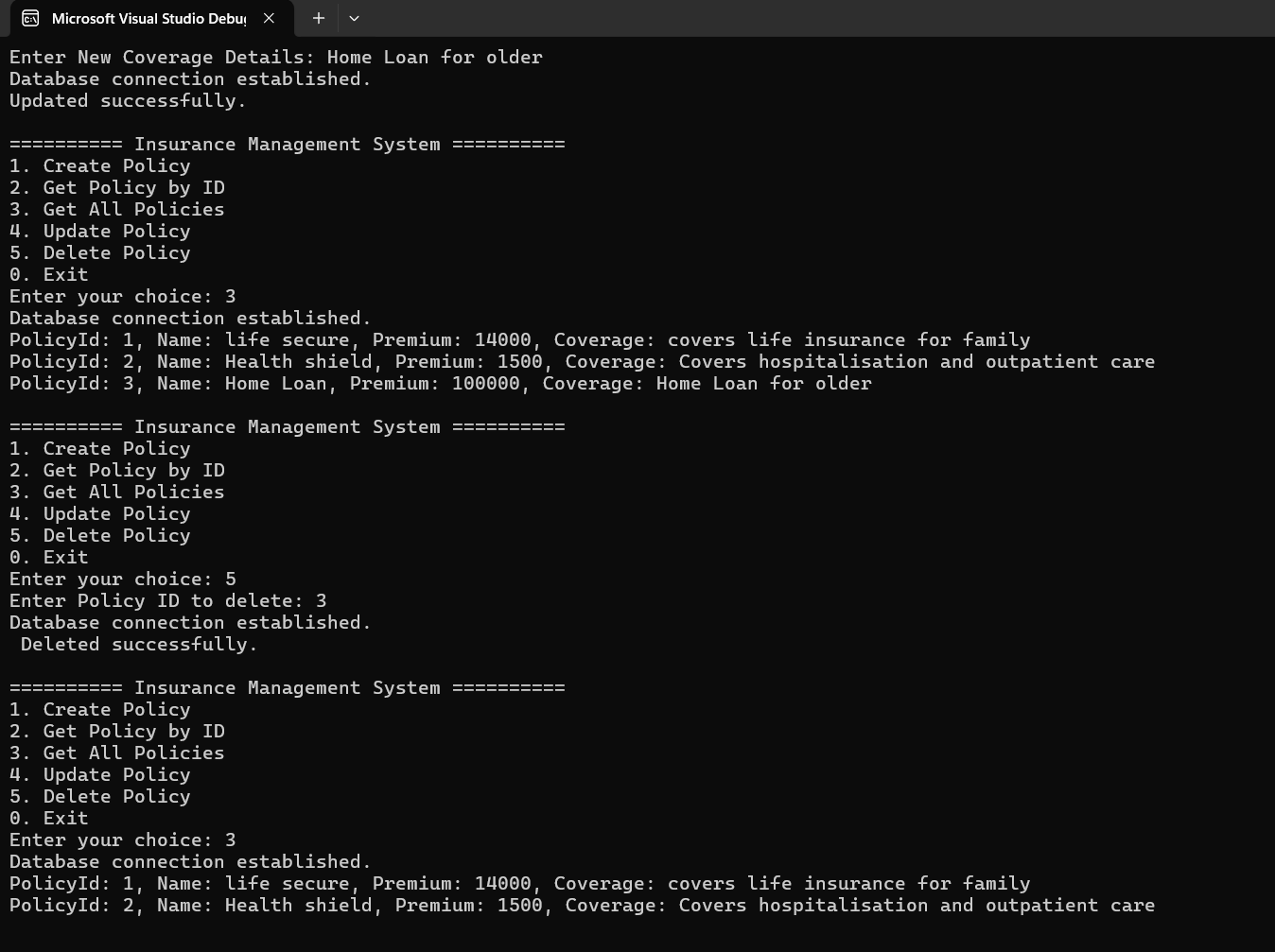
}

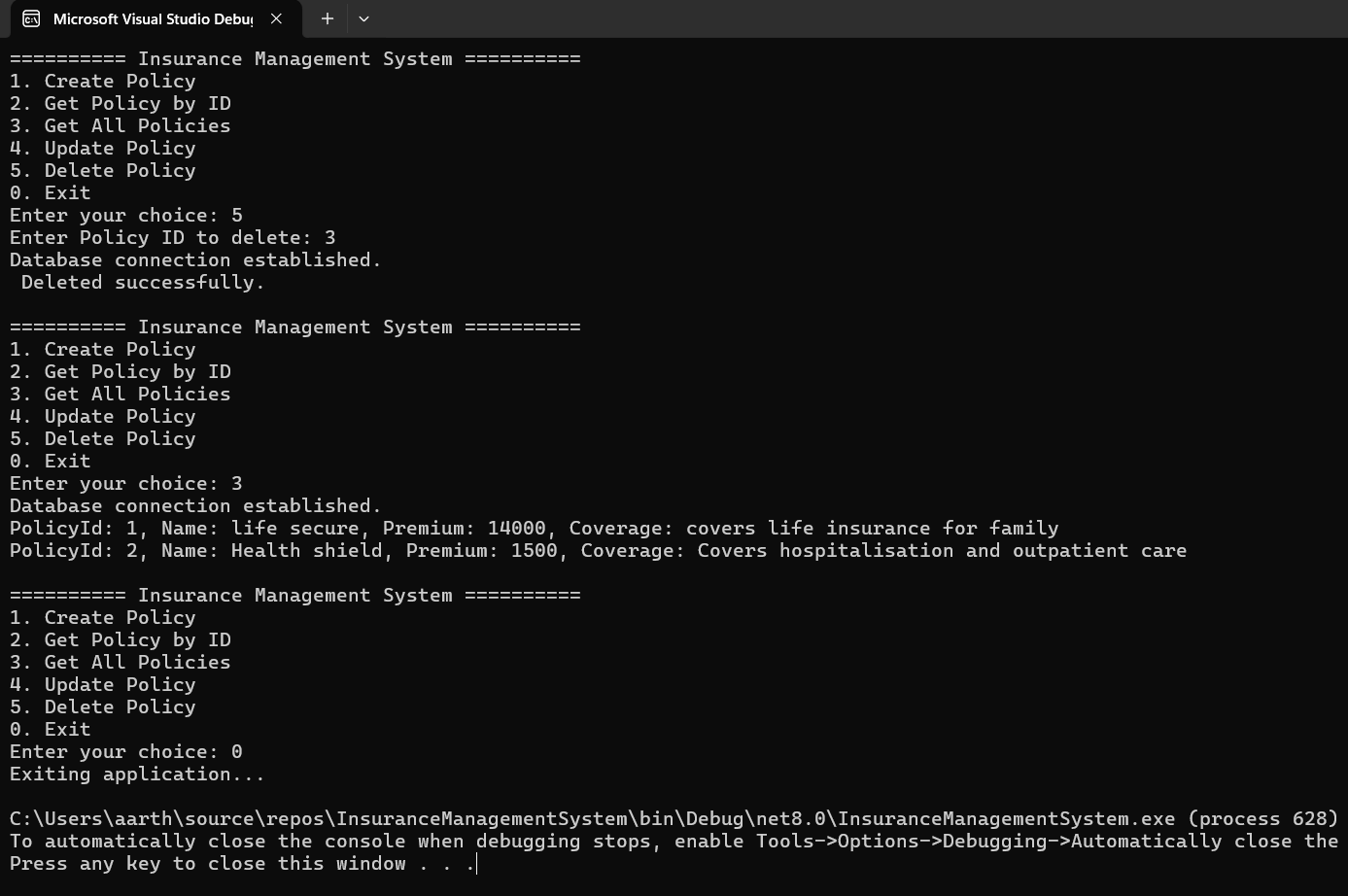
}

**OutPut:**









**Testing:**

