**Design Principles and Patterns**

**Exercise 1: Implementing the Singleton Pattern**

**Logger.cs :-**

using System;

using System.IO;

public class Logger

{

private static readonly Logger \_instance = new Logger();

private readonly string \_logFilePath;

private Logger()

{

\_logFilePath = "log.txt";

if (!File.Exists(\_logFilePath))

{

File.WriteAllText(\_logFilePath, $"[{DateTime.Now:yyyy-MM-dd HH:mm:ss}] Logger initialized.{Environment.NewLine}");

}

}

public static Logger GetInstance()

{

return \_instance;

}

public void Log(string message, string level = "INFO")

{

string logEntry = $"[{DateTime.Now:yyyy-MM-dd HH:mm:ss}] [{level.ToUpper()}] {message}";

Console.WriteLine(logEntry);

File.AppendAllText(\_logFilePath, logEntry + Environment.NewLine);

}

public void Info(string message) => Log(message, "INFO");

public void Warning(string message) => Log(message, "WARNING");

public void Error(string message) => Log(message, "ERROR");

}

**Program.cs :-**

using System;

class Program

{

static void Main(string[] args)

{

Logger logger = Logger.GetInstance();

Console.WriteLine("=== Logger Started ===");

Console.WriteLine("Enter messages to log. Type 'exit' to quit.");

Console.WriteLine("Format: <level> <message>");

Console.WriteLine("Example: info Application started\n");

while (true)

{

Console.Write("Log> ");

string input = Console.ReadLine();

if (string.IsNullOrWhiteSpace(input))

continue;

if (input.Trim().ToLower() == "exit")

break;

string[] parts = input.Split(' ', 2, StringSplitOptions.RemoveEmptyEntries);

if (parts.Length < 2)

{

Console.WriteLine("Invalid input. Format: <level> <message>");

continue;

}

string level = parts[0].ToLower();

string message = parts[1];

switch (level)

{

case "info":

logger.Info(message);

break;

case "warning":

logger.Warning(message);

break;

case "error":

logger.Error(message);

break;

default:

Console.WriteLine("Invalid level. Use info, warning, or error.");

break; } }

logger.Info("Logger stopped by user.");

Console.WriteLine("Exiting logger. Goodbye!");}}

**SingletonPattern.csproj :-**

<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>

<OutputType>Exe</OutputType>

<TargetFramework>net9.0</TargetFramework>

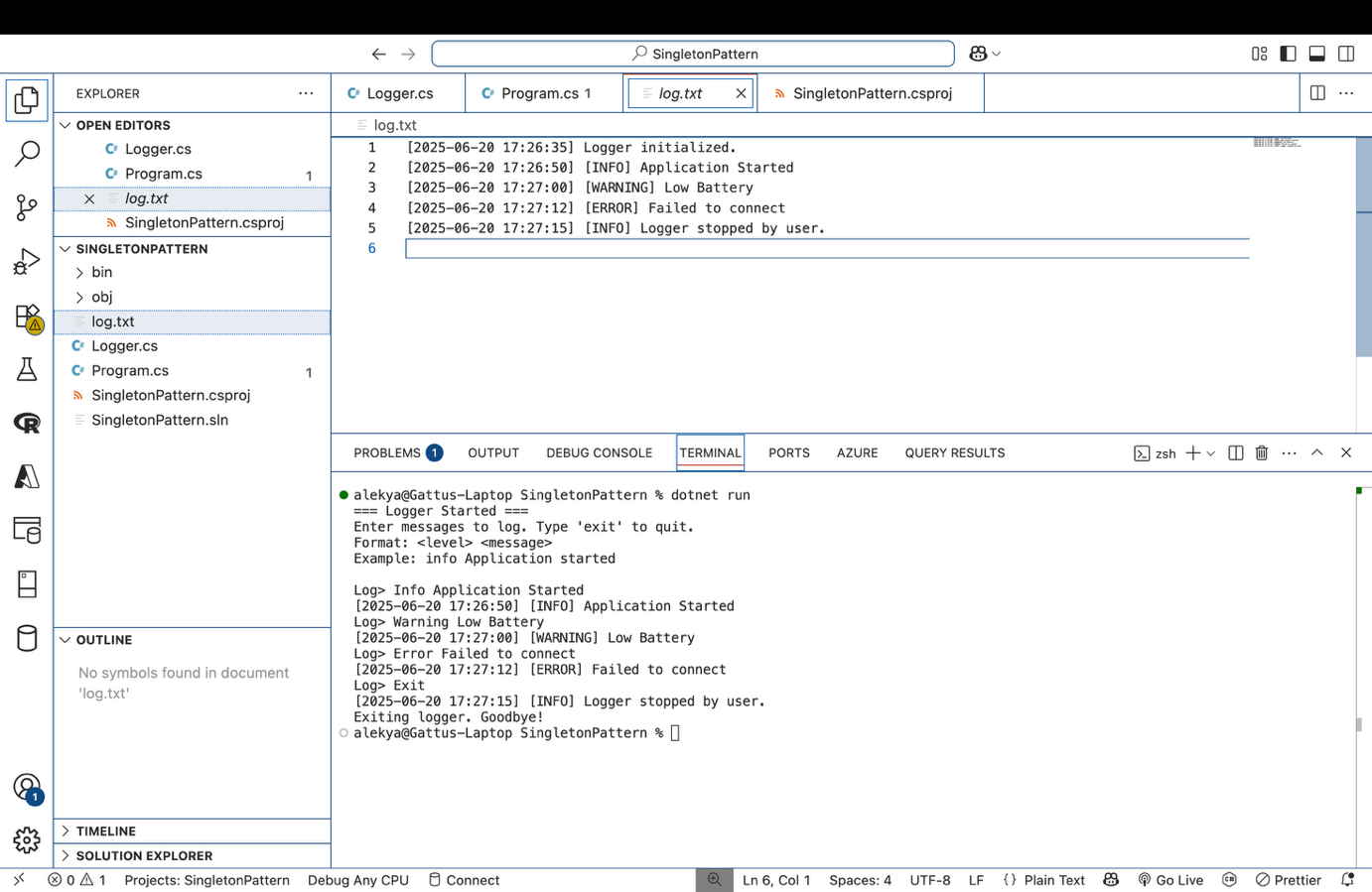
<ImplicitUsings>enable</ImplicitUsings>

<Nullable>enable</Nullable>

</PropertyGroup>

</Project>

**OUTPUT :-**

****

**Log.txt :-**

[2025-06-20 17:26:35] Logger initialized.

[2025-06-20 17:26:50] [INFO] Application Started

[2025-06-20 17:27:00] [WARNING] Low Battery

[2025-06-20 17:27:12] [ERROR] Failed to connect

[2025-06-20 17:27:15] [INFO] Logger stopped by user.

**Exercise 2: Implementing the Factory Method Pattern**

**Program.cs :-**

using System;

class Program

{

static void Main(string[] args)

{

Logger.Log("Choose document type: word / pdf / excel");

string? inputRaw = Console.ReadLine();

if (string.IsNullOrWhiteSpace(inputRaw))

{

Logger.Log("No input given.");

return;

}

string input = inputRaw.ToLower();

DocumentFactory? factory = input switch

{

"word" => new WordDocumentFactory(),

"pdf" => new PdfDocumentFactory(),

"excel" => new ExcelDocumentFactory(),

\_ => null

};

if (factory == null)

{

Logger.Log("Invalid document type.");

return;

}

IDocument document = factory.CreateDocument();

Logger.Log($"Created a {input} document.");

while (true)

{

Logger.Log("Choose an action: open / save / export / exit");

string? actionRaw = Console.ReadLine();

if (string.IsNullOrWhiteSpace(actionRaw))

{

Logger.Log("No action entered.");

continue;

}

string action = actionRaw.ToLower();

switch (action)

{

case "open":

document.Open();

Logger.Log("Action: Open document");

break;

case "save":

document.Save();

Logger.Log("Action: Save document");

break;

case "export":

document.Export();

Logger.Log("Action: Export document");

break;

case "exit":

Logger.Log("Exiting application.");

return;

default:

Logger.Log("Invalid action.");

break;

}

}

}

}

**IDocument.cs :-**

public interface IDocument

{

void Open();

void Save();

void Export();

}

**DocumentFactory.cs :-**

public abstract class DocumentFactory

{

public abstract IDocument CreateDocument();

}

**ExcelDocument.cs :-**

using System;

public class ExcelDocument : IDocument

{

public void Open()

{

Console.WriteLine("Opening an Excel spreadsheet.");

}

public void Save()

{

Console.WriteLine("Saving Excel spreadsheet as .xlsx");

}

public void Export()

{

Console.WriteLine("Exporting Excel sheet to CSV.");

}

}

**ExcelDocumentFactory.cs :-**

public class ExcelDocumentFactory : DocumentFactory

{

public override IDocument CreateDocument() => new ExcelDocument();

}

**FactoryMethodPattern.csproj :-**

<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>

<OutputType>Exe</OutputType>

<TargetFramework>net9.0</TargetFramework>

<ImplicitUsings>enable</ImplicitUsings>

<Nullable>enable</Nullable>

</PropertyGroup>

</Project>

**Logger.cs :-**

using System;

using System.IO;

public static class Logger

{

private static readonly string \_logFilePath = "log.txt";

static Logger()

{

File.AppendAllText(\_logFilePath, $"\n--- Session started: {DateTime.Now} ---\n");

}

public static void Log(string message)

{

string entry = $"[{DateTime.Now:yyyy-MM-dd HH:mm:ss}] {message}";

Console.WriteLine(entry);

File.AppendAllText(\_logFilePath, entry + Environment.NewLine);

}

}

**PdfDocument.cs :-**

using System;

public class PdfDocument : IDocument

{

public void Open()

{

Console.WriteLine("Opening a PDF document.");

}

public void Save()

{

Console.WriteLine("Saving changes to PDF.");

}

public void Export()

{

Console.WriteLine("Exporting PDF to image format.");

}

}

**PdfDocumentFactory.cs :-**

public class PdfDocumentFactory : DocumentFactory

{

public override IDocument CreateDocument() => new PdfDocument();

}

**WordDocument.cs :-**

using System;

public class WordDocument : IDocument

{

public void Open()

{

Console.WriteLine("Opening a Word document.");

}

public void Save()

{

Console.WriteLine("Saving Word document as .docx");

}

public void Export()

{

Console.WriteLine("Exporting Word document as PDF.");

}

}

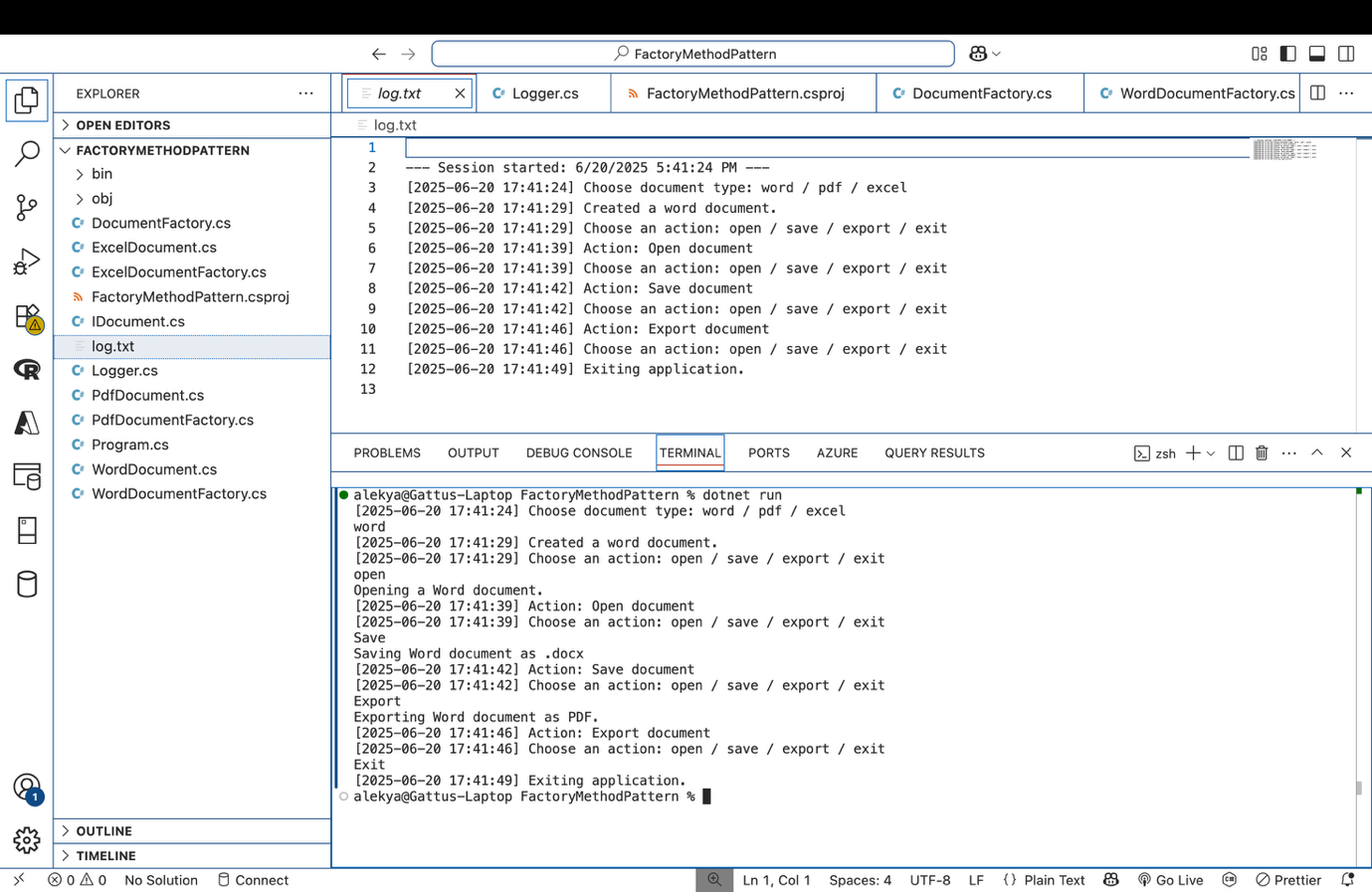
**WordDocumentFactory.cs :-**

public class WordDocumentFactory : DocumentFactory

{

public override IDocument CreateDocument() => new WordDocument(); }

**OUTPUT :-**

****

**log.txt :-**

--- Session started: 6/20/2025 5:41:24 PM ---

[2025-06-20 17:41:24] Choose document type: word / pdf / excel

[2025-06-20 17:41:29] Created a word document.

[2025-06-20 17:41:29] Choose an action: open / save / export / exit

[2025-06-20 17:41:39] Action: Open document

[2025-06-20 17:41:39] Choose an action: open / save / export / exit

[2025-06-20 17:41:42] Action: Save document

[2025-06-20 17:41:42] Choose an action: open / save / export / exit

[2025-06-20 17:41:46] Action: Export document

[2025-06-20 17:41:46] Choose an action: open / save / export / exit

[2025-06-20 17:41:49] Exiting application.

**Exercise 3: Implementing the Proxy Pattern**

**(NOT MANDATORY)**

**Logger.cs :-**

using System;

using System.IO;

public static class Logger

{

public enum LogLevel

{

INFO,

WARNING,

ERROR

}

public static void Log(string imageName, string message, LogLevel level = LogLevel.INFO)

{

string fileName = $"{imageName}.log";

string logMessage = $"[{DateTime.Now:yyyy-MM-dd HH:mm:ss}] [{level}] {message}";

Console.WriteLine(logMessage);

File.AppendAllText(fileName, logMessage + Environment.NewLine);

}

}

**RealImage.cs :-**

public class RealImage : IImage

{

private string fileName;

public RealImage(string fileName)

{

this.fileName = fileName;

LoadFromServer();

}

private void LoadFromServer()

{

Logger.Log(fileName, $"Loading image from remote server: {fileName}", Logger.LogLevel.INFO);

}

public void Display()

{

Logger.Log(fileName, $"Displaying image: {fileName}", Logger.LogLevel.INFO);

}

}

**ProxyPattern.csproj :-**

<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>

<OutputType>Exe</OutputType>

<TargetFramework>net9.0</TargetFramework> <!-- or net8.0 depending on your SDK

<ImplicitUsings>enable</ImplicitUsings>

<Nullable>enable</Nullable>

</PropertyGroup>

</Project>

**ProxyImage.cs :-**

public class ProxyImage : IImage

{

private RealImage? realImage;

private string fileName;

public ProxyImage(string fileName)

{

this.fileName = fileName;

}

public void Display()

{

if (realImage == null)

{

Logger.Log(fileName, $"Lazy loading image.", Logger.LogLevel.INFO);

realImage = new RealImage(fileName);

}

else

{

Logger.Log(fileName, $"Using cached image.", Logger.LogLevel.INFO);

}

realImage.Display();

}

}

**Program.cs :-**

using System;

class Program

{

static void Main()

{

IImage image1 = new ProxyImage("photo1.png");

IImage image2 = new ProxyImage("photo2.png");

image1.Display(); // First load

Console.WriteLine();

image1.Display(); // Cached

Console.WriteLine();

image2.Display(); // First load

}

}

**IImage.cs :-**

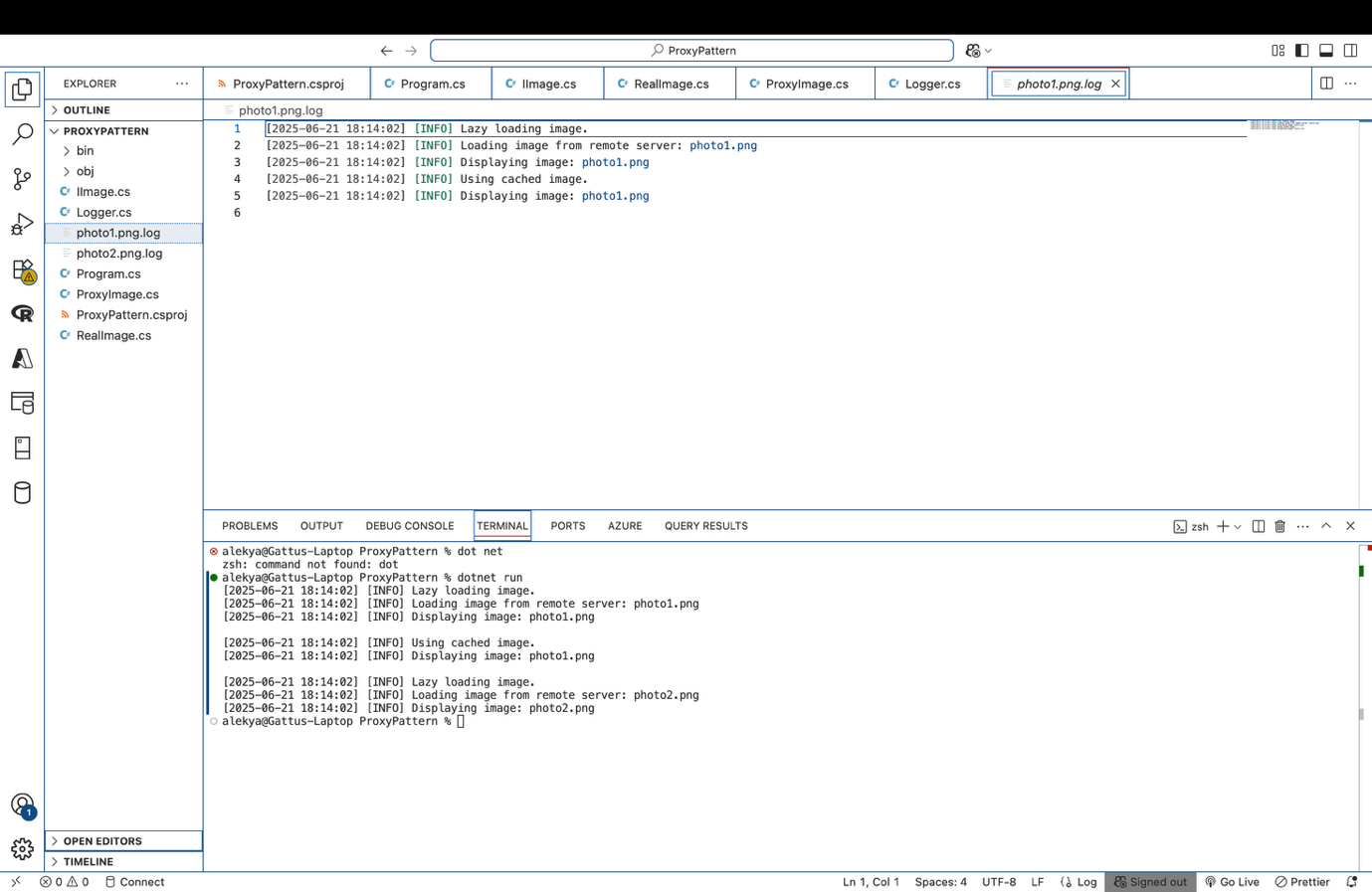
public interface IImage

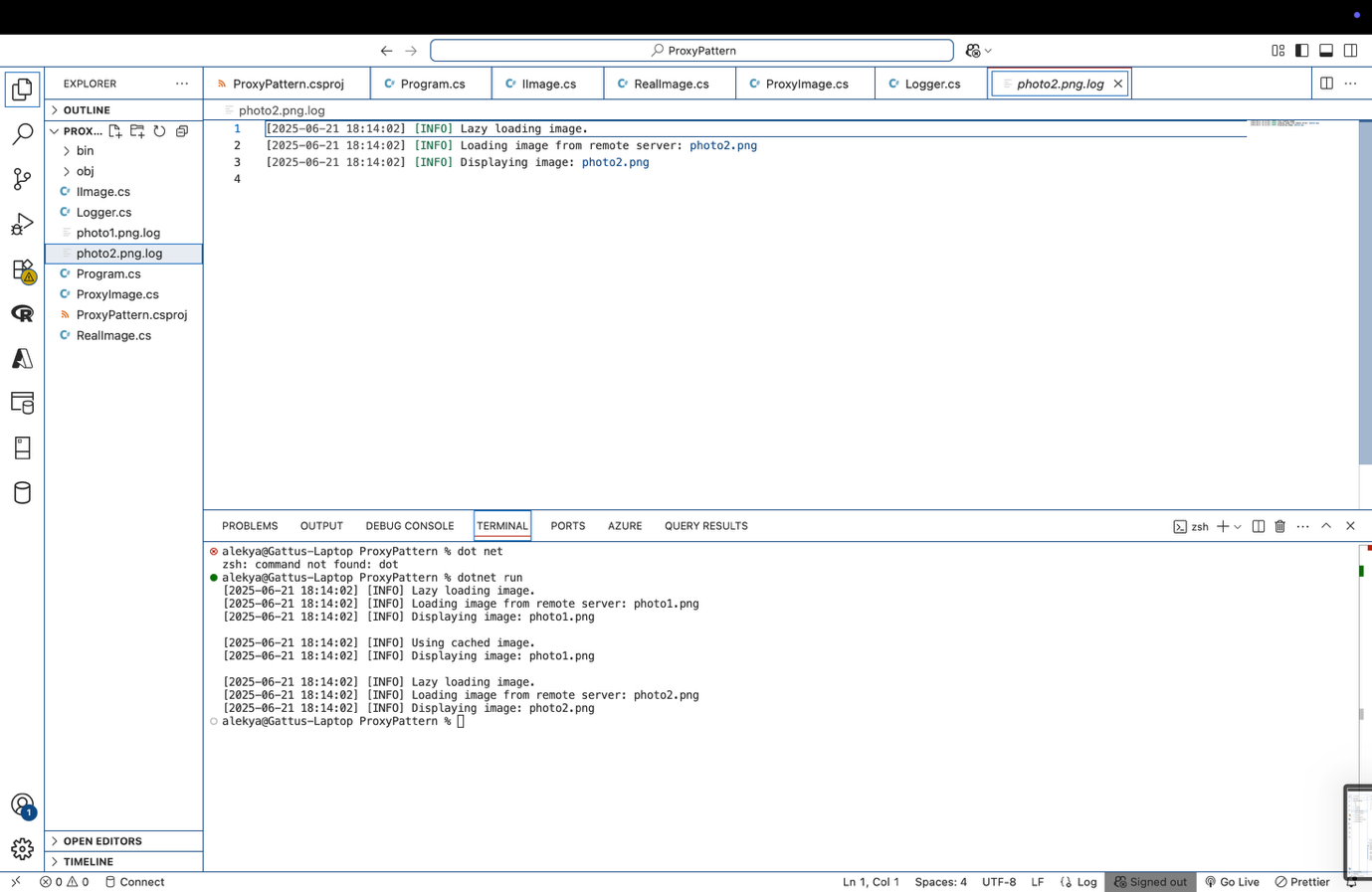
{

void Display();

}

**OUTPUT :-**

****

****

**Photo1.png.log :-**

[2025-06-21 18:14:02] [INFO] Lazy loading image.

[2025-06-21 18:14:02] [INFO] Loading image from remote server: photo1.png

[2025-06-21 18:14:02] [INFO] Displaying image: photo1.png

[2025-06-21 18:14:02] [INFO] Using cached image.

[2025-06-21 18:14:02] [INFO] Displaying image: photo1.png

**Photo2.png.log :-**

[2025-06-21 18:14:02] [INFO] Lazy loading image.

[2025-06-21 18:14:02] [INFO] Loading image from remote server: photo2.png

[2025-06-21 18:14:02] [INFO] Displaying image: photo2.png

**Data Structures and Algorithms**

**Exercise 1: E-commerce Platform Search Function**

**Product.cs :-**

public class Product

{

public int Id { get; set; }

public string Name { get; set; }

public string Category { get; set; }

public double Price { get; set; }

public Product(int id, string name, string category, double price)

{

Id = id;

Name = name;

Category = category;

Price = price;

}

public override string ToString()

{

return $"✅ {Name} (ID: {Id}) – {Category} – ₹{Price}";

}

}

**ProductSearchDemo.csproj :-**

<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>

<OutputType>Exe</OutputType>

<TargetFramework>net9.0</TargetFramework>

<Nullable>enable</Nullable>

</PropertyGroup>

</Project>

**Program.cs :-**

using System;

class Program

{

static void Main()

{

Product[] products = new Product[]

{

new Product(101, "Laptop", "Electronics", 49999),

new Product(102, "Shirt", "Clothing", 999),

new Product(103, "Watch", "Accessories", 1499),

new Product(104, "Camera", "Electronics", 39999),

new Product(105, "Shoes", "Footwear", 1999),

new Product(106, "Mouse", "Electronics", 599),

new Product(107, "Shirt", "Clothing", 899),

new Product(108, "Bag", "Accessories", 1199)

};

Console.WriteLine("\n🔎 Choose Search Type:");

Console.WriteLine("1. Search by Product ID");

Console.WriteLine("2. Search by Product Name");

Console.WriteLine("3. Search by Category");

Console.WriteLine("4. Search by Price Range");

Console.Write("Enter your choice (1–4): ");

string? choice = Console.ReadLine();

switch (choice)

{

case "1":

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

int id = int.Parse(Console.ReadLine() ?? "0");

SearchHelper.SearchById(products, id);

break;

case "2":

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

string name = Console.ReadLine() ?? "";

SearchHelper.SearchByName(products, name);

break;

case "3":

Console.Write("Enter Category: ");

string category = Console.ReadLine() ?? "";

SearchHelper.SearchByCategory(products, category);

break;

case "4":

Console.Write("Enter Min Price: ");

double min = double.Parse(Console.ReadLine() ?? "0");

Console.Write("Enter Max Price: ");

double max = double.Parse(Console.ReadLine() ?? "0");

SearchHelper.SearchByPriceRange(products, min, max);

break;

default:

Console.WriteLine("⚠️ Invalid choice.");

break;

}

Console.WriteLine("\n📁 Results saved to SearchResults.txt");

}

}

**Logger.cs :-**

using System;

using System.IO;

public static class Logger

{

private static readonly string resultFile = "SearchResults.txt";

public static void LogResult(string content)

{

Console.WriteLine(content);

File.AppendAllText(resultFile, content + Environment.NewLine);

}

public static void LogHeader(string header)

{

File.WriteAllText(resultFile, header + Environment.NewLine);

Console.WriteLine(header);

}

public static void LogFooter()

{

string footer = $"🕒 Generated at: {DateTime.Now}";

File.AppendAllText(resultFile, footer + Environment.NewLine);

Console.WriteLine(footer);

}

}

**SearchHelper.cs :-**

using System;

using System.Diagnostics;

using System.Linq;

public static class SearchHelper

{

public static void SearchById(Product[] products, int id)

{

var stopwatch = Stopwatch.StartNew();

var product = products.FirstOrDefault(p => p.Id == id);

stopwatch.Stop();

Logger.LogHeader($"🔍 Search Result – Product ID: {id}");

if (product != null)

Logger.LogResult(product.ToString());

else

Logger.LogResult("❌ Product not found.");

Logger.LogResult($"⌛ Time Complexity: O(n), Time Taken: {stopwatch.Elapsed.TotalMilliseconds:F4} ms");

Logger.LogFooter();

}

public static void SearchByName(Product[] products, string name)

{

var stopwatch = Stopwatch.StartNew();

var matches = products

.Where(p => p.Name.Equals(name, StringComparison.OrdinalIgnoreCase))

.ToList();

stopwatch.Stop();

Logger.LogHeader($"🔍 Search Results – Product Name: {name}");

if (matches.Count > 0)

{

foreach (var product in matches)

Logger.LogResult(product.ToString());

}

else

Logger.LogResult("❌ No products found.");

Logger.LogResult($"⌛ Time Complexity: O(n), Time Taken: {stopwatch.Elapsed.TotalMilliseconds:F4} ms");

Logger.LogFooter();

}

public static void SearchByCategory(Product[] products, string category)

{

var stopwatch = Stopwatch.StartNew();

var matches = products

.Where(p => p.Category.Equals(category, StringComparison.OrdinalIgnoreCase))

.ToList();

stopwatch.Stop();

Logger.LogHeader($"📂 Search Results – Category: {category}");

if (matches.Count > 0)

{

foreach (var product in matches)

Logger.LogResult(product.ToString());

}

else

Logger.LogResult("❌ No products found.");

Logger.LogResult($"⌛ Time Complexity: O(n), Time Taken: {stopwatch.Elapsed.TotalMilliseconds:F4} ms");

Logger.LogFooter();

}

public static void SearchByPriceRange(Product[] products, double minPrice, double maxPrice)

{

var stopwatch = Stopwatch.StartNew();

var filtered = products

.Where(p => p.Price >= minPrice && p.Price <= maxPrice)

.ToList();

stopwatch.Stop();

Logger.LogHeader($"📦 Search Results – Price Range: ₹{minPrice} – ₹{maxPrice}");

foreach (var product in filtered)

Logger.LogResult(product.ToString());

if (filtered.Count == 0)

Logger.LogResult("❌ No products found.");

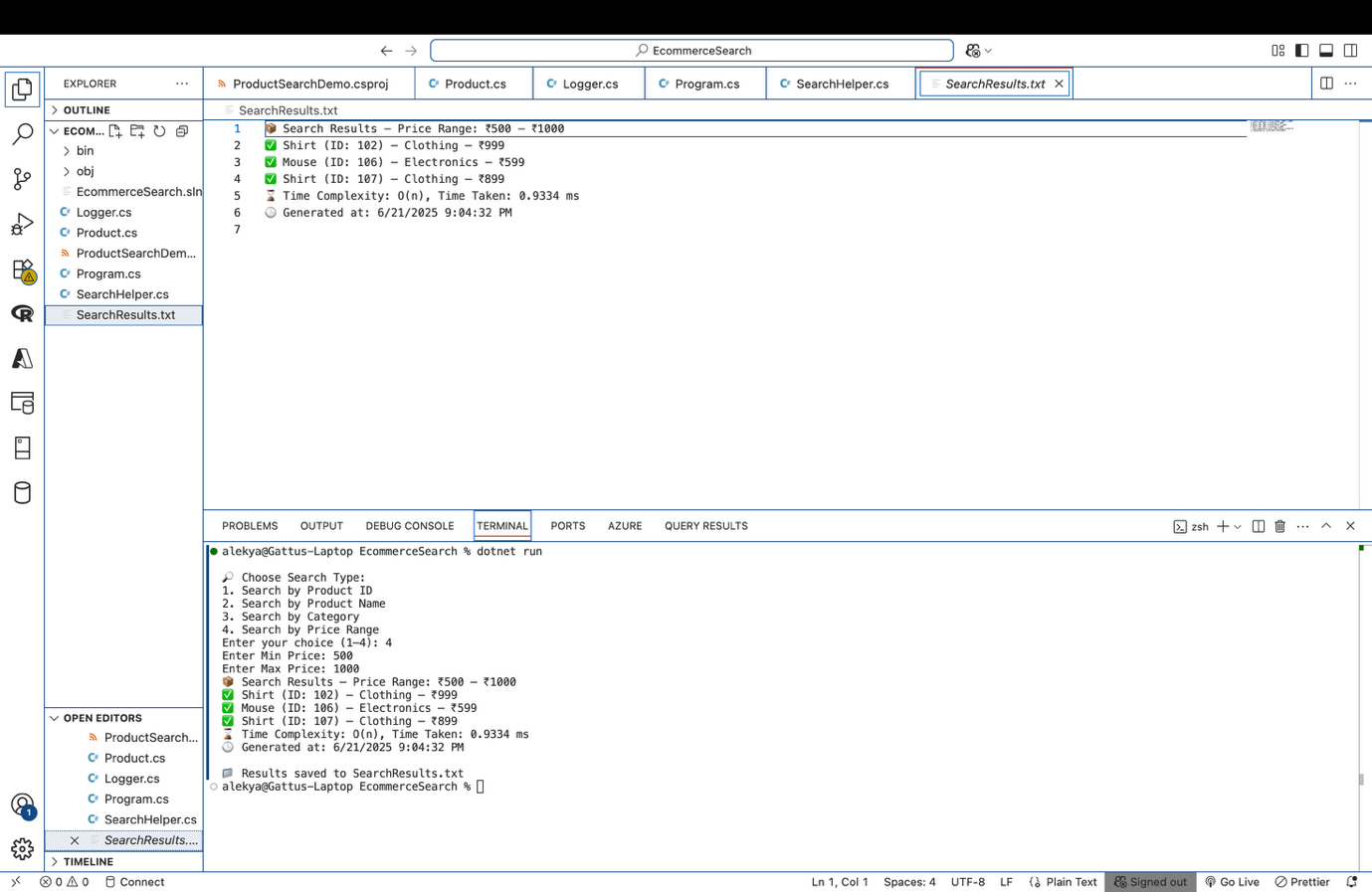
Logger.LogResult($"⌛ Time Complexity: O(n), Time Taken: {stopwatch.Elapsed.TotalMilliseconds:F4} ms");

Logger.LogFooter();

}

}

**OUTPUT :-**

****

**SerachResult.txt :-**

📦 Search Results – Price Range: ₹500 – ₹1000

✅ Shirt (ID: 102) – Clothing – ₹999

✅ Mouse (ID: 106) – Electronics – ₹599

✅ Shirt (ID: 107) – Clothing – ₹899

⌛ Time Complexity: O(n), Time Taken: 0.9334 ms

🕒 Generated at: 6/21/2025 9:04:32 PM

**Exercise 2: Financial Forecasting**

**Program.cs :-**

using System;

using System.Diagnostics;

using System.IO;

class Program

{

static double CalculateForecast(double amount, double rate, int years)

{

// Base case

if (years == 0)

return amount;

// Recursive case: compound annually

return CalculateForecast(amount \* (1 + rate / 100), rate, years - 1);

}

static void Main()

{

Console.Write("Enter initial amount: ");

double initialAmount = double.Parse(Console.ReadLine() ?? "0");

Console.Write("Enter number of years to forecast: ");

int years = int.Parse(Console.ReadLine() ?? "0");

Console.Write("How many growth rates do you want to compare? ");

int rateCount = int.Parse(Console.ReadLine() ?? "0");

double[] rates = new double[rateCount];

for (int i = 0; i < rateCount; i++)

{

Console.Write($"Enter growth rate {i + 1} (as %): ");

rates[i] = double.Parse(Console.ReadLine() ?? "0");

}

string filePath = "forecast.txt";

using StreamWriter writer = new StreamWriter(filePath);

writer.WriteLine($"📈 Forecast for Initial Amount: ₹{initialAmount} over {years} years:");

writer.WriteLine(new string('-', 50));

var stopwatch = Stopwatch.StartNew();

foreach (double rate in rates)

{

double result = CalculateForecast(initialAmount, rate, years);

string line = $"Growth Rate {rate}% → ₹{result:F2}";

Console.WriteLine(line);

writer.WriteLine(line);

}

stopwatch.Stop();

Console.WriteLine("✔ Forecast saved to forecast.txt");

writer.WriteLine(new string('-', 50));

writer.WriteLine($"⌛ Time Complexity: O(n), Time Taken: {stopwatch.Elapsed.TotalMilliseconds:F4} ms");

}

}

**FinancialForecasting.csproj :-**

<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>

<OutputType>Exe</OutputType>

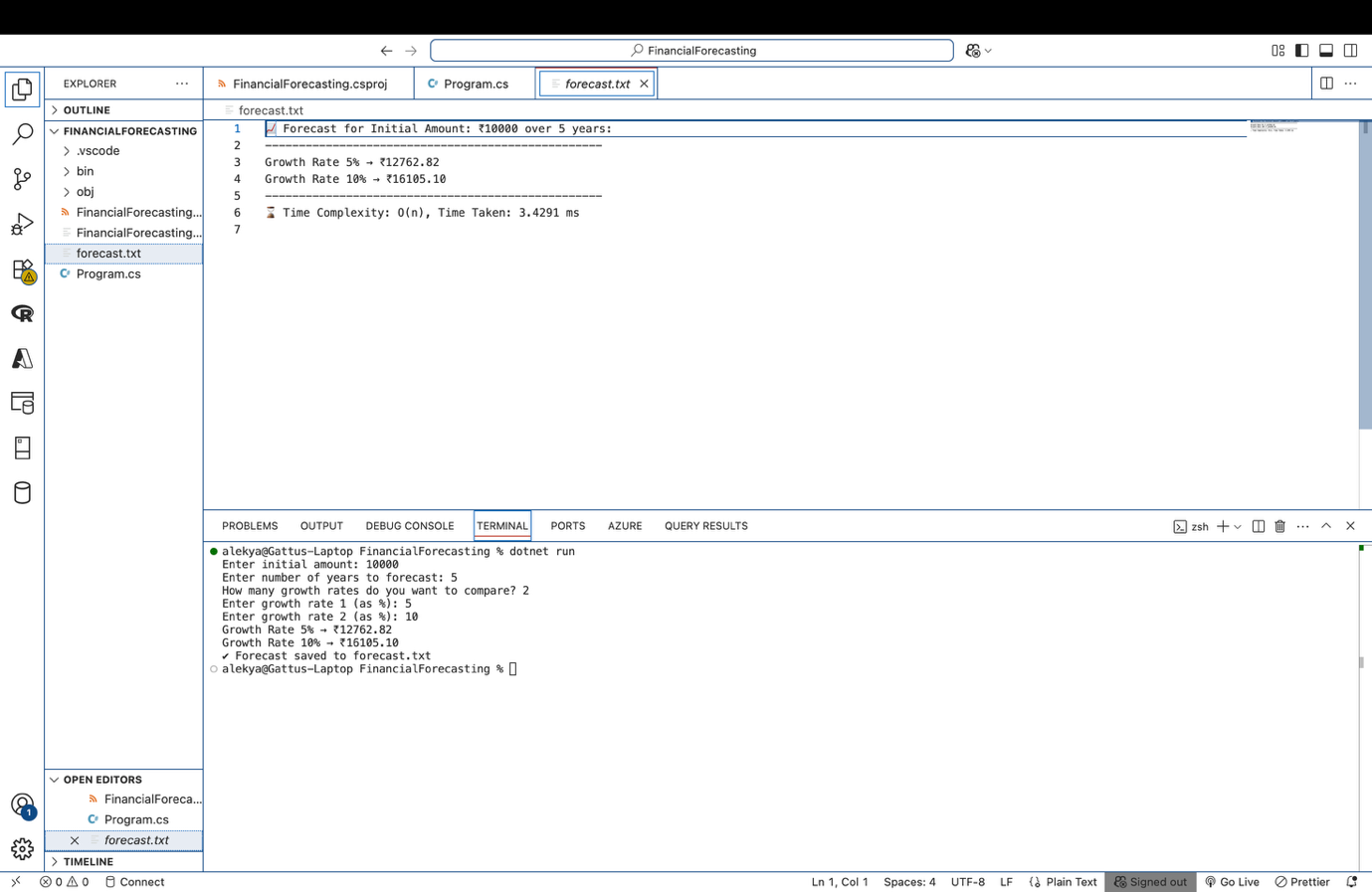
<TargetFramework>net9.0</TargetFramework>

<Nullable>enable</Nullable>

</PropertyGroup>

</Project>

**OUTPUT :-**

****

**Forecast.txt :-**

📈 Forecast for Initial Amount: ₹10000 over 5 years:

--------------------------------------------------

Growth Rate 5% → ₹12762.82

Growth Rate 10% → ₹16105.10

--------------------------------------------------

⌛ Time Complexity: O(n), Time Taken: 3.4291 ms

**Exercise 3: Employee Management System**

**Program.cs :-**

using System;

using System.Diagnostics;

class Program

{

static Employee?[] employees = new Employee?[100];

static int count = 0;

static void AddEmployee(Employee emp)

{

var stopwatch = Stopwatch.StartNew();

if (count >= employees.Length)

{

stopwatch.Stop();

Logger.Log($"❌ Employee array is full. [Time Complexity: O(1), Duration: {stopwatch.ElapsedMilliseconds} ms]");

return;

}

employees[count++] = emp;

stopwatch.Stop();

Logger.Log($"✅ Employee added: {emp.EmployeeId}, {emp.Name}, {emp.Position}, ₹{emp.Salary} [Time Complexity: O(1), Duration: {stopwatch.ElapsedMilliseconds} ms]");

}

static void SearchEmployee(int id)

{

var stopwatch = Stopwatch.StartNew();

for (int i = 0; i < count; i++)

{

var emp = employees[i];

if (emp != null && emp.EmployeeId == id)

{

stopwatch.Stop();

Logger.Log($"🔍 Found Employee ID {id}: {emp.Name}, {emp.Position}, ₹{emp.Salary} [Time Complexity: O(n), Duration: {stopwatch.ElapsedMilliseconds} ms]");

return;

}

}

stopwatch.Stop();

Logger.Log($"❌ Employee ID {id} not found. [Time Complexity: O(n), Duration: {stopwatch.ElapsedMilliseconds} ms]");

}

static void TraverseEmployees()

{

var stopwatch = Stopwatch.StartNew();

if (count == 0)

{

stopwatch.Stop();

Logger.Log($"ℹ️ No employees to display. [Time Complexity: O(1), Duration: {stopwatch.ElapsedMilliseconds} ms]");

return;

}

Logger.Log($"📋 Traversing Employee List: [Time Complexity: O(n)]");

for (int i = 0; i < count; i++)

{

var emp = employees[i];

if (emp != null)

{

Logger.Log($"ID: {emp.EmployeeId}, Name: {emp.Name}, Position: {emp.Position}, Salary: ₹{emp.Salary}");

}

}

stopwatch.Stop();

Logger.Log($"✅ Traversal Complete. Duration: {stopwatch.ElapsedMilliseconds} ms");

}

static void DeleteEmployee(int id)

{

var stopwatch = Stopwatch.StartNew();

for (int i = 0; i < count; i++)

{

var emp = employees[i];

if (emp != null && emp.EmployeeId == id)

{

string deletedInfo = $"ID: {emp.EmployeeId}, Name: {emp.Name}";

for (int j = i; j < count - 1; j++)

{

employees[j] = employees[j + 1];

}

employees[--count] = null;

stopwatch.Stop();

Logger.Log($"🗑 Deleted Employee → {deletedInfo} [Time Complexity: O(n), Duration: {stopwatch.ElapsedMilliseconds} ms]");

return;

}

}

stopwatch.Stop();

Logger.Log($"❌ Employee ID {id} not found for deletion. [Time Complexity: O(n), Duration: {stopwatch.ElapsedMilliseconds} ms]");

}

static void Main()

{

AddEmployee(new Employee(1, "Alekya", "Developer", 65000));

AddEmployee(new Employee(2, "Ravi", "Manager", 85000));

AddEmployee(new Employee(3, "Sara", "Analyst", 55000));

TraverseEmployees();

Console.WriteLine();

SearchEmployee(2);

Console.WriteLine();

DeleteEmployee(2);

TraverseEmployees();

}

}

**Logger.cs :-**

using System;

using System.IO;

public static class Logger

{

private static readonly string logPath = "employee\_log.txt";

static Logger()

{

File.AppendAllText(logPath, $"\n--- Log Started at {DateTime.Now} ---\n");

}

public static void Log(string message)

{

string logMessage = $"[{DateTime.Now:yyyy-MM-dd HH:mm:ss}] {message}";

Console.WriteLine(logMessage);

File.AppendAllText(logPath, logMessage + Environment.NewLine);

}

}

**EmployeeManagement.csproj :-**

<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>

<OutputType>Exe</OutputType>

<TargetFramework>net9.0</TargetFramework>

<ImplicitUsings>enable</ImplicitUsings>

<Nullable>enable</Nullable>

</PropertyGroup>

</Project>

**Employee.cs :-**

public class Employee

{

public int EmployeeId { get; set; }

public string Name { get; set; }

public string Position { get; set; }

public double Salary { get; set; }

public Employee(int employeeId, string name, string position, double salary)

{

EmployeeId = employeeId;

Name = name;

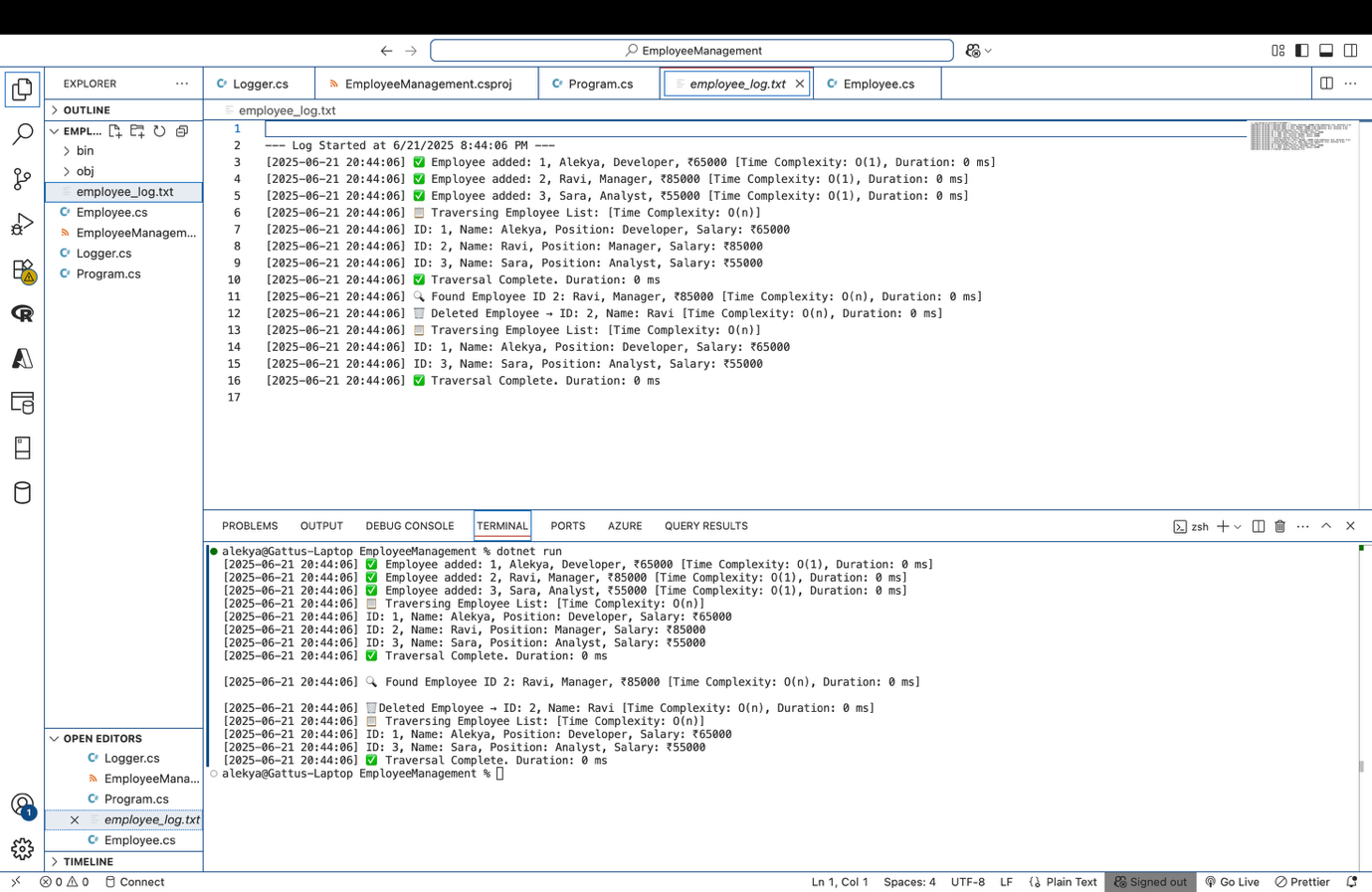
Position = position;

Salary = salary;

}

}

**OUTPUT :-**



**Employee\_log.txt :-**

--- Log Started at 6/21/2025 8:44:06 PM ---

[2025-06-21 20:44:06] ✅ Employee added: 1, Alekya, Developer, ₹65000 [Time Complexity: O(1), Duration: 0 ms]

[2025-06-21 20:44:06] ✅ Employee added: 2, Ravi, Manager, ₹85000 [Time Complexity: O(1), Duration: 0 ms]

[2025-06-21 20:44:06] ✅ Employee added: 3, Sara, Analyst, ₹55000 [Time Complexity: O(1), Duration: 0 ms]

[2025-06-21 20:44:06] 📋 Traversing Employee List: [Time Complexity: O(n)]

[2025-06-21 20:44:06] ID: 1, Name: Alekya, Position: Developer, Salary: ₹65000

[2025-06-21 20:44:06] ID: 2, Name: Ravi, Position: Manager, Salary: ₹85000

[2025-06-21 20:44:06] ID: 3, Name: Sara, Position: Analyst, Salary: ₹55000

[2025-06-21 20:44:06] ✅ Traversal Complete. Duration: 0 ms

[2025-06-21 20:44:06] 🔍 Found Employee ID 2: Ravi, Manager, ₹85000 [Time Complexity: O(n), Duration: 0 ms]

[2025-06-21 20:44:06] 🗑 Deleted Employee → ID: 2, Name: Ravi [Time Complexity: O(n), Duration: 0 ms]

[2025-06-21 20:44:06] 📋 Traversing Employee List: [Time Complexity: O(n)]

[2025-06-21 20:44:06] ID: 1, Name: Alekya, Position: Developer, Salary: ₹65000

[2025-06-21 20:44:06] ID: 3, Name: Sara, Position: Analyst, Salary: ₹55000

[2025-06-21 20:44:06] ✅ Traversal Complete. Duration: 0 ms