# Usage Analyzer Solution Code

## Analyze Project

### Program.cs

using Microsoft.Extensions.Configuration;  
using Microsoft.Extensions.DependencyInjection;  
using Microsoft.Extensions.Logging;  
using Spectre.Console;  
  
namespace Analyze;  
  
public class Program  
{  
 private static async Task Main(string[] args)  
 {  
 var configuration = new ConfigurationBuilder()  
 .SetBasePath(Directory.GetCurrentDirectory())  
 .AddJsonFile("appsettings.json", false)  
 .Build();  
  
 var services = new ServiceCollection();  
 ConfigureServices(services, configuration);  
 var serviceProvider = services.BuildServiceProvider();  
  
 var logger = serviceProvider.GetRequiredService<ILogger<Program>>();  
 var analysisService = serviceProvider.GetRequiredService<AnalysisService>();  
 var consoleUi = new ConsoleUi(logger);  
  
 try  
 {  
 consoleUi.DisplayWelcome();  
  
 // Get all DTO classes  
 var dtoClasses = analysisService.GetDtoClasses().ToList();  
 if (!dtoClasses.Any())  
 {  
 AnsiConsole.MarkupLine("[red]No DTO classes found in the Dto project.[/]");  
 return;  
 }  
  
 // Let user select a class  
 var selectedClass = consoleUi.PromptForClassSelection(dtoClasses);  
 if (selectedClass == null)  
 {  
 return;  
 }  
  
 // Ask user for property usage output format  
 var propertyUsageFormat = consoleUi.PromptForPropertyUsageFormat();  
  
 consoleUi.DisplayAnalysisStart(selectedClass);  
  
 // Analyze usage  
 var (classUsage, propertyUsage) = await analysisService.AnalyzeUsageAsync(  
 selectedClass);  
  
 // Display results  
 consoleUi.DisplayResults(classUsage, propertyUsage, selectedClass, propertyUsageFormat);  
 }  
 catch (Exception ex)  
 {  
 consoleUi.DisplayError(ex);  
 }  
 }  
  
 private static void ConfigureServices(IServiceCollection services, IConfiguration configuration)  
 {  
 services.AddLogging(builder =>  
 {  
 builder.AddConsole();  
 builder.AddConfiguration(configuration.GetSection("Logging"));  
 });  
  
 services.AddSingleton<AnalysisService>();  
 }  
}

### AnalysisService.cs

using System.Reflection;  
using Microsoft.CodeAnalysis;  
using Microsoft.CodeAnalysis.CSharp;  
using Microsoft.CodeAnalysis.CSharp.Syntax;  
using Microsoft.CodeAnalysis.Host.Mef;  
using Microsoft.CodeAnalysis.Text;  
using Microsoft.Extensions.Logging;  
using Spectre.Console;  
  
namespace Analyze;  
  
public class AnalysisService  
{  
 private readonly HashSet<Type> \_analyzedTypes = new();  
 private readonly ILogger<AnalysisService> \_logger;  
 private readonly string \_solutionDir;  
 private readonly string \_solutionPath;  
  
 public AnalysisService(ILogger<AnalysisService> logger)  
 {  
 \_logger = logger;  
 \_solutionPath = FindSolutionFile();  
 \_solutionDir = Path.GetDirectoryName(\_solutionPath) ??  
 throw new InvalidOperationException("Solution directory is null");  
 \_logger.LogInformation("Resolved solution directory: {SolutionDir}", \_solutionDir);  
 }  
  
 public IEnumerable<Type> GetDtoClasses()  
 {  
 // Path to the Dto.dll (adjust if needed)  
 var dtoDllPath = Path.Combine(\_solutionDir, "Dto", "bin", "Debug", "net10.0", "Dto.dll");  
 if (!File.Exists(dtoDllPath))  
 {  
 throw new FileNotFoundException($"Dto.dll not found at {dtoDllPath}. Please build the Dto project first.");  
 }  
  
 var assembly = Assembly.LoadFrom(dtoDllPath);  
 return assembly.GetTypes()  
 .Where(t => t.IsClass && t.Namespace == "Dto")  
 .ToList();  
 }  
  
 public async Task<(Dictionary<string, int>, Dictionary<UsageKey, int>)> AnalyzeUsageAsync(Type selectedClass)  
 {  
 \_logger.LogDebug("Starting analysis for class: {SelectedClassFullName}", selectedClass.FullName);  
 var classUsage = new Dictionary<string, int>();  
 var propertyUsage = new Dictionary<UsageKey, int>();  
 \_analyzedTypes.Clear();  
  
 using var workspace = new AdhocWorkspace(MefHostServices.Create(MefHostServices.DefaultAssemblies));  
 workspace.WorkspaceFailed += (sender, args) => { Console.WriteLine($"Warning: {args.Diagnostic}"); };  
 var projectPaths = GetProjectPathsFromSolution(\_solutionPath);  
 foreach (var projectPath in projectPaths)  
 {  
 LoadProjectIntoWorkspace(workspace, projectPath.Replace("\\", "/"));  
 }  
  
 var solution = workspace.CurrentSolution;  
  
 \_logger.LogDebug("Loaded solution with {ProjectCount} projects.", solution.Projects.Count());  
  
 var shouldSkipTetsProjects = AnsiConsole.Prompt(  
 new SelectionPrompt<string>()  
 .Title("Skip test project analysis?")  
 .AddChoices("Yes", "No")) == "Yes";  
  
 // Queue of types to analyze  
 var typesToAnalyze = new Queue<Type>();  
 typesToAnalyze.Enqueue(selectedClass);  
 \_logger.LogDebug("Initial queue size: {Count}", typesToAnalyze.Count);  
  
 while (typesToAnalyze.Count > 0)  
 {  
 var currentType = typesToAnalyze.Dequeue();  
  
 // Skip if already analyzed  
 if (\_analyzedTypes.Contains(currentType))  
 {  
 \_logger.LogDebug("Skipping already analyzed type: {CurrentTypeFullName}", currentType.FullName);  
 continue;  
 }  
  
 \_analyzedTypes.Add(currentType);  
 \_logger.LogInformation("Analyzing type: {CurrentTypeFullName}", currentType.FullName);  
   
 foreach (var project in solution.Projects)  
 {  
 // Skip Test project  
 if (shouldSkipTetsProjects && project.Name.EndsWith("Tests"))  
 {  
 \_logger.LogInformation("Skipping test project {ProjectName}.", project.Name);  
 continue;  
 }  
   
 // Skip Analyze project  
 if (project.Name == "Analyze" ||  
 project.Name == "Dto")  
 {  
 \_logger.LogInformation("Skipping {ProjectName} project.", project.Name);  
 continue;  
 }  
  
 var dtoAssemblyPath = Path.Combine(\_solutionDir, "Dto", "bin", "Debug", "net10.0", "Dto.dll");  
  
 var compilation = (await project  
 .GetCompilationAsync())?  
 .AddReferences(MetadataReference.CreateFromFile(dtoAssemblyPath));  
 if (compilation == null)  
 {  
 continue;  
 }  
  
 foreach (var document in project.Documents)  
 {  
 var semanticModel = await document.GetSemanticModelAsync();  
 if (semanticModel == null)  
 {  
 continue;  
 }  
  
 var syntaxTree = await document.GetSyntaxTreeAsync();  
 if (syntaxTree == null)  
 {  
 continue;  
 }  
  
 var root = await syntaxTree.GetRootAsync();  
 var filePath = document.FilePath ?? "unknown";  
  
 // Analyze class usage  
 var classUsages = root.DescendantNodes()  
 .OfType<IdentifierNameSyntax>()  
 .Where(usage => semanticModel.GetSymbolInfo(usage).Symbol?.ContainingType == currentType);  
  
 foreach (var usage in classUsages)  
 {  
 if (!classUsage.TryAdd(filePath, 1))  
 {  
 classUsage[filePath]++;  
 }  
 }  
  
 // Analyze property usage  
 var propertyUsages = root.DescendantNodes()  
 .OfType<MemberAccessExpressionSyntax>()  
 .Where(usage => semanticModel.GetSymbolInfo(usage).Symbol?.ContainingType == currentType);  
  
 foreach (var usage in propertyUsages)  
 {  
 var attribute = GetClassAndFieldName(semanticModel, usage);  
 if (string.IsNullOrWhiteSpace(attribute.ClassName))  
 {  
 attribute = attribute with { ClassName = selectedClass.Name };  
 }  
  
 UsageKey key = new(filePath, attribute);  
 if (!propertyUsage.TryAdd(key, 1))  
 {  
 propertyUsage[key]++;  
 }  
 }  
 }  
 }  
  
 // Add nested types to the queue for analysis  
 var nestedTypes = GetNestedTypes(currentType).ToList();  
 foreach (var nestedType in nestedTypes)  
 {  
 if (nestedType.Assembly == selectedClass.Assembly)  
 {  
 typesToAnalyze.Enqueue(nestedType);  
 }  
 }  
 }  
  
 return (classUsage, propertyUsage);  
 }  
  
 private static bool IsPrimitiveOrArrayOfPrimitives(Type type)  
 {  
 if (type.IsPrimitive || type == typeof(string) || type == typeof(decimal) || type == typeof(DateTime))  
 {  
 return true;  
 }  
  
 if (type.IsArray)  
 {  
 var elementType = type.GetElementType();  
 return elementType != null && (elementType.IsPrimitive || elementType == typeof(string));  
 }  
  
 if (type.IsGenericType && type.GetGenericTypeDefinition() == typeof(IEnumerable<>))  
 {  
 var elementType = type.GetGenericArguments()[0];  
 return elementType.IsPrimitive || elementType == typeof(string);  
 }  
  
 return false;  
 }  
  
 private static List<(PropertyInfo Property, string FullPath)> GetDeepProperties(Type type, string prefix = "")  
 {  
 var properties = new List<(PropertyInfo Property, string FullPath)>();  
  
 foreach (var prop in type.GetProperties())  
 {  
 var fullPath = string.IsNullOrEmpty(prefix) ? prop.Name : $"{prefix}.{prop.Name}";  
 properties.Add((prop, fullPath));  
  
 if (!IsPrimitiveOrArrayOfPrimitives(prop.PropertyType))  
 {  
 properties.AddRange(GetDeepProperties(prop.PropertyType, fullPath));  
 }  
 }  
  
 return properties;  
 }  
  
 private HashSet<Type> GetNestedTypes(Type type)  
 {  
 var types = new HashSet<Type>();  
  
 foreach (var prop in type.GetProperties())  
 {  
 var propType = prop.PropertyType;  
  
 if (propType.IsArray)  
 {  
 propType = propType.GetElementType();  
 }  
 else if (propType.IsGenericType && propType.GetGenericTypeDefinition() == typeof(List<>))  
 {  
 propType = propType.GetGenericArguments()[0];  
 }  
  
 if (propType == null)  
 {  
 continue;  
 }  
  
 if (propType.IsClass &&  
 propType != typeof(string) &&  
 !propType.IsPrimitive &&  
 !propType.IsEnum &&  
 (propType.Namespace == null || !propType.Namespace.StartsWith("System")) &&  
 !\_analyzedTypes.Contains(propType))  
 {  
 types.Add(propType);  
 }  
 }  
  
 return types;  
 }  
  
 private static void LoadProjectIntoWorkspace(AdhocWorkspace workspace, string projectPath)  
 {  
 if (!File.Exists(projectPath))  
 {  
 Console.WriteLine($"Warning: Project file not found: {projectPath}");  
 return;  
 }  
  
 var projectName = Path.GetFileNameWithoutExtension(projectPath);  
 Console.WriteLine($"Loading project: {projectName}");  
  
 var projectInfo = ProjectInfo.Create(  
 ProjectId.CreateNewId(),  
 VersionStamp.Create(),  
 projectName,  
 projectName,  
 LanguageNames.CSharp  
 );  
  
 var project = workspace.AddProject(projectInfo);  
  
 var documents = Directory.GetFiles(Path.GetDirectoryName(projectPath)!, "\*.cs", SearchOption.AllDirectories);  
 foreach (var docPath in documents)  
 {  
 var sourceText = SourceText.From(File.ReadAllText(docPath));  
 var documentInfo = DocumentInfo.Create(  
 DocumentId.CreateNewId(project.Id),  
 Path.GetFileName(docPath),  
 loader: TextLoader.From(TextAndVersion.Create(sourceText, VersionStamp.Create())),  
 filePath: docPath  
 );  
  
 workspace.AddDocument(documentInfo);  
 }  
 }  
  
 private static List<string> GetProjectPathsFromSolution(string solutionPath)  
 {  
 var projectPaths = new List<string>();  
 var solutionDir = Path.GetDirectoryName(solutionPath)!;  
  
 foreach (var line in File.ReadAllLines(solutionPath))  
 {  
 if (line.Trim().StartsWith("Project(") && line.Contains(".csproj"))  
 {  
 var parts = line.Split(',');  
 if (parts.Length > 1)  
 {  
 var relativePath = parts[1].Trim().Trim('"');  
 var fullPath = Path.Combine(solutionDir, relativePath);  
 projectPaths.Add(fullPath);  
 }  
 }  
 }  
  
 return projectPaths;  
 }  
}

### ConsoleUi.cs

using Microsoft.Extensions.Logging;  
using Spectre.Console;  
  
namespace Analyze;  
  
public class ConsoleUi  
{  
 public enum PropertyUsageFormat  
 {  
 TotalUsages,  
 UsagesPerFile  
 }  
  
 private readonly ILogger \_logger;  
  
 public ConsoleUi(ILogger logger)  
 {  
 \_logger = logger;  
 }  
  
 public void DisplayWelcome()  
 {  
 AnsiConsole.MarkupLine("[bold blue]Welcome to the DTO Usage Analyzer![/]");  
 AnsiConsole.MarkupLine("This tool will help you analyze the usage of DTO classes in your solution.");  
 }  
  
 public Type? PromptForClassSelection(IEnumerable<Type> dtoClasses)  
 {  
 var classes = dtoClasses.ToList();  
 if (!classes.Any())  
 {  
 AnsiConsole.MarkupLine("[red]No DTO classes found in the Dto project.[/]");  
 return null;  
 }  
  
 var selectedClass = AnsiConsole.Prompt(  
 new SelectionPrompt<Type>()  
 .Title("Select a DTO class to analyze:")  
 .AddChoices(classes)  
 .UseConverter(t => t.Name));  
  
 return selectedClass;  
 }  
  
 public void DisplayAnalysisStart(Type selectedClass)  
 {  
 AnsiConsole.MarkupLine($"[green]Analyzing usage of {selectedClass.Name}...[/]");  
 }  
  
 public void DisplayAnalysisProgress(Action<StatusContext> updateStatus)  
 {  
 AnsiConsole.Status()  
 .Start("Analyzing...", ctx => updateStatus(ctx));  
 }  
  
 public PropertyUsageFormat PromptForPropertyUsageFormat()  
 {  
 var choice = AnsiConsole.Prompt(  
 new SelectionPrompt<string>()  
 .Title("How would you like to display property usage?")  
 .AddChoices("Show usages per file", "Show only total usages"));  
 return choice == "Show usages per file" ? PropertyUsageFormat.UsagesPerFile : PropertyUsageFormat.TotalUsages;  
 }  
  
 public void DisplayResults(  
 Dictionary<string, int> classUsage,  
 Dictionary<UsageKey, int> propertyUsage,  
 Type selectedClass,  
 PropertyUsageFormat propertyUsageFormat)  
 {  
 // Class Usage Table  
 var classTable = new Table()  
 .Border(TableBorder.Rounded)  
 .AddColumn(new TableColumn("[bold]File[/]").LeftAligned())  
 .AddColumn(new TableColumn("[bold]Usages[/]").RightAligned());  
  
 AnsiConsole.MarkupLine("\n[bold blue]Class Usage Analysis[/]");  
 if (classUsage.Any())  
 {  
 foreach (var usage in classUsage.OrderByDescending(u => u.Value))  
 {  
 classTable.AddRow(  
 $"[green]{usage.Key}[/]",  
 $"[yellow]{usage.Value}[/]"  
 );  
 }  
 }  
 else  
 {  
 classTable.AddRow("[red]No direct class usage found[/]", "0");  
 }  
  
 AnsiConsole.Write(classTable);  
  
 // Property Usage Table  
 AnsiConsole.MarkupLine("\n[bold blue]Property Usage Analysis[/]");  
 if (propertyUsageFormat == PropertyUsageFormat.TotalUsages)  
 {  
 var propertyTable = new Table()  
 .Border(TableBorder.Rounded)  
 .AddColumn(new TableColumn("[bold]Property[/]").LeftAligned())  
 .AddColumn(new TableColumn("[bold]Total Usages[/]").RightAligned());  
  
 var propertyUsageData = propertyUsage  
 .Select(u => new { PropertyPath = u.Key.Attribute, Count = u.Value })  
 .GroupBy(x => x.PropertyPath)  
 .OrderBy(x => x.Key.ClassName)  
 .ThenBy(x => x.Key.FieldName)  
 .Select(x => new  
 {  
 PropertyPath = x.Key,  
 Count = x.Sum(y => y.Count)  
 });  
  
 foreach (var usage in propertyUsageData)  
 {  
 var (className, fieldName) = usage.PropertyPath;  
 var totalUsages = usage.Count;  
 propertyTable.AddRow(  
 $"[green]{className}.{fieldName}[/]",  
 $"[yellow]{totalUsages}[/]"  
 );  
 }  
  
 AnsiConsole.Write(propertyTable);  
 }  
 else // UsagesPerFile  
 {  
 var propertyTable = new Table()  
 .Border(TableBorder.Rounded)  
 .AddColumn(new TableColumn("[bold]Property[/]").LeftAligned())  
 .AddColumn(new TableColumn("[bold]File[/]").LeftAligned())  
 .AddColumn(new TableColumn("[bold]Usages[/]").RightAligned());  
  
 var propertyUsageData = propertyUsage  
 .Select(u => new { File = u.Key.FilePath, PropertyPath = u.Key.Attribute, Count = u.Value })  
 .OrderBy(g => g.PropertyPath.ClassName)  
 .ThenBy(g => g.PropertyPath.FieldName)  
 .ThenBy(g => g.File);  
  
 foreach (var usage in propertyUsageData)  
 {  
 var (className, fieldName) = usage.PropertyPath;  
 propertyTable.AddRow(  
 $"[green]{className}.{fieldName}[/]",  
 $"[blue]{usage.File}[/]",  
 $"[yellow]{usage.Count}[/]"  
 );  
 }  
  
 AnsiConsole.Write(propertyTable);  
 }  
 }  
  
 public void DisplayError(Exception ex)  
 {  
 AnsiConsole.MarkupLine($"[red]An error occurred: {ex.Message}[/]");  
 \_logger.LogError(ex, "An error occurred during analysis");  
 }  
}

### ClassAndField.cs

namespace Analyze;  
  
public record ClassAndField(string ClassName, string FieldName) : IComparable<ClassAndField>  
{  
 public int CompareTo(ClassAndField? other)  
 {  
 if (ReferenceEquals(this, other))  
 {  
 return 0;  
 }  
  
 if (other is null)  
 {  
 return 1;  
 }  
  
 var classNameComparison = string.Compare(ClassName, other.ClassName, StringComparison.Ordinal);  
 if (classNameComparison != 0)  
 {  
 return classNameComparison;  
 }  
  
 return string.Compare(FieldName, other.FieldName, StringComparison.Ordinal);  
 }  
}

## Processors Project

### IProcessor.cs

namespace Processors;  
  
public interface IProcessor<TDto>  
{  
 void Process(string jsonInput, TextWriter output);  
}

### BaseProcessor.cs

using System.Text.Json;  
  
namespace Processors;  
  
public abstract class BaseProcessor<TDto> : IProcessor<TDto>  
{  
 protected static readonly JsonSerializerOptions JsonOptions = new()  
 {  
 PropertyNameCaseInsensitive = true  
 };  
  
 public abstract void Process(string jsonInput, TextWriter output);  
  
 protected TDto? Deserialize(string jsonInput)  
 {  
 try  
 {  
 return JsonSerializer.Deserialize<TDto>(jsonInput, JsonOptions);  
 }  
 catch (JsonException ex)  
 {  
 throw new JsonException($"Failed to deserialize JSON input: {ex.Message}", ex);  
 }  
 }  
  
 protected void WriteNoDataMessage(TextWriter output, string dataType)  
 {  
 output.WriteLine($"No {dataType} found.");  
 }  
}

## Project Files

### Analyze.csproj

<Project Sdk="Microsoft.NET.Sdk">  
 <PropertyGroup>  
 <OutputType>Exe</OutputType>  
 <TargetFramework>net10.0</TargetFramework>  
 <ImplicitUsings>enable</ImplicitUsings>  
 <Nullable>enable</Nullable>  
 </PropertyGroup>  
  
 <ItemGroup>  
 <PackageReference Include="Microsoft.Build.Tasks.Core"/>  
 <PackageReference Include="Microsoft.CodeAnalysis.CSharp"/>  
 <PackageReference Include="Microsoft.CodeAnalysis.CSharp.Workspaces"/>  
 <PackageReference Include="Microsoft.CodeAnalysis.Workspaces.MSBuild"/>  
 <PackageReference Include="Microsoft.Extensions.Configuration"/>  
 <PackageReference Include="Microsoft.Extensions.Configuration.FileExtensions"/>  
 <PackageReference Include="Microsoft.Extensions.Configuration.Json"/>  
 <PackageReference Include="Microsoft.Extensions.DependencyInjection"/>  
 <PackageReference Include="Microsoft.Extensions.Logging"/>  
 <PackageReference Include="Microsoft.Extensions.Logging.Console"/>  
 <PackageReference Include="Spectre.Console"/>  
 </ItemGroup>  
  
 <ItemGroup>  
 <None Update="appsettings.json">  
 <CopyToOutputDirectory>PreserveNewest</CopyToOutputDirectory>  
 </None>  
 </ItemGroup>  
</Project>

### Processors.csproj

<Project Sdk="Microsoft.NET.Sdk">  
 <PropertyGroup>  
 <TargetFramework>net10.0</TargetFramework>  
 <ImplicitUsings>enable</ImplicitUsings>  
 <Nullable>enable</Nullable>  
 </PropertyGroup>  
 <ItemGroup>  
 <Reference Include="Dto">  
 <HintPath>..\Dto\bin\Debug\net10.0\Dto.dll</HintPath>  
 </Reference>  
 </ItemGroup>  
</Project>

### Directory.Build.props

<Project>  
 <PropertyGroup>  
 <TargetFramework>net10.0</TargetFramework>  
 <ImplicitUsings>enable</ImplicitUsings>  
 <Nullable>enable</Nullable>  
 <LangVersion>latest</LangVersion>  
 </PropertyGroup>  
</Project>