DTO, ViewModel, AutoMapper, Unit of Work - Documentation

# Data Transfer Object (DTO)

What is a DTO?

A Data Transfer Object (DTO) is a plain object used to encapsulate and transport data between layers of an application.

Key Characteristics

* - No behavior: Contains only fields and properties.
* - Serializable: Suitable for network communication (e.g., APIs).
* - Flattened structure: Simplifies complex domain models for clients.

Purpose

* - Reduce over-fetching and under-fetching of data.
* - Improve performance by sending only required data.
* - Hide internal domain logic from external systems.

Example in C#

public class StudentDTO  
{  
 public int Id { get; set; }  
 public string Name { get; set; }  
 public string Email { get; set; }  
}

Real-World Use Case

In an e-commerce application, a ProductDTO might be used to send only the product's name, price, and availability to the front-end, hiding details like internal stock control.

# ViewModel

What is a ViewModel?

A ViewModel is a model specifically tailored for the view. It represents the data and logic needed for rendering a UI.

Purpose

* - Bind UI elements to data.
* - Encapsulate UI logic.
* - Facilitate testable UI code (MVVM pattern).

Difference from DTO

* DTO: Data transport, no logic, used in API/service layers.  
  ViewModel: View representation, may contain UI logic, used in UI binding.

Example in ASP.NET MVC

public class StudentViewModel  
{  
 public string FullName { get; set; }  
 public string Email { get; set; }  
 public bool IsEnrolled { get; set; }  
}

Real-World Use Case

A StudentViewModel may format FirstName and LastName into FullName, or convert enrollment status into a checkbox UI binding.

# AutoMapper

What is AutoMapper?

AutoMapper is a convention-based object-object mapper. It automates the mapping of properties from one object to another.

Benefits

* - Reduces boilerplate mapping code.
* - Easy to configure and use.

Basic Configuration

var config = new MapperConfiguration(cfg =>  
{  
 cfg.CreateMap<Student, StudentDTO>();  
});  
IMapper mapper = config.CreateMapper();  
var studentDto = mapper.Map<StudentDTO>(student);

When to Use AutoMapper

* - When mapping logic is simple.
* - For mapping between DTOs, ViewModels, and Entities.

When Not to Use AutoMapper

* - When mappings are complex and require custom logic.

Real-World Use Case

In a payroll system, AutoMapper can quickly convert EmployeeEntity to EmployeePayrollDTO by mapping only necessary fields like Salary, TaxRate, and Benefits.

# Unit of Work (UoW)

What is Unit of Work?

Unit of Work is a design pattern that maintains a list of business transactions and coordinates the writing out of changes.

Purpose

* - Manage transactions.
* - Ensure atomic operations.
* - Reduce code duplication for save operations.

Benefits

* - All repository operations under a single transaction.
* - Ensures consistency and rollback on error.

Implementation Structure

public interface IUnitOfWork : IDisposable  
{  
 IStudentRepository Students { get; }  
 ICourseRepository Courses { get; }  
 int Complete();  
}  
  
public class UnitOfWork : IUnitOfWork  
{  
 private readonly ApplicationDbContext \_context;  
 public IStudentRepository Students { get; private set; }  
 public ICourseRepository Courses { get; private set; }  
  
 public UnitOfWork(ApplicationDbContext context)  
 {  
 \_context = context;  
 Students = new StudentRepository(\_context);  
 Courses = new CourseRepository(\_context);  
 }  
  
 public int Complete()  
 {  
 return \_context.SaveChanges();  
 }  
  
 public void Dispose()  
 {  
 \_context.Dispose();  
 }  
}

Real-World Use Case

In a university management system, the Unit of Work pattern allows you to update both student enrollment and course assignment in one transaction—ensuring data consistency if an error occurs.