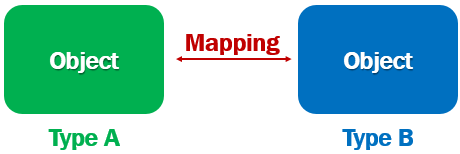
What is AutoMapper and using it in ASP.NET Core

AutoMapper is an object-object mapper i.e it maps an object of one type to another type.



For example, consider the following 2 classes

public class Employee

{

public int EmployeeId { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public string Email { get; set; }

public DateTime DateOfBrith { get; set; }

public Gender Gender { get; set; }

public int DepartmentId { get; set; }

public string PhotoPath { get; set; }

public Department Department { get; set; }

}

public class EditEmployeeModel

{

public int EmployeeId { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public string Email { get; set; }

public string ConfirmEmail { get; set; }

public DateTime DateOfBrith { get; set; }

public Gender Gender { get; set; }

public int? DepartmentId { get; set; }

public string PhotoPath { get; set; }

public Department Department { get; set; } = new Department();

}

Copy

We want to map properties of the Employee class to EditEmployeeModel class. Without a tool like AutoMapper, we have to write the following boring mapping code.

EditEmployeeModel.EmployeeId = Employee.EmployeeId;

EditEmployeeModel.FirstName = Employee.FirstName;

EditEmployeeModel.LastName = Employee.LastName;

EditEmployeeModel.Email = Employee.Email;

EditEmployeeModel.ConfirmEmail = Employee.Email;

EditEmployeeModel.DateOfBrith = Employee.DateOfBrith;

EditEmployeeModel.Gender = Employee.Gender;

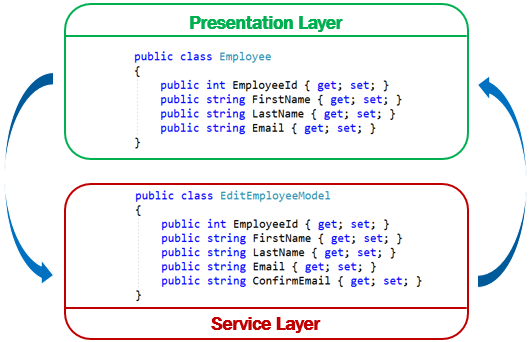
EditEmployeeModel.PhotoPath = Employee.PhotoPath;

EditEmployeeModel.DepartmentId = Employee.DepartmentId;

EditEmployeeModel.Department = Employee.Department;

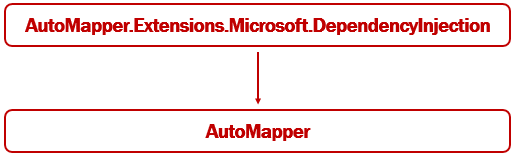
Copy

If you are wondering why is there a need to map objects. Well, a mapping like this is required when we pass data between different layers of an application. For example, when we pass data between data access, business and presentation layers of our application.



Using AutoMapper in ASP.NET Core

To use AutoMapper in an ASP.NET Core project, install AutoMapper.Extensions.Microsoft.DependencyInjection nuget package. This package has a dependency on AutoMapper, so it will also be installed.



Add AutoMapper Services

* + In ConfigureServices method of the Startup class, add AutoMapper services.
* AddAutoMapper method is in AutoMapper namespace.
* The class EmployeeProfile contains the mapping profiles for AutoMapper

public class Startup

{

public void ConfigureServices(IServiceCollection services)

{

services.AddAutoMapper(typeof(EmployeeProfile));

}

}

Copy

AutoMapper Mapping Profile

using AutoMapper;

using EmployeeManagement.Models;

namespace EmployeeManagement.Web.Models

{

public class EmployeeProfile : Profile

{

public EmployeeProfile()

{

CreateMap<Employee, EditEmployeeModel>()

.ForMember(dest => dest.ConfirmEmail,

opt => opt.MapFrom(src => src.Email));

CreateMap<EditEmployeeModel, Employee>();

}

}

}

Copy

* When doing the mapping from one type to another, AutoMapper looks for mapping profiles.
* To create a mapping profile, create a class that derives from AutoMapper Profile class.
* Use CreateMap method to create a mapping from one type to another.
* CreateMap method is called twice to create 2 mappings. Employee to EditEmployeeModel and the reverse.
* Except for one property (ConfirmEmail), we do not need explicit property mappings, because the property names and their data types match. So the AutoMapper does the mapping for us automatically.

Explicit AutoMapper Property Mapping

CreateMap<Employee, EditEmployeeModel>()

.ForMember(dest => dest.ConfirmEmail,

opt => opt.MapFrom(src => src.Email));

Copy

* For ConfirmEmail property in EditEmployeeModel class, we do not have a matching property in Employee class.
* We have an explicit mapping to map the Email property in Employee class to ConfirmEmail property in EditEmployeeModel class.

Using AutoMapper IMapper Service

* + Just like any other service, inject AutoMapper IMapper service in to the blazor component class using thr [Inject] attribute.
  + If you have an ASP.NET Core MVC controller, use the standard constructor injection.
* Use the IMapper service Map method to do the object mapping.
* The first parameter is the source and the second parameter is the destination.

public class EditEmployeeBase : ComponentBase

{

[Inject]

public IMapper Mapper { get; set; }

protected async override Task OnInitializedAsync()

{

Mapper.Map(Employee, EditEmployeeModel);

}

}