Automapper & Mapster & ManualMapper

automapper :

- Convention Model: Follows the convention-over-configuration principle, automatically mapping properties with the same name.
- Fluent API: Uses a fluent API for configuration, allowing customization of mappings as needed.
- Feature-Rich: Comprehensive feature set supporting flattening, conditional mapping, nested mappings, and more.
- Ease of Use: Known for its ease of use, widely adopted, and features a large user base.
- Convention for Naming: Typically follows a naming convention where destination property names match source property names by default.

mapster :

- Fluent API: Similar to AutoMapper, uses a fluent configuration API for mapping definitions.
- Performance: Emphasizes performance, claiming to be faster than AutoMapper in certain scenarios.
- Code Generation: Includes a code generation tool that can generate mapping code during compile time for enhanced performance.
- More Explicit: Considered more explicit, requiring more explicit mapping definitions.
- Immutable Objects: Good support for mapping to and from immutable objects.

manualmapper :

- Full Control: With manual mapping, you have full control over the mapping process. This can be advantageous in cases where you need to implement custom logic or transformations.
- Performance: In some cases, manual mapping might be more performant than using a mapping library, especially for simple mappings with a known and fixed structure.
- Specific Business Logic: Manual mapping allows you to incorporate specific business logic directly into the mapping process, which might be harder to achieve with a generic mapping library.

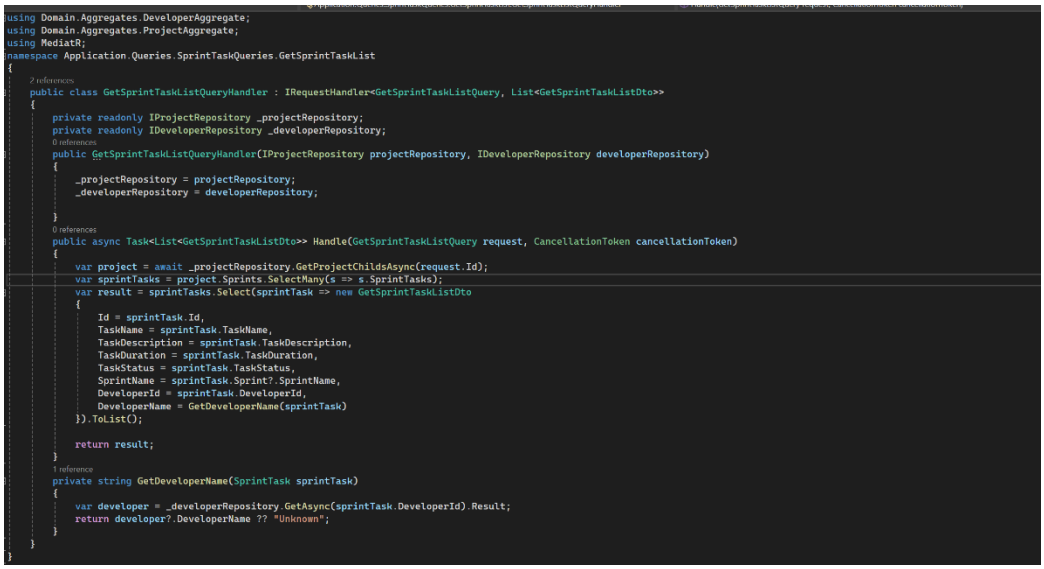
Example to resolve developer name using automapper :

```
using Application.Queries.SprintTaskQueries.GetSprintTaskList;
using AutoMapper;
using Domain.Aggregates.DeveloperAggregate;
using Domain.Aggregates.ProjectAggregate;

3 references
public class DeveloperNameResolver : IValueResolver<SprintTask, GetSprintTaskListDto, string>
{
    private readonly IDeveloperRepository _developerRepository;

    0 references
    public DeveloperNameResolver(IDeveloperRepository developerRepository)
    {
        _developerRepository = developerRepository;
    }

    0 references
    public string Resolve(SprintTask source, GetSprintTaskListDto destination, string destMember, ResolutionContext context)
    {
        var developer = _developerRepository.GetAsync(source.DeveloperId).Result;

        return developer?.DeveloperName ?? "Unknown";
    }

}
```

Example to resolve developer name using mapster:

```
using Application.Queries.SprintTaskQueries.GetSprintTaskDetails;
using Application.Queries.SprintTaskQueries.GetSprintTaskList;
using Domain.Aggregates.DeveloperAggregate;
using Domain.Aggregates.ProjectAggregate;
using Mapster;

namespace Application.MappingProfiles
{
    3 references
    public class SprintTaskProfile : IRegister
    {
        private readonly IDeveloperRepository _developerRepository;

        0 references
        public SprintTaskProfile(IDeveloperRepository developerRepository)
        {
            _developerRepository = developerRepository;
        }

        0 references
        public void Register(TypeAdapterConfig config)
        {
            config.NewConfig<SprintTask, GetSprintTaskListDto>()
                .Map(dest => dest.SprintName, src => src.Sprint.SprintName)
                .Map(dest => dest.DeveloperName, src => GetDeveloperName(src));

            config.NewConfig<SprintTask, GetSprintTaskDetailsDto>()
                .Map(dest => dest.SprintName, src => src.Sprint.SprintName)
                .Map(dest => dest.DeveloperName, src => GetDeveloperName(src));
        }

        2 references
        private string GetDeveloperName(SprintTask source)
        {
            var developer = _developerRepository.GetAsync(source.DeveloperId).Result;

            return developer?.DeveloperName ?? "Unknown";
        }
    }
}
```

Example to resolve developer name using manualmapper:



```
using Domain.Aggregates.DeveloperAggregate;
using Domain.Aggregates.ProjectAggregate;
using MediatR;
namespace Application.Queries.SprintTaskQueries.GetSprintTaskList
{
    2 references
    public class GetSprintTaskListQueryHandler : IRequestHandler<GetSprintTaskListQuery, List<GetSprintTaskListDto>>
    {
        private readonly IProjectRepository _projectRepository;
        private readonly IDeveloperRepository _developerRepository;
        0 references
        public GetSprintTaskListQueryHandler(IProjectRepository projectRepository, IDeveloperRepository developerRepository)
        {
            _projectRepository = projectRepository;
            _developerRepository = developerRepository;

        }
        0 references
        public async Task<List<GetSprintTaskListDto>> Handle(GetSprintTaskListQuery request, CancellationToken cancellationToken)
        {
            var project = await _projectRepository.GetProjectChildsAsync(request.Id);
            var sprintTasks = project.Sprints.SelectMany(s => s.SprintTasks);
            var result = sprintTasks.Select(sprintTask => new GetSprintTaskListDto
            {
                Id = sprintTask.Id,
                TaskName = sprintTask.TaskName,
                TaskDescription = sprintTask.TaskDescription,
                TaskDuration = sprintTask.TaskDuration,
                TaskStatus = sprintTask.TaskStatus,
                SprintName = sprintTask.Sprint?.SprintName,
                DeveloperId = sprintTask.DeveloperId,
                DeveloperName = GetDeveloperName(sprintTask)
            }).ToList();

            return result;
        }
        1 reference
        private string GetDeveloperName(SprintTask sprintTask)
        {
            var developer = _developerRepository.GetAsync(sprintTask.DeveloperId).Result;
            return developer?.DeveloperName ?? "Unknown";
        }
    }
}
```

References:

- MapsterMapper/Mapster: A fast, fun and stimulating object to object Mapper (github.com)

- https://automapper.org/

- Understanding When to Use Mappers vs. Manual Mapping in C# | by J | Nov, 2023 | Medium