1) builder.Services.AddMemoryCache();

builder.Services.AddTransient(

typeof(IPipelineBehavior<,>), typeof(CacheBehavior<,>));

2) public interface ICacheable

{

string CacheKey { get; }

}

3) Modify the query by inheriting above interface

public record GetMProductbyIDQuery(int ID) : IRequest<MProduct>,ICacheable

{

public string CacheKey => $"GetProductByID\_{ID}";

}

4) public class CacheBehavior<TRequest, TResponse> : IPipelineBehavior<TRequest, TResponse> where TRequest : ICacheable

{

private readonly IMemoryCache \_cache;

private readonly ILogger<CacheBehavior<TRequest, TResponse>> \_logger;

public CacheBehavior(ILogger<CacheBehavior<TRequest, TResponse>>

logger, IMemoryCache cache)

{

\_logger = logger;

\_cache = cache;

}

public async Task<TResponse> Handle(TRequest request,

RequestHandlerDelegate <TResponse> next, CancellationToken

cancellationToken)

{

\_logger.LogInformation("{Request} is configured for

caching", request.GetType());

TResponse cacheResponse;

if (\_cache.TryGetValue(request.CacheKey, out cacheResponse!))

{

\_logger.LogInformation("Returning cached value

for {Request}", request.GetType());

return cacheResponse;

}

\_logger.LogInformation("{Request} Cache key: {key} is not inside the

cache,executing request", request.GetType(),request.CacheKey);

cacheResponse = await next();

\_cache.Set(request.CacheKey, cacheResponse);

return cacheResponse;

}

}