Behaviors | PipelineBehaviors

ValidationBehaviors: Debugging flow for validation

builder.Services.AddProblemDetails();

// Register ProblemDetails for global error handling

builder.Services.AddExceptionHandler<GlobalExceptionHandler>();

builder.Services.AddValidatorsFromAssembly(typeof(Program).Assembly);

builder.Services.AddTransient(typeof(IPipelineBehavior<,>), typeof(ValidationBehavior<,>));

app.UseExceptionHandler();

1)var result= await \_sender.Send(new AddMProductCommand(product));

2)public class AddCommandValidator : AbstractValidator<AddMProductCommand>

{

public AddCommandValidator()

{

RuleFor(x => x.mProduct.Name)

.NotEmpty()

.WithMessage("The name of the product can't be empty");

RuleFor(x => x.mProduct.Name)

.MaximumLength(60)

.WithMessage("The length of the name can't be more than 60 characters long");

}

}

3)public class ValidationBehavior<Trequest, TResponse> : IPipelineBehavior

<Trequest, TResponse> where Trequest : IRequest<TResponse>

{

private readonly IEnumerable<IValidator<Trequest>> \_validators;

public ValidationBehavior(IEnumerable<IValidator<Trequest>> validators)

{

\_validators = validators;

}

public Task<TResponse> Handle(Trequest request, RequestHandlerDelegate<TResponse> next, CancellationToken cancellationToken)

{

var context = new ValidationContext<Trequest>(request);

var errorDictionary = \_validators

.Select(x => x.Validate(context))

.SelectMany(x => x.Errors)

.Where(x => x != null)

.GroupBy(

x => x.PropertyName.Substring(x.PropertyName.IndexOf(',') + 1),

x => x.ErrorMessage, (propertyName, errorMessages) => new

{

Key = propertyName,

Values = errorMessages.Distinct().ToArray()

}).ToDictionary(x => x.Key, x => x.Values);

if (errorDictionary.Any())

{

throw new MValidationAppException(errorDictionary);

}

return next();

}

}

4) public class MValidationAppException : Exception

{

public IReadOnlyDictionary<string, string[]> errors { get; }

public MValidationAppException(IReadOnlyDictionary<string, string[]> errors) : base("Validation error(s) occurred.")

{

this.errors = errors;

}

}

5) public class GlobalExceptionHandler : IExceptionHandler

{

private readonly IProblemDetailsService \_problemDetailsService;

public GlobalExceptionHandler(IProblemDetailsService problemDetailsService)

{

\_problemDetailsService = problemDetailsService;

}

public async ValueTask<bool> TryHandleAsync(HttpContext httpContext, Exception exception, CancellationToken cancellationToken)

{

httpContext.Response.ContentType = "application/json";

var excDetails = exception switch

{

MValidationAppException => (Detail: exception.Message, StatusCode: StatusCodes.Status422UnprocessableEntity),

\_ => (Detail: exception.Message, StatusCode: StatusCodes.Status500InternalServerError)

};

httpContext.Response.StatusCode = StatusCodes.Status400BadRequest;

if (exception is MValidationAppException validationException)

{

await httpContext.Response.WriteAsJsonAsync(new { validationException.errors });

return true;

}

return await \_problemDetailsService.TryWriteAsync(new ProblemDetailsContext

{

HttpContext = httpContext,

ProblemDetails =

{

Title = "An error occurred while processing your request.",

Detail = excDetails.Detail,

Type=exception.GetType().Name,

Status = excDetails.StatusCode

},

Exception = exception

});

}

}

If no error found then below command will be executed

public record AddMProductCommand(MProduct mProduct) : IRequest<MProduct>;