Table of Contents

[Current Resources with Changes 2](#_Toc113404504)

[New Resources Added 2](#_Toc113404505)

[Dtos 2](#_Toc113404506)

[UserRegisterDto 2](#_Toc113404507)

[/Core/Dto/ApiExceptionDto.cs [New] 2](#_Toc113404508)

[Constants 2](#_Toc113404509)

[/Core/Constants/ContentTypeConstants.cs 2](#_Toc113404510)

[Controllers 3](#_Toc113404511)

[BuggyController 3](#_Toc113404512)

[Testing with PostMan 3](#_Toc113404513)

[Exception Handling 3](#_Toc113404514)

[The default 3](#_Toc113404515)

[Global Exception Handling – Middleware 4](#_Toc113404516)

[/Core/Middleware/ ExceptionMiddleware.cs 4](#_Toc113404517)

[Add Middleware to Program.cs 5](#_Toc113404518)

[Testing With PostMan 6](#_Toc113404519)

# Current Resources with Changes

1. /Core/Dto/UserRegisterDto
2. Program.cs

# New Resources Added

1. /Controllers/BuggyController
2. /Core/Dto/ApiExceptionDto.cs
3. /Core/Constants/ContentTypeConstants.cs
4. /Core/Middleware folder ExceptionMiddleware

# Dtos

## UserRegisterDto

Add a string length attribute to the password

    [Required(ErrorMessage = "Password is empty")]

    [StringLength(8, MinimumLength = 4)]

    public string Password { get; set; }

## /Core/Dto/ApiExceptionDto.cs [New]

Create a new DTO / model for the ApiException.

namespace MSC.Api.Core.Dto;

public class ApiExceptionDto

{

    public ApiExceptionDto(int statusCode, string message = null, string details = null)

    {

        StatusCode = statusCode;

        Message = message;

        Details = details;

    }

    public int StatusCode { get; set; }

    public string Message { get; set; }

    public string Details { get; set; }

}

# Constants

## /Core/Constants/ContentTypeConstants.cs

Create constants for the content type.

namespace MSC.Api.Core.Constants;

public class ContentTypeConstants

{

    public const string ApplicationJson = "application/json";

}

# Controllers

## BuggyController

A controller to help understands the error handling and auth. It has some common actions to see errors in action.

* AuthError
* NotFound
* ServerError
* BadRequest

Check controller for each action implementation.

### Testing with PostMan

|  |  |
| --- | --- |
| Auth Action Type: Get  URL: <https://localhost:5000/api/buggy/auth>  Result: | Not Found Action Type: Get  Url: <https://localhost:5000/api/buggy/not-found>  Result: |
| Server Error Action Type: Get  URL: <https://localhost:5000/api/buggy/server-error>  Result: | Bad Request Action Type: Get  URL: <https://localhost:5000/api/buggy/bad-request>  Result: |
| Validation Exception Type: POST  URL: <https://localhost:5000/api/account/register>  Body: {}  Result: check the result in the opposite column |  |

# Exception Handling

## The default

The commented line has no affect in .net core 6.0. You’ll see stack trace as in [Server Error Action](#_Server_Error_Action) above.

// Configure the HTTP request pipeline.

if (app.Environment.IsDevelopment())

{

    app.UseSwagger();

    app.UseSwaggerUI();

    //display -- this is the default

    //app.UseDeveloperExceptionPage();

}

## Global Exception Handling – Middleware

### /Core/Middleware/ ExceptionMiddleware.cs

using System;

using System.Net;

using System.Text.Json;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Http;

using Microsoft.Extensions.Hosting;

using Microsoft.Extensions.Logging;

using MSC.Api.Core.Constants;

using MSC.Api.Core.Dto;

namespace MSC.Api.Core.Middleware;

public class ExceptionMiddleware

{

    private readonly RequestDelegate \_next;

    private readonly ILogger<ExceptionMiddleware> \_logger;

    private readonly IHostEnvironment \_env;

    /// <summary>

    /// Receives RequestDelegate which is whats next in the middle ware pipeline

    /// </summary>

    /// <param name="next">What is next in the pipeline</param>

    /// <param name="logger">So to log the exception</param>

    /// <param name="env">The environment development/production</param>

    public ExceptionMiddleware(RequestDelegate next, ILogger<ExceptionMiddleware> logger, IHostEnvironment env)

    {

        \_next = next;

        \_logger = logger;

        \_env = env;

    }

    /// <summary>

    /// The required method to invoke the middleware

    /// </summary>

    /// <param name="context">The http context</param>

    /// <returns></returns>

    public async Task InvokeAsync(HttpContext context)

    {

        try

        {

            //pass the context to the next piece of middleware

            await \_next(context);

        }

        catch(Exception ex)

        {

            //log the error

            \_logger.LogError(ex, ex.Message);

            //set content type

            context.Response.ContentType = ContentTypeConstants.ApplicationJson;

            //set status code

            context.Response.StatusCode = (int)HttpStatusCode.InternalServerError;

            //create the response model

            ApiExceptionDto response = null;

            if(\_env.IsDevelopment())

            {

                //development put out the exact message and stack trace

                response = new ApiExceptionDto(context.Response.StatusCode, ex.Message, ex.StackTrace?.ToString());

            }

            else

            {

                //production do not put out the exact message and stack trace

                response = new ApiExceptionDto(context.Response.StatusCode, "Internal Server Error");

            }

            //want the json responses to go as camel case

            var jsonOptions = new JsonSerializerOptions { PropertyNamingPolicy = JsonNamingPolicy.CamelCase };

            //serialize the response

            var json = JsonSerializer.Serialize(response, jsonOptions);

            //write

            await context.Response.WriteAsync(json);

        }

    }

}

### Add Middleware to Program.cs

Add the custom section for the middle ware to the Program.cs

var app = builder.Build();

//CUSTOM: Middleware Start

app.UseMiddleware<ExceptionMiddleware>();

//CUSTOM: Middleware End

### Testing With PostMan

#### Server Error Action – dev mode

Type: Get

URL: <https://localhost:5000/api/buggy/server-error>

Result:

Graphical user interface, text, application, website

Description automatically generated