# Project

* Site-04-Ng-login-reg-nav-services-templateforms-Api-MiddlewareException
  + Copied from: Site-03-Api-Auth-JWT-Reg-Login-Extensions
* For the “MySocialConnect-API”
  + dotnet restore
  + dotnet build
  + Go to project : MSC.WebApi
    - dotnet build : to build
    - dotnet run : to run the api

# New Resources

|  |  |
| --- | --- |
| MSC.Core | MSC.WebApi |
| Constants/ContentTypeConstants.cs | AccountController.cs |
| Dtos/ExceptionResponsedto.cs |  |
| Middleware/ExceptionMiddleware.cs |  |

# Resources updated

|  |  |
| --- | --- |
| MSC.Core | MSC.WebApi |
|  | Programs.cs |

# MSC.Core

## Constants/ContentTypeConstants.cs

namespace MSC.Core.Constants;

public class ContentTypeConstants

{

    public const string ApplicationJson = "application/json";

}

## Dtos/ExceptionResponseDto.cs

namespace MSC.Core.Dtos;

public class ExceptionResponseDto

{

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

    {

    }

    public int StatusCode { get; set; }

    public string Message { get; set; }

    public string Details { get; set; }

}

## Middleware/ExceptionMiddleware.cs

using System;

using System.ComponentModel.DataAnnotations;

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.Core.Constants;

using MSC.Core.Dtos;

using MSC.Core.ExceptionCustom;

namespace MSC.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 (DataFailException dfe)

        {

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

            await WriteError(context, dfe.Message, dfe.StackTrace?.ToString(), HttpStatusCode.BadRequest);

        }

        catch (ValidationException vex)

        {

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

            await WriteError(context, vex.Message, vex.StackTrace?.ToString(), HttpStatusCode.BadRequest);

        }

        catch (UnauthorizedAccessException uex)

        {

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

            await WriteError(context, uex.Message, uex.StackTrace?.ToString(), HttpStatusCode.Unauthorized);

        }

        catch (Exception ex)

        {

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

            await WriteErrorException(context, ex, HttpStatusCode.InternalServerError);

        }

    }

    private async Task WriteError(HttpContext context, string exMessage, string exStackTrace, HttpStatusCode code)

    {

        //set content type

        context.Response.ContentType = ContentTypeConstants.ApplicationJson;

        //set status code

        context.Response.StatusCode = (int)code;

        //write

        await context.Response.WriteAsync(exMessage);

    }

    private async Task WriteErrorException(HttpContext context, Exception ex, HttpStatusCode code)

    {

        //set content type

        context.Response.ContentType = ContentTypeConstants.ApplicationJson;

        //set status code

        context.Response.StatusCode = (int)code;

        //create the response model

        ExceptionResponseDto response = null;

        if (\_env.IsDevelopment())

        {

            //development put out the exact message and stack trace

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

        }

        else

        {

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

            response = new ExceptionResponseDto(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);

    }

}

# MSC.WebApi

## Programs.cs

Add ExceptionMiddleware to programs.cs

var app = builder.Build();

/\*\*\*Custom Section Middleware Start\*\*\*/

app.UseMiddleware<ExceptionMiddleware>();

/\*\*\*Custom Section Middleware End\*\*\*/

// Configure the HTTP request pipeline.

if (app.Environment.IsDevelopment())

{

    app.UseSwagger();

    app.UseSwaggerUI();

}

## AccountController.cs

New end point to check the user name available or taken

    [HttpGet("checkUser/{userName}")] // /api/account/checkUser/name

    public async Task<ActionResult<bool>> CheckUserName(string userName)

    {

        var userExists = await \_userBusinessLogic.UserExists(userName);

        return Ok(userExists);

    }