GLOBAL EXCEPTION ERROR HANDLINE USING MIDDLEWARE IN ADO.NET

public class StartUp

{

private readonly IWebHostEnvironment \_env;

public IConfiguration Configuration { get; }

public StartUp(IConfiguration configuration, IWebHostEnvironment env )

{

Configuration = configuration;

\_env = env;

}

public void ConfigureServices(IServiceCollection services)

{

// read jsonfile config:

// ConfigJsonFiles

var path = Path.Combine(Environment.CurrentDirectory, @"JSONFiles\");

// // for RSAClient.json

IConfiguration config = new ConfigurationBuilder()

.SetBasePath(path)

.AddJsonFile("RSAClientData.json").Build();

var rSAClients = config.GetSection("RSAClientData").Get<List<RSAClient>>();

// // Register the list of items as a singleton

services.AddSingleton(rSAClients);

// // configure json data

// //services.Configure<RSAClientData>(config);

// //services.Configure<List<RSAClient>>(configRoot.GetSection("RSAClient"));

// // Users.json

// IConfiguration usersCongfig = new ConfigurationBuilder()

//.SetBasePath(path)

//.AddJsonFile("Users.json").Build();

// var users = usersCongfig.GetSection("UsersData").Get<List<User>>();

// services.AddSingleton(users);

// dependeny injections

services.AddScoped<IAuthRepo, AuthRepo>();

services.AddScoped<IAuthService, AuthService>();

services.AddScoped<ITokenService, TokenService>();

services.AddScoped<Helpers>();

services.AddScoped<CustomerDAL>();

services.AddScoped<CustomerService>();

services.AddScoped<ValidationHelper>();

services.AddScoped<IFileRepo, FileRepo>();

services.AddScoped<IFileService, FileService>();

// configure

services.AddOptions();

services.Configure<AppSettings>(Configuration.GetSection("AppSettings"));

services.Configure<ConnectionStrings>(Configuration.GetSection("ConnectionStrings"));

services.Configure<JWT>(Configuration.GetSection("JWT"));

var key = Encoding.ASCII.GetBytes(Configuration["JWT:Secret"]);

// services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)

//.AddJwtBearer(options =>

//{

// options.TokenValidationParameters = new TokenValidationParameters

// {

// ValidateIssuer = true,

// ValidateAudience = true,

// ValidateLifetime = true,

// ValidateIssuerSigningKey = true,

// ValidIssuer = Configuration["Jwt:Issuer"],

// ValidAudience = Configuration["Jwt:Issuer"],

// IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(Configuration["Jwt:Key"]))

// };

//});

services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)

.AddJwtBearer(x =>

{

x.RequireHttpsMetadata = false; // Set to true in production

x.SaveToken = true;

x.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuerSigningKey = true,

IssuerSigningKey = new SymmetricSecurityKey(key),

ValidateIssuer = true,

ValidateAudience = true,

ValidIssuer = Configuration["JWT:ValidIssuer"],

ValidAudience = Configuration["JWT:ValidAudience"]

};

});

services.AddMvc();

services.AddAuthorization(options =>

{

options.AddPolicy("RequireAdminRole", policy => policy.RequireRole("Admin"));

options.AddPolicy("RequireUserRole", policy => policy.RequireRole("User"));

});

services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new OpenApiInfo { Title = "APIApplication", Version = "v1" });

c.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme

{

Description = @"Please provide authorization token to access restricted features.",

Name = "Authorization",

In = ParameterLocation.Header,

Type = SecuritySchemeType.Http,

Scheme = "Bearer",

BearerFormat = "JWT",

});

c.AddSecurityRequirement(new OpenApiSecurityRequirement

{

{

new OpenApiSecurityScheme

{

Reference = new OpenApiReference

{

Type = ReferenceType.SecurityScheme,

Id = "Bearer"

}

},

new string[] {}

}

});

});

services.AddControllers();

services.AddHttpContextAccessor();

services.AddCors(options =>

{

options.AddDefaultPolicy(

policy =>

{

policy.AllowAnyOrigin()

.AllowAnyHeader()

.AllowAnyMethod();

});

});

}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

app.UseSwagger();

app.UseSwaggerUI(c => c.SwaggerEndpoint("/swagger/v1/swagger.json", "My API V1"));

//app.UseSwagger();

//app.UseSwaggerUI(c =>

//{

// c.SwaggerEndpoint("/swagger/v1/swagger.json", "APIApplication v1");

// c.RoutePrefix = string.Empty; // To serve the Swagger UI at the app's root URL

//});

}

// using middleware for refresh tokens:

//app.UseMiddleware<RefreshTokenMiddleware>();

// app.UseMiddleware<GlobalExceptionHandlerMiddleware>();

// Exception Handler:

//app.Use(async (context, next) =>

//{

// try

// {

// await next(context);

// }catch(Exception ex)

// {

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

// if(ex.InnerException != null)

// {

// await context.Response.WriteAsync(ex.Message.ToString());

// }

// }

//});

//app.UseExceptionHandler(options =>

//{

// options.Run(async (context) =>

// {

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

// var re = context.Features.Get<IExceptionHandlerFeature>();

// if (re != null)

// {

// await context.Response.WriteAsync(re.Error.Message.ToString());

// }

// });

//});

app.UseHttpsRedirection();

app.UseRouting();

app.UseCors();

app.UseAuthentication();

app.UseAuthorization();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllers();

});

}

}

Using middlewares:

using Microsoft.AspNetCore.Mvc;

using System.Data.SqlClient;

using System.Reflection;

using System.Text.Json;

namespace JWTRoleAuthentication.CommonLayer.Models

{

public class GlobalExceptionHandlerMiddleware

{

private readonly RequestDelegate \_next;

public GlobalExceptionHandlerMiddleware(RequestDelegate next)

{

\_next = next;

}

public async Task InvokeAsync(HttpContext context)

{

try

{

await \_next(context);

}

catch (Exception error)

{

var response = context.Response;

response.ContentType = "application/json";

int statusCode;

string message;

switch (error)

{

case BadRequestException badEx:

statusCode = StatusCodes.Status400BadRequest;

message = badEx.Message;

break;

case NotFoundException notFoundEx:

statusCode = StatusCodes.Status404NotFound;

message = notFoundEx.Message;

break;

case InternelException internelEx:

statusCode = StatusCodes.Status500InternalServerError;

message = internelEx.Message;

break;

default:

statusCode = StatusCodes.Status500InternalServerError;

message = "An internal server error occurred.";

break;

}

response.StatusCode = statusCode;

var result = JsonSerializer.Serialize(new {

StatusCode = response.StatusCode,

StatusMessage = message

});

await response.WriteAsync(result);

}

}

}

public class AppException : Exception

{

public AppException() : base()

{

}

public AppException(string message) : base(message)

{

}

public AppException(string message, params object[] args) : base(String.Format(message, args)) { }

}

public class NotFoundException : Exception

{

public NotFoundException(string message) : base(message)

{

}

}

public class BadRequestException : Exception

{

public BadRequestException(string message) : base(message)

{

}

}

public class InternelException : Exception

{

public InternelException(string message) : base(message)

{

}

}

}

Usage in controllers:

[HttpPost]

[Route("InsertData")]

public async Task<IActionResult> InsertData(SampleDataTypes model)

{

var result = await \_authService.InsertSampleData(model);

switch (result.StatusCode)

{

case 200:

return Ok(result);

case 400:

return BadRequest(result);

// throw new BadRequestException(result.StatusMessage);

case 500:

return StatusCode(result.StatusCode, new { StatusCode = result.StatusCode, StatusMessage = result.StatusMessage });

//throw new InternelException( result.StatusMessage );

default:

return StatusCode(result.StatusCode, result);

}

}