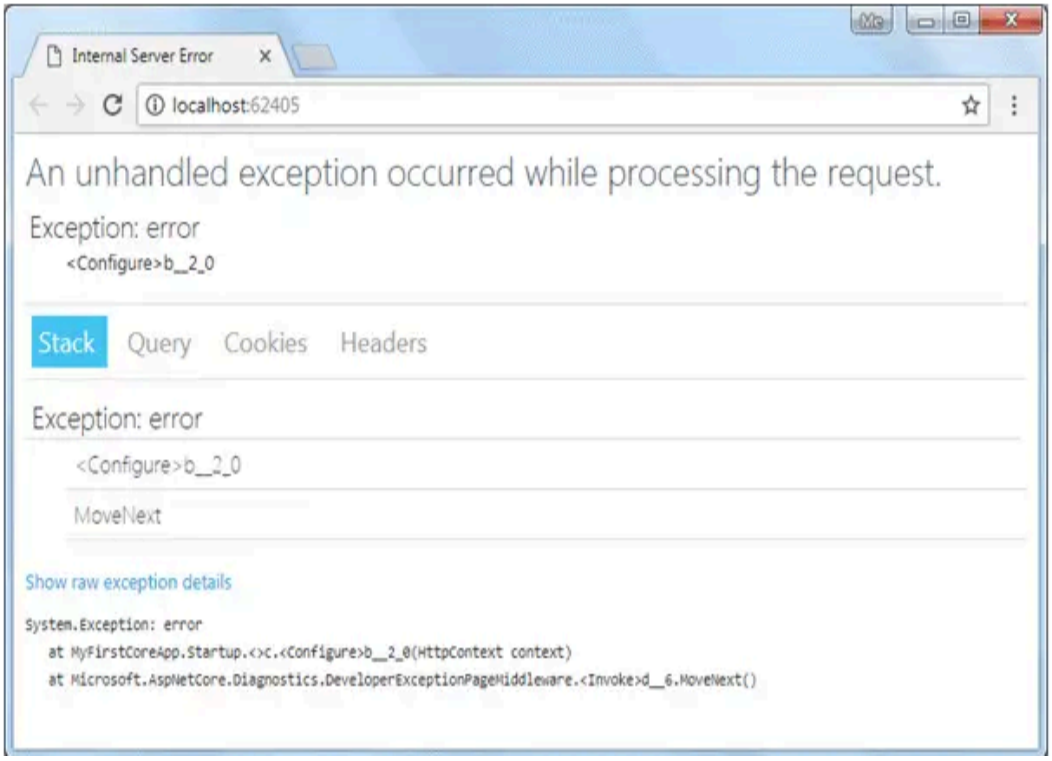
# Exception Handling

# 1. UseDeveloperExceptionPage

- The Developer Exception Page in ASP.NET Core provides detailed information about exceptions. Because developers need detailed information about exception.
- The ASP.NET Core templates enable the Developer Exception Page only when an application is running in the Development environment.

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment en
      if (env.IsDevelopment()) {
            app.UseDeveloperExceptionPage();
      }
      else {
            app.UseExceptionHandler("/Home/Error");
      }
      app.UseStaticFiles();
      app.UseRouting();
      app.UseAuthorization();
      app.UseEndpoints(endpoints = >{
            endpoints.MapControllerRoute(
            name: "default", pattern:
"{controller=Home}/{action=Index}/{id?}");
      });
}
```

- As the preceding code indicates, the Developer Exception Page will be enabled only when the application is running in the Development environment.
- The Developer Exception Page is not enabled when the application is running in the Production environment.
- Naturally, the purpose is to avoid displaying exception messages publicly.
- Note that the **UseDeveloperExceptionPage** extension method is called at the beginning of the pipeline, before any other middleware is called. This is done to ensure that exceptions in the following middleware will be caught.




- The developer exception page includes 4 tabs:
  - **Stack,**
  - **Query,**
  - **Cookies,**
  - **Headers,**
  - **Routing**
- Stack tab displays information of stack trace, which indicates where exactly an error occurred.
- Query tab displays information about query string.
- Cookies tab displays information about cookies set by the request and
- Headers tab displays information about headers.
- Routing refers to how ASP.NET Core maps incoming requests to route handlers.

## 2. UseExceptionHandler

- For handling exceptions in the Production environment, you should take advantage of the UseExceptionHandler extension method.

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment en
      if (env.IsDevelopment()) {
            app.UseDeveloperExceptionPage();
      }
      else {
            app.UseExceptionHandler("/Home/Error");
      }
      app.UseStaticFiles();
      app.UseRouting();
      app.UseAuthorization();
      app.UseEndpoints(endpoints = >{
            endpoints.MapControllerRoute(
            name: "default", pattern:
            "{controller=Home}/{action=Index}/{id?}");
      });
}
```

- The UseExceptionHandler extension method can be used to configure custom error handling routes.

```
//Using UseDeveloperExceptionPage Middleware to Show Exception Details
app.UseExceptionHandler(a => a.Run(async context =>
{
    var exceptionHandlerPathFeature = context.Features.Get<IExceptionHandlerPathFeature>
();
    var exception = exceptionHandlerPathFeature.Error;

    // Custom logic for handling the exception
    // ...

    context.Response.ContentType = "text/html";
    await context.Response.WriteAsync("<html><body>\r\n");
    await context.Response.WriteAsync("Custom Error Page<br><br>\r\n");

    // Display custom error details
    await context.Response.WriteAsync($"<strong>Error:</strong> {exception.Message}
<br>\r\n");
    await context.Response.WriteAsync("</body></html>\r\n");
}));
```