**ASP.NET Core developer exception page**

we will discuss **Developer Exception Page** and it's use for us as a developer.  
  
  
**UseDeveloperExceptionPage Middleware**  
  
**Consider the following code in Configure() method in Startup class**

public void Configure(IApplicationBuilder app, IHostingEnvironment env)  
{  
    if (env.IsDevelopment())  
    {  
        app.UseDeveloperExceptionPage();  
    }  
  
    app.UseFileServer();  
  
    app.Run(async (context) =>  
    {  
        throw new Exception("Some error processing the request");  
        await context.Response.WriteAsync("Hello World!");  
    });  
}

If we run our application with the above code we do not see the exception, instead we see **"Hello from Default.html"**. If you understand how asp.net core request processing pipeline works, then you might already know the reason why we do not see the exception that we are throwing.   
  
UseFileServer middleware combines the functionality of UseDefaultFiles and UseStaticFiles middleware. In our previous videos in this series we have included a default html document with name **default.html**in **wwwroot folder**.   
  
So, a request to the application root URL i.e http://localhost:49119 is handled by UseFileServer middleware and the pipeline reverses from there. So the next middleware in the request pipeline that we have registered using the Run() method will not get the chance to execute and hence we do not see the exception thrown by this middleware.  
  
Now, if we issue a request to http://localhost:49119/abc.html, we see the exception as expected. So, in this case, UseFileServer middleware will not find the file with name **abc.html**. It gives control to the next middleware in the pipeline, which in our case is the middleware that we have registered using Run() method. This middleware throws an exception and we see the exception details as expected.  
  
If you have any experience with classic asp.net, then you must be very familiar with this page. This is similar to that **yellow screen of death**in classic asp.net.   
  
As the name implies, this **Developer Exception Page**contains exception details like 

* Stack trace including the file name and line number that caused the exception
* Query String, Cookies and HTTP headers

At the moment, on the exception page, on the **"Query"** tab we see **"No QueryString data"**. If you have any query string parameters in the request URL like the following, you will see them under the **"Query"** tab.  
http://localhost:48118/abc.html?country=usa&state=iowa  
  
**Customising UseDeveloperExceptionPage Middleware**  
  
Like most other middleware components in ASP.NET Core, we can also customize UseDeveloperExceptionPage middleware. Whenever you want to customize a middleware component, always remember you may have the respective OPTIONS object. So, to customize UseDeveloperExceptionPage middleware, use DeveloperExceptionPageOptions object as shown below.

DeveloperExceptionPageOptions developerExceptionPageOptions = new DeveloperExceptionPageOptions  
{  
    SourceCodeLineCount = 10  
};  
app.UseDeveloperExceptionPage(developerExceptionPageOptions);

SourceCodeLineCount property specifies how many lines of code to include before and after the line of code that caused the exception.  
  
**How UseDeveloperExceptionPage Middleware works**  
  
UseDeveloperExceptionPage Middleware must be plugged into the request processing pipeline as early as possible, so it can handle the exception and display the Developer Exception Page if the subsequent middleware components in the pipeline raises an exception.  
  
**Consider the following code :**UseDeveloperExceptionPage() middleware is present after the middleware registered using Run() method. So in this case, developer exception page will not be displayed. This is the reason it must be plugged in as early as possible in the request processing pipeline.

public void Configure(IApplicationBuilder app, IHostingEnvironment env)  
{  
    //app.UseFileServer();  
  
    app.Run(async (context) =>  
    {  
        throw new Exception("Some error processing the request");  
        await context.Response.WriteAsync("Hello World!");  
    });  
  
    if (env.IsDevelopment())  
    {  
        DeveloperExceptionPageOptions developerExceptionPageOptions = new DeveloperExceptionPageOptions  
        {  
            SourceCodeLineCount = 10  
        };  
        app.UseDeveloperExceptionPage(developerExceptionPageOptions);  
    }  
}