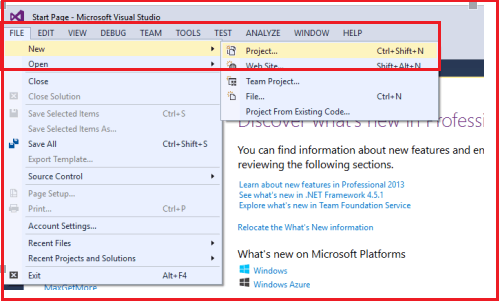
**What is Exception Filter in ASP.NET MVC Application?**

The Exception Filter in the ASP.NET MVC Application is used to handle any exceptions that occur during the ASP.NET MVC Request processing pipeline. The ASP.NET MVC Framework provides one in-built attribute called **HandleError** which is basically used to handle the unhandled exception in the MVC application.

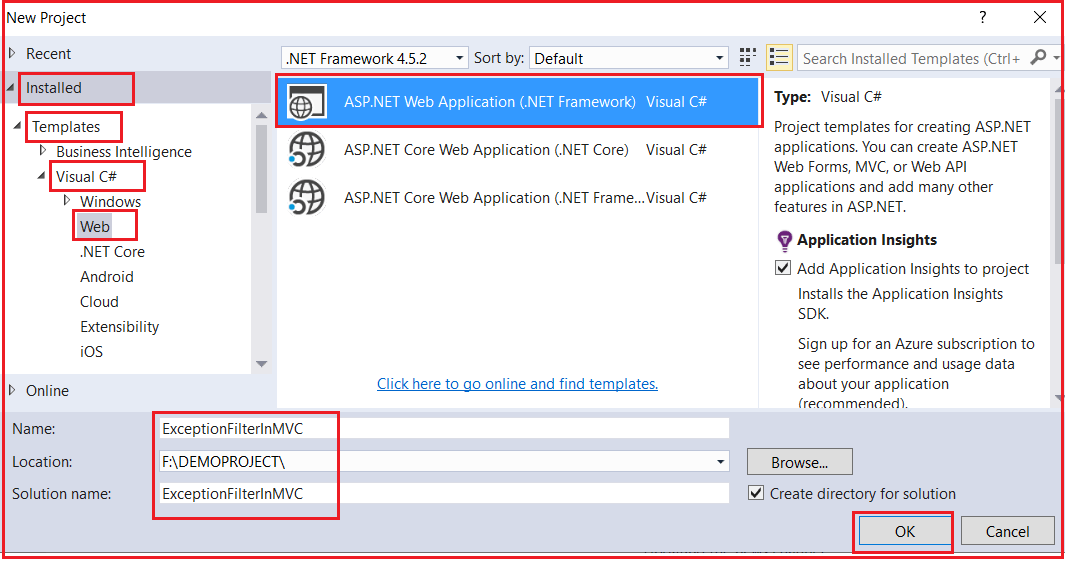
As part of this article, we will see how to use the **HandleError**attribute to display friendly error pages to the end-user when there is an unhandled exception occurs during the request processing pipeline. Let us understand this with an example.

**Create a new Empty MVC application:**

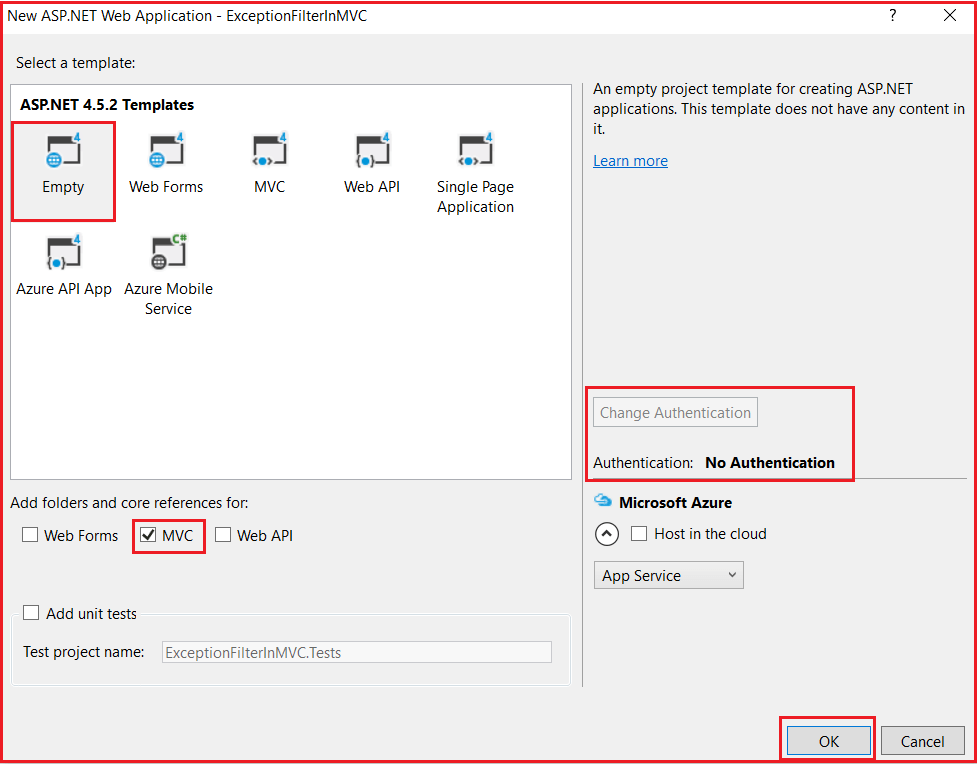
Open visual studio and then select **File => New => Project**option from the context menu as shown in the below image.



Once you click on the “**Project**” link, a new window will open. From that window, we need to select the “**Web**” templates from the left pane. From the middle pane, select the “**ASP.NET Web Application**“. Then provide a meaningful name to the project such as “**ExceptionFilterInMVC**”. Finally, click on the “**OK**” button as shown in the below image



Once you click on the “**OK”** button a new dialog window will open with the name “**New ASP.NET Project**” for selecting project Templates as shown in the below image.



From the above window, we need to choose the “**Empty”** and “**MVC”** project template with the Authentication type as “**No Authentication”** and then click on the “**OK**” button. Once you click on the **OK** button it will take some to time create the project for us.

**Creating Controller:**

Create an Empty MVC5 controller with the name HomeController within the Controllers Folder. Once you create the Controller copy and paste the following code into it.

**using** *System;*

**using** *System.Web.Mvc;*

**namespace** *ExceptionFilterInMVC.Controllers*

**{**

**public** **class** HomeController : Controller

**{**

**public** ActionResult Index**()**

**{**

**throw** new Exception**(**"Something went wrong"**)**;

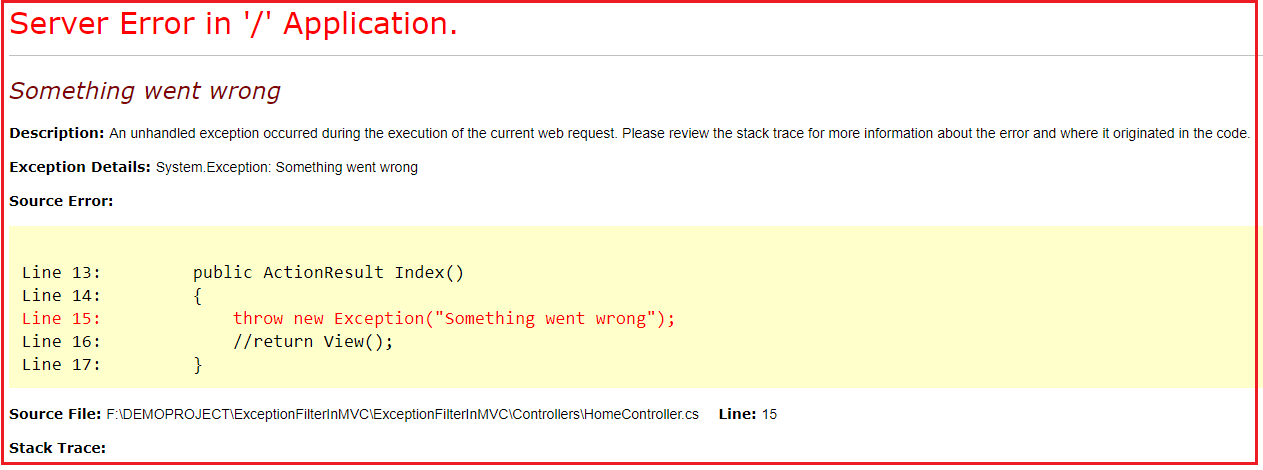
//return View();

**}**

**}**

**}**

As you can see, from the **Index**() action method, we intentionally throw an exception. As we have not handled this exception, so when we run the application, we will get the default **“yellow screen of death”** error page as shown in the below image.



The above error page does not make any sense for the end-user to understand. Now, let us see how to replace the above yellow screen of death error page with a friendly error page that can be understood by the end-user.

**How to use Handle Error Attribute in ASP.NET MVC Application?**

We can use the Handle Error attribute in three simple steps:

**Step1: Creating Error.cshtml view**

Create Shared Folder within the Views folder if it does not exist already. Then create one view with the name **Error.cshtml** within the shared folder. Once you create the **Error.cshtml** view then copy and paste the following code in it.

@{

Layout = null;

}

<!DOCTYPE html>

**<html>**

**<head>**

**<meta** name="viewport" content="width=device-width" **/>**

**<title>**Error**</title>**

**</head>**

**<body>**

**<hgroup>**

**<h1>**Unknwo Error**</h1>**

**<h2>**An unknown error has occurred. We are working on it. Please try after some time**</h2>**

**</hgroup>**

**</body>**

**</html>**

**Step2: Enable Custom Errors in the web.config file**

To enable Custom Errors for your application, open the web.config file that is present in the root directory and then adds the following “**customErrors**” element under the “**<system.web>**” section.

**<customErrors mode=”On”></customErrors>**

**Step3: Apply Handle Error Attribute in ASP.NET MVC**

You can apply the HandleError attribute at all three different locations i.e. At the Action Method Level, at the Controller level, and Global Level.

**Applying at Controller Level:**

Please modify the Home Controller as shown below to apply the Handle Error attribute to handle the unhandled exception that occurred during the execution of this Controller actions.

**using** *System;*

**using** *System.Web.Mvc;*

**namespace** *ExceptionFilterInMVC.Controllers*

**{**

**[**HandleError**]**

**public** **class** HomeController : Controller

**{**

**public** ActionResult Index**()**

**{**

**throw** new Exception**(**"Something went wrong"**)**;

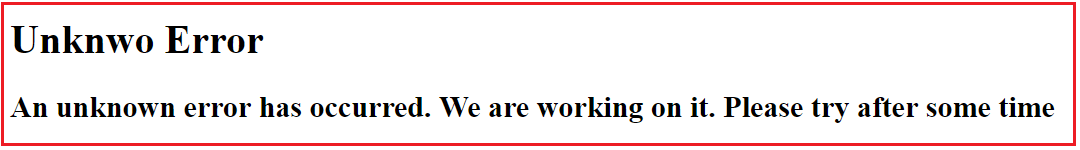
//return View();

**}**

**}**

**}**

Now run the application and you will see that instead of the yellow screen of death error page, you will now get the generic error page as shown below.



**Using HandleError Globally in ASP.NET MVC:**

If you want to use the **HandleError** attribute globally, then you need to register it within the **GlobalFilters**. You can do this within the **Application\_Start()** method of the **Global.asax file** as shown below.

**using** *System.Web.Mvc;*

**using** *System.Web.Routing;*

**namespace** *ExceptionFilterInMVC*

**{**

**public** **class** MvcApplication : System.Web.HttpApplication

**{**

**protected** **void** Application\_Start**()**

**{**

AreaRegistration.RegisterAllAreas**()**;

RouteConfig.RegisterRoutes**(**RouteTable.Routes**)**;

//Adding Handle Error attribute Globally

GlobalFilters.Filters.Add**(**new HandleErrorAttribute**())**;

**}**

**}**

**}**

Now remove the HandleError Attribute from the Controller. With the above changes in place, now it is going to handle all the exceptions raised by all the action methods of all the controllers and return the error view which is present inside the shared folder.

**How to register Filters Using FilterConfig in ASP.NET MVC?**

You can also register the Filters globally by using the FilterConfig class. Let see how we can do this. Add a class file with the name **FilterConfig.cs** within the **App\_Start** folder of your application. Once you create the class file then copy and paste the following code into it.

**using** *System.Web.Mvc;*

**namespace** *ExceptionFilterInMVC.App\_Start*

**{**

**public** **class** FilterConfig

**{**

**public** **static** **void** RegisterGlobalFilters**(**GlobalFilterCollection filters**)**

**{**

filters.Add**(**new HandleErrorAttribute**())**;

**}**

**}**

**}**

**Modifying the Global.asax file:**

Modify the **Application\_Start** event of the **Global.asax** file as shown below. Here we are just calling the RegisterGlobalFilters method of FilterConfig class.

**using** *System.Web.Routing;*

**using** *ExceptionFilterInMVC.App\_Start;*

**namespace** *ExceptionFilterInMVC*

**{**

**public** **class** MvcApplication : System.Web.HttpApplication

**{**

**protected** **void** Application\_Start**()**

**{**

AreaRegistration.RegisterAllAreas**()**;

RouteConfig.RegisterRoutes**(**RouteTable.Routes**)**;

//calling RegisterGlobalFilters to register filters globally

FilterConfig.RegisterGlobalFilters**(**GlobalFilters.Filters**)**;

**}**

**}**

**}**

With the above changes, now run the application and it should display the error page as expected.

**How to** **display error detail in the Error view?**

If you want to display the error details in the error view, then you need to make the error view a strongly typed view of the Model **System.Web.Mvc.HandleErrorInfo**. Then as usual by using the **@Model** keyword, you can access the necessary properties to display errors. So, modify the **Error.cshtml** view which is present in the Shared folder as shown below.

@{

Layout = null;

}

@model System.Web.Mvc.HandleErrorInfo

<!DOCTYPE html>

**<html>**

**<head>**

**<meta** name="viewport" content="width=device-width" **/>**

**<title>**Error**</title>**

**</head>**

**<body>**

**<hgroup>**

**<h1>**Erro Occured **</h1>**

**<h2>**Controller Name: @Model.ControllerName**</h2>**

**<h2>**Action Name: @Model.ActionName**</h2>**

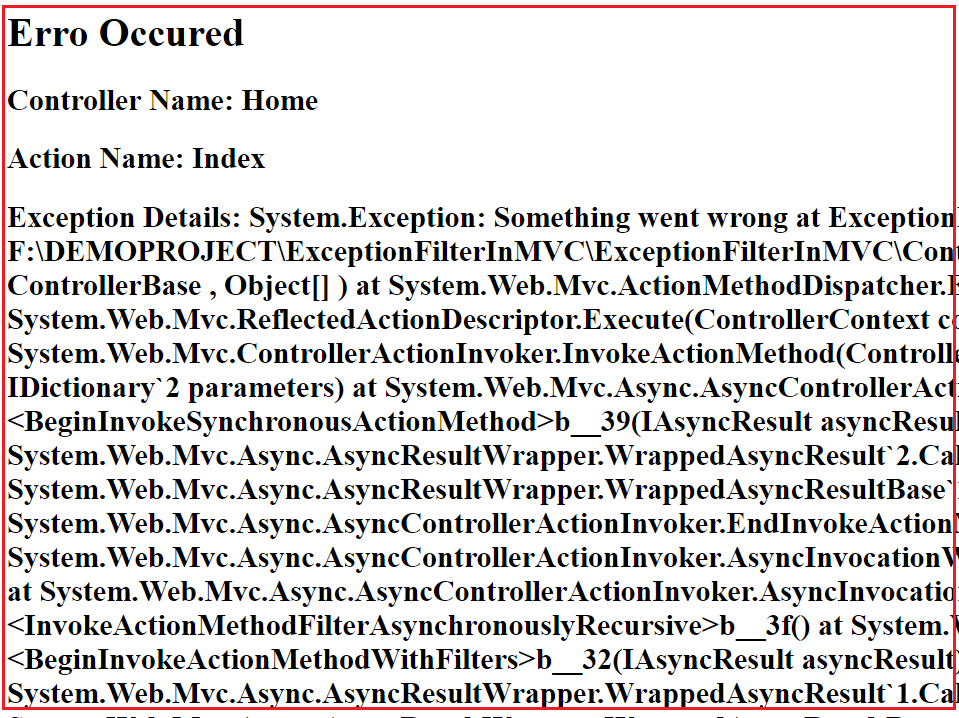
**<h2>**Exception Details: @Model.Exception**</h2>**

**</hgroup>**

**</body>**

**</html>**

Now run the application and navigate to **Home/Index** it will display the following error page.



**How to display Different Error Page for Different Exceptions?**

Let understand this with an example. First, add two views in the shared folder. Create **NullReference.cshtml** view within the Shared folder. Then copy and paste the following code into it.

@{

Layout = null;

}

<!DOCTYPE html>

**<html>**

**<head>**

**<meta** name="viewport" content="width=device-width" **/>**

**<title>**Null Reference**</title>**

**</head>**

**<body>**

**<hgroup>**

**<h1>**Erro Occured **</h1>**

**<h2>**Null reference Exception occurred**</h2>**

**</hgroup>**

**</body>**

**</html>**

Create **DivideByZero.cshtml** view within the Shared Folder. Then copy and paste the following code in it.

@{

Layout = null;

}

<!DOCTYPE html>

**<html>**

**<head>**

**<meta** name="viewport" content="width=device-width" **/>**

**<title>**DivideByZero**</title>**

**</head>**

**<body>**

**<hgroup>**

**<h1>**Erro Occured **</h1>**

**<h2>**Divide by zero Exception occurred**</h2>**

**</hgroup>**

**</body>**

**</html>**

**Modifying the HomeController:**

Please modify the Home Controller as shown below. As you can see we have applied the HandleError attribute at the Controller level. While applying the HandleError attribute, we have also checked the Exception type, and based on the Exception Type we have specified the view name. So, in that case, if the Exception type is **DivideByZeroException** then the **DivideByZero** view is going to be rendered. In the same line if the Exception type is **NullReferenceException** then the **NullReference** view is going to be rendered. Except for these two any other exception occurred then the **Error** view is going to be displayed.

**[**HandleError**(**ExceptionType = typeof**(**DivideByZeroException**)**, View = "DivideByZero"**)]**

**[**HandleError**(**ExceptionType = typeof**(**NullReferenceException**)**, View = "NullReference"**)]**

**[**HandleError**]**

**public** **class** HomeController : Controller

**{**

**public** ActionResult Index**()**

**{**

**throw** new Exception**(**"Something went wrong"**)**;

**}**

**public** ActionResult TestMethod1**()**

**{**

**throw** new NullReferenceException**()**;

**}**

**public** ActionResult TestMethod2**()**

**{**

**throw** new DivideByZeroException**()**;

**}**

**}**

**What are the Limitations of HandleErrorAttribute in MVC?**

Following are the limitations of the built-in HandleError attribute in MVC.

1. We cannot log the error anywhere using the built-in HandleError attribute.
2. It is not possible to handle the exceptions raised outside the controllers. For example- we cannot handle the exception because of the invalid URL.
3. Scenario-based Exception Handling is not possible. For example – display one error page when the request comes via AJAX and show a different error page when it comes via other than AJAX.