

Programación Web .NET Core

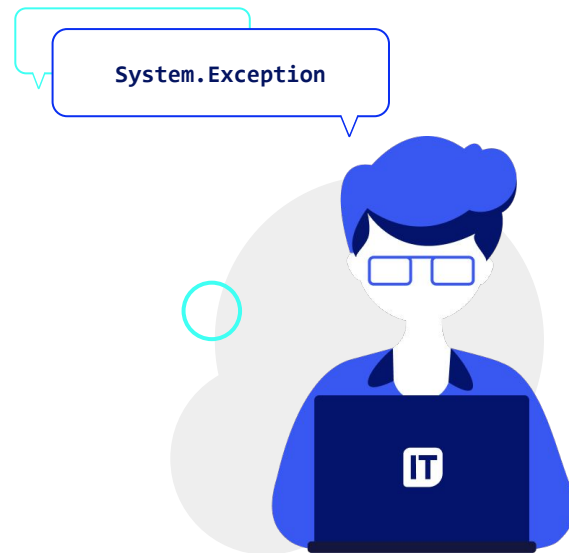
Módulo 9

Manejo de excepciones

Generación y captura de excepciones


Un **error** es un evento inesperado en tiempo de ejecución que evita que una solicitud complete una operación. Cuando una línea de código provoca un error, **ASP.NET Core** o el llamado **CommonLanguageRuntime(CLR)** crea una **excepción**. Esta excepción es un **objeto de una clase que hereda de la clase base denominada `System.Exception`**.

Hay muchas clases de excepción. A menudo, **la clase de objeto identifica qué salió mal**. Por ejemplo, si hay un `ArgumentNullException`, indica que un valor nulo se envió a un método que no acepta un valor nulo como argumento.

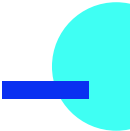


Excepciones no controladas

Cuando una aplicación no maneja explícitamente una excepción, la aplicación se detiene y el usuario ve un **mensaje de error**. En las aplicaciones **ASP.NET Core MVC**, este mensaje de error tiene el **formato de una página web**. Puedes anular las páginas de error predeterminadas de ASP.NET Core para mostrar tu propia información de error a los usuarios.



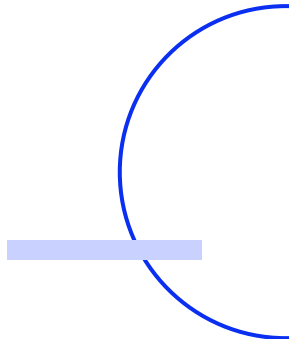
Si surge una excepción no controlada mientras se estás depurando la aplicación en **Visual Studio**, la ejecución se rompe en la línea que generó la excepción. Puedes utilizar las herramientas de depuración de Visual Studio para investigar qué salió mal, aislar el problema y depurar el código.



Generar tus propias excepciones

Es posible que también desees generar tus propias excepciones para **alertar a los componentes de que algo salió mal**.

Por ejemplo, considera una aplicación que calcula los precios de los productos después de agregar impuestos. En el modelo **producto**, implementas un método **GetPriceWithTax** que agrega un porcentaje de impuesto al precio base de un producto. Si el porcentaje de impuestos es un número negativo, es posible que desees generar una excepción. Este enfoque habilita el código en el controlador que llama al método denominado **GetPriceWithTax**, para manejar el caso de que se haya utilizado un porcentaje de impuestos negativo. Puedes usar la palabra clave **throw** para generar errores en código C#.



Detección de errores con bloques Try / Catch

La forma más utilizada para **detectar una excepción**, que funciona en cualquier código de Microsoft .NET Framework, es usar el **bloque try/catch**. Se ejecuta el código en el bloque **try**. Si alguna sentencia de ese código genera una excepción, el tipo de excepción se compara con el tipo declarado en el bloque **catch**. Si el tipo coincide o es de un tipo derivado del tipo declarado en el bloque catch, se ejecuta el código en el bloque catch. Puedes usar el código en el bloque de captura para obtener información sobre lo que salió mal y resolver la condición de error.

Veamos un ejemplo en la próxima pantalla.



El siguiente ejemplo de código muestra un bloque **try/catch** que detecta una **ArgumentNullException**:

```
try
{
    Productproduct = FindProductFromComment(comment);
}
catch (ArgumentNullException ex)
{
    // Handletheexception
}
```

En este ejemplo, se detectará cualquier excepción del tipo **ArgumentNullException**. Si ocurre una excepción de un tipo que no sea **ArgumentNullException**, la excepción no se detectará.



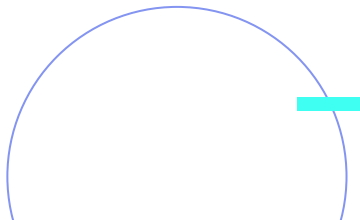
Crear tipos de excepciones personalizados

Si bien hay muchos tipos de excepciones integradas en el sistema, a veces es posible que se deba generar una excepción que no se ajusta a ninguno de los estándares. Para este propósito, puedes crear una clase excepción personalizada. Esto puede permitirte **crear excepciones que tengan sentido para tu aplicación**.

Para crear una **excepción personalizada**, deberás crear una nueva clase que herede de **Exception**, o cualquier otra clase que se derive de **Exception**.

El siguiente código es un ejemplo de una clase de excepción personalizada:

```
public class InvalidTaxException : Exception
{
}
```



El siguiente código es un ejemplo de un método que lanza una excepción personalizada:

Lanzar excepción personalizada

```
public float GetPriceWithTax(float taxPercent)
{
    if (taxPercent < 0)
    {
        throw new InvalidTaxException();
    }
    return BasePrice + (BasePrice * (taxPercent / 100));
}
```

El siguiente código es un ejemplo de cómo detectar una excepción personalizada:

Detectar una excepción personalizada

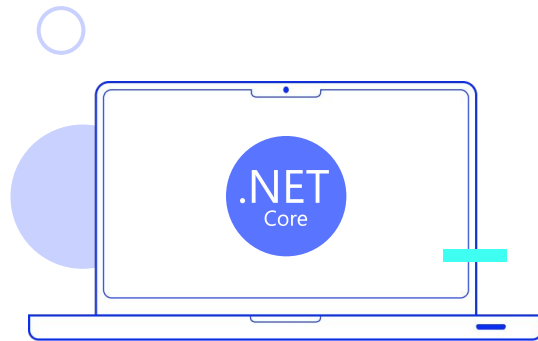
```
try
{
    float price = product.GetPriceWithTax(-20);
}
catch (InvalidTaxException ex)
{
    return Content("Tax cannot be negative");
}
```



Trabajar con varios entornos

En ASP.NET Core, la variable de entorno llamada **ASPNETCORE_ENVIRONMENT** debe usarse para **determinar el entorno de la aplicación ASP.NET Core**. Si no se establece, el valor predeterminado de la variable de entorno cuyo nombre es **ASPNETCORE_ENVIRONMENT** será **Producción**. Puedes, en cualquier momento, especificar su entorno de trabajo.

En una aplicación ASP.NET Core, al declarar el middleware, es posible usar el actual entorno de ejecución para tomar decisiones que afecten el comportamiento de la aplicación.



Un uso muy común para el manejo de excepciones.

Ejecuta una aplicación en modo de desarrollo, a menudo querrás que los errores de las páginas contengan el seguimiento de pila detallado con tanta información como sea posible sobre la excepción que ocurrió.

Sin embargo, al hacerlo en un entorno de producción, los clientes recibirán páginas llenas de detalles de errores que no significan nada y no son deseables para la mayoría de los usuarios. Estas páginas contienen información que no se debe exponer a usuarios malintencionados para que puedan explotarla.

Para resolver esto, puedes utilizar el servicio **IHostingEnvironment**. Este servicio se inyecta con frecuencia en el método **Configure** de la clase **Startup**. La interfaz **IHostingEnvironment** expone métodos útiles:

IsDevelopment, **IsStaging**, **IsProduction** e **IsEnvironment (* EnvironmentNombre*)**.

Puedes utilizar estos métodos para determinar el entorno que se está ejecutando actualmente.

El siguiente código es un ejemplo del uso del servicio **IHostingEnvironment**:

Uso del servicio IHostingEnvironment

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    elseif (env.IsStaging() || env.IsProduction() ||
        env.IsEnvironment("ExternalProduction"))
    {
        app.UseExceptionHandler("/error");
    }
    app.UseMvcWithDefaultRoute();
    app.Run(async (context) =>
    {
        await context.Response.WriteAsync("...");
    });
}
```

En este ejemplo, puedes ver que en un entorno de desarrollo, se llama al método denominado **UseDeveloperExceptionPage**, y en los entornos de **Staging**, **Production** y el llamado **ExternalProduction**, en su lugar, se llama a **UseExceptionHandler** con una ruta de **/error**.

Usar entornos en vistas

Otro lugar en el que puede resultar útil diferenciar entornos pueden ser las **vistas**, especialmente al configurar la aplicación. A menudo querrás usar versiones sin comprimir de archivos JavaScript y CSS mientras se ejecutan en un entorno de desarrollo, en comparación con un entorno de producción donde querrás utilizar archivos empaquetados y minificados en su lugar. Para lograr esto, puedes utilizar el **Tag Helper de entorno**.

El **Tag Helper de entorno** te permite utilizar y establecer la **propiedad de inclusión o exclusión**, puedes especificar en qué entornos se ejecutará el código. Esto es particularmente útil para cargar **scripts** y **archivos CSS**.

Tag helper de entorno



El siguiente código es un ejemplo del uso del asistente de etiquetas de entorno:

Tag Helper de entorno

```
<!DOCTYPEhtml>
<html>
<head>
<environmentinclude="Development">
<linkrel="stylesheet" href="~/Styles/bootstrap.css" />
</environment>
<environmentexclude="Development">
<linkrel="stylesheet" href="~/Styles/vendor-min.css" />
</environment>
</head>
<body>
<div class="containerbody-content">
@RenderBody()
</div>
```



...

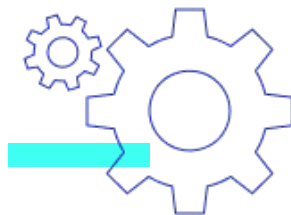
```
<environmentinclude="Development">
<scriptsrc="~/Scripts/jquery.js"></script>
<scriptsrc="~/Scripts/popper.js"></script>
<scriptsrc="~/Scripts/bootstrap.js"></script>
</environment>
<environmentinclude="Production,Staging">
<scriptsrc="~/Scripts/vendor.min.js"></script>
</environment>
</body>
</html>
```



Configurar el manejo de errores

En un entorno de desarrollo, a menudo se desea **información detallada sobre errores**. Esto te permite extrapolar información importante sobre qué salió mal y cómo. Se añade el método **Configure UseDeveloperExceptionPage** en el middleware, para ver todas las excepciones que ocurren después de que el middleware **UseDeveloperExceptionPage** es redirigido, por defecto, a una página que tiene información detallada sobre la excepción. Dentro de la página, puedes cambiar entre varias pestañas para ver información importante, como el seguimiento de la pila de la excepción, la cadena

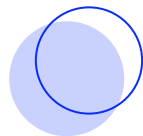
de consulta, datos de cookies y encabezados enviados como parte de la solicitud. Todo esto puede ser de gran ayuda mientras depuras la aplicación y para reproducir errores conocidos porque la información relevante para la excepción está presente y es de fácil acceso.



El siguiente código es un ejemplo del uso del middleware
UseDeveloperExceptionPage:

El middleware UseDeveloperExceptionPage

```
public void Configure(IApplicationBuilder app,
    IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    app.Run(async (context) =>
    {
        await Task.Run(() => throw new NotImplementedException());
    });
}
```



Página del controlador de excepciones personalizado

Mientras estés en un entorno de desarrollo, a menudo querrás **ver las excepciones y errores detallados**. En otros entornos, querrás usar páginas más fáciles de usar y de ayuda inmediata. Para ese propósito, puedes utilizar el middleware **UseExceptionHandler**. Este middleware recibe una URL de cadena a la que será redirigida la solicitud. Esta URL se puede utilizar para redirigir la solicitud problemática a un controlador que maneja las excepciones proporcionando una vista, o en un archivo HTML estático.

El código de la siguiente pantalla, es un ejemplo del uso de un controlador de excepciones personalizado.



El middleware UseExceptionHandler

```
public void Configure(IApplicationBuilder app,
    IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    elseif (env.IsStaging() || env.IsProduction() ||
        env.IsEnvironment("ExternalProduction"))
    {
        app.UseExceptionHandler("/error");
    }
    app.UseMvcWithDefaultRoute();
    app.Run(async (context) =>
    {
        await Task.Run(() => throw new NotImplementedException());
    });
}
```



Se debe tener en cuenta que, para implementar un controlador de excepciones, se deberá llamar a **UseMvc** o el middleware denominado **UseMvcWithDefaultRoute**. Si prefieres usar un archivo HTML estático, puedes utilizar un controlador de excepciones que redirija a un archivo estático junto con el middleware llamado **UseStaticFiles**.

Como regla general, querrás mantener esta página lo más estática posible, ya que si se produce un error arroja excepciones, en su lugar se mostrará una página de error genérica. Al crear controladores y vistas para mostrar errores, asegúrate de mantener la **lógica simple**.

El siguiente código muestra un controlador que funciona como un controlador de excepciones:

Controlador de controlador de excepciones

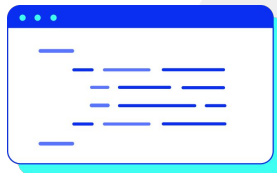
```
public class ErrorController : Controller
{
    public IActionResult Index()
    {
        return View();
    }
}
```



Páginas de códigos de estado

Un método adicional de manejo de errores es el middleware **UseStatusCodePages**. Este middleware está diseñado para proporcionar un manejo básico de varios errores que pueden ocurrir al mostrar una breve explicación del código de error al usuario. En general, el middleware **UseStaticCodePages** está diseñado para manejar los casos en los que alguna parte de la solicitud salió mal sin generar una excepción.

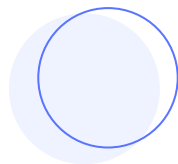
Este método captura códigos de estado html entre 400 y 599, que generalmente cubren todos los casos en los que algo resultó incorrecto (los 400 representan errores del lado del cliente, mientras que los 500 cubren los errores del lado del servidor).



El siguiente código es un ejemplo del uso del middleware **UseStatusCodePages**:

Uso del middleware UseStatusCodePages

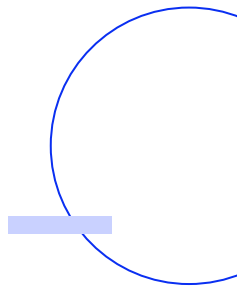
```
public void Configure(IApplicationBuilder app,
    IHostingEnvironment env)
{
    app.UseStatusCodePages();
    app.Run(async (context) =>
    {
        await Task.Run(() => context.Response.StatusCode = 404);
    });
}
```



No estás limitado a llamar solo al middleware `UseStatusCodePages` para manejar códigos de estado HTTP.

También están disponibles los siguientes métodos:

- **`UseStatusCodePagesWithRedirects`**: Permite enviar una URL de redireccionamiento al cliente. Esta nueva URL se utilizará en lugar del original. El uso del parámetro `{0}` en la URL se puede utilizar para establecer el estado del código.
- **`UseStatusCodePagesWithReExecute`**: Similar a la redirección. Sin embargo, esta redirección ocurre directamente en el pipeline sin afectar al cliente.



Manejo de excepciones en el servidor

A veces, en la mayoría de las aplicaciones ASP.NET Core MVC, puede ocurrir una excepción en el servidor web, y no quedar atrapado dentro de la aplicación. En este caso, si se detecta la excepción antes de que se envíen los encabezados de respuesta, **la respuesta se enviará sin cuerpo y con un código de estado 500 (error interno del servidor)**. Por otro lado, si los encabezados de respuesta se enviaron antes de que el servidor detecte la excepción, **la conexión se cierra de inmediato**.

Por lo tanto, es importante evitar cualquier **excepción que salga de la aplicación** porque tales errores pueden ser muy difíciles de rastrear.



Manejo de errores de estado del modelo

Otra opción para manejar excepciones es a través del **modelo**. Como parte de la acción del controlador, **el modelo se valida mediante el uso de la propiedad `IsValid`** dentro de la acción del controlador. El modelo puede dejar de procesar datos inválidos y tomar las acciones en el controlador.

`IsValid`



Usar filtros de excepción

También puedes configurar un filtro especial llamado ***filtro de excepción***.

Puedes crear un filtro de excepciones creando una clase que herede de la clase denominada **ExceptionHandlerAttribute**. Este filtro se puede aplicar como un atributo a acciones específicas y controladores, lo que permite manejar **casos de excepción específicos**.

Veamos un ejemplo de código de filtro de excepción, en la próxima pantalla.



Clase de filtro de excepción

```
public class MyExceptionHandler : ExceptionFilterAttribute
{
    public override void OnException(ExceptionContext context)
    {
        if (!context.ModelState.IsValid)
        {
            var result = new ViewResult { ViewName = "InvalidModel" };
            context.Result = result;
            context.ExceptionHandled = true;
        }
    }
}
```



El siguiente código es un ejemplo de cómo aplicar un filtro de excepción a una acción:

Acción con filtro de excepciones

```
public class ProductController : Controller
{
    [MyExceptionHandler]
    public IActionResult Index(Product product)
    {
        if (!ModelState.IsValid)
        {
            throw new Exception();
        }
        return View(product);
    }
}
```

En el ejemplo, se observa un filtro de error que redirige a la vista **InvalidModel** siempre que una excepción ocurre con un modelo inválido.

**¡Sigamos
trabajando!**