

Tipo : Guía de Laboratorio
Capítulo : Aplicando técnicas en una aplicación ASP.NET MVC
Duración : 180 minutos

I. OBJETIVO

Crear el Action Filters.

II. REQUISITOS

Los siguientes elementos de software son necesarios para la realización del laboratorio:

- Windows 10 (como mínimo Windows 8)
- Visual Studio 2017 (como mínimo Visual Studio 2015)

III. EJECUCIÓN DEL LABORATORIO

- Ejercicio N° 5.3: Crear el Action Filters.
1. Abrir la solución **Cibertec**.
 2. Instalar el paquete de Log4Net en el proyecto: **Cibertec.Mvc**
 - Install-Package log4net
 3. Configuramos el uso de Log4Net de la siguiente manera:
(1) Agregar sección en el web.config:

```
<configSections>
  <section name="log4net"
type="log4net.Config.Log4NetConfigurationSectionHandler, log4net" />
</configSections>
<log4net>
  <appender name="RollingFile"
type="log4net.Appender.RollingFileAppender">
    <file value="C:\Logs\Cibertec.Mvc.log"/>
    <appendToFile value="true"/>
    <maximumFileSize value="10000KB"/>
    <maxSizeRollBackups value="10"/>
    <layout type="log4net.Layout.PatternLayout">
      <conversionPattern value="%date %-5level %logger -
%message%newline"/>
    </layout>
  </appender>
  <root>
    <level value="DEBUG"/>
    <appender-ref ref="RollingFile"/>
    <appender-ref ref="aiAppender"/>
  </root>
</log4net>
```

Al finalizar debemos de tener algo como la siguiente imagen:

```

<configuration>
  <configSections>
    <section name="log4net" type="log4net.Config.Log4NetConfigurationSectionHandler, log4net" />
  </configSections>
  <log4net>
    <appender name="RollingFile" type="log4net.Appender.RollingFileAppender">
      <file value="C:\Logs\Cibertec.Mvc.log"/>
      <appendToFile value="true"/>
      <maximumFileSize value="10000KB"/>
      <maxSizeRollBackups value="10"/>
      <layout type="log4net.Layout.PatternLayout">
        <conversionPattern value="%date %-5level %logger - %message%newline"/>
      </layout>
    </appender>
    <root>
      <level value="DEBUG"/>
      <appender-ref ref="RollingFile"/>
      <appender-ref ref="aiAppender"/>
    </root>
  </log4net>
  <connectionStrings>
    <add name="NorthwindConnection" connectionString="Server=.\SQLEXPRESS;Database=Northwind_Lite" />
  </connectionStrings>

```

- (2) En el fichero **Global.asax** agregar la siguiente línea al método **"Application_Start"**:
 log4net.Config.XmlConfigurator.Configure();

Ver imagen:

```

protected void Application_Start()
{
    DIConfig.ConfigureInjector();
    AreaRegistration.RegisterAllAreas();
    FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
    RouteConfig.RegisterRoutes(RouteTable.Routes);
    BundleConfig.RegisterBundles(BundleTable.Bundles);

    log4net.Config.XmlConfigurator.Configure();
}

```

4. Procedemos a inyectar el log4net a todos los constructores que los usaran, para lo cual tenemos que crear un controlador base:

- (1) En la carpeta **Controllers** agregar un controlador con el nombre **"BaseController"**, con el siguiente código:

```

using Cibertec.UnitOfWork;
using log4net;
using System.Web.Mvc;

namespace Cibertec.Mvc.Controllers
{
    public class BaseController : Controller
    {
        protected readonly IUnitOfWork _unit;
        protected readonly ILog _log;
        public BaseController(ILog log, IUnitOfWork unit)
        {
            _log = log;
            _unit = unit;
        }
    }
}

```

- (2) En el **CustomerController** realizamos el siguiente cambio:

```

public class CustomerController : BaseController
{
    0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public CustomerController(ILog log, IUnitOfWork unit) : base(log, unit)
    {
    }
}

0 references | Julio Velarde, 1 day ago | 1 author, 1 change | 1 request | 0 exceptions
public ActionResult Index()
{
    _log.Info("Execution of Customer Controller OK");
    return View(_unit.Customers.GetList());
}

```

- Ahora heredará de la clase base.
- El constructor tiene ahora 2 parámetros.

- (3) En el fichero “**DIConfig**” de la carpeta **App_Start** realizamos los siguientes cambios:

```
using System.Web.Mvc;
using SimpleInjector;
using SimpleInjector.Integration.Web;
using SimpleInjector.Integration.Web.Mvc;
using System.Reflection;
using Cibertec.UnitOfWork;
using Cibertec.Repositories.Dapper.Northwind;
using System.Configuration;
using log4net;
using log4net.Core;

namespace Cibertec.Mvc
{
    public class DIConfig
    {
        public static void ConfigureInjector()
        {
            var container = new Container();
            container.Options.DefaultScopedLifestyle = new
WebRequestLifestyle();
            container.Register<IUnitOfWork>(() => new
NorthwindUnitOfWork(ConfigurationManager.ConnectionStrings["NorthwindCo
nnection"].ToString()));

            container.RegisterMvcControllers(Assembly.GetExecutingAssembly());
            container.RegisterConditional(typeof(ILog), c =>
typeof(Log4NetAdapter<>).MakeGenericType(c.Consumer.ImplementationType)
, Lifestyle.Singleton, c => true);
            container.Verify();
            DependencyResolver.SetResolver(new
SimpleInjectorDependencyResolver(container));
        }

        public sealed class Log4NetAdapter<T> : LogImpl
        {
            public Log4NetAdapter() :
base(LogManager.GetLogger(typeof(T)).Logger) { }
        }
    }
}
```

- (4) Para validar que todo este bien ejecutamos la aplicación y nos dirigimos al controlador de customer. Si verificamos la ruta “**C:\Logs**” ubicaremos un archivo Cibertec.Mvc.log con el siguiente contenido:

➤ *Cibertec.Mvc.Controllers.CustomerController - Execution of Customer Controller OK*

5. En el proyecto **Cibertec.Mvc** agregar la carpeta “**ActionFilters**” y en ella crea la clase “**ErrorActionFilter**” con el siguiente código:

```
using log4net;
using System.Web.Mvc;

namespace Cibertec.Mvc.ActionFilters
{
    public class ErrorActionFilter: HandleErrorAttribute
    {
        public override void OnException(ExceptionContext filterContext)
        {
            var log = LogManager.GetLogger(typeof(ErrorActionFilter));
            filterContext.ExceptionHandled = true;
            log.Error(filterContext.Exception);
            filterContext.Result = new ViewResult()
            {
                TemplateName = "Error"
            };
        }
    }
}
```

```

        ViewName = "Error"
    };
}
}
}

```

6. Y en el “**CustomerController**” procede a agregar la siguiente notación: **[ErrorActionFilter]**

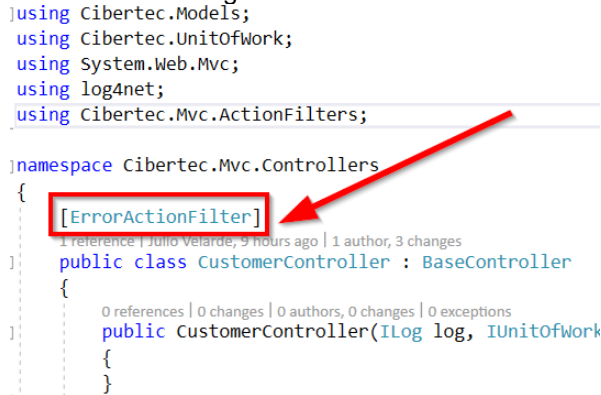
Validar con la imagen:

```

using Cibertec.Models;
using Cibertec.UnitOfWork;
using System.Web.Mvc;
using log4net;
using Cibertec.Mvc.ActionFilters;

namespace Cibertec.Mvc.Controllers
{
    [ErrorActionFilter]
    public class CustomerController : BaseController
    {
        public CustomerController(ILog log, IUnitOfWork
    {
    }
}

```



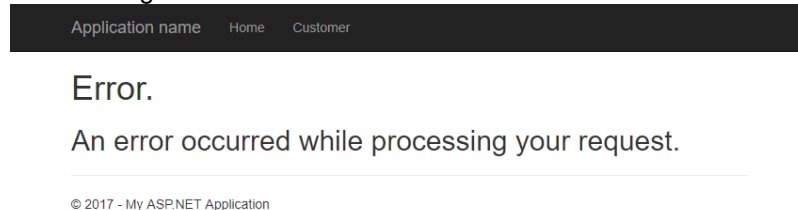
7. En el mismo controlador agregamos un ActionResult para provocar el error y validar que todo marche bien:

```

public ActionResult Error()
{
    throw new System.Exception("Test error to validate Action Filter");
}

```

8. Ahora procedemos a validar los resultados al llamar a la url /Customer/Error debemos de ver la siguiente ventana:



9. En el fichero de Log visualizaremos todo el stacktrace del error:

```

2017-09-24 23:56:40,146 ERROR Cibertec.Mvc.ActionFilters.ErrorActionFilter - System.Exception: Test error
to validate Action Filter
    at Cibertec.Mvc.Controllers.CustomerController.Error() in
C:\Cibertec\Repos\WebDeveloper\WebDeveloper\Cibertec\Cibertec.Mvc\Controllers\CustomerController.cs:li
ne 17
    at lambda_method(Closure , ControllerBase , Object[] )
    at System.Web.Mvc.ActionMethodDispatcher.Execute(ControllerBase controller, Object[] parameters)
    at System.Web.Mvc.ReflectedActionDescriptor.Execute(ControllerContext controllerContext, IDictionary`2
parameters)
    at System.Web.Mvc.ControllerActionInvoker.InvokeActionMethod(ControllerContext controllerContext,
ActionDescriptor actionDescriptor, IDictionary`2 parameters)
    at
System.Web.Mvc.Async.AsyncControllerActionInvoker.<BeginInvokeSynchronousActionMethod>b__39(IAsyn
cResult asyncResult, ActionInvocation innerInvokeState)
    at System.Web.Mvc.Async.AsyncResultWrapper.WrappedAsyncResult`2.CallEndDelegate(IAsyncResult
asyncResult)
    at System.Web.Mvc.Async.AsyncResultWrapper.WrappedAsyncResultBase`1.End()

```

```
    at System.Web.Mvc.Async.AsyncControllerActionInvoker.EndInvokeActionMethod(IAsyncResult
asyncResult)
    at
System.Web.Mvc.Async.AsyncControllerActionInvoker.AsyncInvocationWithFilters.<InvokeActionMethodFilter
AsynchronouslyRecursive>b__3d()
    at
System.Web.Mvc.Async.AsyncControllerActionInvoker.AsyncInvocationWithFilters.<>c__DisplayClass46.<Inv
okeActionMethodFilterAsynchronouslyRecursive>b__3f()
    at
System.Web.Mvc.Async.AsyncControllerActionInvoker.<>c__DisplayClass33.<BeginInvokeActionMethodWith
Filters>b__32(IAsyncResult asyncResult)
    at System.Web.Mvc.Async.AsyncResultWrapper.WrappedAsyncResult`1.CallEndDelegate(IAsyncResult
asyncResult)
    at System.Web.Mvc.Async.AsyncResultWrapper.WrappedAsyncResultBase`1.End()
    at System.Web.Mvc.Async.AsyncControllerActionInvoker.EndInvokeActionMethodWithFilters(IAsyncResult
asyncResult)
    at
System.Web.Mvc.Async.AsyncControllerActionInvoker.<>c__DisplayClass21.<>c__DisplayClass2b.<BeginIn
vokeAction>b__1c()
    at
System.Web.Mvc.Async.AsyncControllerActionInvoker.<>c__DisplayClass21.<BeginInvokeAction>b__1e(IAs
yncResult asyncResult)
```

IV. EVALUACIÓN

1. ¿Cuáles son las carpetas principales en una aplicación MVC?

Las carpetas principales son: Controllers, Views, Models y wwwroot

2. ¿Qué pasa si al nombre del controlador no se le coloca el prefijo Controller?

Por convención todos los archivos que van a ser controladores deben tener el prefijo Controller al final del nombre para que puedan ser reconocidos por Asp.NET como tales.

3. ¿Cuál es el tipo de datos de los métodos del controller?

El tipo base de todos los controladores es IActionResult.

4. ¿Qué es una vista Layout?

Es la vista maestra, en ella se establece la estructura base (html) de todas las vistas que tendrá la aplicación web. Algunas vistas pueden no utilizar la vista base.

5. Los modelos, ¿pueden ser librería de clases?

Sí, los modelos son un concepto que involucra entidades, acceso a datos, reglas de negocio y pueden ser implementadas en librería de clases (dll).

6. ¿Qué son las vistas parciales?

Al igual que las vistas normales, estas permiten ser reutilizadas en ciertas partes de la aplicación.