

Tipo : Guía de Enunciado
Capítulo : ASP NET Web Api
Duración : 60 minutos

I. OBJETIVO

Implementando Autenticación por Token.

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

1. Abrir la solución del módulo anterior.
2. Ubicarse en el proyecto “**Cibertec.WebApi**”
3. Instalar en el proyecto de WebApi la siguiente librería:
 - Install-Package log4net
4. Una vez instalado el paquete procedemos a editar el “**web.config**” de la siguiente manera:

```
<configSections>
  <section name="log4net"
type="log4net.Config.Log4NetConfigurationSectionHandler, log4net" />
</configSections>
<log4net>
  <appender name="RollingFile"
type="log4net.Appender.RollingFileAppender">
    <file value="C:\Logs\Cibertec.WebApi.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>
```

Guiarse por la imagen

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <!--
3 For more information on how to configure your ASP.NET application, please visit
4 https://go.microsoft.com/fwlink/?LinkId=169433
5 -->
6 <configuration>
7 <configSections>
8 <section name="log4net" type="log4net.Config.Log4NetConfigurationSectionHandler, log4net" />
9 </configSections>
10 <log4net>
11 <appender name="RollingFile" type="log4net.Appender.RollingFileAppender">
12 <file value="C:\Logs\Cibertec.WebApi.log" />
13 <appendToFile value="true" />
14 <maximumFileSize value="1000KB" />
15 <maxSizeRollBackups value="10" />
16 <layout type="log4net.Layout.PatternLayout">
17 <conversionPattern value="%date %-5level %logger - %message%newline" />
18 </layout>
19 </appender>
20 <root>
21 <level value="DEBUG" />
22 <appender-ref ref="RollingFile" />
23 <appender-ref ref="aiAppender" />
24 </root>
25 </log4net>
26 <connectionStrings>
27 <add name="NorthwindConnection" connectionString="Server=.\SQLExpress;Database=Northwind_Lite;
28 </connectionStrings>
```

5. Procedemos a editar el fichero “**Startup.cs**” de la siguiente manera:

```
using Cibertec.WebApi.Handlers;
using Microsoft.Owin;
using Owin;
using System.Web.Http;
using System.Web.Http.ExceptionHandling;

[assembly: OwinStartup(typeof(Cibertec.WebApi.Startup))]

namespace Cibertec.WebApi
{
    public class Startup
    {
        public void Configuration(IAppBuilder app)
        {
            log4net.Config.XmlConfigurator.Configure();
            var log = log4net.LogManager.GetLogger(typeof(Startup));
            log.Debug("Logging is enabled");

            var config = new HttpConfiguration();
            config.Services.Replace(typeof(IExceptionHandler), new
            GlobalExceptionHandler());

            DIConfig.ConfigureInjector(config);
            TokenConfig.ConfigureOAuth(app, config);
            RouteConfig.Register(config);
            app.UseWebApi(config);
        }
    }
}
```

6. Editamos el fichero “**DIConfig.cs**” con el siguiente código:

```
using SimpleInjector;
using System.Configuration;
using SimpleInjector.Lifestyles;
using Cibertec.UnitOfWork;
using Cibertec.Repositories.Dapper.Northwind;
using System.Web.Http;
using SimpleInjector.Integration.WebApi;
using log4net;
using log4net.Core;
```

```

namespace Cibertec.WebApi
{
    public class DIConfig
    {
        public static void ConfigureInjector(HttpConfiguration config)
        {
            var container = new Container();
            container.Options.DefaultScopedLifestyle = new
            AsyncScopedLifestyle();
            container.Register<IUnitOfWork>(() => new
            NorthwindUnitOfWork(ConfigurationManager.ConnectionStrings["NorthwindConne
            ction"].ToString()));
            container.RegisterConditional(typeof(ILog), c =>
            typeof(Log4NetAdapter<>).MakeGenericType(c.Consumer.ImplementationType),
            Lifestyle.Singleton, c => true);

            container.Verify();
            config.DependencyResolver = new
            SimpleInjectorWebApiDependencyResolver(container);
        }

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

```

7. Para poder hacer uso de nuestro Logger en todos los controladores que hereden de nuestro controlador base ("**BaseController**"), procedemos a cambiar el constructor de nuestro controlador de la siguiente manera:

```

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

namespace Cibertec.WebApi.Controllers
{
    [Authorize]
    public class BaseController : ApiController
    {
        protected readonly IUnitOfWork _unit;
        protected readonly ILog _log;

        public BaseController(IUnitOfWork unit, ILog log)
        {
            _unit = unit;
            _log = log;
        }
    }
}

```

8. Actualizamos los constructores de los controladores que hereden de nuestra clase base.

```
➤ CustomerController
public CustomerController(IUnitOfWork unit, ILog log) : base(unit, log)
{
    _log.Info($"{typeof(CustomerController)} in Execution");
}

➤ OrderController
public OrderController(IUnitOfWork unit, ILog log) : base(unit, log)
{
    _log.Info($"{typeof(OrderController)} in Execution");
}
```

9. En la raíz del proyecto crea la carpeta “**Handlers**” y en ella crea la clase “**GlobalExceptionHandler.cs**” con el siguiente código:

```
using log4net;
using System.Web.Http.ExceptionHandling;
using System.Web.Http.Results;

namespace Cibertec.WebApi.Handlers
{
    public class GlobalExceptionHandler : ExceptionHandler
    {
        private readonly ILog log =
            LogManager.GetLogger(typeof(GlobalExceptionHandler));
        public override void Handle(ExceptionHandlerContext context)
        {
            log.Error(context.Exception);
            context.Result = new
                InternalServerErrorResult(context.Request);
        }
    }
}
```

10. Para poder testear nuestro log creamos un endpoint error en “**CustomerController**” de la siguiente manera:

```
[HttpGet]
[AllowAnonymous]
[Route("error")]
public IHttpActionResult CreateError()
{
    throw new System.Exception("This is an unhandled error.");
}
```

11. En la ruta configurada en el “**web.config**”. “**C:\Logs\Web.Api.log**” validamos que contenga el siguiente resultado:



```
2017-11-24 10:53:01,046 DEBUG Cibertec.WebApi.Startup - Logging is enabled
2017-11-24 10:53:59,717 INFO Cibertec.WebApi.Controllers.CustomerController - Cibertec.WebApi.Controllers.CustomerController in Execution
2017-11-24 10:54:01,950 ERROR Cibertec.WebApi.Handlers.GlobalExceptionHandler - System.Exception: This is an unhandled error:
at Cibertec.WebApi.Controllers.CustomerController.CreateError() in C:\Repos\WebDeveloper\Cibertec\Cibertec.WebApi\Controllers\CustomerController.cs:line 63
at lambda_method(closure, Object, Object[])
at System.Web.Http.Controllers.ReflectedHttpActionDescriptor.ActionExecutor.<>c__DisplayClass8.<GetExecutor>b__9(Object instance, Object[] methodParameters)
at System.Web.Http.Controllers.ReflectedHttpActionDescriptor.ActionExecutor.Execute(Object instance, Object[] arguments)
at System.Web.Http.Controllers.ApiControllerActionInvoker.InvokeAsync(HttpControllerContext controllerContext, IDictionary`2 arguments, CancellationToken cancellationToken)
--- End of stack trace from previous location where exception was thrown ---
at System.Runtime.CompilerServices.TaskAwaiter.ThrowForNonSuccess(Task task)
at System.Runtime.CompilerServices.TaskAwaiter.HandleOnSuccessAndDebuggerNotification(Task task)
at System.Web.Http.Controllers.ApiControllerActionInvoker.InvokeAsync(CancellationToken cancellationToken)
--- End of stack trace from previous location where exception was thrown ---
at System.Runtime.CompilerServices.TaskAwaiter.ThrowForNonSuccess(Task task)
at System.Runtime.CompilerServices.TaskAwaiter.HandleOnSuccessAndDebuggerNotification(Task task)
at System.Web.Http.Controllers.ActionFilterResult.ExecuteAsync(CancellationToken cancellationToken)
--- End of stack trace from previous location where exception was thrown ---
at System.Runtime.CompilerServices.TaskAwaiter.ThrowForNonSuccess(Task task)
at System.Runtime.CompilerServices.TaskAwaiter.HandleOnSuccessAndDebuggerNotification(Task task)
at System.Web.Http.Filters.AuthorizationFilterAttribute.ExecuteAuthorizationFilterAsync(CancellationToken cancellationToken)
--- End of stack trace from previous location where exception was thrown ---
at System.Runtime.CompilerServices.TaskAwaiter.ThrowForNonSuccess(Task task)
at System.Runtime.CompilerServices.TaskAwaiter.HandleOnSuccessAndDebuggerNotification(Task task)
at System.Web.Http.Dispatcher.HttpControllerDispatcher.SendAsync(CancellationToken cancellationToken)
```

IV. EVALUACIÓN

1. ¿Por qué para vistas parciales se deben de usar “_” el guion bajo?

Bajo el lenguaje Razor, el guion bajo permite evitar mostrar Web Pages cuando son solicitadas directamente como parte de un request.