Prueba Práctica: Vueling

**Guía**

Aparte del objetivo principal, que es crear un servicio WCF Web RestFul (<http://gvasquezf12.blogspot.com.es/2013/02/como-crear-un-simple-servicio-wcf-rest.html>) que funcione, es muy importante que se implemente explícitamente cada uno de los apartados que se mencionan a continuación.

         Utilizar try/catch en todas las funciones del código

         Crear un CustomException : Exception (Incluyendo explícitamente el control de valores null)

         Guardar todas las excepciones con log4net categorizadas (la configuracion de log4net debe estar en el web.config del wcfservice o en otro fichero de configuracion log4net):

o    Error: Graves

o    Warning: Leves

o    Info: Inicio y fin de funciones

o    Debug: Con valores para poder ver desde el log

         Crear un fichero de configuración donde guardar valores a recuperar en el resto del programa.

         Poner a todos los elementos del componente un nombre significativo:

o    Aplicación

o    Proyectos

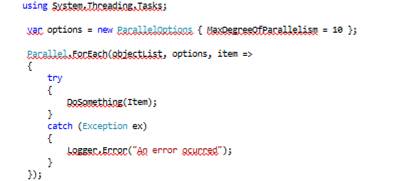
o    Clases

o    Funciones

o    Variables

         No poner el nombre helper o utils a ninguna clase.

         Utilizar multithreading



         **Implementar test unitarios para cada proyecto.**

         **Implementar, al menos, un test de integración** (un método que se pueda ejecutar desde un test para probar toda la aplicación).

         Utilizar los principios SOLID (<http://www.genbetadev.com/paradigmas-de-programacion/solid-cinco-principios-basicos-de-diseno-de-clases>)

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| S | Principio de Única Responsabilidad | Un objeto solo debería tener una única responsabilidad. | Las funciones tienen que ser sencillas y concretas. |
| O | Principio Abierto/Cerrado | Las entidades de software deben estar abiertas para su extensión, pero cerradas para su modificación. | Pasar por parámetro a las clases interfaces que se implementan en otras clases. |
| L | Principio de sustitución de Liskov | Cuando creamos clases derivadas debemos asegurarnos de no reimplementar métodos que hagan que los métodos de la clase base no funcionases si se tratasen como un objeto de esa clase base. | Las clases hijas se deben poder utilizar como las padres. |
| I | Principio de Segregación de la Interface | Muchas interfaces cliente específicas son mejores que una interfaz de propósito general. | Las interfaces tienen que ser sencillas y concretas. |
| D | Principio de Inversión de Dependencia | Desacoplar las clases. | Inyección de dependencias (autofac). |

**CRITERIOS de VALORACIÓN**

         Instrumentalization (2p)

o   Exception management - Thrown exceptions when wrong behaviours are found. Create custom exception fro them and just

o   catch them. Left application to thrwon not managed exceptions.

o   Error and Warning logging - Separate errors, warnings, infos and debug. No acoplish libraries to logging implementations.

o   Usage of standards.

o   Configuration definition - Usage of standards. Don't expose changes of configuration.

         Software Design (1p)

o   Algorithm design - Analytic thinking and Simplicity.

         Practices and Patterns (4p)

o   Naming - Good naming convections.

o   Clear flows - Low level of cyclomatic complexity.

o   Null returned values - Don't use null as returned values.

o   No helper classes - Don't abuse of helper classes.

o   SOLID principles (2/4p) - SOLID implementation.

         Scalability (1p)

o   No bottlenecks

o   Workers tasks - Use of paralle programming to improve performace

         Testeability (2p)

o   Unit Testing - Existence and best practices implementation.

o   Integration Testing - Existence and best practices implementation.