

## Contenido

Entity Framework.....	2
JavaScript.....	3
JQuery .....	4
MVC.....	5
Ordenamientos .....	6
Patrones de diseño.....	6
Programación Orientada a Objetos.....	7
SOLID .....	11
Scrum.....	12
INVEST .....	15
TDD y Refactoring.....	16
Casos de Uso .....	17
SQL.....	17
Entidad relación .....	17
Transacciones en SQL Server.....	19
ACID.....	20
VISUAL BASIC.....	22
GAC Global Assembly Cache .....	29
Pruebas de estrés.....	30
WCF .....	35
Tracing:.....	42
HTML 5 .....	44
XML.....	46

## RESUMEN:

### Entity Framework

Es una tecnología de Microsoft para el acceso a datos puede ser de tres tipos:

- ❖ **DatabaseFirst:** genera código a partir de una base de datos.
- ❖ **ModelFirst:** genera la base de datos y código a partir de un asistente visual.
- ❖ **CodeFirst:** Puede generar la base de datos y tablas a partir del código. Este es el más utilizado por los programadores porque les brinda más control y flexibilidad.

Entre las ventajas de entity framework están:

- ❖ Fácil de usar no se deben aprender a configurar esquemas.
- ❖ Fácil instalación por medio de nuget
- ❖ Es flexible y fácil de modificar.

Se encarga de generar las sentencias para creación y actualización de la BD y también genera consultas pero necesita tres componentes:

- ❖ **String de conexión:** El string de conexión puede estar en el app.config, web.config o machine.config.
- ❖ **Las clases de entidades que representan las tablas:** Cada entidad debe tener un constructor público y una propiedad que represente la llave primaria.
- ❖ **La clase contexto:** Une las demás piezas, usa el string para comunicarse con la BD y las entidades se deben agregar al contexto como propiedades tipo DbSet.

### Reglas en Entity Framework

- ❖ Siempre se busca un string de conexión con el nombre de la clase contexto.
- ❖ Se toma de llave primaria alguna propiedad llamada id.
- ❖ Cuando se va a modificar o eliminar un registro se debe usar **tracking**, cuando se consultan datos que se van a enviar como argumentos a otra función, o cuando se sabe de antemano que no se van a modificar se debe usar **asNoTracking**. Por ejemplo si es un listado.
- ❖ Cuando se quiere traer el resultado de una consulta y sus asociados debemos utilizar el **eager loading**, esto se hace en Linq mediante el uso del método **include**. En cambio el **lazy loading** no utiliza el método **include** aunque obtiene resultados por partes y no trae asociados inmediatamente.

### Bases de datos en Entity Framework

Tiene 3 componentes:

- ❖ **Conexión:** Objeto que se encarga de manejar la excepción y abrir y cerrar la conexión además de declarar inicio y fin de transacciones.
- ❖ **Command:** Objeto encargado de ejecutar las consultas.
- ❖ **DataReader:** Carga e itera los datos como un cursor, lee los registros obtenidos de uno en uno.

Para SQL la conexión se hace usando el **SqlConnection**, es importante abrir y cerrar la conexión e implementarla dentro de un **Using**, se debe usar **Commit** y **Rollback**.

El **SqlCommand** tiene varios métodos entre ellos **ExecuteReader** que retorna un **objeto** y **executeXMLReader** retorna un **XML**.

También se pueden enviar parámetros se debe agregar el **@**.





El **DataReader** devuelve los registros y se pueden guardar en **DataSet** para cargarlos se necesita un **DataAdapter**.

Si se usa **transactionScope** se debe agregar dentro de un **Using** de lo contrario cuando se haga **Dispose** se haría un **RollBack**

## JavaScript

Es un lenguaje interpretado, que se agrega directamente a las páginas y se usa para aportar interactividad a las paginas HTML.

Entre sus principales funciones están:

-  Capturar eventos.
-  Validar datos
-  Crear cookies
-  Y agregar texto dinámico a la página

Como curiosidad si se añade un **número** y una **cadena**, el resultado será una **cadena**.

Entre sus tipos de datos se encuentran los operadores aritméticos como por ejemplo **+** **-** **/** **\*** **^** **++**

Operadores de asignación entre ellos = **+=** **-=**

De comparación como por ejemplo **==** **===** **<** **>** **<=** **>=**




Operadores lógicos como **||** **o** **&**

Estructuras de condición como los **if else ifelse**

Tiene mensajes de alerta como el **confirm** (agrega dos botones un cancelar y un aceptar), **prompt** (para agregar texto e incluye un botón aceptar y cancelar)

Tiene también estructuras de iteración como **for While, Do While**

Cada elemento de una página WEB tiene una serie de eventos que pueden ser manejados vía JS, por ejemplo: Click, Carga de página, pasar el puntero por algún punto, seleccionar algún campo, teclear un botón.

-  **onLoad y onUnload:** Se dispara cada vez que el usuario entra o sale de la página.
-  **onSubmit:** Este evento se dispara cuando se requiere validar todos los campos de un formulario antes de enviarlo
-  **onMouseOver y onMouseOut:** Usados generalmente para generar animaciones y comportamiento relativo al cursor, en interacción con elementos de la página WEB.

## JQuery

Es una biblioteca de JavaScript que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, y agregar Ajax.





Se diferencia de JavaScript en que JQuery puede hacer lo mismo que JavaScript pero con menos líneas de código.

El símbolo `$()`, reemplaza a la función JQuery `()`.

Ejemplo agregar color al fondo de una página con JQuery: `$('body').css('background', '#ccc');`

Ocultar algún elemento con JQuery: `$("elemento").hide();`

## Ventajas de JQuery

-  Es flexible y rápido para el desarrollo web.
-  Es Open Source.
-  Es una excelente integración con AJAX.
- 

## ¿Qué es el DOM?

Significa **Modelo de Objetos de Documento**, es un **protocolo** del World Wide Web Consortium y brinda la comunicación entre JavaScript y HTML.

## Manipulación CSS con JQuery

jQuery permite la manipulación de las propiedades CSS que se le puedan dar a cierto elemento a través de los métodos: `.css ()`, `.attr ()`, `.addClass ()`, `.removeClass ()`, `toggleClass ()`, `removeAttr ()`, entre otros.

## MVC

Es un patrón estructural, el cual plantea la separación del problema en tres capas.

Está conformado por:

- **Model:** Esta capa representa la realidad y la lógica que se aplicara en nuestro sistema, este devuelve al controller el modelo generado.
- **Controller:** Capta los eventos producidos por el usuario y por medio de **actions** pueden generar las vistas (HTML) correspondientes a la acción a realizar o devolver al cliente. Además es el que comunica al modelo con la vista.
- **Vista:** Se encarga de mostrar visualmente en pantalla nuestro modelo. Las vistas usan *helpers que se encargan de generar cadenas de texto que se pueden convertir en código HTML, scripts...* de forma simple. Las vistas pueden recibir un modelo enviado desde su controlador.

## Ventajas de usar MVC

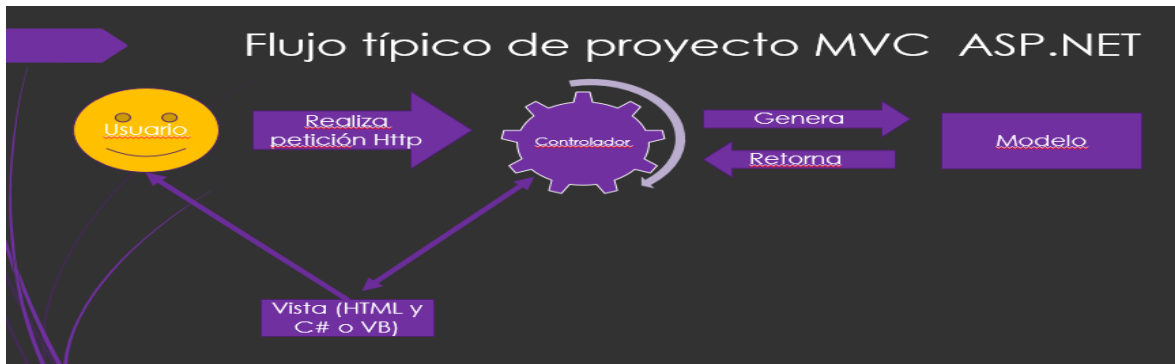
- **Menor acoplamiento:** MVC al tener bien en claro la funcionalidad de cada uno de sus capas nos provee de un menor acoplamiento entre componentes.
- **Facilidad para realizar pruebas Unitarias:** AL crear un proyecto MVC, se nos ofrece la posibilidad de crear un proyecto de pruebas unitarias.
- **Mayor control de lo que se programa:** Al tener la vista completamente separada de la lógica, unos **no “interfieren”** en las tareas de los otros.
- Uso de un **“FrontController”** que procesa todas las solicitudes a nuestra aplicación. Esto permite diseñar una infraestructura con un sistema de **enrutamiento avanzado**.

## Desventajas de usar MVC

- Es necesario una **mayor dedicación** en los tiempos iniciales del desarrollo. Normalmente el patrón exige al programador desarrollar un mayor número de clases que, en otros entornos de desarrollo, no son necesarias.
- MVC es un **patrón de diseño orientado a objetos** por lo que su **implementación** es sumamente **costosa y difícil** en lenguajes que no siguen este paradigma.

**Es bueno utilizar MVC cuando**

Si ya conoces de MVC y te sientes cómodo con su uso en desarrollo. Si ocurre lo contrario y afecta negativamente a tu productividad, deberías buscar otra opción. Necesitas crear **test unitarios** eficientes para la interfaz de usuario y deseas tener un control total sobre como las URLs serán **formateadas o mostradas en el navegador**.



## Ordenamientos

Es el procedimiento en el cual se **agrupan los registros en orden definido**, con el fin de facilitar la búsqueda de datos ordenados en secuencia, no importa su tipo.

El propósito principal de un ordenamiento es el de facilitar las búsquedas de los miembros del conjunto ordenado y se debe hacer cuando es una cantidad considerable de búsquedas y el tiempo es un factor importante.

Es un ordenamiento eficiente cuando se hacen la menor cantidad de movimientos.

### Algoritmos de ordenamiento Internos

Son aquellos en los que los valores a ordenar están en memoria principal, por lo que se asume que el tiempo que se requiere para acceder cualquier elemento sea el mismo por ejemplo el **quicksort** o el de **burbuja**.

## Patrones de diseño

Son soluciones generales y de eficacia comprobada para problemas comunes de la programación orientada a objetos. Ayudan a que el código sea más fácil de mantener. También se puede decir que son el esqueleto de las soluciones a problemas comunes en el desarrollo de software.

Hay tres tipos de patrones

- **Creacionales:** Resuelven problemas de creación de objetos.

- **Estructurales:** Estructuran el código para que los objetos trabajen juntos, pero sin saber el funcionamiento interno de los otros.
- **De Comportamiento:** Cambian el funcionamiento de un algoritmo en tiempo de ejecución.

Entre los más importantes están:

- **Fabrica Abstracta:** es de tipo **creacional**, define una interfaz para la creación de objetos relacionados o similares.
- **Singleton:** es de tipo **creacional**, una sola instancia global del objeto para todas las solicitudes, su base es un constructor privado y una propiedad de acceso estática.
- **Modelo Vista Controlador (MVC):** es de tipo **creacional**, se basa en la separación de responsabilidades.
- **Adaptador:** es de tipo **estructural**, permite que dos clases con interfaces diferentes o incompatibles trabajen juntas.
- **Puente (Bridge):** es de tipo **estructural**, desacopla una abstracción de su implementación permitiendo modificarlas independientemente.
- **Decorador:** es de tipo **estructural**, añade funcionalidad a un objeto de forma dinámica y sin modificarlo.
- **Recuerdo (Memento):** es de tipo de comportamiento, Almacena el estado de un objeto y lo restaura posteriormente.

Elementos de un patrón de diseño:

- **Nombre.**
- **Problema:** Cuando aplicar el patrón.
- **La solución:** descripción del patrón.
- **Consecuencias:** costos y beneficios

## Programación Orientada a Objetos

Es un paradigma de desarrollo de software en donde el programa se divide en componentes con responsabilidades bien definidas. Se compone de Herencia, acoplamiento y cohesión, ocultamiento y encapsulamiento, abstracción, y polimorfismo.

- ✓ **Abstracción:** Consiste en entender Que es lo que hace una clase, cuáles son sus características esenciales y comportamientos esperados, pero sin saber Cómo lo hace, es decir, sin conocer los detalles de la implementación.
- ✓ **Clase:** Es la definición de los atributos y comportamientos de un objeto.

- ✓ **Objeto:** es la instancia en memoria de una clase, se crea por medio del operador New. Se pueden crear muchos objetos de la misma clase pero inicializados con valores diferentes.

Los atributos estáticos (o **Shared** en VB) no requieren de una instancia de clase para accederse o ejecutarse.

### **Garbage Collector**

En .Net es el encargado de liberar la memoria que ya no se está utilizando. Usa un motor de optimización que decide cual es el momento más adecuado para efectuar la recolección, pero también se puede invocar manualmente usando **System.GC.Collect**.

**Ocultamiento:** Consiste en cambiar la visibilidad y accesibilidad de los atributos o métodos de una clase. Se hace por medio de los modificadores de acceso como Public, Private, Protected o Friend

**Encapsulamiento:** Consiste en impedir la modificación directa de los atributos de la clase. La modificación y acceso solo se pueden hacer a través de métodos o de **Gets y Sets**.

**Public:** El elemento es visible fuera de la clase.

**Protected:** El elemento solo es accesible dentro de la clase y en clases derivadas. Las clases derivadas no pueden acceder elementos privados de la clase base.

**Friend:** El elemento solo es visible dentro del mismo ensamblado (dll).

**Protected Friend:** El elemento solo es visible por clases derivadas en el mismo ensamblado

**Private:** El elemento solo es visible y accesible dentro de la clase.

**Cohesividad:** Es el grado en que una clase está bien definida y con una única responsabilidad. Es bueno que haya alta cohesividad.

**Acoplamiento:** es el grado de interdependencia entre componentes de un sistema, el alto acoplamiento es malo ya que puede causar que cambios en un componente obliguen a cambiar los componentes dependientes. Se debe buscar que haya bajo acoplamiento.

**Herencia:** En VB se utiliza herencia simple, es decir, solo se puede heredar de una clase a la vez, esto evita tener que repetir código en clases muy similares, al heredar, la clase derivada solo debe declarar sus atributos nuevos, una clase derivada puede redefinir miembros de la clase base.

Se puede hacer una referencia explícita a un método o propiedad de la clase base usando **"MyBase"**.

Una clase abstracta no se puede instanciar, solo heredarse.



Si deseamos que una clase no pueda ser heredada, la debemos declarar con el operador **NotInheritable**.

**Polimorfismo:** Es la capacidad que tienen los objetos de tener métodos con nombre y parámetros idénticos, pero con comportamiento diferente dependiendo del objeto. El polimorfismo también se puede lograr a través de interfaces. En .NET se implementa mediante herencia o Interfaces.

### **Tipos de polimorfismo**

**Por medio de la sobrecarga de métodos:** ocurre cuando las funciones del mismo nombre existen, con función similar, en clases que son completamente independientes unas de otras (estas no tienen que ser clases secundarias de la clase objeto).

**Polimorfismo Paramétrico:** es la capacidad para definir varias funciones utilizando el mismo nombre, pero usando parámetros diferentes (nombre y/o tipo). El polimorfismo paramétrico selecciona automáticamente el método correcto a aplicar en función del tipo de datos pasados en el parámetro.

**Polimorfismo de Subtipado:** es poder llamar un método de objeto sin tener que conocer su tipo intrínseco.

**Clase abstracta:** Es aquella que no puede ser instanciada directamente. En C# se declara como “**abstract**” y en Visual Basic como “**MustInherit**”. Aunque las clases abstractas si pueden tener constructor, este solo puede invocar desde sus clases derivadas. Una clase abstracta no se puede instanciar, solo heredarse. Una clase abstracta puede tener propiedades y métodos abstractos que deben ser obligatoriamente sobrescritos. **La clase abstracta se diferencia de las interfaces en que un objeto solo puede heredar de una clase, pero puede implementar muchas interfaces a la vez.**

**Campos o Miembros de clase:** Variables que definen el estado de una clase.

**Propiedad:** Define métodos de acceso para un miembro de clase: Lectura/Escritura, Solo lectura, Solo escritura.

**Constructor:** Inicializa los miembros de la clase. En VB el constructor se declara con de la siguiente manera: **Public Sub New ()**

Una clase puede tener varios constructores, uno por defecto que no recibe parámetros y uno o más constructores que reciban parámetros. Si declaramos un constructor con parámetros el compilador ya no creará uno por defecto.

Un **sub** es un método que no devuelve resultados, no pueden tener **Return**

Una **función** es un método que si devuelve resultados y lleva **Return**.

Se diferencia de las interfaces en que un objeto solo puede heredar de una clase, pero puede implementar muchas interfaces a la vez.

**Método abstracto:** Se declaran como “**abstract**” en C# y “**MusOverride**” en Visual Basic, los métodos abstractos solo se pueden declarar dentro de clases abstractas.

**Protected:** Los atributos privados de una clase no son accesibles por sus clases derivadas, a menos que se declaren como “Protected”, tanto en C# como en Visual Basic.

**Sobrecarga:** es cuando en una clase se tienen métodos con el mismo nombre, pero que reciben parámetros diferentes. La sobrecarga permite que una clase pueda tener varios constructores. Para que la sobrecarga sea válida, la cantidad o tipo de parámetros debe ser diferente. No se pueden declarar dos métodos con el mismo nombre y además con el mismo tipo y cantidad de parámetros.

Dim p1 as Persona = New Persona ()

Lo anterior significa que se está creando un puntero de tipo Persona que apunta a una ubicación de memoria creada para el objeto Persona. Realmente p1 no es nuestro objeto, es el puntero al objeto creado en memoria.

Cuando una función recibe un objeto como parámetro, siempre recibe su puntero esto implica que el objeto siempre corre el riesgo de modificarse.

Si se usa **ByVal**, se pasa una copia del puntero pero no se hace una copia del objeto en memoria

Si se usa **ByRef**, se pasa una referencia al puntero lo cual es ineficiente

Este comportamiento de punteros solo aplica con objetos y no con los tipos que no requieren del operador New, por ejemplo estructuras o tipos básicos como integer y string.

En VB Se puede hacer una referencia explícita a un método o propiedad de la clase base usando “**MyBase**” además se utiliza herencia simple, es decir, solo se puede heredar de una clase a la vez.

**Herencia con sobre escritura:** Una clase derivada puede sobrescribir métodos y propiedades de la clase padre. Para ello se usa la palabra **Overrides**, siempre se ejecutará lo que tenga **Overrides**. Si una clase hija declara un método o propiedad con el mismo nombre que el de la clase base o padre, utilizara la clase del hijo y no la del padre. Esto se conoce como **Shadowing**.

En VB se utilizan los siguientes modificadores de acceso para controlar la visibilidad de los elementos:

- ✓ Public, Private, Protected, Friend, Protected Friend

Por defecto las clases son **Friend** y sus elementos **Public**, esto quiere decir que a menos que se declare la clase explícitamente como Public, esta no será visible fuera del ensamblado.

Se maneja el acrónimo **SOLID** para dar algunas instrucciones de cómo se debería hacer la programación orientada a objetos.

## SOLID

### Principio de responsabilidad única

- ✓ Una clase debe tener una única responsabilidad y ser altamente cohesivas.
- ✓ Se deben dividir clases en clases más pequeñas.
- ✓ La programación en capas favorece este principio.
- ✓ Si una clase tiene demasiados métodos, puede que también tenga más de una responsabilidad y deba separarse en dos o más clases.
- ✓ Igualmente si un método es muy extenso, debe separarse en dos o más métodos.

### Principio de Abierto/Cerrado

- ✓ La idea de este principio es que las clases están abiertas a la extensión pero cerradas a la modificación.
- ✓ Las clases deben poder extender su funcionalidad sin tener que ser modificadas.
- ✓ Se deben declarar sus variables como privadas y usar **getters** y **setters públicos**.
- ✓ Además se debe aprovechar el polimorfismo.

### Principio de sustitución de Liskov

- ✓ Los objetos deberían poder reemplazarse por instancias de sus subtipos sin alterar lo que el programa sea correcto. Esto significa que al implementar una interfaz o heredar un objeto, los métodos no deben generar errores, excepciones, o resultados incorrectos.

### Principio de segregación de la interfaz

- ✓ Muchas interfaces puntuales son mejor que una genérica.
- ✓ No se debe forzar a los objetos a implementar métodos de una interfaz que no necesitan.

- ✓ En lugar de modificar una interfaz para agregarle una nueva funcionalidad, debería extenderse por una nueva interfaz.

### Principio de inversión de dependencias

- ✓ La abstracción no debe depender de los detalles, los detalles deben depender de la abstracción.

### Principios de programación para crear código mantenible

- Se pueden aplicar desde el inicio del desarrollo o durante refactorizaciones
- No se puede pretender escribir código perfecto desde el inicio, las refactorizaciones siempre son necesarias.

Aclarado lo anterior, las siguientes serian buenas prácticas de código mantenible

- **Use nombres significativos:** El nombre debe indicar la intención y no requerir comentarios que explique para qué es la variable o función. No ponga el nombre del contenedor en el nombre.
- Las líneas de código deben tener una **responsabilidad única**.
- Se deben **evitar** los “**scroll**” horizontales y verticales, ya que si un método tiene más de 20 líneas debería revisarse para hacer un método más pequeño y hacer el llamado desde el método inicial.
- Una **función no debe tener más de dos niveles de anidamiento** por ejemplo no tener un if, dentro de un for, dentro de un try.
- **No use valores directamente** como por ejemplo If (monto < 500).
- No haga global algo que es usado solo por una clase.
- **No implemente lo que no va a utilizar, ni lo que nadie le pidió:** no hay que hacer desarrollos partiendo de suposiciones.

## Scrum

Es una **metodología de desarrollo ágil**. Scrum en sí **no es un proceso o técnica**, es un **marco de trabajo** (framework) en el cual se pueden emplear varias técnicas y procesos para el desarrollo de productos complejos. El marco de trabajo consiste en los equipos, eventos y artefactos de Scrum, y las reglas que los relacionan. Scrum permite gestionar el tiempo y entregar productos de manera ágil. Scrum **No es un metodología** de gestión de proyectos y tampoco es un remplazo de las mismas

### Proceso de SCRUM

En cada iteración se realizan tareas de planeación, diseño, desarrollo y pruebas.

Cada iteración produce un incremento al producto, dicho incremento tiene un alto potencial de ser liberado. Scrum es ideal para desarrollar productos complejos además es fácil de aprender, pero difícil de dominar.

## **Algunos roles de la metodología SCRUM**

### **Product Owner**

- Es quien gestiona el Product Backlog o Lista de Producto

La gestión de Product Backlog incluye

- Optimizar el valor del trabajo realizado por el equipo de desarrollo.
- Asegurar que lista muestre claramente lo que el equipo debe hacer.
- Asegurarse de que el equipo de desarrollo entienda los elementos de la lista.
- Es una sola persona, no puede ser un comité.

### **SCRUM MASTER**

- Es el responsable de asegurar que Scrum es entendido y adoptado
- Ayuda a los involucrados a entender Scrum y ayuda al Equipo de Scrum a entender la necesidad de contar con elementos de Lista de Producto claros y concisos.
- Entender y practicar la agilidad.
- Facilita los eventos de Scrum según se requiera o necesite.

### **Servicios del Scrum Master al equipo de desarrollo**

- Guiar al Equipo de Desarrollo en ser auto organizado y multifuncional.
- Ayudar al Equipo de Desarrollo a crear productos de alto valor.
- Eliminar impedimentos para el progreso del Equipo de Desarrollo.

### **Servicios del Scrum Master a la organización**

- Liderar y guiar a la organización en la adopción de Scrum.
- Planificar las implementaciones de Scrum en la organización.
- Ayudar a los empleados e interesados a entender y llevar a cabo.
- Motivar cambios que incrementen la productividad del Equipo Scrum.

### **Equipo de desarrollo**

- Son los profesionales que llevan a cabo los incrementos al producto.
- Son los únicos que pueden llevar a cabo el desarrollo.
- Deben organizar su propio trabajo ni siquiera el Scrum Master debe indicarles cómo convertir historias en incrementos.
- No hay títulos para los miembros del equipo, todos son desarrolladores.
- Los miembros del equipo pueden tener especialidades diferentes, pero la responsabilidad del desarrollo recae en el equipo como un todo.
- El Tamaño óptimo del equipo sería de 4 a 7 personas sin contar al Product Owner ni al Scrum Master.

### **Un Sprint o Iteración**

- Es un bloque de tiempo de 2 a 4 semanas en el que se crea un incremento de producto.
- Al finalizar una iteración comienza de inmediato la siguiente.
- Cada sprint se puede considerar como un proyecto.

### **En una iteración se realizan**

#### **► Reunión de planificación del sprint:**

- En esta reunión se define el trabajo a realizar durante el sprint
  - ¿Qué se puede entregar en el sprint?
  - ¿Cómo se conseguirá hacer el trabajo necesario?
  - Tiene un máximo de duración de 8 horas para un sprint de un mes
  - El Scrum Master se asegura de que esta reunión se lleve a cabo y dentro del tiempo asignado

### **Scrum Diario**

- Es una reunión diaria de aproximadamente 15 minutos.
- En esta reunión el equipo sincroniza sus actividades y planea lo que hará durante las siguientes 24 horas.
- El Scrum diario se debe llevar a cabo siempre a la misma hora y en el mismo lugar.
- El Scrum master se asegura que el equipo realice la reunión y no sobrepasen los 15 minutos, también que solo participen los miembros del equipo de desarrollo.

### **Revisión del sprint**

- ❖ Es una reunión informal para revisar el resultado del sprint.
- ❖ Su duración se restringe a 4 horas para Sprint de un mes.
- ❖ Se discute que hacer a continuación.
- ❖ Se revisa que cambios en el entorno afectan la lista del producto.

### **Retrospectiva del sprint**

- ❖ Se realiza luego de la revisión del sprint y antes de la reunión de planificación.
- ❖ Tiene un máximo de tres horas para Sprint de un mes.
- ❖ El Scrum Master participa como un miembro del equipo ya que la responsabilidad del proceso se Scrum recae sobre él.

### **Propósito**

- ❖ Inspeccionar como fue el sprint en cuanto a personas, herramientas y procesos.
- ❖ Identificar qué cosas se hicieron bien y se deben seguir haciendo.
- ❖ Identificar qué cosas se hicieron mal, y planear como mejorarlas en el siguiente sprint.

### **Herramientas de SCRUM**

- ❖ **Tarjeta de Product Backlog:**
  - Es una tarjeta en donde se escribe una historia de usuario.
  - Las historias de usuario son los requerimientos, o aspectos deseados del producto.
  - En estas tarjetas se agrega el puntaje asignado por el equipo de desarrollo ya sea en esfuerzo o en horas.
  - Estas historias pueden ser actualizadas en cada Sprint.

### **INVEST**

- ✓ Es una técnica para analizar y refinar las historias de usuario.

Las historias se analizan en base a los siguientes principios:

- ✓ **I -Independent (Independiente):** No depende de otra.
- ✓ **N -Negotiable (Negociable):** Se puede reemplazar por otra de diferente prioridad.

- ✓ **V -Valuable (De valor):** Que sea necesaria y de valor para el producto.
- ✓ **E –Estimable:** El equipo se siente tranquilo y seguro al estimarla.
- ✓ **S –Small (Pequeña):** Hace algo simple que funciona, se puede hacer en dos semanas.
- ✓ **T -Testable (Se puede probar):** Se le pueden hacer pruebas.

Si la historia no cumple alguno de los principios, debería replantearse o dividirse en historias más pequeñas.

**Disponibilidad del equipo:** Es la suma de las horas dedicadas por cada miembro del equipo de desarrollo al proyecto. Las horas de las tareas tampoco deben ser iguales a las de la disponibilidad, se debe usar cierto margen de seguridad.

**Velocidad del equipo:** La suma del puntaje asignado a las historias de la iteración representa el esfuerzo total de la iteración. Si el equipo logra completar con éxito una iteración, para la siguiente se podría tratar de aumentar el esfuerzo total, es decir, agregar más historias.

#### **Diferencia entre desarrollo incremental y desarrollo iterativo**

- ✓ En los incrementos se tiene una idea completa del producto final.
- ✓ Al comenzar hay certeza absoluta sobre el resultado final deseado.
- ✓ En las iteraciones se va construyendo un borrador, se valida, y luego se sigue agregando calidad al producto.
- ✓ Al comenzar no hay certeza absoluta sobre el resultado deseado, sino que se va construyendo a medida que se avanza y se va viendo el producto.

#### **TDD y Refactoring**

Hay dos niveles de TDD

##### **Acceptance TDD (ATDD)**

- ✓ Se escribe una sola prueba de aceptación y luego el código que la satisface.
- ✓ También se conoce como BDD (Behavior Driven Development).
- ✓ Su objetivo es describir requerimientos ejecutables detallados.

##### **Developer TDD (o solo TDD)**

- ✓ Se escribe una prueba unitaria y luego el código para hacer que la prueba pase.
- ✓ El objetivo es crear un diseño ejecutable detallado.

##### **Aspectos a considerar sobre TDD**

- ✓ Para que TDD funcione debe emplearse desde el inicio de un proyecto, antes de escribir una sola línea de código.
- ✓ Se requiere tener bien definidos los requisitos de la funcionalidad a realizar.



- ✓ Los criterios de aceptación deben estar bien definidos y contemplar casos de éxito y error (tantos como sea posible).
- ✓ TDD se basa en los principios de SOLID, por lo que si estos no se aplican o no se dominan, puede que no se logre que el código sea robusto y mantenible.

### Refactorización

- ✓ Consiste en modificar código de manera que se eliminen redundancias, malas prácticas y todo aquello que pueda afectar su mantenimiento; sin alterar su funcionalidad.
- ✓ Es un proceso iterativo, el código no queda perfecto a la primera.
- ✓ Antes de refactorizar hay que recordar que ya deben existir las pruebas unitarias.
- ✓ Recordemos la frase: Si algo funciona, no lo toques.
- ✓ TDD y refactoring requieren emplear los principios de SOLID.

### Casos de Uso

- ✓ Es la descripción de un comportamiento que debe tener un sistema en respuesta a una interacción con un actor.
- ✓ El actor inicia la interacción con el sistema para cumplir con un objetivo. El caso de uso describe los posibles escenarios producto de esa interacción e indica cómo se cumple con los objetivos esperados.
- ✓ Los casos de uso no son las opciones del sistema, son lo que el usuario necesita del sistema.

### SQL

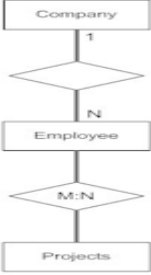
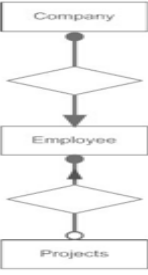
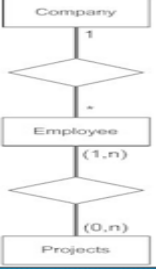
### Entidad relación

- ✓ El modelo Entidad/Relación se usa para el diseño conceptual. Tiene 3 partes fundamentales:
  - **Entidades:** Objetos o ítems de interés. Las entidades representan objetos o ítems de interés. Pueden ser elementos físicos o abstractos. Se representan como cajas con las esquinas redondeadas.
  - **Atributos:** Hechos sobre propiedades de una Entidad. Los Atributos son hechos, aspectos, propiedades o detalles sobre una Entidad. Se representan como círculos u óvalos.
  - **Relaciones:** Enlaces entre Entidades. Las Relaciones son una asociación entre dos o más Entidades. Tienen un nombre, cardinalidad y entidades que

participan en la relación. El nombre se indica en una caja en forma de diamante.

- **Cardinalidad:** Cada Entidad en una Relación puede participar en cero, una o más instancias de la relación. Uno a uno (1:1), Uno a muchos (1: M), Muchos a muchos (M: M) (se saca una tabla intermedia).

Existen varias formas de representar la cardinalidad

Chen Style	Bachman Style	Martin Style
		

Ejemplo entidad relación



## Guías generales

- Las Entidades pueden tener atributos, pero los atributos no tienen partes más pequeñas.
- Las Entidades pueden tener relaciones entre ellas, pero un atributo pertenece a una entidad en particular.

## Optimización de consultas SQL

- Usar Exists en lugar de Count ().
- No utilizar Not Exists.

- Usar joins en lugar de selects anidados.
- No usar Select \*, siempre especificar las columnas.

### ¿Qué es el plan de ejecución?

- Es el resultado de la forma en que el optimizador de consultas planea ejecutar el query solicitado.
- Muestra el orden de las acciones, el costo asociado a cada acción y las operaciones intermedias que se utilizarán.

**Analizar el plan de ejecución:** la mínima instrucción puede generar planes sorpresivamente complejos. Revisar si hay tablas de más involucradas y como podría mejorarse el diseño.

**Joins:** Son instrucciones que permiten combinar registros entre diferentes tablas. Especifican cuales son los criterios para asociar un registro de una tabla con uno o varios registros en otra tabla.

### Los tipos de Join son:

- **Inner Join:** Equivale a hacer una intersección entre dos tablas. Muestra únicamente los registros que cumplen una condición específica en ambas tablas. Los que no cumplen el criterio se excluyen del resultado.
- **Cross Join:** Muestra el producto cartesiano entre dos tablas. Combina todos los registros de la tabla A contra todos los registros de la tabla B.
- **Outer Join:** Un outer join siempre trae los registros de una tabla, aunque no cumplan el criterio de unión en la otra tabla. Las columnas de la tabla que no cumple el criterio se muestran como NULL.

Hay tres tipos de Outer Join:

- **Left Outer Join, o solo Left Join:** Incluye todos los registros de la tabla A y de la tabla B solo los que cumplen el criterio de unión. Cuando el criterio no se cumple, las columnas de la tabla B se muestran con valor NULL.
- **Right Outer Join, o solo Right Join:** El resultado incluye todos los registros de la tabla B (la que está después del Join), y los de la tabla A que cumplan el criterio de unión. Cuando el criterio no se cumple, las columnas de la tabla A se muestran con valor NULL.
- **Full Outer Join, o solo Full Join:** El resultado siempre incluye los registros de ambas tablas. Cuando no se cumple el criterio de unión, se agregan nulos ya sea en la tabla A o en la tabla B

- ✓ Una transacción es una unidad única de trabajo. Si tiene éxito, todas las operaciones realizadas se mantienen, de lo contrario, se regresa la base de datos al estado anterior a la transacción.
- ✓ El termino **Commit** se refiere al acto de persistir todos los cambios realizados.
- ✓ El termino **Rollback** se refiere al acto de deshacer todos los cambios realizados.
- ✓ Algunos términos de Transacciones SQL Server:
  - **Set Xact\_Abort On**: Hace rollback automático de la transacción si ocurre un error durante su ejecución. Si no se utiliza esta opción, se debe manejar manualmente en qué casos llamar a Commit o a Rollback.
  - **Begin Transaction**: Indica el inicio de la transacción. Se puede especificar un alias para la transacción para que no tenga conflicto con otras transacciones.
  - **Commit Transaction**: Indica que se deben persistir los cambios, solo debe llamarse si no hubo errores.
  - **Rollback Transaction**: Indica que se deben deshacer todos los cambios, se debe llamar cuando se detecta un error.

#### ACID

- ✓ **Atomicity**: Una transacción es una unidad única de trabajo.
- ✓ **Consistency**: Todas las operaciones realizadas se mantienen, si se presenta un error, se regresa la base de datos al estado anterior a la transacción.
- ✓ **Isolation**: Los niveles de aislamiento controlan el comportamiento del bloqueo de datos usado en las transacciones, para evitar que se ralentice el sistema y evitar los deadlocks.
- ✓ **Durability**: Cuando se utiliza commit es para hacer que persistan los cambios, el commit se debe utilizar solo cuando no hay errores.

#### Niveles de aislamiento

- ✓ Los niveles de aislamiento controlan el comportamiento del bloqueo de datos usado en las transacciones.
- ✓ En ocasiones, para asegurar la integridad de las operaciones, se pueden bloquear los registros leídos o recién modificados durante el tiempo que dure la transacción, para evitar que alguien los modifique y se produzcan resultados indeseables tales como tuplas fantasma o lecturas sucias.
- ✓ Sin embargo, un bloqueo excesivo de datos puede causar que otras transacciones se bloqueen haciendo muy lento un sistema, o causando que se produzcan Deadlocks en la BD.

**Deadlocks**: Es cuando dos o más procesos quedan detenidos esperando el uno por el otro. Esto suele resolverlo el Microsoft SQL eligiendo cuál de los procesos debe detenerse para que otro termine con normalidad.

### Formas de evitarlos

- ✓ Una seria manteniendo las transacciones abiertas la menor cantidad de tiempo posible.
- ✓ Usando los niveles de aislamiento para evitar los deadlocks o que el sistema se vuelva muy lento.

**BlockLocks:** es cuando dos o más procesos quedan detenidos esperando el uno por el otro pero el Microsoft SQL no detiene ningún proceso solo elige cual debe terminar primero que el otro.

**Bloqueo de datos:** Un bloqueo es aquella demora que se produce en el acceso a un recurso de una base de datos (tablas, índices, etc.), cuando dos procesos “compiten” por tener la exclusividad sobre el mismo. En este caso se produce el famoso “bloqueo” que hace que uno de los procesos deba esperar a que el otro termine de utilizar el recurso, para poder hacer uso del mismo.

Existen dos tipos:

**Compartido o Shared Lock (S):** es el bloqueo que establece cualquier SELECT sobre una tabla (lectura).

**Exclusivo o Exclusive Lock (X):** es el nivel de bloqueo necesario para cualquier instrucción de modificación de data (INSERT/UPDATE/DELETE).

Los tipos de niveles de aislamiento son:

- ✓ **READ UNCOMMITTED:** Se pueden leer cambios que aún no se han confirmado, pueden haber dirty reads.
- ✓ **READ COMMITTED:** No se pueden leer cambios que no se hayan confirmado por otras transacciones. Evita los dirty reads pero no las tuplas fantasma ni las lecturas no repetibles.
- ✓ **REPEATABLE READ:** No se pueden leer cambios sin confirmar, y además los datos leídos se mantienen bloqueados para que otra transacción no los modifique. Evita las lecturas sucias, y las lecturas no repetibles, Pero si puede haber lecturas fantasma.
- ✓ **SNAPSHOT:** Actúa sobre una “instantánea” de los datos, no hace bloqueos.
- ✓ **SERIALIZABLE:** No solo bloquea los datos que lee, sino también el rango de esos datos, de modo que no pueden nidificarse ni se pueden agregar registros dentro de ese rango. Se evitan las lecturas sucias, lecturas no repetibles y las tuplas fantasma.

### ¿Qué se puede hacer con un Group By?

- ✓ Se pueden agrupar registros que tengan valores repetidos, Soporta funciones de agregación como SUM, AVG, COUNT, MIN, MAX, VAR. Se pueden hacer filtros en

base al resultado de las funciones de agrupación por medio de la cláusula having.  
Los resultados se pueden ordenar con Order By

Aspectos a considerar:

- ✓ La cláusula Where se ejecuta antes que Group BY.
- ✓ Las funciones de agregación se ejecutan luego de que se han obtenido y agrupado los registros, por lo que no se pueden referenciar en la cláusula Where.
- ✓ La cláusula Having se aplica luego del agrupamiento, por lo que si pueden emplear funciones de agregación.
- ✓ El orden correcto de declaración es: Select, From, Where, Group By, Having
- ✓ El orden de ejecución es: From, Where, Group By, Having, Select

### Índices en SQL Server

- ✓ Es una estructura de datos que mejora la velocidad de las operaciones, permitiendo un rápido acceso a los registros de una tabla en una base de datos.
- ✓ Al aumentar drásticamente la velocidad de acceso, se suelen usar, sobre aquellos campos sobre los cuales se hacen búsquedas frecuentes.

Una tabla o una vista, puede contener los siguientes tipos de índices:

- ✓ **Agrupado o clustered**
  - Los índices clúster ordenan y almacenan las filas de los datos de la tabla o vista de acuerdo con los valores de la clave del índice.
  - Solo puede haber un índice clúster por cada tabla, porque las filas de datos solo pueden estar ordenadas de una forma.
  - La única ocasión en la que las filas de datos de una tabla están ordenadas es cuando la tabla contiene un índice clúster.
  - Cuando una tabla tiene un índice clúster, la tabla se denomina tabla agrupada.
  - Si una tabla no tiene un índice clúster, sus filas de datos están almacenadas en una estructura sin ordenar denominada montón.
- ✓ **No agrupado o no clustered**
  - Los índices no clúster tienen una estructura separada de las filas de datos.
  - Un índice no clúster contiene los valores de clave de índice no clúster y cada entrada de valor de clave tiene un puntero a la fila de datos que contiene el valor clave.
  - Puede contener tantos índices no agrupados se requieran.

### VISUAL BASIC

### Expresiones Lambda

- ✓ Funciones o rutinas sin nombre e **inline** (funciones que, al **compilar**, no son llamadas en el **código objeto**, sino *insertadas* en la sección del código donde se las llame).
- ✓ Se pueden usar en lugar de un **delegado** (declaración de una función pero sin implementación, se puede utilizar luego como un tipo).
- ✓ Usarlas permite el ahorro de tener que escribir una función con nombre y luego asignarla a un evento, o enviarla como argumento de una función.
- ✓ También ayuda a evitar escribir funciones de una sola línea de código y que solo se utilizan una única vez.
- ✓ Se declaran con la palabra **Function** cuando debe retornar un valor, o con **Sub** cuando son void. Si se declaran en una sola línea se pueden omitir el **End Function** o **End Sub** al final y si son **inline** tampoco usan **Return**.
- ✓ No aceptan parámetros **Optional** o **Paramarray**. Un delegado hace match de forma automática con cualquier método que reciba los mismos parámetros en cantidad, orden y tipo, y retorne el mismo tipo de datos.
- ✓ La declaración de la función Lambda debe tener la misma cantidad de argumentos que el delegado y debe retornar una expresión del mismo tipo. No es necesario declarar el tipo de los argumentos puesto que se infiere del delegado.

### Declaración Inline de delegados

- ✓ Cuando se debe retornar un valor se utiliza la palabra **Func**, cuando es void se utiliza la palabra **Action (Action (Of T))**, no declaran nombres de argumentos, solo sus tipos.
- ✓ Ambos aceptan parámetros genéricos.

Ejemplos:

- *Action (Of String, Integer).*
- *Func (Of Integer, **Boolean**) el último argumento indica el tipo de retorno.*

### Delegados en Colecciones de datos

- **Comparison (Of T)**: Se utiliza en el método *Sort* para comparar elementos.
- **Predicate (Of T)**: Se utiliza para verificar que un objeto cumpla ciertos criterios. Se utiliza en varios métodos.
- **Converter (Of Tinput, TOutput)**: Convierte un objeto de un tipo a otro. Se utiliza en el método *ConvertAll* de la clase *List*.

### Lambda en Entity Framework

- Las expresiones lambda se pueden aprovechar en Entity Framework para realizar consultas con la cláusula *Where* y para proyectar datos.

### Lambda en MVC

- Lambda se utiliza en varios métodos helpers de Razor como por ejemplo `LabelFor` y `TextboxFor`.

### Programación Concurrente en VB.NET

#### TPL

- ✓ Task Parallel Library, API impulsado por Microsoft para ejecutar código paralelo, multi-hilo y asíncrono.
- ✓ Simplifica los procesos de agregar paralelismo y concurrencia de aplicaciones.
- ✓ Sus clases y APIs están contenidos en los ensamblados **System.Threading** y **System.Threading.Tasks**

#### Características:

- Usa los procesadores disponibles de forma más eficiente
- Usa un pool de hilos (ThreadPool)
- Tiene soporte para cancelar procesos
- Maneja el particionamiento del trabajo

#### Paralelismo de Datos

- ✓ Ejecución de acciones concurrentes sobre colecciones de datos.
- ✓ Usa la clase **System.Threading.Tasks.Parallel** que incluye los métodos **For** y **ForEach**.
- ✓ Cuando se invoca a uno de estos métodos, TPL particiona los datos de modo que los métodos puedan operar sobre ellos de manera concurrente.
- ✓ El calendarizador de tareas distribuye el trabajo en hilos y nivela las cargas de ser necesario.
  - **Parallel.For:**
    - Es similar al `For` que se utiliza normalmente con la diferencia de que aparte de recibir un índice inicial y uno final, se debe indicar la acción a ejecutar.
  - **Parallel.ForEach**
    - Sirve para realizar tareas en paralelo sobre estructuras que implementen la interfaz `IEnumerable` o `IEnumerable(Of T)`.
    - En su forma más básica recibe como primer argumento la estructura de datos y luego la acción.



- La acción típicamente es una expresión Lambda, o una expresión lambda que invoca a otro método.
- **Parallel.Invoke:**
  - Recibe como argumentos una o más funciones a ejecutar.
- **Task y Task(Of T)**
  - Representan tareas que se ejecutan de forma asíncrona, con la diferencia de que el primero es void y el segundo tiene un valor de retorno del tipo de T.
  - El objeto **Task** permite iniciar tareas asíncronas de tres maneras:
    - ▶ **Task.Run:** Crea, ejecuta y retorna un task.
    - ▶ **Task.Factory.StartNew:** Crea, ejecuta y retorna un task que puede calendarizarse.
    - ▶ **Task.Start:** Inicia la ejecución de un task previamente creado.
- **Async y Await**
  - Son la base de la programación asíncrona.
  - Cuando se utilizan métodos asíncronos no se bloquea el UI mientras se ejecuta una tarea que consuma mucho tiempo o recursos.
  - El flujo de la aplicación tampoco se interrumpe.
  - Los métodos asíncronos se definen con la palabra **Async** y al invocarse se antecede la palabra **Await**.
  - El tipo de retorno de un método asíncrono es un objeto de tipo **Task** o **Task (Of T)**.
  - Si se marca un método como Async permite utilizar el operador Await en su interior.
  - **Await** se utiliza antes de ejecutar una instrucción o método y hace que se suspenda la ejecución del método Async actual hasta que la operación finalice.
  - También se puede usar para ejecutar expresiones lambda u otros métodos que puedan tardar mucho en ejecutarse.
  - No es obligatorio invocar a los métodos Async usando Await, esto solo se hace cuando se quiere esperar a que finalice el método asíncrono.
  - Al invocar un método Async se puede guardar el Task que retorna, o usar Await para esperar y obtener el valor de retorno final.

## Manejo de excepciones

- ✓ Las excepciones de métodos invocados con Await se pueden manejar con un Try/Catch.

- ✓ Si en un Task ocurre un error, Await va a devolver ese error, sin embargo Await no puede ir dentro de un bloque Catch o Finally, únicamente se puede declarar en el cuerpo del bloque Try.

### **Malas prácticas y errores comunes**

- ▶ **Paralelizar tareas pequeñas o con pocos datos**
  - En este caso el costo de manejar hilos hace que la solución sea menos eficiente que una síncrona o secuencial.
- ▶ **Paralelizar en desorden**
  - El Parallel.For es más eficiente cuando se ejecuta de forma ascendente e incrementa de uno en uno que cuando se cambia a descendente o se incrementa de dos en dos o en pasos mayores.
- ▶ **Ejecutar muchos task void o no esperar sus resultados**
  - Ejecutar task sin esperar un resultado puede causar que se pierda el control de los procesos que se ejecutan y el momento en que finalizan.
  - Es recomendable esperar en algún punto a que retornen un valor, o confirmar su finalización exitosa.

### **Inyección de Dependencias**

#### **Inversión de Control (IoC)**

- ✓ Es cuando se delega el control o flujo de trabajo a otro componente.
- ✓ Los objetos consumidores no crean los objetos consumidos, estos son creados por un contenedor.

#### **Inyección de Dependencias (DI)**

- ✓ Una clase se encarga de inicializar o inyectar las dependencias de otra.
- ✓ Se utiliza para desacoplar componentes.
- ✓ En .Net se logra a través del uso de Interfaces.
- ✓ Es una forma de IoC.
- ✓ Para que una clase sea inyectable, debe tener un constructor que reciba sus dependencias. El problema es que la clase consumidora debe saber las dependencias de la clase a invocar para poder inyectarlas, a menos que se utilice un contenedor de IoC.

#### **Contenedor de IoC**

- ✓ Es un componente especializado en la creación de instancias de objetos.
- ✓ También administra el ciclo de vida de las instancias creadas.

- ✓ Se basa en los principios de IoC y DI.

Cuando se utiliza inyección de dependencias se puede utilizar también un contenedor de IoC para que se encargue de inicializar las dependencias de las clases. De esta manera la clase consumidora no necesita saber ni inicializar las dependencias de la clase consumida, puesto que esto se le delega al contenedor de IoC

### **Unity**

- ✓ Es un contenedor de IoC desarrollado por Microsoft Patterns and Practices.
- ✓ Las dependencias se pueden especificar en tiempo de ejecución, o en un archivo de configuración.
- ✓ En MVC se debe instalar el paquete Unity.Mvc(C#) o Unity.Mvc.Vb desde Nuget, esto crea el archivo UnityConfig, dentro de la carpeta App\_Start, en el que se debe agregar la configuración de dependencias.
- ✓ Si se hacen pruebas unitarias al proyecto MVC también se debe instalar Unity.Mvc en el proyecto de pruebas, aunque este no sea un proyecto MVC.
- ✓ Unity puede configurarse de forma simple para resolver la instanciación de dependencias.
- ✓ Otras configuraciones más avanzadas permiten que cada instancia se resuelva como un Singleton.
- ✓ También permite definir métodos explícitos para crear instancias que requieran de argumentos adicionales en su constructor.

### **Notas finales**

- El repositorio Mock bien puede no hacer nada, o podría conectarse a una BD de pruebas usando entity, Ado.Net, u algún otro medio.
- Aprovechando que la configuración se hace mediante .configs, podemos cambiar la instancia del repositorio en cualquier momento sin tener que recompilar la aplicación.
  - Esto nos permitiría tener una configuración para desarrollo y otra para reléase.
  - Inclusive podríamos agregar una dll nueva con otra implementación del repositorio y modificar el .config para utilizarla.
- Unity también puede usarse en cualquier parte del código para obtener una instancia de la interfaz mediante el método Resolve de la clase UnityContainer.
- Si no queremos que las dependencias se inyecten en el constructor, podemos usar siempre el método Resolve en cada método que utilice la dependencia.

### **Pila**

- ✓ Este tipo de estructuras son utilizadas cuando se quiere recordar una secuencia de acciones u objetos en el orden inverso del ocurrido.
- ✓ Son estructuras LIFO (Last In, First Out). Último en entrar, primero en salir.

### Cola

- ✓ Una cola es una colección de elementos homogéneos.
- ✓ En la misma se pueden insertar elementos por uno de los extremos, llamado frente, y retirar los mismos por el otro extremo, denominado final.
- ✓ Son estructuras FIFO (First In, First Out). Primero en entrar, primero en salir.

### ¿Qué son micro servicios?

- ✓ Son un estilo arquitectural en el cual los sistemas se dividen en servicios pequeños.
- ✓ Cada servicio es independiente y se ejecuta en su propio proceso separado de los demás.
- ✓ Entre sus características están
  - Componentes vía servicios.
  - Productos en lugar de proyectos.
  - Endpoints inteligentes.
  - Diseño tolerante al fallo.
  - Diseño evolutivo.
- ✓ Al trabajar con micro servicios es común (pero no obligatorio) usar APIs basadas en HTTP y REST debido a que se simplifica y estandariza la forma de comunicación.

### Verbos de HTTP

- **Post:** crear nuevos recursos.
- **Get:** consultar recursos, se pueden enviar parámetros adicionales de paginación.
- **Put:** actualizar o crear un recurso si el id no existe, se envían todos los campos.
- **Patch:** actualizar, solo envían los campos modificados.
- **Delete:** eliminar.

Las solicitudes HTTP también tienen códigos de respuesta estandarizados

- **200:** Ok
- **201:** Created
- **204:** No Content
- **400:** Invalid Request
- **404:** Not Found

- **405:** Method Not Allowed
- **409:** Conflict
- **500:** Internal Server Error

### Serialización

- Es la capacidad de convertir la información en un conjunto de bytes para su envío o transporte.

Para realizar un **debug** en un servidor remoto se deben seguir algunos **pasos**:

- Primero se debe instalar un tool en visual studio.
- Luego configurar el firewall para que permita la comunicación.
- Activar el modo debug tanto en visual como en el servidor remoto.
- Configurar el visual studio para conectarnos al servidor.

### Función del bloque Using

- ✓ Se utiliza para liberar recursos no manejados, con el fin de que estén nuevamente disponibles para su uso.
- ✓ Fuerza a que se liberen de inmediato en lugar de esperar al recolector de basura.
- ✓ Para esto necesita que el objeto que consume los recursos no manejados implemente la interfaz IDisposable, esta interfaz garantiza que el objeto tenga una implementación del método **Dispose** (Se encarga de liberar recursos).
- ✓ El método Dispose es invocado automáticamente al finalizar el bloque using.
- ✓ Aunque el bloque using capture el error, no lo notifica.
- ✓ Es como usar un bloque Try sin definir el Catch.
- ✓ El bloque using también define un alcance o scope, lo que significa que las variables declaradas en su interior dejarán de existir al finalizar el bloque.
- ✓ Se debe incluir la menor cantidad de código posible, solamente lo necesario.

### GAC Global Assembly Cache

- ✓ Como su nombre lo indica es un cache de DLLs que se utiliza por las aplicaciones de .NET.
- ✓ Las DLLs del GAC son compartidas por varias aplicaciones.
- ✓ Registrar DLLs en el GAC solo debe hacer cuando es estrictamente necesario compartirlas entre varias aplicaciones.

- ✓ El GAC puede tener muchas copias de una DLL pero con diferente versión.

### **Registrar DLLs en el GAC**

- ✓ Para registrar una DLL en el GAC se utiliza el programa gacutil en la línea de comandos (usando la ventana de comandos de VS).
- ✓ El primer argumento es '-i' seguido de un espacio y de la ruta completa del ensamblado a registrar.

### **Eliminar DLLs del GAC**

- ✓ Para eliminar una dll del gac se utiliza el programa gacutil desde la línea de comandos (usando la ventana de comandos de VS).
- ✓ El primer argumento es '-u' seguido del nombre del ensamblado.
- ✓ No es necesario indicar la ruta completa del ensamblado, solo su nombre.

### **Para ver contenido del GAC**

- ✓ Para ver las DLLs registradas en el gac se utiliza el programa gacutil y una L minúscula:
  - Gacutil -l
- ✓ Este comando lista todas las DLLs del GAC

Para eliminar el caché de ASP.NET se debe reiniciar el servidor de IIS (o el pool de la app) y proceder a borrar los archivos del cache.

### **Pruebas de estrés.**

#### **Características de funciones testeables**

- ▶ Tienen una única y clara responsabilidad
- ▶ Retornan algo

### **Pruebas de carga VS Pruebas de Estrés**

- **Prueba de carga:** es aquella en que una funcionalidad se somete a cierta carga esperada de datos con el fin de determinar su comportamiento (tiempos de respuesta, tamaño de la respuesta)
- **Prueba de estrés:** es cuando una funcionalidad se somete a cargas altas o anormales, usualmente en escenarios de concurrencia, con el fin de encontrar el punto en que falla
- **LoadTest:** de Visual Studio permite hacer ambas cosas
  - Se puede hacer una prueba unitaria que representa la prueba de carga y un **LoadTest** con un usuario y una única iteración
  - Se puede hacer una prueba unitaria que realice una carga y un **LoadTest** con muchos usuarios concurrentes realizando varias solicitudes con esa carga
- El valor de **sample rate** determina que tan seguido se recolectan los contadores de desempeño, en pruebas de corta duración se usan valores bajos, que indican que se deben recolectar contadores más a menudo.
- Los valores bajos requieren de más espacio en la base de datos de resultados de la prueba, por lo que para pruebas largas se recomienda usar un **sample rate** mayor.
- Una prueba se puede configurar para ejecutarse cierta cantidad de veces (iteraciones), o por tiempo de duración.
- En el asistente podemos especificar un tiempo de duración de la prueba.
- Si además indicamos el tiempo de calentamiento, no se recogerán datos hasta que haya pasado el tiempo de calentamiento.
- El tiempo de calentamiento se usa cuando queremos excluir de la prueba los tiempos de inicialización del servicio ya que estos podrían sesgar la prueba.
- Dependiendo de lo que queramos probar puede que sea relevante incluir los tiempos de inicialización o que sea mejor excluirlos.
- El perfil de tiempo de pensar se utiliza para simular los tiempos de espera o de reacción que utiliza un usuario antes de seleccionar una acción.
- Una carga constante significa que todos los usuarios están activos desde el inicio de la prueba. Una carga en pasos significa que la cantidad de usuarios aumenta poco a poco hasta llegar a un máximo que se mantendrá hasta el final de la prueba.
- En una prueba de carga se pueden agregar una o varias pruebas unitarias o web test.

## Test mix Model

- **Basado en el total de pruebas**
  - Cada vez que un usuario ejecuta una prueba se toma como un iteración, si hay 25 a la vez, se toma como 25 iteraciones.
  - Esto influye cuando se ha indicado una cantidad máxima de iteraciones.

- **Basado en número de usuarios**
  - Cada vez que **todos** los usuarios ejecutan la prueba se toma como una iteración.
- **Basado en ritmo de usuario**
  - Cada usuario ejecuta 'n' cantidad de veces una prueba por hora.
- **Basado en orden secuencial**
  - Cada usuario ejecuta las pruebas en el orden en que fueron definidas en el escenario, el usuario vuelve a repetir esas pruebas hasta que el tiempo definido se haya agotado.

### Network Mix Model

- Podemos indicar como simular la conexión a internet.
- Se puede usar un solo modelo, o combinar y distribuir varios modelos según el escenario que se desee probar.

Los **counter sets** permiten simular la máquina que queremos probar, en este caso, un servidor de aplicaciones web.

Los contadores a recolectar para un servidor de aplicaciones son: ASP.NET, IIS y .NET

### Excepciones

- ✓ Una excepción es cualquier condición de error o comportamiento inesperado que encuentra un programa en ejecución.
- ✓ En .NET hay una tabla de manejo de información sobre excepciones para cada aplicación.
- ✓ En .NET se representan con el objeto **Exception**. SystemException hereda de Exception y es la base de todas las demás excepciones, que se debe declarar de último en caso de usarse, sus propiedades son:
  - **Message**: Proporciona información sobre la causa del error.
  - **StackTrace**: Seguimiento de pila del origen del error. Contiene el archivo, clase, método y línea donde ocurrió el error.
  - **InnerException**: Se utiliza para anidar excepciones, por ejemplo, cuando se arroja una excepción personalizada y se quiere conservar la información de la original.
- ✓ **Data**: Es un diccionario que puede contener datos adicionales sobre la excepción.
- ✓ Otras jerarquías de excepciones heredan de SystemException, y así sucesivamente.



- ✓ Para manejar las excepciones y permitir el continuo funcionamiento de la aplicación o la finalización agraciada, se utiliza el Bloque Try/Catch/Finally.
- ✓ **Try:** Contiene el código manejado. Además, contiene los bloques Catch y Finally.
- ✓ El bloque Try puede tener uno o varios catch para dar un trato especial a diferentes excepciones.
- ✓ **Catch:** Atrapa y maneja la excepción. Puede haber varios o ninguno en un bloque Try.
- ✓ **Finally:** Opcional. Se ejecuta ocurra o no un error. Se utiliza para liberar recursos.

### Reenvío de excepciones

- ✓ **Throw:** mantiene el StackTrace sin modificaciones, se utiliza cuando se quiere loguear la excepción, pero sin detenerla e indicando el StackTrace original.
- ✓ **Throw ex:** sobrescribe el StackTrace. Nuevamente se utiliza para loguear una excepción sin detenerla, pero reiniciando el StackTrace de manera que se oculta la línea original donde ocurrió el error.
- ✓ **Throw New [Exception]:** Reemplaza por completo la excepción original con una nueva.
- ✓ La excepción original se puede conservar en el **InnerException** de la nueva excepción.
- ✓ Se utiliza para arrojar excepciones personalizadas, o reenviar una excepción, pero con mensaje diferente.

### Errores comunes al usar el manejo de excepciones

- ✓ Revelar demasiada información.
- ✓ No hay que mostrarle al cliente un error de la forma `ex.ToString ()` ya que se revelan detalles de la implementación como el StackTrace, el nombre de clases, librerías y métodos.
- ✓ El StackTrace completo solo debe mostrarse o guardarse para efectos de soporte.
- ✓ Querer atrapar todas las excepciones.
- ✓ Poner un Try/Catch en cada método tiene un costo a nivel de rendimiento, en especial si se hace repetidamente en ciclo For o While.
- ✓ Dejar el Catch vacío o Anidar un bloque Try/Catch en un ciclo.

### Postback

- ✓ Propiedad que obtiene un valor que indica si la página se carga por primera vez o se carga en respuesta a una devolución de datos.
- ✓ Es una comunicación entre cliente servidor.

- ✓ Se valida generalmente cuando se ejecuta la acción de algún evento, ya sea el Click de un botón, o el cambio de selección de un combobox.
- ✓ Se valida con if (! IsPostBack) en c# y If Not IsPostBack en VB.

### **Razor**

- ✓ Tipo de Vista de MVC.
- ✓ Es una forma que se tiene para escribir el código de las vistas.
- ✓ Se utiliza para mezclar código de servidor y cliente en una vista.
- ✓ El Razor es código server.

### **Herencia**

- ✓ Una clase puede heredar de otra usando la palabra "Inherits" en la siguiente línea.
- ✓ Si se quiere que una clase no se pueda heredar se debe decorar con la palabra NotInheritable.
- ✓ En VB una clase solo puede extender una clase a la vez y no está obligada a sobrescribir todos los métodos.
- ✓ Una clase abstracta no se puede instanciar.
- ✓ Una clase puede implementar varias interfaces, y en cada caso está obligada a implementar todos los métodos de la interfaz.

### **ForEach, Next**

- ✓ Se utiliza para recorrer colecciones que implementen la interfaz IEnumerable.

### **Strings**

- ✓ Se pueden concatenar Strings usando el operador &.
- ✓ Cuando un string se modifica, realmente se destruye el objeto original y se reemplaza por el nuevo valor.
- ✓ Cuando un string se debe modificar muchas veces se recomienda usar StringBuilder, en especial cuando se realizan muchas concatenaciones.
- ✓ El objeto string también posee los siguientes métodos utilitarios:
  - **ToUpper**: convierte todo el texto a mayúscula, devuelve el string modificado.
  - **ToLower**: convierte todo el texto a minúsculas, devuelve el string modificado
  - **Trim**: Elimina los espacios en blanco al inicio y final del texto, devuelve el string modificado.
  - **Replace**: reemplaza caracteres o secuencias, devuelve el string modificado.
  - **Split**: Convierte el string en un array de strings, utiliza uno o varios caracteres como separadores.
  - **Join**: Une los valores de un array de strings en un solo string, se puede indicar un separador.
  - **IndexOf**: devuelve el índice del valor especificado.

- **Length:** devuelve el tamaño del string.

### Clase genérica

- ✓ Es una clase que usa el polimorfismo para permitir la instanciación con varios tipos de datos y una vez instanciada, da seguridad de tipo.
- ✓ Se declaran con la instrucción: (Of T), T puede ser cualquier nombre, y pueden ser varios tipos separados por coma, Una vez que se define el tipo de dato en la clase genérica, este no puede cambiar, por lo que la clase se vuelve fuertemente tipada.

### WCF

- ✓ Viene de las siglas Windows Communication Foundation.
- ✓ Es un framework para construir aplicaciones orientadas a servicios.
- ✓ Soporta patrones de mensajería como request/reply, dúplex y one-way.
- ✓ Los mensajes se pueden encriptar y se puede solicitar que el usuario se autentique para recibirlos.
- ✓ Soporta transacciones tanto atómicas como distribuidas. Soporta Ajax y REST.

### Ejemplo de WCF completo

```
<bindings>
  <wsHttpBinding>
    <binding name="WS_Bccr" maxReceivedMessageSize="2147483646">
      <security mode="Message">
        <message clientCredentialType="Certificate"
          negotiateServiceCredential="False" /> Que el cliente no necesita una un certificado
          de servicio.
          algorithmSuite="Default" define el algoritmo de encapsulamiento del mensaje.
          establishSecurityContext ="False"/> Valor booleano que determina si el canal de
          seguridad establece una sesión segura
          negotiateServiceCredential = "True"/> Valor booleano que determina si el
          credencial del servicio es proporcionado por el cliente o si es obtenido por el servicio a
          través de un proceso de negociación.
        </security>
      </binding>
    </wsHttpBinding>
  </bindings>
</behaviors>
```

```

<serviceBehaviors>
  <behavior name =" BccrServidor">
    <serviceCredentials findValue="CN=SRV, OU=SINPE, O=BCCR, L=SJ,C=CR"
      storeLocation="LocationMachine"
      storeName="My"
      x509FindType="FindBySubjectDistinguishedName">
      <clientCertificate>
        <authentication certificateValidationMode="Custom" Especifica el nivel de
confiabilidad usado para la autenticación por certificado. Si es custom, deberá setear la
etiqueta customCertificateValidaroType

        customCertificateValidaroType="Bccr.Seuridad.Validador" Atributo a un
ensamblador y tipo usado para validar el certificado
        revocationMode="NoCheck" Especifica como los certificados son
chaqueados por revocación: Online (indica que el certificado será chequeado
automáticamente por revocación)

        includeWindowsGroups="False" /> Valor booleano que especifica si el
sistema incluye grupos de Windows en el contexto de seguridad
      </clientCertificate>
    </serviceCredentials>
    <serviceMetadata httpGetEnabled ="true" />
    <serviceDebug includeExceptionDetailInFaults="True"/> Es utilizada para habilitar a los
servicios de WCF el envío de información detallada cuando una excepción ocurre en objetos
SOAP fault hacia el cliente. El propósito de esta acción, es brindarle al cliente más
información en pruebas de la aplicación.

  </behavior>
</serviceBehaviors>
</behaviors>
<services>
  <service name="Bccr.Sinpe.Servicio" behaviorConfiguration="BccrServidor">
    <endpoint name ="WSHttpBindinf_ITransferencia"
      contract="Bccr.Sinpe.Srv"
      binding="wsHttpBinding"

      bindingConfiguration="WS_Bccr" />
  </service>
</services>

```

### **IncludeExceptionDetailsInFaults**

- ✓ Si sería capaz, puesto que **IncludeExceptionDetailsInFaults** muestra los mensajes de error detallados al cliente para conocer y depurar el error.
- ✓ Este atributo solo debe de ser activado cuando el servicio se encuentre en desarrollo, y desactivada, cuando el servicio se lance a producción.

Ejemplo de error en el tipo de error guardado en diferentes partes:

myTrace.TraceEvent ([TraceEventType.Error](#), 0, "Error al obtener datos")

El problema es que el mensaje no aparece en el archivo de tracing. Analice la siguiente configuración e indique cual es el error y como corregirlo

```
<source name="System.ServiceModel.MessageLogging"
  switchValue="Critical, ActivityTracing">
  <listeners>
    <add name="log" type="System.Diagnostics.XmlWriterTraceListener"
      initializeData="Traces.svclog" />
  </listeners>
</source>
```

- ✓ El problema se debe a que a nivel de código se están traceando los eventos de tipo Error y en el web.config en el value de la propiedad del switchValue, se encuentra el value de Critical, lo cual hace que el listener solo escuche y guarde los eventos de este tipo.
- ✓ Se resolvería, cambiando el value del switchValue por Error, para que el listener escuche este tipo de eventos y los almacene en el archivo especificado por el log

```

1  <?xml version="1.0" encoding="utf-8" ?>
2
3  <configuration>
4    <system.diagnostics>
5      <sources>
6        <source name="System.ServiceModel"
7              switchValue="Information, ActivityTracing"
8              propagateActivity="true">
9          <listeners>
10             <add name="xml" />
11          </listeners>
12        </source>
13      </sources>
14      <switches>
15        <add name="Activated" value="1" />
16      </switches>
17      <trace autoflush="true">
18        <listeners>
19          <add name="TextWriterTraceListener"
20                type="System.Diagnostics.TextWriterTraceListener" initializeData="C:\log.txt"/>
21        </listeners>
22      </trace>
23    </system.diagnostics>
24  </configuration>
25

```

La línea 15 se encarga de activar el trace de eventos producidos por el sistema.

La línea 17 se encarga de activar el auto limpiado del buffer de eventos.

La línea 20 se encarga de escribir el archivo log.txt, los eventos de tipo información que se disparan en el sistema y pasen por el trace y los listener.

```

Imports System.ServiceModel
Imports System.ServiceModel.Web

```

```

<ServiceContract()> _
Public Interface IService1

```

```

    <OperationContract()> _ |
    Function GetProducts(ByVal categoryId As String) As List(Of Products)

```

```
End Interface
```

## Operation Contract

- ✓ Expone al mundo las operaciones o métodos disponibles en el servicio, para poder ser consumidos por los clientes.

## Mensajes y EndPoints

- ✓ La comunicación entre clientes y servicios se realiza por medio de mensajes.

## ENDPOINT

- ✓ Es el lugar donde el mensaje es enviado o recibido.
- ✓ Punto de acceso o de entrada que un servicio expone para ser consumido por el mundo externo.

Los componentes básicos de un endpoint son:

- ✓ **Address:** URL del servicio, indica donde se recibe el mensaje.
- ✓ **Binding:** define como se comunica el endpoint, incluye el transporte (HTTP, TCT) y el encoding (texto, binario), y la seguridad (SSL).
- ✓ **Contract:** Identifica las operaciones disponibles.

## Otros componentes de wcf

- ✓ **Behaviors:** Especifican comportamientos adicionales de la implementación del endpoint.
- ✓ **Client:** Atributo que configura al cliente que se comunica o intercambia mensajes con uno o más EndPoint.

## Tipos de Bindings

- ✓ **BasicHttpBinding:** Utiliza HTTP y SOAP 1.1, sirve para comunicarse con servicios ASMX y otras basados en WS-I Basic Profile 1.1. Por defecto la seguridad está apagada y usa Text encoding.
- ✓ **WSHttpBinding:** Es seguro, interoperable y tiene soporte para transacciones. Se puede comunicar con servicios basados en los protocolos WS-\*.
- ✓ **WSDualHttpBinding:** Seguro e interoperable con soporte para comunicación Dúplex
- ✓ **NetTcpBinding:** Es seguro y optimizado para comunicación entre máquinas de la misma Intranet.

Los mensajes se envían mediante protocolos de transporte tales como HTTP o TCP además se puede codificar como Texto, MTOM y Binario.

Los Bindings permiten configurar aspectos tales como:

- Tamaño de los mensajes
- Tiempos de espera
- Formato del mensaje
- Tipo de seguridad

Por ejemplo, se quiere implementar un servicio que no se comunica con clientes externos.

Únicamente se solicita que el cliente tenga implementado tecnologías del tipo Microsoft .Net. Además se quiere garantizar que el traspaso de mensajes se realice de la manera más

**rápida posible** y se desea que la información transmitida **permanezca íntegra, segura y confidencial.**

- Para resolver el caso anterior debemos utilizar el protocolo NetTcpBinding y el modo de seguridad a utilizar sería el Transport, ya que este encripta todo el canal de comunicación incluyendo el mensaje, cumpliendo con lo establecido de que la información se transmita íntegra, segura y confidencial.

#### **Manejo de instancia en WCF**

- **InstanceContextMode.PerCall:** Una instancia del servicio por cada llamado. Usar cuando no se ocupe compartir estados entre llamados.
- **InstanceContextMode.PerSession:** Una instancia del servicio por sesión para mantener el estado entre llamados.
- **InstanceContextMode.Single:** Una instancia del servicio para todas las llamadas. Usar cuando se necesite compartir información global.

#### **Manejo de concurrencia**

- **ConcurrencyContextMode.Multiple:** Cada instancia del servicio puede tener muchos hilos (threads), y procesar muchos mensajes a la vez.
- **ConcurrencyContextMode.Single:** Cada instancia solo puede tener un hilo y procesar un mensaje a la vez (más seguro).

#### **Tipos de sesión en el IIS**

- **Modo InProc:** almacena el estado de sesión en memoria en el servidor web. Éste es el valor predeterminado.
- **Modo StateServer:** almacena el estado de sesión en un proceso distinto denominado "servicio de estado de ASP.NET". Este modo garantiza que el estado de sesión se mantiene si se reinicia la aplicación Web y que esté disponible también para varios servidores web en una granja de servidores web.
- **Modo SQLServer:** almacena el estado de sesión en una base de datos de SQL Server. Este modo garantiza que el estado de sesión se mantiene si se reinicia la aplicación Web y que esté disponible también para varios servidores Web en una granja de servidores Web.

#### **Behavior**

- En el web.config vamos a configurar un service Behavior para limitar el acceso al servicio.



- Cuando declaremos un behavior no olvidemos la sección de publicación de metadatos: `<serviceMetadata httpGetEnabled="true" />`.
- La sección de Bindings al igual que la de Behaviors, va dentro del elemento `system.serviceModel` en el `web.config`.
- Para configurar un endpoint debemos agregar la sección `services` en el `web.config`, dentro la sección `system.serviceModel`.
- Nuestro service behavior debe ir en la sección `<service>` y la configuración del Binding en la sección `<endpoint>`.
- Para probar la accesibilidad de un endpoint utilizamos `WcfTestClient`.

### Seguridad en WCF

- ✓ La transferencia de los mensajes entre el cliente y el servicio debe realizarse de forma segura.
- ✓ Los 3 puntos a considerar para tener un WCF seguro son:
  - **Integridad del mensaje:** El mensaje no fue modificado.
  - **Privacidad del mensaje:** El mensaje es confidencial y ningún ente externo debe leer su contenido.
  - **Autenticación mutua:** El cliente se conecta al servicio correcto y el servicio autentica al cliente.

### Tipos de seguridad

Los modos de seguridad se configuran a nivel de Binding

- ✓ **None:**
  - La seguridad está apagada.
- ✓ **Transport:**
  - Se utiliza un protocolo de comunicación seguro, todo el canal de comunicación está encriptado.
  - Los protocolos disponibles son: HTTPS, TCP, IPC y MSMQ. Es la forma más simple y ligera de implementar la seguridad.
  - Solo es seguro de punto a punto y se utiliza en intranets.
- ✓ **Message:**
  - Solo se encripta el mensaje, obteniendo integridad, privacidad y autenticación mutua.
  - Es seguro de extremo a extremo, no importa si el mensaje viaja a través de varios intermediarios, es utilizado en aplicaciones de internet donde el canal no es necesariamente seguro.
- ✓ **Mixto (TransportWithMessageCredential):**

- Utiliza el modo Transport y el modo Message para los credenciales de usuario.
- Raramente usado
- ✓ **Ambos (Both):**
  - Se encripta el mensaje y se envía en un canal igualmente encriptado.

### **Autenticación**

Los modos de seguridad se basan en la encriptación del mensaje y/o el canal de comunicación.

- ✓ **Sin autenticación:**
  - Todos los clientes son permitidos.
- ✓ **Autenticación de Windows:**
  - El cliente usa sus credenciales de Windows para autenticarse.
- ✓ **Usuario y Contraseña:**
  - El cliente envía un nombre de usuario y contraseña al servicio.
  - El servicio valida los datos contra una base de datos u otro almacén de usuarios.
- ✓ **Certificado digital (Certificado X509):**
  - El cliente se identifica mediante un certificado digital, el servicio conoce de antemano ese certificado.

### **Tracing:**

- ✓ Consiste en recopilar información y generar diagnósticos sobre la ejecución de una aplicación o servicio web.
- ✓ Durante la ejecución de un programa se producen eventos que pueden ser capturados por un proceso "listener", esta información puede almacenarse de manera temporal en memoria, o de forma persistente en archivos de texto, xml, e inclusive en bases de datos.
- ✓ La información obtenida puede utilizarse para rastrear el origen de un error durante la ejecución de la aplicación.
- ✓ El tracing se puede habilitar mediante el archivo de configuración web.config, en la sección <system.diagnostics>.
- ✓ A diferencia del debug, el tracing se utiliza cuando la aplicación esta liberada en producción y puede activarse o desactivarse cuando se desee.

### **Niveles de trazabilidad**

- ✓ **Off:** No se emite ningún trace.
- ✓ **Critical:** Errores no manejados.
- ✓ **Error:** Errores manejados.
- ✓ **Warning:** Posibles problemas detectados que podrían causar un error.
- ✓ **Information:** Eventos positivos, por ejemplo, la finalización exitosa de un proceso.

Estos errores contienen al anterior.

Un servicio puede tener varios listener, y pueden ser personalizados.

### Opciones de logueo de mensajes

- ✓ **LogEntireMessage:** Por defecto se guarda el encabezado y cuerpo del mensaje.
- ✓ **LogMalformedMessages:** Por defecto si se loguean los mensajes mal formados, pero se puede desactivar esta función.
- ✓ **LogMessagesAtServiceLevel:** Por defecto desactivado, al activarse se incluyen los mensajes de infraestructura.
- ✓ **LogMessagesAtTransportLevel:** Por defecto activado, incluye los mensajes a nivel de transporte.
- ✓ **maxMessagesToLog:** Por defecto 10.000, si se excede, no se guardan más mensajes.
- ✓ **maxSizeOfMessageToLog:** Por defecto 256 KB, los mensajes que lo excedan no se registran.

El **tracing** se puede **guardar en una ruta específica** por ejemplo c:\Traces.svclog, los mensajes se guardan como una sola línea, así que tener un formateador de XML será de gran utilidad.

**AutoFlush** en true, sirve para limpiar el buffer de memoria automáticamente.

En escenarios de mucha concurrencia, usar un archivo para el log puede producir bloqueos si muchos threads tratan de escribir al mismo tiempo.

### Web Config y Machine Config

- ✓ En ASP.NET, se puede compartir información y configuración entre páginas Web, guardando la configuración de la aplicación web en una localización central llamada archivo Web.config.
- ✓ Un solo archivo Web.config siempre se ubicará en la carpeta raíz de la aplicación Web.
- ✓ La configuración del archivo Machine.config se aplica a todas las aplicaciones asp.net en su servidor, puede haber uno por cada framework instalado, mientras que las configuraciones realizadas en el archivo Web.config solo se aplican a esa aplicación web en particular.

### **WCF RestFul**

- ✓ Los servicios rest operan exclusivamente sobre el protocolo Http.
- ✓ Tienen menos overhead que SOAP por lo que son más ligeros y veloces.
- ✓ Además se basan en los verbos HTTP Post, Get, Put, Delete.

### **Diferencia wcf y webservice**

- ✓ **WCF**
  - Es un conjunto de librerías aportadas por el frameWork .Net, el cual cuenta con varios protocolos de comunicación como HttpBinding, NetTcpBinding, WSHttpBinding.
  - Este se expone al mundo a través de interfaces.
  - Robusto y seguro.
- ✓ **Web Services:**
  - Es una capa de lógica de negocio que se puede acceder mediante protocolos web como SOAP.
  - Se solicita o envía las peticiones mediante protocolos como get, put, post y delete.
  - Solo se comunica por http, donde la seguridad no se encuentra activa por default.

### **Diferencia entre soa y soap**

- ✓ SOA es una arquitectura para crear sistemas orientados a servicios.
- ✓ SOAP es un paradigma o protocolo de mensajería estándar basado en XML, es el medio por el cual un servicio comunica con clientes u otros servicios para enviar data.

### **Diferencia entre soap y rest**

- ✓ Los servicios rest operan exclusivamente sobre el protocolo Http y tienen menos overhead que SOAP por lo que son más ligeros y veloces.

### **HTML 5**

- ✓ Implementa funcionalidades que antes dependían de CSS y JavaScript.
- ✓ El contenido de un sitio debe tener 4 características:
  - Perceptible
  - Operable

- Entendible
- Robusto.
- ✓ La sección `<script>` puede declararse tanto en head como dentro de body y en ella se pueden referenciar scripts o agregar el código directamente. Se recomienda declararla al final del `<body>`.



## CSS Cascading Style Sheets

- ✓ Tecnología desarrollada para separar la presentación de la estructura HTML.
- ✓ Esta tecnología aplica reglas de estilo a los elementos HTML. Así, a los elementos de la página web creados con HTML se les dará la apariencia que se desee utilizando CSS: colores, espacios entre elementos, tipos de letra separando de esta forma la estructura de la presentación.
- ✓ En css el modelo de cajas o "box Model" es seguramente la característica más importante del lenguaje de hojas de estilos CSS, ya que condiciona el diseño de todas las páginas web.
- ✓ El modelo de cajas es el comportamiento de CSS que hace que todos los elementos de las páginas se representen mediante cajas rectangulares. Tiene 4 componentes:
  - Margin
  - Border
  - Padding
  - Content

## XML

- ✓ XML es un subconjunto de SGML.
- ✓ Especificación para diseñar lenguajes de marcado, que permite definir etiquetas personalizadas para descripción y organización de datos.

### Características

- ✓ Es un estándar internacionalmente conocido, que no pertenece a ninguna compañía.
- ✓ Es Extensible, estructurado y cuenta con validación.
- ✓ Diseñado para cualquier lenguaje y alfabeto.
- ✓ Basado en texto. El XML es Case – Sensitive.
- ✓ Orientado a los contenidos no a la presentación.
- ✓ Todo elemento tiene que tener su correspondiente etiqueta de inicio y de cierre y debe haber un elemento llamado **raíz** de documento, que contenga a los demás.
- ✓ Cuando un nodo contiene a otro se le denomina **rama**.
- ✓ Los nodos finales, que no contienen otros nodos, se llaman **hojas**.
- ✓ Todos los valores de los atributos deberán ir entre comillas.
- ✓ El XML sirve para representar información estructurada en la web (todos documentos), de modo que esta información pueda ser almacenada, transmitida, procesada, visualizada e impresa, por muchos tipos de aplicaciones.

### Tipo de Documento DTD

- ✓ Define los elementos, atributos, entidades y notaciones que pueden utilizarse para construir un tipo de documentos, así como las reglas para su utilización.
- ✓ Mediante ellas se comprueba la validez de un documento y posee una sintaxis especializada.

### XSD (XML SCHEMA DEFINITION)

- ✓ XSD es un formato para definir la estructura de un documento XML. XSD sustituye al anterior formato DTD, y añade funcionalidad para definir la estructura XML con más detalle.

### XSLT

- ✓ Es un lenguaje que se usa para convertir documentos XML en otros documentos XML
- ✓ Puede convertir un documento XML que obedezca a un DTD a otro que obedezca otro diferente, un documento XML bien formado a otro que siga un DTD, o, lo más habitual, convertirlo a "formatos finales", tales como WML (usado en los móviles WAP) o XHTML.



**CRUX Consultores**

Edificio Las Terrazas A, Piso 5. Plaza Roble, Escazú  
San José, Costa Rica ▪ Apartado: 147-3019  
Tel.: (506) 2494-3415 ▪ Fax: (506) 2505-5601

**DTSX**

- ✓ Es un formato de archivo basado en XML que almacena las instrucciones para el procesamiento de un flujo de datos.