**TUTORIAL DEV.**

**SERVIDORES WEB**

*Un servidor web es un software que forma parte del servidor y tiene como misión principal devolver información (páginas) cuando recibe peticiones por parte de los usuarios.*

**IIS :** Internet Information Services o IIS es un servidor web y un conjunto de servicios para el sistema operativo Microsoft Windows.

**Kestrel:** Es el servidor web multi-plataforma que ha creado Microsoft y que sirve para albergar las aplicaciones web creadas con ASP.NET Core y dotarlas de manera sencilla de la infraestructura para poder ejecutarse.

**BASE DE DATOS**

**Normalización:** La normalización es el proceso o forma de organizar datos en una DB. Esto incluye la creación de tablas ,el establecimiento de relaciones entre esas tablas y la eliminación de la redundancia y la dependencia incoherente.

**Indexación**: La **indexación es el proceso mediante el cual se examinan ordenadamente los datos e informes para elaborar los índices de las tablas y realizar consultas más rápidas.**

**HASH:** Un valor de este tipo hace referencia a una posición o dirección en la base de datos para realizar consultas más rápidas.

**ORM:** Object Relational Mappings.Es un modelo de programación que permite mapear las estructuras de una base de datos relacional ,en adelante RDBMS (Relational Database Management System), sobre una estructura lógica de entidades con el objeto de simplificar y acelerar el desarrollo de nuestras aplicaciones.Las estructuras de la base de datos relacional quedan vinculadas con las entidades lógicas o base de datos virtual definida en el ORM, de tal modo que las acciones CRUD (Create, Read, Update, Delete) a ejecutar sobre la base de datos física se realizan de forma indirecta por medio del ORM.

**Relaciones:**

En las bases de datos relacionales, las relaciones (también denominadas asociaciones) entre tablas se definen mediante claves externas.

Una clave externa (FK) es una columna o combinación de columnas que se utiliza para establecer y exigir un vínculo entre los datos de dos tablas.

Por lo general, hay tres tipos de relaciones:

**uno a uno, uno a varios y varios a varios.**

En una relación uno a varios, la clave externa se define en la tabla que representa el gran final de la relación.

La relación de varios a varios implica definir una tercera tabla (denominada unión o tabla de combinación), cuya clave principal se compone de las claves externas de ambas tablas

En una relación uno a uno, la clave principal actúa además como una clave externa y no hay ninguna columna de clave externa independiente para ninguna tabla.

**ENTITY FRAMEWORK**

Es un conjunto de tecnologías ADO .NET orientada a modelos de datos de forma simplificada

*-Modelo de dominio -> Entidades y relaciones lógicas*

*-Modelo lógico -> Normalización de la base de datos y restricciones de claves externas*

*-Modelo Físico -> Capacidades del motor de base de datos, peticiones e indexación*

**DB CONTEXT**

Es la interfaz entre el dominio o clases y la base de datos, es la clase principal en la que reside Entity Framework , convierte consultas Linq to Sql en consultas Sql y las envia a la base de datos.

**EntityState** es una **enumeración** con 5 valores posibles e incluso combinaciones entre sus valores:

**Added**

**Modified**

**Deleted**

**Unchanged**

**Detached**

Cuando haces operaciones usando **EntityFramework** se producen cambios en la entidad donde se están realizando esos cambios, y esos cambios tienen un estado, ej. si se modificó un registro el estado de la entidad es **Modified**, o si se borró es **Deleted**.

Cuando haces **SaveChange** al contexto, ej.:

db.SaveChanges();

El **EntityFramework** recorre todas las entidades para ver su estado y si detecta que alguna cambió su estado, entonces hace efectivo este cambio y guarda dicho dato a la **base de datos**, recuerda que en **EntityFramework** hasta que no hagas **SaveChanges** no se harán efectivos los cambios en la **base de datos**.

Al marcar los cambios hechos sobre las entidades con un estado, el **EntityFramework**, sabrá exactamente que hacer con esa entidad, es decir si el estado de una entidad es **Modified**, sabe que debe modificar ese dato en la **base de datos** y si es **Added**, sabe que tiene que añadirlo, por lo que solo hará alguna acción sobre las entidades que han sido cambiadas de alguna forma, es decir la que su estado no sea **Unchanged**, esto es eficiencia, ya que si el estado de la entidad no ha cambiado, pues el **EntityFramework** sencillamente no la toca, el solo trabaja con las entidades cambiadas para registrar esos cambios en la **base de datos**.

Una vez que se hace **SaveChange**, pues el estado de todas las entidades será **Unchanged**.

**POCO** (Plain Old C# Object, Simple antiguo C# objeto), o sea, clases normales y corrientes de C#) que representan dicho modelo, usando plantillas T4.

**Plantillas T4**

Esta plantillas usadas para clases de entidad POCO simples y un contexto que se deriva de DbContext usando EF. Esta es la plantilla recomendada a menos que tenga una razón para usar otro tipo de plantillas. También es la plantilla de generación de código que obtiene de forma predeterminada si está utilizando versiones recientes de Visual Studio cuando crea un nuevo modelo, esta plantilla se utiliza de forma predeterminada y los archivos T4 se anidan en su archivo **.edmx.**

**.EDMX**

Este archivo contiene información que describe el modelo conceptual de la base de datos, el modelo de almacenamiento y las asignaciones entre ellos.

**LINQ**

(Language Integrated Query) es un conjunto de extensiones integradas en el lenguaje C#, que nos permite trabajar de manera cómoda y rápida con colecciones de datos, como si de una base de datos se tratase.

**LINQ TO SQL**

En LINQ TO SQL, el modelo de datos de una DB relacional se asigna a un modelo de objetos expresado en el lenguaje de programación del desarrollador.

Cuando se ejecuta la aplicación, LINQ to SQL traduce a SQL las consultas

las envía a la base de datos para su ejecución. Cuando la base de datos devuelve los resultados,

**Microsoft SQL Server es un sistema de gestión de base de datos relacional.**

**ORM** (**Object Relational Mapping**)

Es un modelo de programación que consiste en la transformación de las tablas de una base de datos, en una serie de entidades (Clases) que simplifiquen las tareas básicas de acceso a los datos para el programador.

**Database First - DB Primero**

*Es el método que nos permite primero crear la base de datos con sus tablas, relaciones e indexación y luego incorporarlas a la aplicación.*

Database First le permite realizar ingeniería inversa de un modelo a partir de una base de datos existente. El modelo se almacena en un archivo EDMX y se puede ver y editar en Entity Framework Designer. Las clases con las que interactúa en su aplicación se generan automáticamente a partir del archivo EDMX.

**Model First - Modelo Primero**

Permite crear un nuevo modelo utilizando Entity Framework Designer y luego generar un esquema de base de datos a partir del modelo. El modelo se almacena en un archivo EDMX y se puede ver y editar en Entity Framework Designer.

**Code First - Codigo Primero**

Definimos nuestras clases mediante código, y EF se encarga de generar la base de datos y todo lo necesario para encajar las clases en ellas.

**EXPRESION LAMBDA**

Es una función anónima que puede contener expresiones e instrucciones y se puede utilizar para crear delegados o tipos de árboles de expresión la mayoría de las expresiones devuelven (más o menos) un valor

**METODOS ANONIMOS**

Son aquellos que carecen de nombre y solo estan constituidos por un cuerpo (codigo)

**DELEGADOS:**

Podemos indicar que un delegado es una referencia a un método.

Cuando definimos un delegado lo que estamos haciendo es declarar una variable que apunta a otro método.

***Un delegado es un tipo que define una firma de método y se puede asociar a cualquier método con una firma compatible. Puede invocar (o llamar) al método a través del delegado.***

**Firma:**

*public void MostrarMensaje(string mensaje, int cantidadVeces) { ... }*

*public string MostrarMensaje(string mensaje) { ... }*

**Declaración:**

*public delegate void MostrarMensajeDelegado(string mensaje, int cantidadVeces);*

**Invocar Delegado:**

*MostrarMensajeDelegado miDelegado = new MostrarMensajeDelegado(MostrarMensaje);*

*miDelegado("hola mundo", 5);*

**Ejemplo:**

*Y porque no simplemente se llama el método directamente?*

***Vamos a ver porque:***

*Dicho antes, con un delegado puedes invocar cualquier método con la única condición de que la firma sea idéntica. El ejemplo más practico que puedo darte son los eventos. Cuando creas un evento onclick a un botón lo que haces es crear un método, y "decirle" al botón que cuando hagan clic en él, invoque dicho método. Cómo crees que lo hace?.*

*Otro ejemplo muy utilizado es cuando utilizas hilos (Threads) y desde un hilo necesitas utilizar un control de formulario (como un TextBox o un DataGridView), ya que esto no es permitido se deben utilizar dichos controles desde el mismo contexto de la interfaz gráfica haciendo la invocación del delegado a través de un control de formulario. así:*

object[] parametros = new object[] { "hola mundo", 5};

this.textBox1.Invoke(miDelegado, parametros);

**LENGUAJE DE ALTO NIVEL**

Se caracteriza por expresar los algoritmos de una manera adecuada a la capacidad cognitiva humana, en lugar de la capacidad con que los ejecutan las máquinas

**LENGUAJE DE BAJO NIVEL**

Es aquel en el que sus instrucciones ejercen un control directo sobre el hardware y están condicionados por la estructura física de las computadoras que lo soportan

**MODELO - MVC**

Modelo Vista Controlador (MVC) es un estilo de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos.

**El Modelo** que contiene una representación de los datos que maneja el sistema, su lógica de negocio, y sus mecanismos de persistencia.

**La Vista o interfaz de usuario**, que compone la información que se envía al cliente y los mecanismos interacción con éste.

**El Controlador**, que actúa como intermediario entre el Modelo y la Vista, gestionando el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno.

**Razor**

En una sintaxis basada en C# que permite usarse como motor de programación en las vistas o plantillas de nuestros controladores.

**Blazor**

Es un proyecto desarrollado por Microsoft creado para permitir crear SPAs únicamente usando como lenguajes de programación C# y Razor Pages, haciendo nula la necesidad de programar en Javascript o frameworks derivados.

**Modelos de hospedaje de Blazor**

**Blazor Server:** Se construye el DOM que se ha de enviar al cliente desde el servidor. Es el modelo más tradicional, cuyo objetivo es sustituir el modelo Web Forms de .NET. Su principal fuerte es la interacción en tiempo real entre cliente y servidor a través de SignalR.

**Blazor WebAssembly:** modelo SPA basado en WebAssembly, es decir, la construcción del DOM se realizará en el lado del cliente. Permite a su vez realizar operaciones en el lado del servidor, llamando a APIs para solicitar datos, con la intencionalidad de obtener información sensible que no se quiera calcular en el cliente. Para entender esto, hay que comprender qué es WebAssembly.

**.NET CORE**

Un nuevo framework de código abierto y multiplataforma para la creación de aplicaciones.

Se pueden desarrollar y ejecutar aplicaciones ASP.NET Core en Windows, Mac y Linux.

ASP.NET Core es un rediseño completo de ASP.NET. por lo que su arquitectura ha sido diseñada para resultar más ligera y modular.

**DIFERENCIA ENTRE ASP NET CORE Y ASP NET MVC**

ASP.NET MVC está basado en System.Web.dll que aportaba un exceso de funcionalidad. ASP.NET CORE se basa en un conjunto de paquetes NuGet granulares y bien factorizados. Esto te permite optimizar tu aplicación para incluir solo los paquetes NuGet que necesitas.

**Beneficios de ASP.NET CORE contra ASP.NET MVC**

-Seguridad más estricta: Menor intercambio de información y rendimiento mejorado

-Una plataforma unificada para la creación de interfaz web y las APIs web.

-Un sistema de configuración basado en la nube. Preparado para su integración de forma sencilla en entornos en la nube.

Inyección de dependencias incorporada.

-Las peticiones HTTP se procesan siguiendo un flujo que puede ser modificado de forma modular para adaptarse a nuestras necesidades y que nos permite poder controlar el procesado de las peticiones HTTP en nuestra aplicación.

-Capacidad para alojar en IIS u otros servidores web como Apache. o self-host, kestrell en su propio proceso.

**INYECCION DE DEPENDENCIAS**

Es un patrón de diseño orientado a objetos, en el que se suministran objetos a una clase en lugar de ser la propia clase la que cree dichos objetos. Esos objetos cumplen contratos que necesitan nuestras clases para poder funcionar Nuestras clases no crean los objetos que necesitan, sino que se los suministra otra clase 'contenedora' que inyectará la implementación deseada a nuestro contrato.

**VENTAJAS**

Bajo acoplamiento entre los componentes

La Inyección de Dependencias es un patrón de diseño ampliamente conocido, por ende es fácil de adaptar en múltiples lenguajes de programación

Facilidad para pruebas, ya que al tener componentes más desacoplados estos son más independientes,

El software se hace más mantenible a medida que va creciendo, ya que si se implementa una buena arquitectura con DI, la responsabilidad de cada uno de los componentes será muy clara y un cambio podrá ser más fácil de implementar.

Al usar DI debemos pensar un poco más y planear mejor las dependencias de una clase, ya que la idea es utilizar solo las que sean necesarias.

**PRINCIPIO DE INVERSION DE DEPENDENCIAS** es una forma específica de desacoplar módulos de software. Al seguir este principio, las relaciones de dependencia convencionales establecidas desde los módulos de alto nivel de establecimiento de políticas a los módulos de dependencia de bajo nivel se invierten, lo que hace que los módulos de alto nivel sean independientes de los detalles de implementación del módulo de bajo nivel.

**WEB API**

**API (interfaz de programación de aplicaciones)**

La interfaz de programación de aplicaciones, abreviada como API del inglés: Application Programming Interface, es un conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

**Servicio Web**

Un servicio web es una interfaz de software que describe un conjunto de operaciones a las cuales se puede acceder por la red a través de mensajería XML estandarizada. Usa protocolos basados en el lenguaje XML con el objetivo de describir una operación para ejecutar o datos para intercambiar con otro servicio web.

**API RESTFULL - transferencia de estado representacional**

Es un estilo arquitectónico para una interfaz de programa de aplicación ( API ) que usa solicitudes HTTP para acceder y usar recursos. Utilizando verbos HTTP: GET, PUT, POST, DELETE.

**HTTP**

El http (del inglés HyperText Transfer Protocol o Protocolo de Transferencia de Hiper Textos) es el protocolo de transmisión de información de la World Wide Web, es decir, el código que se establece para que el computador solicitante y el que contiene la información solicitada puedan “hablar” un mismo idioma a la hora de transmitir información por la red.

No es sólo para servir páginas web. También es una potente plataforma para crear API que exponen servicios y datos. HTTP es simple, flexible y ubicuo. Casi cualquier plataforma que se pueda imaginar tiene una biblioteca HTTP, por lo que los servicios HTTP pueden llegar a una amplia gama de clientes, incluidos navegadores, dispositivos móviles y aplicaciones de escritorio tradicionales.

.- GET Obtiene el recurso indicado. Es el método que se utiliza cuando se pide el contenido de una página web, por ejemplo.

.- HEAD: Similar a GET, pero no se obtiene el cuerpo de respuesta, únicamente los metadatos de la cabecera.

.- POST: añade datos al servidor. Siempre es un método de creación.

.- PUT: es una solicitud para almacenar la entidad suministrada en el URI indicado. Si la entidad no existe, se crea. Si la entidad existe, se actualiza.

.- DELETE: elimina el recurso indicado.

.- TRACE: devolverá la misma información que se ha enviado en la solicitud. Es una especie de eco. Sirve para comprobar si la solicitud se ha visto modificada por servidores intermedios.

.- OPTIONS: Devuelve los métodos HTTP soportados por el servidor para la URL especificada.

.- CONNECT: Convierte la solicitud en un tunel TCP/IP. Normalmente se usa para crear comunicaciones HTTPS a través de proxys HTTP sin encriptación.

.- PATCH; Aplica modificaciones parciales al recurso especificado.

**SOAP: protocolo simple de acceso a objetos**

SOAP es un protocolo estándar que se creó originalmente para permitir la comunicación entre las aplicaciones que se diseñaban con diferentes lenguajes y en diferentes plataformas. Como es un protocolo, impone reglas integradas que aumentan la complejidad y la sobrecarga, lo cual puede retrasar el tiempo que tardan las páginas en cargarse. Sin embargo, estos estándares también ofrecen normas integradas que pueden ser ideales para el sector empresarial. Los estándares de cumplimiento integrados incluyen la seguridad, la atomicidad, la uniformidad, el aislamiento y la durabilidad (ACID), que forman un conjunto de propiedades que garantizan operaciones confiables de las bases de datos.

**Idepotencia**

Es la propiedad para realizar una acción determinada varias veces y aun así conseguir el mismo resultado.

El principio de idempotencia aplicado a REST (realmente es a los verbos HTTP) nos dice que, la ejecución repetida de una petición con los mismos parámetros sobre un mismo recurso tendrá el mismo efecto en el estado de nuestro recurso en el sistema si se ejecuta 1 o N veces

**HATEOAS:** Es que el cliente pueda moverse por la aplicación web únicamente siguiendo a los identificadores únicos URI en formato hipermedia. Cuando se aplica este principio, el cliente, aparte de una comprensión básica de los hipermedia, no necesita más información para poder interactuar con la aplicación y el servidor.

**Hipermedia:**  Sirve para designar al conjunto de métodos o procedimientos para escribir, diseñar o componer contenidos que integren soportes tales como: texto, imagen, video, audio, mapas y otros soportes de información emergentes, de tal modo que el resultado obtenido, además, tenga la posibilidad de interactuar con los usuarios.

**Atomicidad:**  Este principio asegura que las operaciones de múltiples pasos se realizan completas, es decir, o se ejecutan o todos los pasos o no se ejecuta ninguno, de manera que los datos nunca quedan en medio de una transacción en caso de que el sistema se detenga y puedan perderse o corromperse

**PROGRAMACION ORIENTADA A OBJETOS**

**Un paradigma es una teoría o modelo explicativo de las realidades físicas. Con el significado de paradigma científico, se usa hoy en la investigación científica y fue introducido por Thomas Kuhn (1975) para explicar o los cambios en teorías**

**Atributos** Características o variables que tiene la clase.

**Propiedad** Contenedor de un tipo de datos asociados a un atributo u objeto utilizados para realizar operaciones con los atributos.

**Una clase es una plantilla** que define de manera genérica cómo van a ser los objetos de determinado tipo.

**Instancia** es un objeto del tipo de una clase

**La encapsulación** es la característica que permite que todas las propiedades de un objeto este aislado dentro de su respectiva clase.

**Abstraccion**  característica que implica que la clase debe contener toda la lógica aislado del resto de elementos que lo acompañan.

**Herencia**  es uno de los mecanismos de los lenguajes de programación orientada a objetos basados en clases, por medio del cual una clase se deriva de otra de manera que extiende su funcionalidad.

**Polimorfismo** En programación orientada a objetos, el polimorfismo se refiere a la propiedad por la que es posible enviar mensajes sintácticamente iguales a objetos de tipos distintos. El único requisito que deben cumplir los objetos que se utilizan de manera polimórfica es saber responder al mensaje que se les envía.

**Principio de ocultación** Cada objeto está aislado del exterior, de manera natural es un módulo

**Clase Abstracta** Una clase que declara la existencia de métodos, pero no la implementación de dichos métodos (o sea, las llaves { } y las sentencias entre ellas), se considera una clase abstracta. Cuando declaramos una clase como abstracta estamos indicado que esa clase va a ser utilizada como clase base de otras clases, ya que ella misma no se puede instanciar

**Clase Estatica** En términos generales es similar a una clase normal, pero tiene una diferencia crucial. Y es que no puede ser instanciada. Lo que quiere decir que para acceder a los miembros de una clase estática utilizando directamente el nombre de la clase estática.

**UI / UX**

**UI (User Interface)** hace referencia a la experiencia y sensación del usuario, amigabilidad del sitio.

En el campo del diseño industrial de la interacción humano-computadora, una interfaz de usuario (UI) es el espacio donde ocurren las interacciones entre humanos y máquinas. El objetivo de esta interacción es permitir la operación y el control efectivos de la máquina desde el extremo humano, mientras que la máquina retroalimenta simultáneamente información que ayuda en el proceso de toma de decisiones de los operadores.

**UX User Experience:** el diseño UX tiene como objetivo proporcionar la experiencia positiva del usuario. Como se siente mientras utiliza nuestro producto, La experiencia de usuario es el conjunto de factores y elementos relativos a la interacción del usuario con un entorno o dispositivo concretos, dando como resultado una percepción positiva o negativa de dicho servicio.

**INGENIERIA INVERSA – RETROINGENIERIA**

Es el proceso llevado a cabo con el objetivo de obtener información o un diseño a partir de un producto, con el fin de determinar cuáles son sus componentes y de qué manera interactúan entre sí y cuál fue el proceso de fabricación.

**JWT**

JSON Web Token (JWT) es un estándar abierto que define una forma compacta y autónoma de transmitir información de forma segura entre las partes como un objeto JSON. ***“Los tokens firmados pueden verificar la integridad de los reclamos contenidos en él”***

**Escenarios en los que los tokens web JSON son útiles:**

**Autorización** : este es el escenario más común para usar JWT. Una vez que el usuario haya iniciado sesión, cada solicitud posterior incluirá el JWT, lo que le permitirá acceder a rutas, servicios y recursos que están permitidos con ese token. El inicio de sesión único es una función que utiliza ampliamente JWT en la actualidad, debido a su pequeña sobrecarga y su capacidad para usarse fácilmente en diferentes dominios.

**Intercambio de información** : los tokens web JSON son una buena forma de transmitir información de forma segura entre las partes. Debido a que los JWT se pueden firmar, por ejemplo, utilizando pares de claves públicas / privadas, puede estar seguro de que los remitentes son quienes dicen ser. Además, como la firma se calcula utilizando el encabezado y la carga útil, también puede verificar que el contenido no haya sido manipulado.

**Estructura JWT**

**Encabezado:** Generalmente consta de dos partes: el tipo de token, que es JWT, y el algoritmo de firma que se utiliza, como HMAC SHA256 o RSA.

**{**

**"alg": "HS256",**

**"typ": "JWT"**

**}**

**Carga útil:** La segunda parte del token es la carga útil, que contiene las notificaciones. Las afirmaciones son declaraciones sobre una entidad (normalmente, el usuario) y datos adicionales. Hay tres tipos de reclamaciones: reclamaciones registradas , públicas y privadas .

**Reclamaciones registradas** : Se trata de un conjunto de reclamaciones predefinidas que no son obligatorias, pero sí recomendadas, para proporcionar un conjunto de reclamaciones útiles e interoperables. Algunos de ellos son: iss (emisor), exp (tiempo de vencimiento), sub (asunto), aud (audiencia) y otros .

Tenga en cuenta que los nombres de las notificaciones tienen solo tres caracteres, ya que JWT debe ser compacto.

**Reclamos públicos** : Estos pueden ser definidos a voluntad por aquellos que usan JWT. Pero para evitar colisiones, deben definirse en el Registro de tokens web JSON de

**IANA(La coordinación global de la raíz del DNS, el direccionamiento IP y otros recursos del protocolo de Internet se realiza como funciones de la Autoridad de números asignados de Internet (IANA))** o definirse como un URI que contiene un espacio de nombres resistente a colisiones.

**Reclamos privados** : Estos son reclamos personalizados creados para compartir información entre las partes que acuerdan usarlos y no son reclamos registrados ni públicos .

**Un ejemplo de carga útil podría ser:**

**{**

**"sub": "1234567890",**

**"name": "John Doe",**

**"admin": true**

**}**

**Firma**

Para crear la parte de la firma, debe tomar el encabezado codificado, la carga útil codificada, un secreto, el algoritmo especificado en el encabezado y firmarlo.

Por ejemplo, si desea utilizar el algoritmo HMAC SHA256, la firma se creará de la siguiente manera:

HMACSHA256(

base64UrlEncode(header) + "." +

base64UrlEncode(payload),

secret)

La firma se usa para verificar que el mensaje no se haya modificado en el camino y, en el caso de los tokens firmados con una clave privada, también puede verificar que el remitente del JWT es quien dice ser.

**PROTOCOLOS DE TRASMISION DE DATOS**

**El protocolo UDP (User Datagram Protocol)**

UDP es un protocolo no orientado a conexión. Es decir cuando una maquina A envía paquetes a una maquina B, el flujo es unidireccional. La transferencia de datos es realizada sin haber realizado previamente una conexión con la máquina de destino (maquina B), y el destinatario recibirá los datos sin enviar una confirmación al emisor (la maquina A). Esto es debido a que la encapsulación de datos enviada por el protocolo UDP no permite transmitir la información relacionada al emisor. Por ello el destinatario no conocerá al emisor de los datos excepto su IP.

**El protocolo TCP (Transmission Control Protocol)**

Contrariamente a UDP, el protocolo TCP está orientado a conexión. Cuando una máquina A envía datos a una máquina B, la máquina B es informada de la llegada de datos, y confirma su buena recepción. Aquí interviene el control CRC de datos que se basa en una ecuación matemática que permite verificar la integridad de los datos transmitidos. De este modo, si los datos recibidos son corruptos, el protocolo TCP permite que los destinatarios soliciten al emisor que vuelvan a enviar los datos corruptos.

**SOCKET**

Un socket es un punto final de comunicación al que se puede vincular un nombre. Cada socket en uso tiene un tipo y uno o más procesos asociados. Los sockets existen dentro de los dominios de comunicación. Un dominio de comunicación es una abstracción introducida para agrupar propiedades comunes de procesos que se comunican a través de sockets.

El establecimiento de una conexión suele ser asimétrico; un proceso es un cliente y el otro es un servidor. El servidor, cuando está dispuesto a ofrecer sus servicios anunciados, vincula un socket a una dirección conocida asociada con el servicio y luego escucha pasisamente en su socket.

**TIPOS DE SOCKET**

Stream socket

Datagram socket

Raw socket

**Toma de corriente**

Proporciona el flujo de datos bidireccional, confiable, secuenciado y no duplicado sin límites de registro. Aparte de la naturaleza bidireccional del flujo de datos, un par de sockets de flujo conectados proporciona una interfaz casi idéntica a la de las tuberías.

**Toma de datagrama**

Admite un flujo de datos bidireccional que no se garantiza que sea secuenciado, confiable o no duplicado. Es decir, un proceso que recibe mensajes en un socket de datagrama puede encontrar mensajes duplicados y posiblemente en un orden diferente al que fueron enviados. Una característica importante de un socket de datagrama es que se conservan los límites de registro en los datos. Los sockets de datagramas modelan de cerca las instalaciones que se encuentran en muchas redes de conmutación de paquetes contemporáneas (por ejemplo, Ethernet).

**Toma sin procesar**

Proporciona a los usuarios acceso a los protocolos de comunicación subyacentes que admiten abstracciones de socket. Estos sockets están normalmente orientados a datagramas, aunque sus características exactas dependen de la interfaz proporcionada por el protocolo. Los sockets sin procesar no están destinados al usuario general; se han proporcionado principalmente para cualquier persona interesada en desarrollar nuevos protocolos de comunicación o en obtener acceso a algunas de las instalaciones más esotéricas de un protocolo existente. El uso de sockets sin formato se analiza en la sección `` Temas avanzados '' de este capítulo.

**TEST UNIT – PRUEBAS UNITARIAS**

Se usan para conservar el estado del código, garantizar la cobertura del código y detectar errores y fallos antes de que lo hagan los clientes. Ejecute las pruebas unitarias con frecuencia para asegurarse de que el código funciona correctamente.

En programación, una prueba unitaria es una forma de comprobar el correcto funcionamiento de una unidad de código. Esto sirve para asegurar que cada unidad funcione correctamente y eficientemente por separado. Además de verificar que el código hace lo que tiene que hacer, verificamos que sea correcto el nombre, los nombres y tipos de los parámetros, el tipo de lo que se devuelve, que si el estado inicial es válido, entonces el estado final es válido también.

**CARACTERISTICAS – TEST UNIT**

**Automatizable**: No debería requerirse una intervención manual. Esto es especialmente útil para integración continua.

**Completas**: Deben cubrir la mayor cantidad de código.

**Repetibles o Reutilizables**: No se deben crear pruebas que sólo puedan ser ejecutadas una sola vez. También es útil para integración continua.

**Independientes**: La ejecución de una prueba no debe afectar a la ejecución de otra.

**Pruebas integrales o pruebas de integración**

Son aquellas que se realizan en el ámbito del desarrollo de software una vez que se han aprobado las pruebas unitarias y lo que prueban es que todos los elementos unitarios que componen el software, funcionan juntos correctamente probándolos en grupo. Se centra principalmente en probar la comunicación entre los componentes y sus comunicaciones ya sea hardware o software.

**La prueba de subsistemas** es un tipo de prueba de integración donde se prueba el contenido de un subsistema.

**SOA**

**La Arquitectura Orientada a Servicios (Service Oriented Architecture)** Es un estilo de arquitectura de TI que se apoya en la orientación a servicios. La orientación a servicios es una forma de pensar en servicios, su construcción y sus resultados.

***Un servicio es una representación lógica de una actividad de negocio que tiene un resultado de negocio específico*** (ejemplo: comprobar el crédito de un cliente, obtener datos de clima, consolidar reportes de perforación)

El estilo de arquitectura SOA se caracteriza por:

* Estar basado en el diseño de servicios que reflejan las actividades del negocio en el mundo real.
* Representar los servicios utilizando descripciones de negocio para asignarles un contexto.
* Tener requerimientos de infraestructura específicos y únicos para este tipo de arquitectura, en general se recomienda el uso de estándares abiertos para la interoperabilidad y transparencia en la ubicación de servicios.

**MICROSERVICIOS**

Los microservicios son un tipo de arquitectura que sirve para diseñar aplicaciones. Lo que distingue a la arquitectura de microservicios de los enfoques tradicionales y monolíticos es la forma en que desglosa una aplicación en sus funciones principales. Cada función se denomina servicio y se puede diseñar e implementar de forma independiente. Esto permite que funcionen separados (y también, fallen por separado) sin afectar a los demás.

*Cuanto más grande es la aplicación, más difícil es solucionar los problemas que se presentan y agregar funciones nuevas rápidamente. En cambio, si las aplicaciones se diseñan con microservicios, se resuelven dichos problemas y se impulsa el desarrollo y la capacidad de respuesta.*

**ESB**

En informática un Bus de Servicio Empresarial (ESB por sus siglas en inglés) es un modelo de arquitectura de software que gestiona la comunicación entre servicios web. Es un componente fundamental de la Arquitectura Orientada a Servicios.

Un ESB generalmente proporciona una capa de abstracción construida sobre una implementación de un sistema de mensajes de empresa que permita a los expertos en integración explotar el valor del envío de mensajes sin tener que escribir código. Al contrario que sucede con la clásica integración de aplicaciones de empresa (IAE) que se basa en una pila monolítica sobre una arquitectura hub and spoke, un bus de servicio de empresa se construye sobre unas funciones base que se dividen en sus partes constituyentes, con una implantación distribuida cuando se hace necesario, de modo que trabajen armoniosamente según la demanda.

**Un ESB no implementa en sí mismo una arquitectura orientada a servicios (SOA), sino que proporciona las características mediante las cuales sí se puede implementar. Un ESB debería basarse en normas y proporcionar flexibilidad, dando cobertura a distintos medios de transporte que sean capaces de implementar tanto patrones de SOA tradicionales como arquitectura de negocios con una SOA 2.0 enriquecida. El ESB trata de aislar el acoplamiento entre el servicio solicitado y el medio de transporte. La mayoría de los proveedores de ESB incorporan principios de SOA y permiten formatos de mensaje independientes.**

**Patrones de diseño PRINCIPIOS SOLID.**

**Los Principios SOLID son uno de los conceptos de programación y diseño de software**

1. Principio de responsabilidad única.

El Principio de Responsabilidad Única nos dice que una clase tiene una única razón para cambiar

2. Abierta para extensiones, cerrada para modificaciones.

Una clase debe estar abierta a extensiones, pero cerrada a las modificaciones.

OCP es la respuesta a la pregunta que hacíamos anteriormente, ya que argumenta que deberíamos diseñar clases que nunca cambien, y que cuando un requisito cambie, lo que debemos hacer es extender el comportamiento de dichas clases añadiendo código, no modificando el existente.

3. Principio de sustitución de Liskov

Si S es un subtipo de T, entonces los objetos de tipo T pueden ser reemplazados con objetos de tipo S, sin romper el programa.

Nos dice que si en alguna parte de nuestro código estamos usando una clase, y esta clase es extendida, tenemos que poder utilizar cualquiera de las clases hijas y que el programa siga siendo válido.

4. Principio de segregación de interfaces

El principio de segregación de interfaces viene a decir que ninguna clase debería depender de métodos que no usa. Por tanto, cuando creemos interfaces que definan comportamientos, es importante estar seguros de que todas las clases que implementen esas interfaces vayan a necesitar y ser capaces de agregar comportamientos a todos los métodos. En caso contrario, es mejor tener varias interfaces más pequeñas.

5. Principio de inversión de dependencias

Establece que las dependencias deben estar en las abstracciones, no en las concreciones. Es decir:

-. Los módulos de alto nivel no deberían depender de módulos de bajo nivel. Ambos deberían depender de abstracciones.

-. Las abstracciones no deberían depender de detalles. Los detalles deberían depender de abstracciones.

**GraphQL**

Es un lenguaje de consulta para API y un tiempo de ejecución para completar esas consultas con sus datos existentes. GraphQL proporciona una descripción completa y comprensible de los datos en su API, les da a los clientes el poder de pedir exactamente lo que necesitan y nada más, facilita la evolución de las API con el tiempo y habilita poderosas herramientas para desarrolladores.

* Las aplicaciones que usan GraphQL son rápidas y estables porque controlan los datos que obtienen, no el servidor.
* Si bien las API REST típicas requieren la carga desde varias URL, las API GraphQL obtienen todos los datos que su aplicación necesita en una sola solicitud. Esto implica mayor velocidad.
* Las API de GraphQL están organizadas en términos de tipos y campos, no de puntos finales. Acceda a todas las capacidades de sus datos desde un único punto final.
* Agregue nuevos campos y tipos a su API GraphQL sin afectar las consultas existentes.

**BLAZOR -> Sigle Page Application**

**Interfaz de usuario web interactiva con C #**

Blazor le permite crear interfaces de usuario web interactivas utilizando C # en lugar de JavaScript. Las aplicaciones Blazor se componen de componentes de interfaz de usuario web reutilizables implementados con C #, HTML y CSS. Tanto el código del cliente como el del servidor están escritos en C #, lo que le permite compartir código y bibliotecas.

Blazor es una característica de ASP.NET , el popular marco de desarrollo web que amplía la plataforma de desarrollo .NET con herramientas y bibliotecas para crear aplicaciones web.

**Ejecutar en WebAssembly o en el servidor**

Blazor puede ejecutar el código C # del lado del cliente directamente en el navegador mediante WebAssembly. Debido a que es .NET real que se ejecuta en WebAssembly, puede reutilizar el código y las bibliotecas de las partes del lado del servidor de su aplicación.

Alternativamente, Blazor puede ejecutar la lógica de su cliente en el servidor. Los eventos de la IU del cliente se envían de vuelta al servidor mediante SignalR, un marco de mensajería en tiempo real. Una vez que se completa la ejecución, los cambios requeridos en la interfaz de usuario se envían al cliente y se combinan en el DOM.

**WebAssembly**

Esta tecnología tiene la clave del funcionamiento de Blazor WebAssembly, el modelo más novedoso y en el que nos vamos a enfocar de ahora en adelante. WebAssembly (conocido también por su abreviatura Wasm) es un estándar que permite ejecutar código binario en un navegador web para ofrecer un rendimiento a priori mayor que Javascript.

El servidor web se encargará de enviar al navegador cliente que realice una petición a nuestra aplicación las librerías directamente compiladas (dlls), y el navegador a través de esta tecnología sabrá interpretar lo que queremos ejecutar. Como consecuencia, el servidor web se liberará de procesar lógica, ya que se realizará en el cliente.

Con esta aproximación, se sobreentiende la necesidad de mantener un back-end con el procesamiento de información que no se quiera llevar al cliente. Recordemos que las librerías compiladas son fácilmente decompilables, exponiendo nuestro código fuente a cualquier usuario de nuestra aplicación

**Warehouse:**

Una definición de warehouse en relación a datos, o más concretamente un data warehouse, podría ser la siguiente: un data warehouse es un sistema utilizado para reportar y analizar datos que funciona como un repositorio central de datos integrando una o más fuentes de información dispares. Almacenan datos actuales e históricos y son utilizados para realizar procesamientos y crear informes analíticos para la toma de decisiones de las organizaciones.

Un data warehouse cuenta con tres capas generales que, aunque no son obligatorias, sí son frecuentemente utilizadas, dependiendo de la arquitectura:

* Capa de integración en la que se almacenan y preparan los datos brutos extraídos.
* Capa principal del data warehouse donde los datos se adaptan al modelo de datos homogéneo de la empresa.
* Capa de data marts y strategic marts, que proporciona extractos específicos del data warehouse corporativo