1. ¿Qué es la programación Orientada a Objetos?

Es un paradigma de programación donde se describe un programa en términos de objetos o entidades que intervienen en el problema y sus responsabilidades.

El objetivo es simplificar la complejidad abtrayendo su conocimiento y su comportamiento y encapsulándolo como objetos. De alguna manera los objetos de la vida real se personifican aunque no siempre sean objetos concretos, algunas veces pueden ser objetos mas abstractos.

1. ¿Qué es una clase?

Es un molde a partir del cual puedo crear objetos. En la misma agrupo las características y comportamientos comunes a todos los objetos que instancien dicha clase.

1. ¿Qué es un objeto?

Es una instancia de una clase donde defino su estado (características) y su comportamiento (acciones que puede realizar el objeto).

**PILARES DEL POO (PROGRAMACION ORIENTADA A OBJETOS)**

1. ¿que es encapsulamiento?

Es el mecanismo por el cual aislamos el estado interno y el comportamiento de un objeto. Se busca proteger los datos y que solo se puedan modificar por usuarios que tengan el permiso para hacerlo.

1. ¿Qué es la herencia?

Nos permite crear clases hijas a partir de una super clase padre. Una clase hija hereda el estado y el comportamiento de la clase padre y agrega nuevos estados o funcionalidad. La herencia es muy importante para la reutilización de código.7

1. ¿Qué es la abstracción?

Cuando queremos modelar un objeto debemos pensarlo abstractamente y no en algo concreto. De esta manera extraemos sus características principales y su comportamiento (solo lo relevante para nuestra solución) y lo encapsulamos en un objeto

5) ¿Qué es el polimorfismo?

Es la capacidad de que objeto de diferentes clases respondan a un mismo método. Es muy usado en jerarquías de clases. Por ej podría crar una super clase vehiculo con un método Girar() y 2 clases hijas, motociclieta y automóvil. Ambas clases tendrán que implementar el método Girar() aunque cada clase tendrá una lógica diferente para el mismo método.

1. ¿Cuál es la diferencia entre una clase abstracta y una interface?

Las clases abstractas son aquellas que tienen algún método abstracto. Quiere decir que el o los métodos abstractos solo cuentan con la firma del mismo. Y la implementación del mismo se delega a la subclase hija que la herede. Por lo tanto una clase abst. No puede ser instanciada. No todos los métodos de una clase abstracta tienen que ser abstractos algunos métodos puede ser implementados dentro de la misma.

En cambio una interfaz es un contrato donde constan los métodos (y sus firmas) que debe implementar una clase que la implemente.

1. ¿Qué modificadores de acceso conocés? Explicar.

Public: puede ser accedido y modificado desde cualquier lugar.

Private: solo es accesible desde la misma clase.

Protected: solo es accesible desde la misma clase o sus clases heradades

Internal: Solo accesible para el mismo ensamblado. Dll o .exe.

1. ¿Qué es LINQ?

Es un lenguaje de consultas integrado propio de C# que nos permite obtener datos en diferentes fuentes.

1. ¿Qué es la programación Asíncrona? ¿Cual es la diferencia entre programación asíncrona y síncrona?

La programación asíncrona rompe con la secuencialidad del la programación síncrona al no producirse bloqueos a la espera de la resolución de una llamada o solicitud. Esto nos permite ejecutar solicitudes en paralelo lo que se traducirá en una mayor velocidad de carga.

1. ¿Qué ORM conocés? ¿Qué formas de trabajar en EntityFramework Conocés? ¿En qué casos de uso corresponde cada uno y cuales son las ventajas?

Un *ORM* es un modelo de programación que permite mapear las estructuras de una base de datos relacional (*SQL Server*, *Oracle*, MySQL, etc.), 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 con algunas ventajas de seguridad como ser la eliminación de inyección SQL.

En EF. Se puede trabajar como

code First: Primero creamos el código en nuestra apicacion y luego hacemos una migración a una base de datos. Útil cuando creamos un proyecto de cero.

Data base first: Partimos de una base de datos ya existente y a partir de ella se mapea a objetos en nuestra solucion.

11) ¿Qué patrones de diseño conocés? Repository, Unit of Work, CQRS.

SOLID

* [Principio de Responsabilidad Única](https://devexperto.com/principio-responsabilidad-unica/) (Single Responsibility Principle)

El principio de responsabilidad única se basa en que cada clase o método **sólo debe hacer una cosa**, sencilla y concreta. Si un objeto tiene un sólo cometido, éste será más fácil de mantener.

* [Principio Open/Closed](https://devexperto.com/principio-open-closed) (Open/Closed Principle)

El principio Open/Closed dice que una clase/método debe estar abierto a extensiones pero cerrado a modificaciones.

* [Principio de Sustitución de Liskov](https://devexperto.com/principio-de-sustitucion-de-liskov) (Liskov Substitution Principle)

Este principio trata sobre que los objetos de un desarrollo deberían ser reemplazables por instancias de sus subtipos sin alterar el correcto funcionamiento del desarrollo. Dicho de otro modo: **cualquier subclase debería poder ser sustituible por la clase padre**.

* [Principio de Segregación de Interfaces](https://devexperto.com/principio-de-segregacion-de-interfaces) (Interface Segregation Principle)

Este principio trata sobre algo parecido a SRP. Es mejor definir una serie de métodos abstractos a través de una serie de interfaces para que implementen nuestras clases. El objetivo no es otro que poder reutilizar estas interfaces en otras clases. Cada clase implementará las interfaces que necesite y use, **ninguna más**. No contendrá métodos que no utilice. De nuevo con el objetivo de aplicar sentido común y no ensuciar una clase con métodos o propiedades que no se utilizan.

* [Principio de Inversión de Dependencias](https://devexperto.com/principio-de-inversion-de-dependencias) (Dependency Inversion Principle)

El objetivo de este principio es conseguir desacoplar las clases de nuestro desarrollo. **Que una clase pueda funcionar por sí sola sin depender de otra**. Es difícil, pero en todo diseño de software al final suele existir un acoplamiento, pero hay que evitarlo en la medida de lo posible. Un sistema altamente acoplado es muy difícil de mantener.

1. ¿Cómo funciona la inyección de dependencias en C#? ¿Cuáles son las distintas formas de registrar un servicio?

**La inversión de dependencias es un principio** que describe un conjunto de técnicas destinadas a **disminuir el acoplamiento entre los componentes de una aplicación**. Un objeto no debe tener la necesidad de conocer como crar un objeto del cual depende. En C# podemos inyectar las dependencias como servicio. Ya sea en la class program o en start up dependiendo de la versión de . net core. Los servicios los podemos registrar con los siguientes ciclos de vida:

**Sigleton: solo se crea una instancia y siempre se inyecta esa misma.**

Scoped : Cuando registramos un tipo como Scoped, una instancia está disponible en toda la aplicación por solicitud. Cuando llega una nueva solicitud, se crea la nueva instancia. Add scoped especifica que un solo objeto está disponible por solicitud.

Transient :Cuando registramos un tipo como Transient, cada vez que se crea una nueva instancia. Transient crea una nueva instancia para cada servicio/controlador, así como para cada solicitud y cada usuario.

13) ¿Si tenés un servicio singleton que posee dependencias, las cuales son transient o scoped, que mecanismo utilizarías para poder realizar la inyección de dependencia?

1. ¿Qué es una web api? ¿Con qué protocolo se comunican? ¿Verbos?

es un mecanismo por el cual una organización puede exponer los servicios que brinda.

Se comunican mediante el protocolo Http.

Los verbos de uso mas frecuentes son:

GET: obtención de datos

POST: creación o modificación de datos

PUT: modificación de datos

DELETE: eliminación de un recurso.

PACH

15) ¿Qué es una arquitectura de microservicios? Diferencia entre microservicios y monolitos, ventajas y desventajas.

La arquitectura de microservicios es un **método de desarrollo de aplicaciones** software que funciona como un **conjunto de pequeños servicios que se ejecutan de manera independiente y autónoma, proporcionando una funcionalidad de negocio completa.** En ella, **cada microservicio es un código** que puede estar en un lenguaje de programación diferente, y **que desempeña una función específica**. Los microservicios **se comunican entre sí a través de APIs, y cuentan con sistemas de almacenamiento propios**, lo que evita la sobrecarga y caída de la aplicación.

Los microservicios han creado infraestructuras IT más adaptables y flexibles. Porque **si se quiere** **modificar solamente un servicio, no es necesario alterar el resto de la infraestructura**. Cada uno de los servicios se puede desplegar y modificar sin que ello afecte a otros servicios o aspectos funcionales de la aplicación.

Veamos qué ventajas y desventajas tiene la aplicación de una arquitectura de microservicios frente a otros tipos de arquitectura. Hay que tener en cuenta estos puntos a la hora de identificar qué tipo de arquitectura software será la mejor para un determinado proyecto u organización.

**Ventajas**

* **Modularidad:** al tratarse de servicios autónomos, se pueden desarrollar y desplegar de forma independiente. Además un error en un servicio no debería afectar la capacidad de otros servicios para seguir trabajando según lo previsto.
* **Escalabilidad:** como es una aplicación modular, se puede escalar horizontalmente cada parte según sea necesario, aumentando el escalado de los módulos que tengan un procesamiento más intensivo.
* **Versatilidad:** se pueden usar diferentes tecnologías y lenguajes de programación. Lo que permite adaptar cada funcionalidad a la tecnología más adecuada y rentable.
* **Rapidez de actuación:** el reducido tamaño de los microservicios permite un desarrollo menos costoso, así como el uso de “contenedores de software” permite que el despliegue de la aplicación se pueda llevar a cabo rápidamente.
* **Mantenimiento simple y barato:** al poder hacerse mejoras de un solo módulo y no tener que intervenir en toda la estructura, el mantenimiento es más sencillo y barato que en otras arquitecturas.
* **Agilidad:** se pueden utilizar funcionalidades típicas (autenticación, trazabilidad, etc.) que ya han sido desarrolladas por terceros, no hace falta que el desarrollador las cree de nuevo.

**Desventajas**

* **Alto consumo de memoria:** al tener cada microservicio sus propios recursos y bases de datos, consumen más memoria y CPU.
* **Inversión de tiempo inicial:** al crear la arquitectura, se necesita más tiempo para poder fragmentar los distintos microservicios e implementar la comunicación entre ellos.
* **Complejidad en la gestión:** si contamos con un gran número de microservicios, será más complicado controlar la gestión e integración de los mismos. Es necesario disponer de una centralización de trazas y herramientas avanzadas de procesamiento de información que permitan tener una visión general de todos los microservicios y orquesten el sistema.
* **Perfil de desarrollador:**los microservicios requieren desarrolladores experimentados con un nivel muy alto de experiencia y un control exhaustivo de las versiones. Además de conocimiento sobre solución de problemas como latencia en la red o balanceo de cargas.
* **No uniformidad:** aunque disponer de un equipo tecnológico diferente para cada uno de los servicios tiene sus ventajas, si no se gestiona correctamente, conducirá a un diseño y arquitectura de aplicación poco uniforme.
* **Dificultad en la realización de pruebas:** debido a que los componentes de la aplicación están distribuidos, las pruebas y test globales son más complicados de realizar.
* **Coste de implantación alto:** una arquitectura de microservicios puede suponer un alto coste de implantación debido a costes de infraestructura y pruebas distribuidas.

16) Comunicación entre microservicios.

17) ¿Conocés OpenShift?

18) ¿Qué es la normalización en base de datos?

La normalización de la base de datos es el proceso de organizar los datos en tablas de tal manera que los resultados del uso de la base de datos sean siempre inequívocos y según lo previsto.

Evitar la redundancia

Dependencia únicamente de la PK.

Valores atomicos.

Cumplir con las 3FN.

19) ¿Qué es una tabla desnormalizada? ¿Casos de uso?

Es una tabla que no cumple con las 3FN, (hay redundancia de datos). Esto puede hacerse deliberadamente con el objetivo de acelerar una búsqueda en el caso de un dato muy consoltado. Asi trabaja Nosql.

20) ¿Qué son los índices?

Son estructuras que nos facilitan y aceleran el acceso a datos. Ademas de brindarnos otras funcionalidades como la unicidad.

21) ¿Qué es una vista?

Es una visión parcial o completa de nuestra base de datos que se brinda generalmente con el objetivo de proteger datos sensibles del negocio.

22) ¿Qué es un Stored Procedure?

Es un código SQL pre guardado el cual puede ser llamado y ejecutado. Esto nos ayuda a reutilizar código ya que no tenemos la necesidad de escribir la misma query varias veces o para querys muy costosas nos brinda una mejor performance.

23) ¿Cuál es la diferencia entre correr una query y un stored procedure?

El sp es mas rápido ya que se compila solo una vez y luego en cada llamada solo se ejecuta. En cambio una query se debe compilar cada vez que se ejecute.

24) Tablas virtuales.