TEORICA

Describa los conceptos de los siguientes temas:

1. **Microservicios**

Los microservicios son una arquitectura de desarrollo de software donde una aplicación grande se divide en componentes más pequeños, independientes y que se comunican entre sí a través de APIs. Cada microservicio es responsable de una funcionalidad específica y se puede desarrollar, desplegar y escalar de manera independiente. Esta arquitectura ofrece flexibilidad, escalabilidad y facilita el mantenimiento y actualización del sistema.

1. **SQL Server**

SQL Server es un sistema de gestión de bases de datos relacional (RDBMS) desarrollado por Microsoft. Es utilizado para almacenar y recuperar datos según lo soliciten otras aplicaciones de software. SQL Server soporta una amplia variedad de transacciones, permite consultas complejas y proporciona herramientas para el análisis y reporte de datos. Ofrece características como la integridad de los datos, la seguridad y la recuperación ante desastres.

1. **.NET Core**

.NET Core es un framework de desarrollo de código abierto y multiplataforma creado por Microsoft. Permite a los desarrolladores crear aplicaciones modernas, de alto rendimiento y seguras para Windows, macOS y Linux. .NET Core es la base para el desarrollo de aplicaciones web, servicios en la nube, aplicaciones de escritorio, juegos y más. Es conocido por su rendimiento mejorado y su flexibilidad en comparación con su predecesor, .NET Framework.

1. **Azure DevOps**

Azure DevOps es un conjunto de herramientas de desarrollo en la nube de Microsoft que facilita la colaboración en equipo y la entrega continua de software. Incluye servicios para la planificación de proyectos, la gestión de código fuente, la integración y entrega continuas (CI/CD), y la monitorización de aplicaciones. Azure DevOps proporciona un entorno completo para el desarrollo ágil y DevOps, permitiendo a los equipos desarrollar, probar y desplegar aplicaciones de manera eficiente.

1. **Componente (Frontend)**

Un componente en el contexto del frontend es una unidad modular y reutilizable de la interfaz de usuario (UI). Los componentes encapsulan el comportamiento y la presentación de un elemento de la UI, permitiendo a los desarrolladores construir interfaces complejas a partir de piezas pequeñas y mantenibles. Los componentes pueden incluirse en bibliotecas y frameworks como React, Angular y Vue.js, y facilitan el desarrollo de aplicaciones escalables y fáciles de mantener.

1. **React Hooks**

React Hooks son funciones especiales introducidas en React a partir de la versión 16.8, que permiten a los desarrolladores utilizar estado y otras características de React sin escribir clases. Los hooks más comunes son **useState** para manejar el estado local, **useEffect** para realizar efectos secundarios en componentes funcionales, y **useContext** para acceder al contexto. Los hooks simplifican el código y promueven la reutilización de lógica entre componentes.

1. **Principios SOLID**

SOLID es un acrónimo que representa cinco principios de diseño de software orientado a objetos que promueven la legibilidad, escalabilidad y mantenimiento del código. Los principios son:

* **S**ingle Responsibility Principle (SRP): Un módulo o clase debe tener una única responsabilidad o motivo para cambiar.
* **O**pen/Closed Principle (OCP): El software debe estar abierto a la extensión, pero cerrado a la modificación.
* **L**iskov Substitution Principle (LSP): Los objetos de una clase derivada deben poder reemplazar a los objetos de la clase base sin alterar el comportamiento del programa.
* **I**nterface Segregation Principle (ISP): Los clientes no deben estar obligados a depender de interfaces que no usan.
* **D**ependency Inversion Principle (DIP): Los módulos de alto nivel no deben depender de módulos de bajo nivel. Ambos deben depender de abstracciones.

1. **TDD (Test-Driven Development)**

TDD es una metodología de desarrollo de software en la que las pruebas unitarias se escriben antes de que se escriba el código de producción. El ciclo TDD consiste en tres pasos: escribir una prueba que falle, escribir el código mínimo necesario para pasar la prueba, y refactorizar el código para cumplir con los estándares de calidad. TDD ayuda a garantizar que el código es testable y que cumple con los requisitos desde el principio, mejorando la calidad y la mantenibilidad del software.

**Practico**

La empresa NEXTI te encarga la construcción de un aplicativo para la venta de entradas para acontecimientos deportivos y culturales. Se quiere que la aplicación funcione en la web, de tal manera que los usuarios accedan a través de un navegador.

* 1. Los usuarios podrán registrar, eliminar, modificar, consultar un evento y consultar todos los eventos
  2. Las características del evento son:
     + Fecha evento
     + Lugar evento
     + Descripción del evento
     + Precio
  3. La eliminación del evento debe ser lógica a nivel de BDD.

Se valora la implementación de los siguientes puntos:

1. Implementación de estado global de la aplicación con Redux o Zustand
2. Consulta masiva con paginación
3. Implementación de variables de entorno
4. Pruebas unitarias
5. Docker
6. Clean Architecture

Tecnología a utilizar:

* + Frontend con - React js
  + Backend C# Asp .Net Core 6 (API Rest) - Microservicios
  + Bases de datos SQL Server