**API Rest (Conceptos) :**

● Explique los principios básicos de REST y los diferentes métodos HTTP utilizados en una API REST ful.

**REST (Representational State Transfer)** es un estilo arquitectónico para servicios web que define cómo interactúan cliente y servidor de manera stateless y escalable.

Métodos HTTP:

GET Obtener recursos

POST Crear nuevos recursos

PUT Actualizar/reemplazar recurso completo

PATCH Actualización parcial

DELETE Actualización parcial

● ¿Qué es CORS (Cross-Origin Resource Sharing) y cómo se configura en una API REST? Realizar un ejemplo de configuración de CORS

**CORS (Cross-Origin Resource Sharing)** es un mecanismo de seguridad del navegador que controla qué dominios pueden acceder a recursos de tu API desde JavaScript.

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers();

builder.Services.AddCors(options =>

{   
options.AddPolicy("MiPoliticaCORS", policy =>

{   
policy.WithOrigins("https://miapp.com", "https://admin.miapp.com")

.WithMethods("GET", "POST", "PUT", "DELETE")

.WithHeaders("Content-Type", "Authorization")

.AllowCredentials();

});

});

var app = builder.Build();

*// Aplicar la política específica*

app.UseCors("MiPoliticaCORS");

app.UseRouting(); // app.MapControllers(); // app.Run();

**1.2. Arquitectura y Patrones (Conceptos):**

● Diferencias entre una arquitectura monolítica y una arquitectura de microservicios.

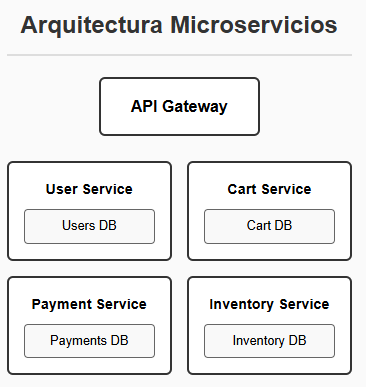
Ejemplos de aplicaciones.

Realizar un diagrama para diferenciar

| **Aspecto** | **Monolítico** | **Microservicios** |
| --- | --- | --- |
| **Estructura** | Una aplicación única | Múltiples servicios independientes |
| **Base de datos** | Una sola, compartida | Una por servicio |
| **Despliegue** | Todo junto | Cada servicio por separado |
| **Escalabilidad** | Escala toda la app | Escala servicios específicos |
| **Comunicación** | Llamadas internas | HTTP/REST, mensajería |
| **Tecnología** | Una sola (ej: .NET) | Múltiples (Java, Python, Node.js) |
| **Complejidad** | **Simple** al inicio | **Compleja** desde el inicio |
| **Equipos** | Un equipo grande | Múltiples equipos pequeños |
| **Fallas** | Si falla, todo se cae | Falla un servicio, otros siguen |
| **Testing** | Más fácil | Más complejo |

**Regla simple**: **Empieza monolítico**, migra a microservicios cuando sea necesario.





● Gráfica un diagrama de arquitectura considerando el uso de dockers, microservicios, APIS, Integraciones, Controlador, base de datos, monitoring, etc. para un proyecto web.

Cliente

Load Balancer

API Gateway

APIs Externas

Controllers

Microservicios Docker

Base de Datos

Monitoring

**1.3. SQL Server:**

● Crear un trigger que genere un registro en otra tabla cuando se actualizan los libros, guardando los datos del registro antes de actualizar, incluir el usuario y fecha de actualización.

create database biblioteca

use biblioteca;

CREATE TABLE Libro (

ID int IDENTITY(1,1) PRIMARY KEY,

Titulo varchar(200),

Autor varchar(100),

Precio decimal(10,2),

Stock int

);

-- 2. CREAR TABLA DE LIBROS

CREATE TABLE HistorialLibros (

ID int IDENTITY(1,1) PRIMARY KEY,

LibroID int,

TituloAnterior varchar(200),

AutorAnterior varchar(100),

PrecioAnterior decimal(10,2),

StockAnterior int,

Usuario varchar(100),

FechaActualizacion datetime

);

go

CREATE TRIGGER TR\_Actualizarlibros

ON Libro

AFTER UPDATE

AS

BEGIN

INSERT INTO HistorialLibros (

LibroID,

TituloAnterior,

AutorAnterior,

PrecioAnterior,

StockAnterior,

Usuario,

FechaActualizacion

)

SELECT

d.ID,

d.Titulo,

d.Autor,

d.Precio,

d.Stock,

SYSTEM\_USER,

GETDATE()

FROM deleted d;

END;

------------ejemplo de uso----------------------

INSERT INTO Libro (Titulo, Autor, Precio, Stock)

VALUES ('El Quijote', 'Cervantes', 25.50, 10);

-------------- Activacion del Trigger------------

UPDATE Libro

SET Precio = 30.00, Stock = 8

WHERE ID = 1;

-------------ver el registro-----------------

SELECT \* FROM Libro;

**● Mencionar 5 buenas prácticas a considerar en una sentencia de Base de Datos (por ejemplo: no aplicar cursores.)**

1- Poner condición con WHERE al usar UPDATE o DELETE

2- Evitar SELECT \*, usar solo columnas necesarias:

3- Usar LIMIT o TOP para pruebas (evitar traer muchos datos):

4- Agregar índices en columnas que usas mucho en WHERE o JOIN

5- Hacer copias de seguridad antes de hacer cambios grandes:

● **Mencionar 3 casos donde se recomendaría utilizar Triggers.**

* Cuando quieres que ciertos campos se completen solos:
* Calcular el promedio de calificaciones de un estudiante
* Notificar cuando se registra una venta grande

**1.4 Herramientas:**

Lista herramienta que utilizas para las siguientes tareas:

● Gestión de Proyectos / Actividades

● Gestión de Tareas

● Editores, IDE

● Herramientas para Desarrollo

● Herramientas para Base de Datos

● Herramientas para API

| **Tarea** | **Herramienta recomendada** |
| --- | --- |
| **Gestión de Proyectos / Actividades** | Trello, Jira |
| **Gestión de Tareas** | Notion, ClickUp |
| **Editores / IDE** | Visual Studio, VS Code, IntelliJ, NetBeans |
| **Herramientas para Desarrollo** | Git, GitHub, Postman |
| **Herramientas para Base de Datos** | SQL Server Management Studio (SSMS) ,oracle, MongoDB |
| **Herramientas para API** | Postman, Insomnia, Swagger |