Evaluacion Teorica

1.1. API Rest (Conceptos):

- Explique los principios básicos de REST y los diferentes métodos HTTP utilizados en una API RESTful.

PRINCIPIOS

- Arquitectura Cliente – Servidor: Podemos separar frontend y backend, podemos escalar independientemente.
- Sin Estado: El servidor no guarda sesión en cada petición, solo depende de un token, ejemplo claro en un token JWT.
- Representación. – El request y response se envían en formato Json.

METODOS HTTP

Los verbos mas usados son

- GET: Usado para obtener datos
- POST: Usado para crear y actualizar dato
- PUT: Usado Para actualizar dato
- DELETE: Usado para eliminar un dato

NOTA: Para mejorar la seguridad podemos sectorizando los verbos por roles cliente, empleado y administrador.

En donde el cliente puede solo visualizar (GET).

El empleado puede consultar,crear y editar (GET, POST, PUT)

El administrador de sistema puedo usar todos los verbos.

- ¿Qué es CORS (Cross-Origin Resource Sharing) y cómo se configura en una API REST?
  Controla la seguridad bloqueando las aplicaciones web que no estén registradas.

  Configuracion:

  - Instalar o configurar CORS en el backend
  - Agregar un middleware de CORS

    a) En ASP.NET en el archivo Program.cs registro la política de esta manera

```
builder.Services.AddCors(options =>
{
  options.AddPolicy("PoliticaFrontend",
    policy =>
    {
      policy.WithOrigins("https://prueba.com")
        .WithHeaders("Content-Type", "Authorization");
        . WithMethods("GET", "POST")
    });
});
```

b) En el pipeline
app.UseCors("PoliticaFrontend");

Con esto digo que todos los que vienen de https://prueba.com aceptare Header de tipo "Content-Type", "Authorization" y solo los métodos GET Y POST

1.2. **Arquitectura y Patrones:**

1.2.1. Diferencias entre una arquitectura monolítica y una arquitectura de microservicios. Ejemplos de aplicaciones.-

Diferencia	Arquitectura Monolitica	Arquitectura Microservicios
Aplicacion	Toda la aplicación se construye en una sola solución (front y back juntos)	Las aplicaciones son independientes
Dependencias	Los módulos están unidos sin un estándar, un cambio en un servicio puede afectar a otro servicio.	Los servicios son independientes, cada uno puede escalar sin afectar a los demas
Escalabilidad	Se requiere escalar toda la aplicación	Solo se escala el servicio que lo necesita
Despliegue	Solo tiene un solo despliegue para toda la aplicación	Cada servicio puede desplegarse de forma independiente
Mantenimiento	Se vuelve complicado cuando la aplicación crece	Mas fácil de mantener, cada despliegue puede ser independiente sin afectar la funcionalidad de la aplicacion
Tecnologia	Solo tiene un stack	Cada servicio puede usar usar diferente tecnología (.NET, NodeJs, Spring Boot)
Riesgo	Al compilar se obtiene 1 solo dll, si tiene error un método el sistema se cae	Al tener separado los servicios ya no dependen de un archivo, solo se cae el servicio con error los demás servicios siguen ejecutándose, la aplicación sigue ejecutanose normalmente

Ejemplos:

Monoliticas (IDE antiguos)

- VB6
- Visual Fox Pro
- COBOL
- Desarrollo en .Net Framework MVC web y capas dentro de la misma solución

Aplicaciones del estado antiguos (RENIEC)

Microservicios (IDE modernos)

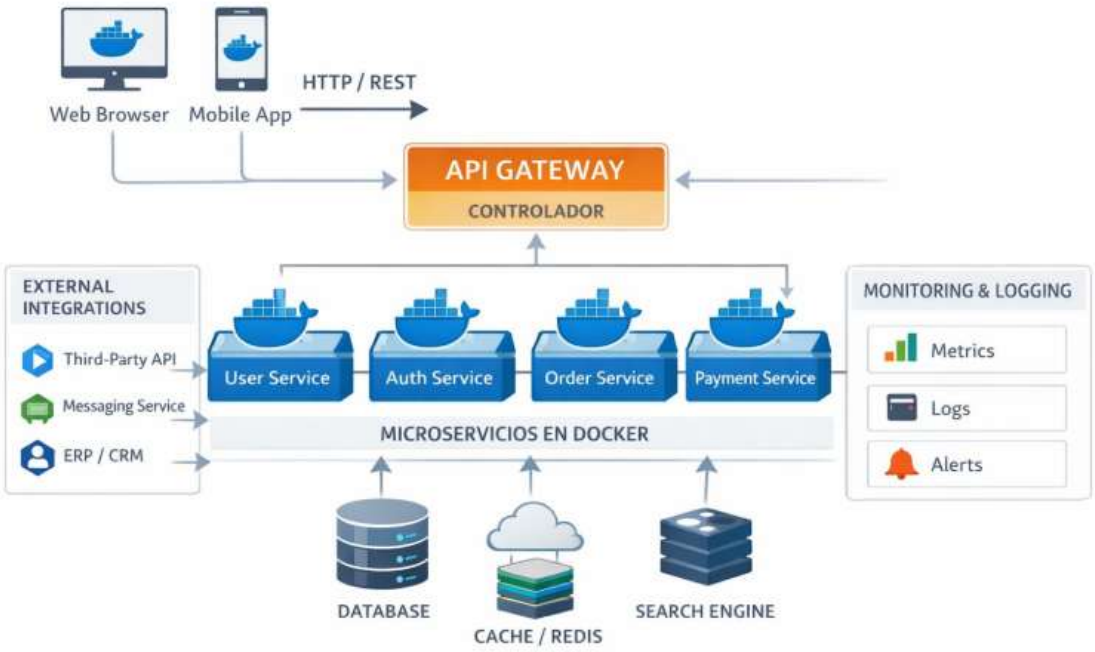
- Visual Studio 2015 en adelante
- Visual Code
- IntelliJ IDEA
- Eclipse y Net Beans a partir de JAVA 8 en adelante

Whatsapp

NetFlix

Facebook + Messenger

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



1.2.3. Explica qué es el patrón Backend for Frontend (BFF), incluyendo su propósito y diferencias con una API tradicional monolítica. Describe cómo lo implementarías en un software de ventas que soporta múltiples interfaces.

Es un backend especifico para cada tipo de frontend.

PROPOSITO

- Optimiza respuesta de cada interfaz
- Reduce lógica con el frontend
- Evita sobrecarga de datos
- Mejora seguridad y control

DIFERENCIAS

Caracteristica	API Monolitica	BFF
Cantidad de apis	Una sola	Varias por frontend
Acoplamiento	Alto	Bajo
Optimizacion por cliente	No	Si
Logica en frontend	Alta	Baja
Escalabilidad	Limitada	Alta
Evolucion de UI	Dificil	Flexible

1.2.4. Describe qué es el patrón Domain-Driven Design (DDD), incluyendo sus conceptos claves (entidades, agregados, contextos delimitados). Explica cómo lo aplicarías para modelar el dominio de un software de ventas.

1.3. SQL Server

1.3.1. Escriba una consulta T-SQL que recupere los 10 productos más vendidos en el último trimestre, incluyendo el nombre del producto, la cantidad vendida y el total de ingresos generados. La base de datos tiene las tablas Productos (con columnas IdProducto, NombreProducto,

```
SELECT TOP 10
  p.NombreProducto,
  SUM(v.Cantidad) AS CantidadVendida,
  SUM(v.Cantidad * p.Precio) AS TotalIngresos
FROM Ventas v
INNER JOIN Productos p
  ON v.IdProducto = p.IdProducto
WHERE v.FechaVenta >= DATEADD(QUARTER, -1, GETDATE())
GROUP BY p.NombreProducto
ORDER BY CantidadVendida DESC
```

1.3.2. Cree un procedimiento almacenado que reciba como parámetro 2 fechas (inicio y Fin) y devuelva una lista de todos los préstamos de libros realizados agrupados por libro y por autor, incluyendo los libros que no se prestaron, ordenarlos por el libro más prestado y apellidos del autor.

```
CREATE PROCEDURE sp_PrestamosPorLibroYAutor
  @FechaInicio DATE,
  @FechaFin    DATE
AS
BEGIN
  SET NOCOUNT ON
```

```
  SELECT
    l.Titulo AS Libro,
    a.Apellido + ', ' + a.Nombre AS Autor,
    COUNT(p.IdPrestamo) AS TotalPrestamos
  FROM Libros l
  INNER JOIN Autores a
    ON l.IdAutor = a.IdAutor
  LEFT JOIN Prestamos p
    ON l.IdLibro = p.IdLibro
   AND p.FechaPrestamo BETWEEN @FechaInicio AND @FechaFin
  GROUP BY
    l.Titulo,
    a.Apellido,
    a.Nombre
  ORDER BY
    TotalPrestamos DESC,
    a.Apellido ASC
END
```

1.3.3. 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 TRIGGER trg_AuditoriaUpdateLibros
ON Libros
AFTER UPDATE
AS
BEGIN
  SET NOCOUNT ON

  INSERT INTO Libros_Auditoria
  (
    IdLibro,
    TituloAnterior,
    IdAutorAnterior,
    AnioAnterior,
    PrecioAnterior,
    UsuarioModifico,
    FechaModifico
  )
  SELECT
    d.IdLibro,
    d.Titulo,
    d.IdAutor,
    d.AnioPublicacion,
    d.Precio,
```

```
    SYSTEM_USER,
    GETDATE()
  FROM DELETED d;
END;
GO
```