Microservicios-REST

Grupo 4

karol Geraldine Ceballos Castro
Angela Catalina Llaña Arciniegas
Brayan Steven Carrillo Mora

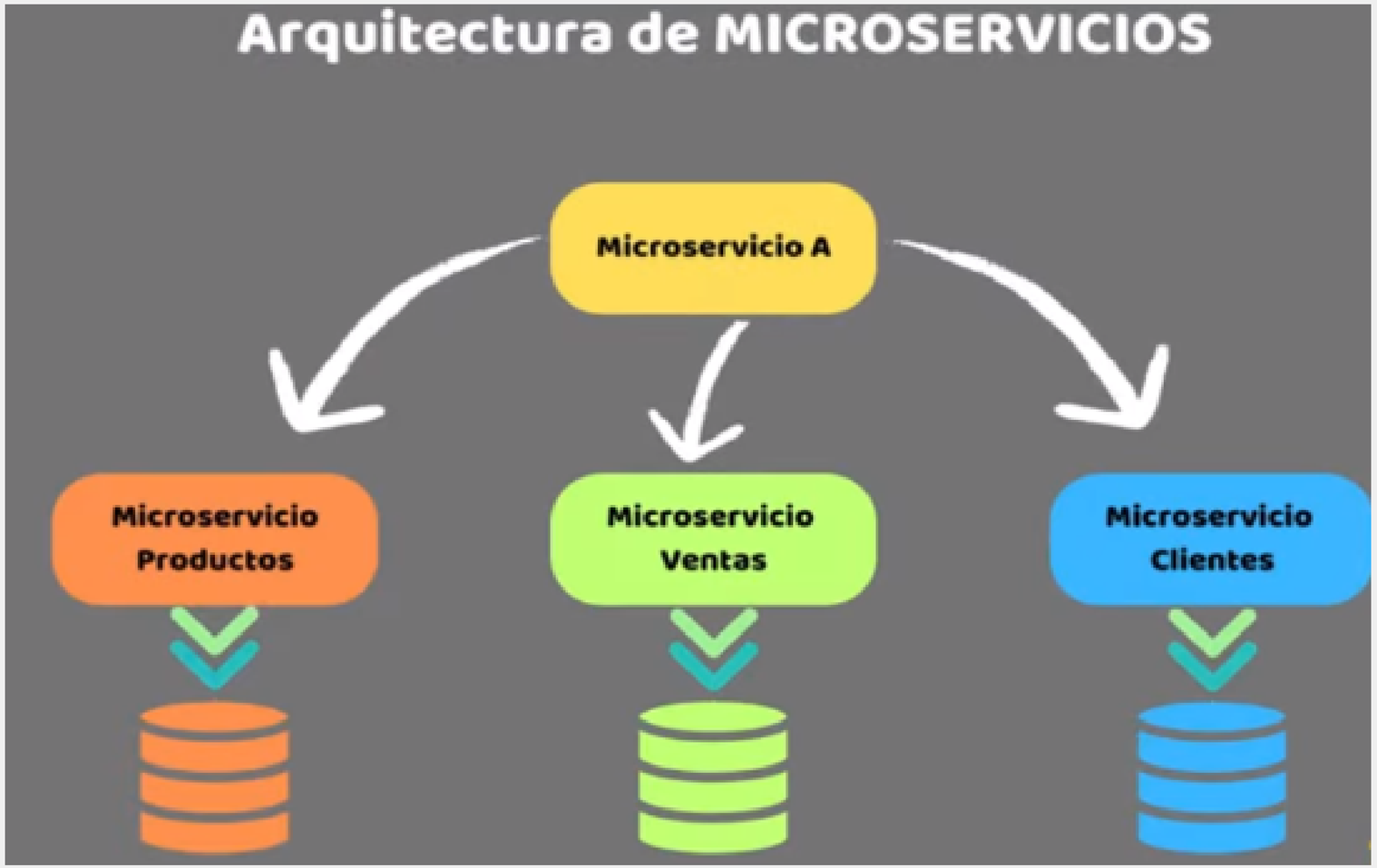
- toda la aplicación está contenida en un solo modelo de clase o proyecto.
- La lógica de negocio de la aplicación está integrada en una sola entidad, lo que puede dificultar su mantenimiento y escalabilidad a medida que la aplicación crece.
- Todos los datos de la aplicación se almacenan en una única base de datos centralizada, lo que puede generar cuellos de botella y problemas de rendimiento a medida que aumenta la carga de datos y usuarios.
- el codigo de la aplicación se compila en un solo archivo ejecutable lo que puede dificultar la distribución y la actualización de los datos

SOA(Arquitectura orientada a servicios)

- se centra en la creación de servicios independientes y reutilizables que se pueden compartir entre diferentes sistemas dentro de una organización
- .
- Los servicios suelen ser grandes y encapsulan una parte específica de la lógica empresarial. Estos servicios se comunican entre sí a través de interfaces bien definidas
- los servicios pueden depender uno del otro para funcionar correctamente

MICROSERVICIOS

- Cada microservicio se desarrolla, empaqueta y despliega de forma independiente.
- A diferencia de los servicios en una arquitectura SOA, los microservicios son más pequeños y encapsulan funcionalidades específicas de negocio..
- Se comunican entre sí mediante interfaces claramente definidas, como servicios web (SOAP o REST), lo que facilita la interoperabilidad.
- La comunicación entre los diferentes microservicios suele ser asincrónica, lo que permite una mayor flexibilidad y eficiencia en el intercambio de datos y la gestión de solicitudes.


Arquitectura de MICROSERVICIOS
Microservicio A
Microservicio Productos
Microservicio Ventas
Microservicio Clientes

Ventajas

- Escalabilidad
- Despliegue independiente
- Resistencia a fallos
- Reutilización de código
- Mejora la colaboración
- flexibilidad
- Reutilización del codigo

Desventajas

- Fallos de comunicación entre microservicios
- Requiere mucha más infraestructura
- Complejidad que puede llegar a alcanzar

REST/JSON

1990

Surgimiento del Protocolo de Transferencia de Hipertexto (HTTP) como el protocolo fundamental para la comunicación en la World Wide Web.

1990s:

Aparición del Protocolo Simple de Acceso a Objetos (SOAP) como estándar para el intercambio de mensajes entre aplicaciones distribuidas, utilizando XML como formato de datos

2000s

Propuesta de Roy Fielding de la arquitectura REST (Transferencia de Estado Representacional) como alternativa a SOAP, basada en principios web fundamentales como el uso de URIs y métodos HTTP.

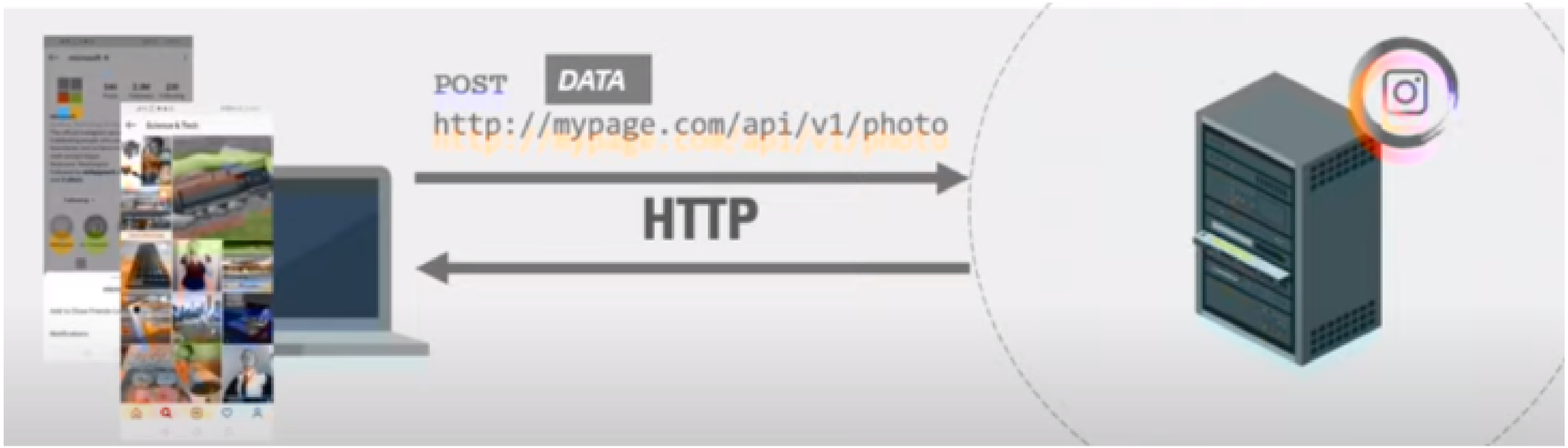
2000s

Incremento de la popularidad de JSON como un formato ligero y legible para el intercambio de datos, superando en algunos casos a XML.

actualidad

Estandarización de REST y JSON con la aparición de especificaciones comunes como OpenAPI y JSON Schema.

Arquitectura REST Api



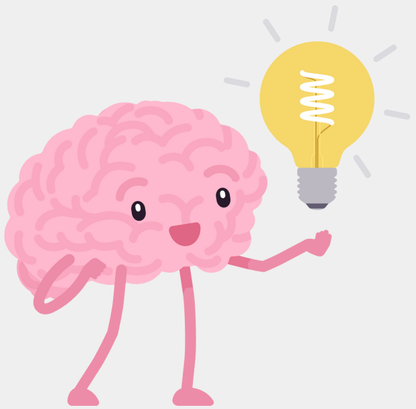
- Operaciones http: GET, POST, PUT/PATCH, DELETE
- formato de intercambio de datos: Json,XML,texto plano,texto binario
- Separación clara entre el cliente (aplicación web o móvil) y el servidor, permitiendo una comunicación fluida y eficaz.
- uso de cache para mejorar la eficiencia reduciendo la carga del servidor

ventajas

- Simplicidad y uniformidad
- Visibilidad y transparencia
- Interoperabilidad
- Facilidad de prueba y depuración
- Uso de caché
- Protección de datos durante la transferencia

desventajas

- Posible rigidez en el desarrollo
- Falta de conocimientos
- Complejidad en la gestión del estado



Spring boot

Origenes de spring boot

Este framework surgió en 2014 como un proyecto para simplificar el desarrollo de aplicaciones Spring, construido sobre el éxito del framework Spring original en 2002.

Caracteristicas

- Autoconfiguración: Spring Boot detecta automáticamente la configuración requerida según las dependencias y la infraestructura, minimizando la necesidad de configuración manual.
- Inicio rápido: Proporciona servidores web embebidos, permitiendo el inicio rápido de aplicaciones sin la necesidad de configurar servidores por separado.
- Gestión de Dependencias: Utiliza Maven o Gradle para gestionar dependencias, e incluye iniciadores de dependencias para simplificar el proceso de agregar librerías compatibles.

Ventajas

- Facilidad de uso
- Configuración automática
- Compatibilidad con microservicios
- Servidor integrado

Desventajas

- Spring Boot puede consumir muchos recursos
- Escalabilidad limitada
- Curva de aprendizaje pronunciada

```java
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Producto {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nombre;
    private double precio;

    // Getters y Setters
}
```

```java
import org.springframework.data.jpa.repository.JpaRepository;

public interface ProductoRepository extends JpaRepository<Producto, Long> {
    // Métodos adicionales de consulta si es necesario
}
```

```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class ProductoService {
    @Autowired
    private ProductoRepository productoRepository;

    public Producto guardarProducto(Producto producto) {
        return productoRepository.save(producto);
    }

    public Producto obtenerProductoPorId(Long id) {
        return productoRepository.findById(id).orElse(null);
    }

    // Otros métodos de servicio según sea necesario
}
```

```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/productos")
public class ProductoController {
    @Autowired
    private ProductoService productoService;

    @PostMapping
    public Producto agregarProducto(@RequestBody Producto producto) {
        return productoService.guardarProducto(producto);
    }

    @GetMapping("/{id}")
    public Producto obtenerProducto(@PathVariable Long id) {
        return productoService.obtenerProductoPorId(id);
    }

    // Otros métodos del controlador según sea necesario
}
```

Angular

Historia de Angular:

- Inicio en 2010 por Misko Hevery y Adam Abrons en Google.
- Evolución desde AngularJS (2012) a Angular 2 (2016) y versiones posteriores.
- Enfoque en la creación de aplicaciones web dinámicas y ricas en funcionalidades.

MVC

Modelo: TypeScript que define la estructura y el comportamiento de los datos.

- También puede ser un servicio que interactúa con el backend para obtener o manipular los datos.

Vista: Las vistas están representadas por componentes, clases TypeScript que contienen la lógica de presentación y el HTML asociado.

- Cada componente está vinculado a una vista específica y maneja las interacciones del usuario en esa vista.

Controlador: Maneja la lógica de presentación y el comportamiento del componente.

- En Angular, los componentes encapsulan tanto la lógica de presentación como el comportamiento del controlador.
- Para la lógica de negocio más compleja o compartida entre varios componentes, se pueden utilizar servicios,

Ventajas

- Tipado estático con TypeScript
- Arquitectura basada en componentes
- Inyección de dependencias
- Soporte de bibliotecas y herramientas
- Desarrollo de aplicaciones de una sola página (SPA).

DESVENTAJAS

- tamaño de la aplicación
- Dependencia de angular
- Documentación y recursos complejos
- Complejidad en proyectos pequeños


MySQL®

Orígenes:

- Surgió del modelo relacional propuesto por Edgar F. Codd en la década de 1970.
- IBM System R fue el primer sistema RDBMS basado en este modelo, seguido por el producto exitoso de Oracle Corporation en 1979.

Aplicaciones Actuales:

- Ampliamente utilizado en una variedad de aplicaciones, desde sistemas empresariales hasta aplicaciones web y móviles.
- Sistemas populares que admiten SQL incluyen PostgreSQL, SQLite, MySQL y Microsoft SQL Server.

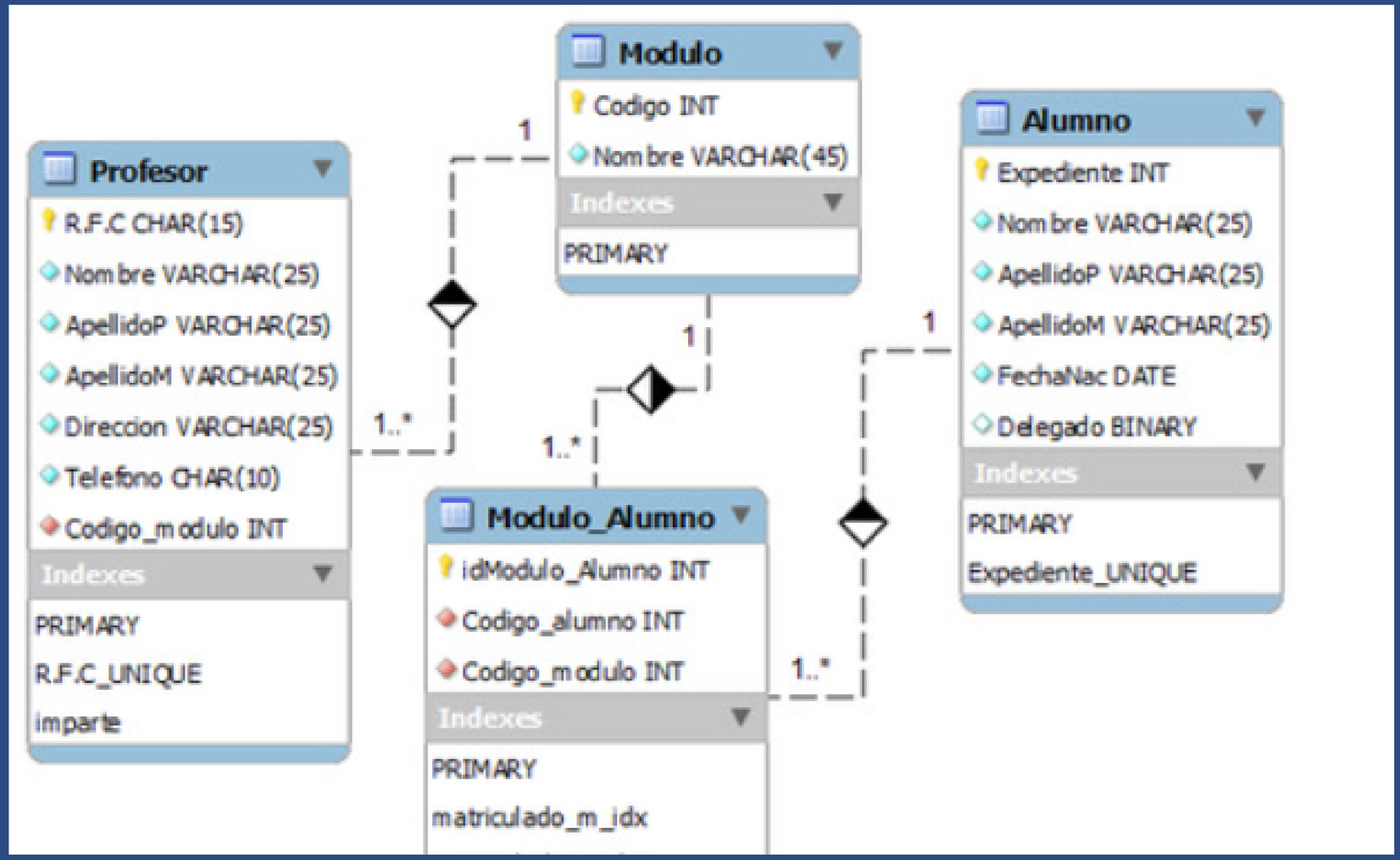
- Funcionalidades Clave:
- Operaciones CRUD (Creación, Lectura, Actualización, Eliminación) son fundamentales en la administración de datos.
- Capacidad para realizar consultas (queries) para obtener información específica de la base de datos.

Ventajas

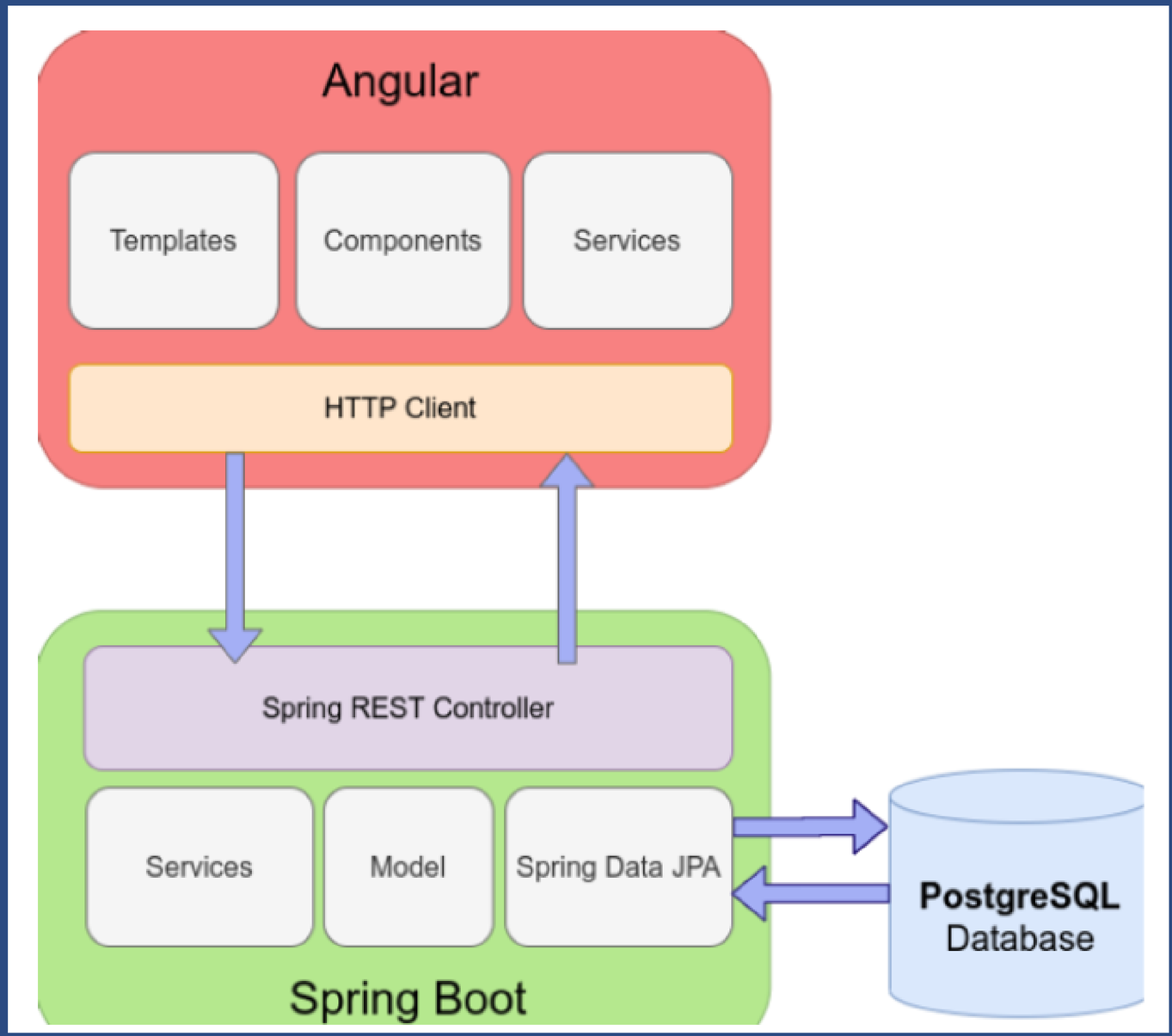
- Bajo costo en requerimientos
- Baja probabilidad de corromper datos
- Inyección de dependencias
- Es encriptada y confible

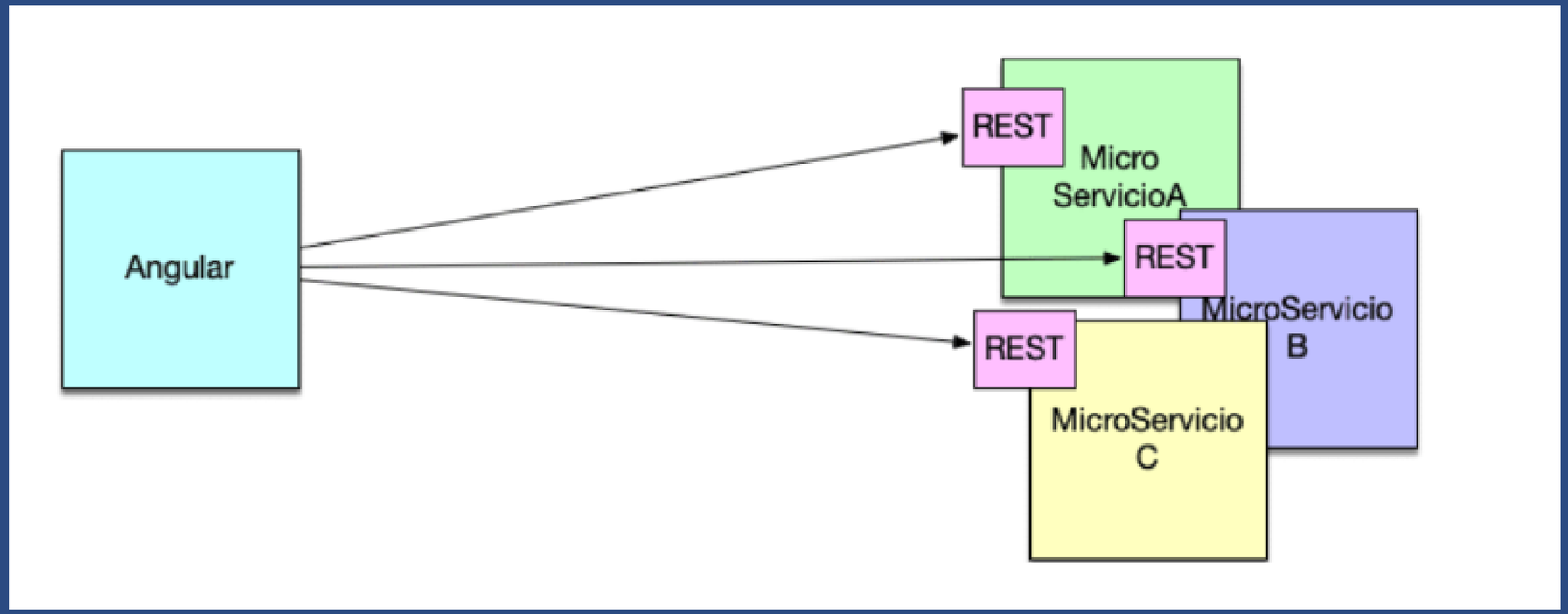
DESVENTAJAS

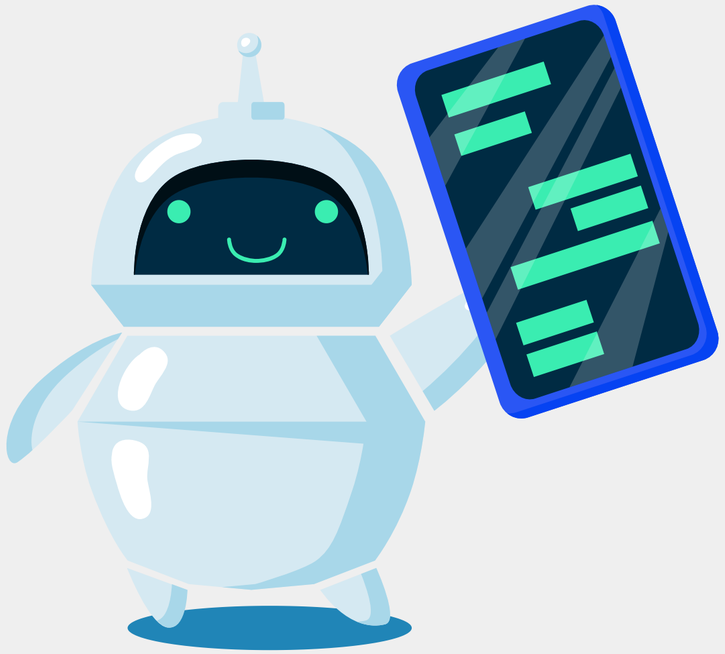
- No maneja de manera tan eficiente una base de datos con un tamaño muy grande.
- varias de las utilidades de MySQL no están documentadas
- No es del todo intuitivo en comparación de otros programas


Modulo
Codigo INT
Nombre VARCHAR(45)
Indexes
PRIMARY
Profesor
R.F.C CHAR(15)
Nombre VARCHAR(25)
ApellidoP VARCHAR(25)
ApellidoM VARCHAR(25)
Direccion VARCHAR(25)
Telefono CHAR(10)
Codigo_modulo INT
Indexes
PRIMARY
R.F.C_UNIQUE
imparte
Alumno
Expediente INT
Nombre VARCHAR(25)
ApellidoP VARCHAR(25)
ApellidoM VARCHAR(25)
FechaNac DATE
Delegado BINARY
Indexes
PRIMARY
Expediente_UNIQUE
Modulo_Alumno
idModulo_Alumno INT
Codigo_alumno INT
Codigo_modulo INT
Indexes
PRIMARY
matriculado_m_idx
1
1..*
1
1..*
1
1..*

¿Como se relacionan?


Angular
Templates
Components
Services
HTTP Client
Spring REST Controller
Services
Model
Spring Data JPA
Spring Boot
PostgreSQL
Database


Angular
REST
Micro
ServicioA
REST
MicroServicio
B
REST
MicroServicio
C



ALGUNAS SITUACIONES Y/O PROBLEMAS DONDE SE PUEDEN APLICAR

- Sistema de gestión de inventarios
- Sistema de gestión de usuarios y autenticación:
- Gestión de libros

Principios SOLID

¿Cómo se pueden aplicar en los microservicios?

S - Responsabilidad única

Los micorservicios deben ser especialistas o responsables de una única funcionalidad.

O - Abierto/Cerrado

Toda nueva responsabilidad debería agregarse a un nuevo microservicio en lugar de ampliar la responsabilidad de uno ya existente.

L - Sustitución de Liskov

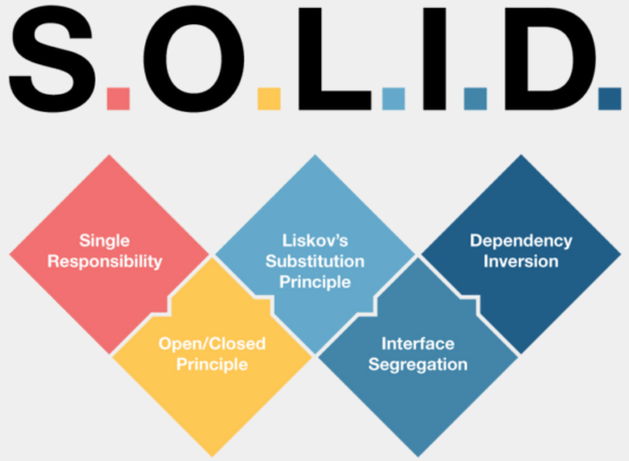
Una nueva versión de un microservicio siempre debería poder sustituir a una versión anterior sin romper nada.

I- Segregación de interfaz

Un microservicio no debe exponer métodos que no estén directamente relacionados.

D - Inversión de dependencia

Un microservicio no debe llamar directamente a otro microservicio.



Atributos de Calidad

Atributos de calidad asociados con los microservicios

Escalabilidad

Eficiencia de desempeño

Modularidad

Compatibilidad

Funcionalidad

Reusabilidad

Confiabilidad

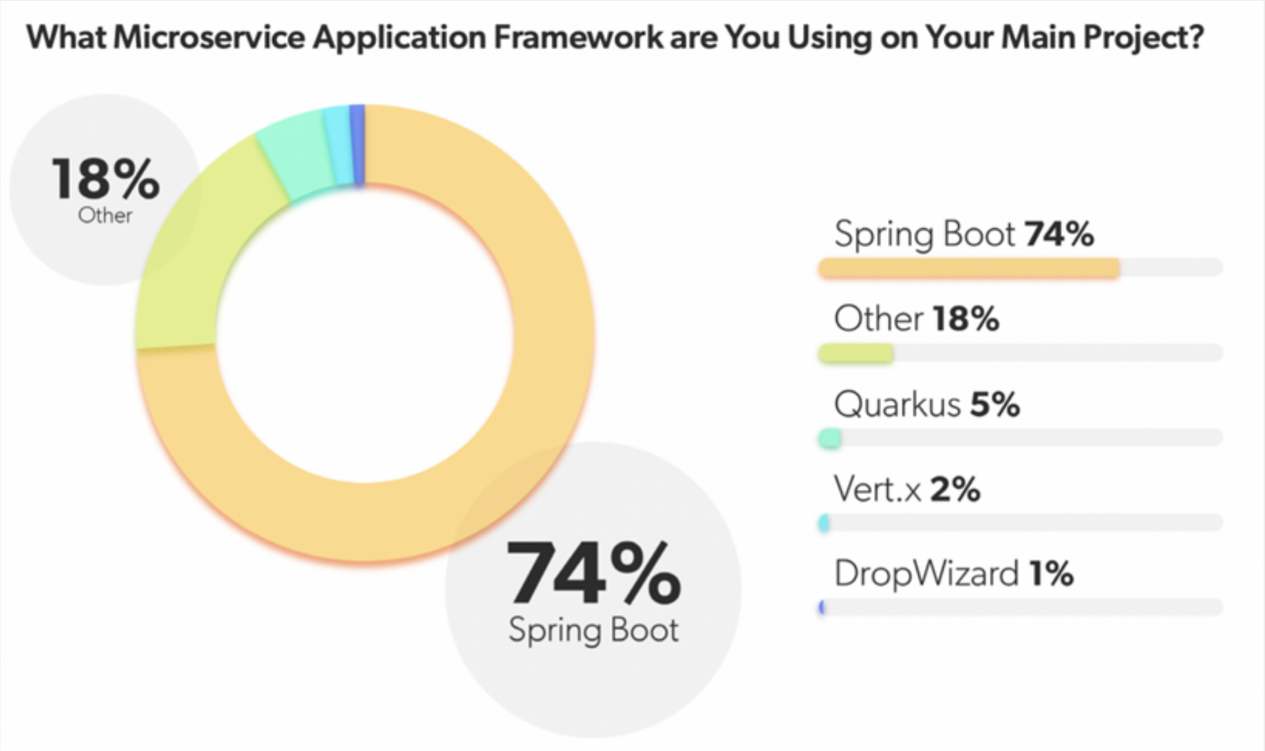
Portabilidad

Casos de estudio

Análisis de mercado

Spring Boot:

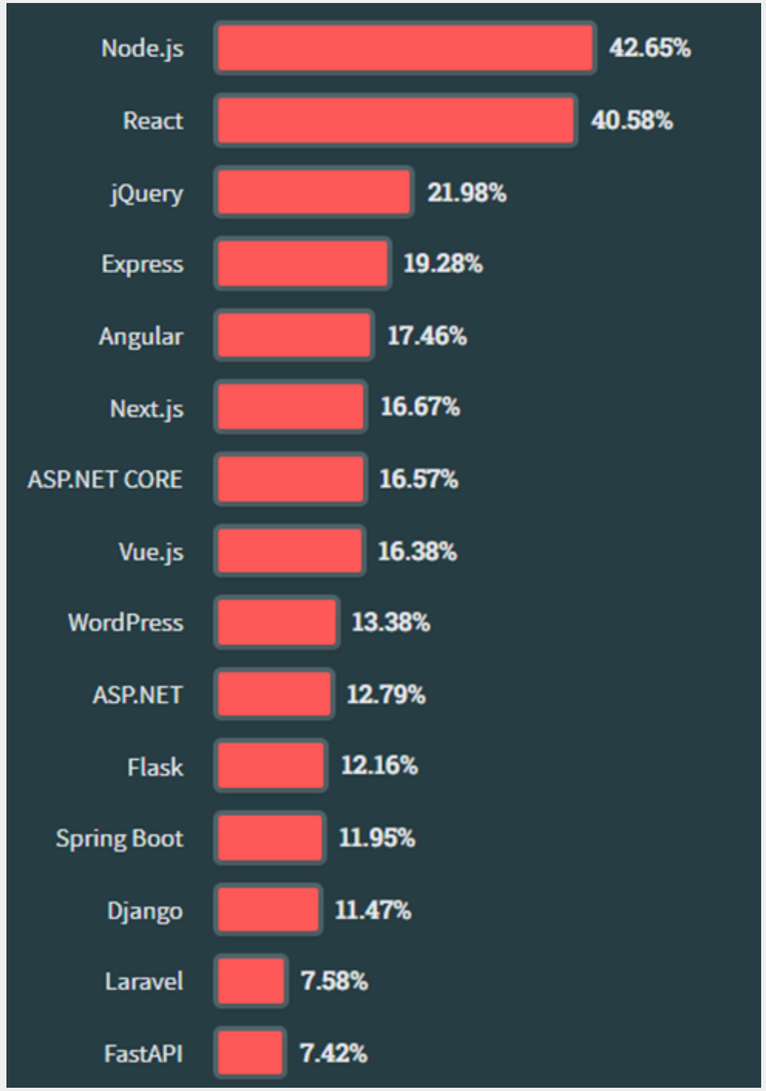
- Informe de Productividad del Desarrollador de Java 2022, el 32% de los desarrolladores utilizan los microservicios como arquitectura principal y el framework más utilizado es Spring Boot con el 74%.
- Salario para un desarrollador Spring Boot en Estados Unidos se encuentra entre los $64.000 y $120.000 USD al año.



Análisis de mercado

Angular:

- Según la Encuesta para Desarrolladores hecha por Stack Overflow de 2023, entre los frameworks web y tecnologías más utilizadas, Angular ocupó el 5.° puesto con un porcentaje del 17.28%. Los frameworks y tecnologías que superaron a Angular fueron Node.js, React, jQuery y Express.
- El rango de salario para un desarrollador de Angular en los Estados Unidos se encuentra entre $69.000 y $129.000 USD al año.



Análisis de mercado

MySQL:

- Cuota de mercado del 32.36%, lo convierte en el segundo sistema más popular después de Microsoft SQL Server.
- Es utilizado por el 79.6% de todos los sitios web que utilizan un sistema de gestión de bases de datos conocido.
- El 53% de las empresas de la industria TI y servicios utilizan MySQL como solución de gestión de bases de datos.
- Es utilizado por más del 25% de los sitios web que funcionan con tecnologías Java.
- El salario promedio de un desarrollador de MySQL en los Estados Unidos es de $87.000 USD al año.