

# Arquitectura MVC

Descubre los patrones de software y la arquitectura Modelo-Vista-Controlador para el desarrollo web profesional



# Patrones de Software

## Soluciones Comprobadas

Respuestas estandarizadas a problemas comunes en desarrollo de software

## Reutilizables

Fácilmente aplicables a diferentes circunstancias y proyectos

## Bien Documentados

Probados en múltiples sistemas con documentación completa

Los patrones incluyen arquitectura, diseño, creación de objetos, estructura de clases y comportamiento.

# Tipos de patrones

Arquitectura

Diseño

Creación de objetos

Estructura de clases

Comportamiento

Dialectos

Interacción o interfaz de usuario

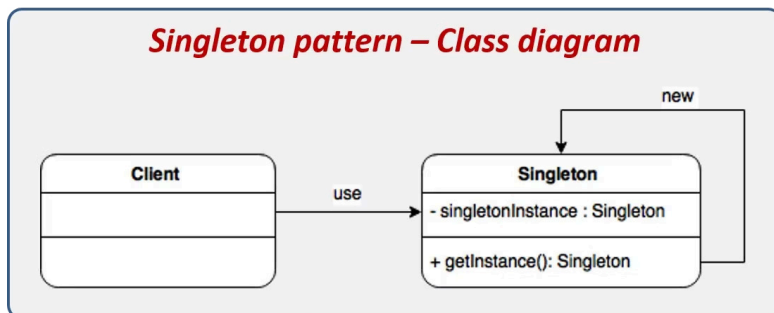
Análisis

Dominio

# Ejemplo: Patrón Singleton

Patrón de software considerado dentro de patrones de diseño

Asegura que una aplicación sólo puede generar una instancia de este objeto. Muy usado para gestionar conexiones a BBDD



```
class Singleton
{
    private static $instancia; // Referencia a la única instancia de este objeto. Es private
                                // para que nadie pueda usarla desde fuera del objeto

    // Constructor privado. Nadie podrá crear objetos desde fuera de la clase.
    private function __construct()
    {
        $this->contador = 0;
    }

    // Este método comprueba si existe ya una instancia del objeto Singleton.
    // Si existe, la devuelve. Si no existe, la crea antes de devolverla.
    public static function getInstance()
    {
        if ( self::$instancia instanceof self )
        {
            self::$instancia = new self;
        }

        return self::$instancia;
    }
}
?>
```



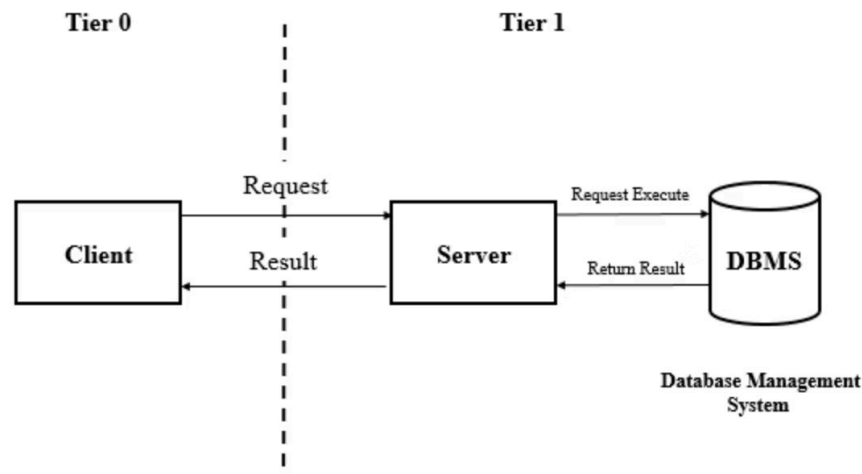
# Arquitecturas Web

## Arquitectura Física (Multitier)

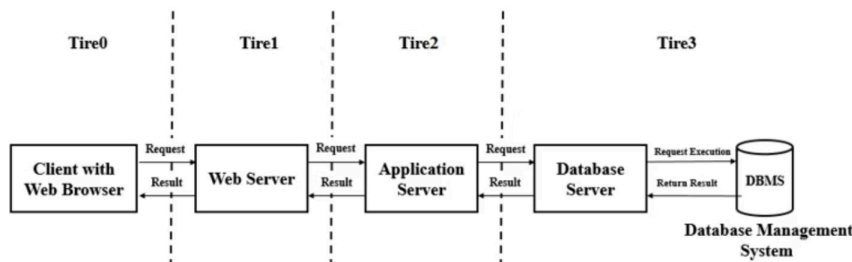
Ordenadores conectados en red ejecutando conjuntamente una aplicación

- Cliente-servidor
- N niveles físicos

1 nivel físico



3 niveles físicos



## Arquitectura Lógica (Multilayer)

Capas de software que colaboran mediante interfaces definidos

- Capa abstracta (usuario)
- Capas intermedias
- Capa menos abstracta (hardware)

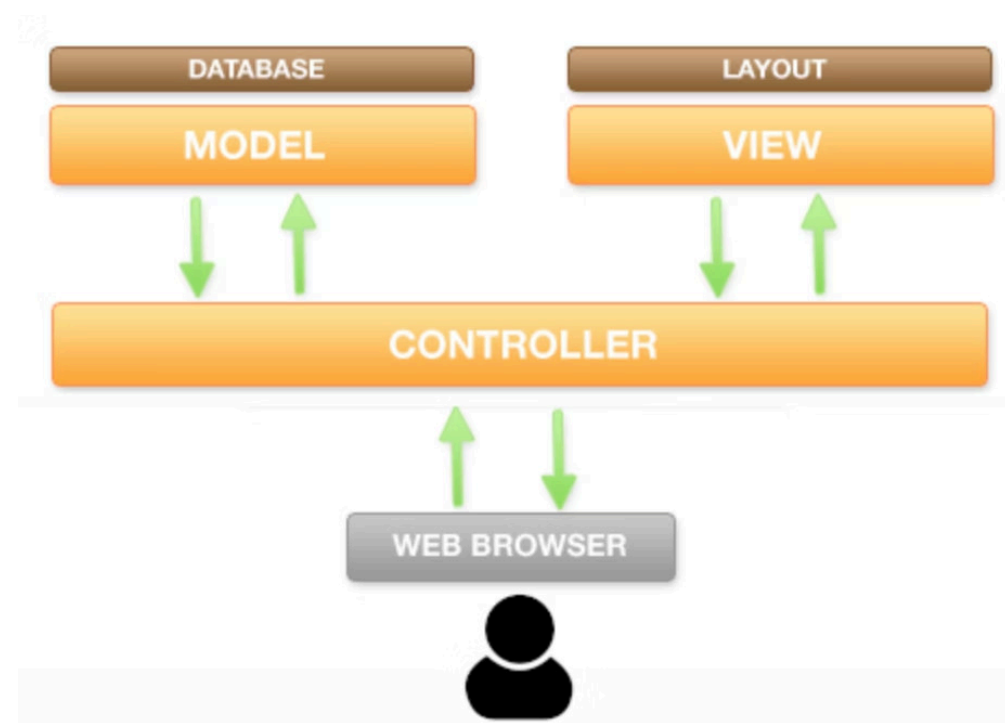
## Ventajas de las arquitecturas multicapa

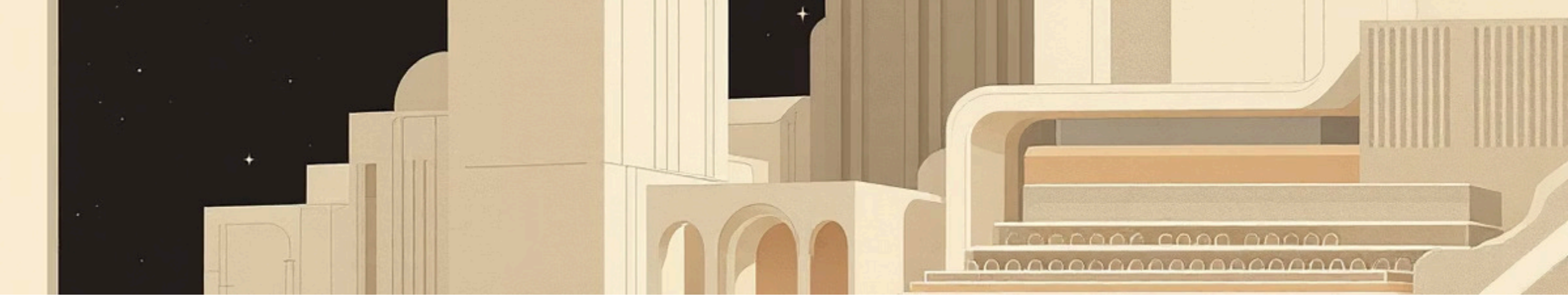
- **Desarrollar en paralelo** cada capa (mayor rapidez de desarrollo).
- Aplicaciones más **robustas** gracias al encapsulamiento. ¿Te suena? ¡Programación orientada a objetos! Cada capa se implementa en una clase, y cada clase hace su trabajo sin importunar a las demás y sin preocuparse por cómo funcionan las otras internamente.
- El **mantenimiento** es más sencillo.
- Más **flexibilidad** para añadir módulos.
- Más **seguridad**, al poder aislar (relativamente) cada capa del resto.
- Mejor **escalabilidad**: es más fácil hacer crecer al sistema.
- Mejor **rendimiento** (aunque esto podría discutirse: puedes hacer un sistema multicapa con un rendimiento desastroso y un sistema monolítico que vaya como un tiro. Pero, en general, es más fácil mejorar el rendimiento trabajando en cada capa por separado).
- Es más fácil hacer el **control de calidad**, incluyendo la fase de pruebas.

# ¿Qué es MVC?



El MVC es una arquitectura multicapa estandarizada de 3 capas que encaja perfectamente en aplicaciones web.





# Evolución del Código

## Código Monolítico

Todo el código en un solo bloque. Simple pero inmanejable en sistemas complejos.

## Modelo-Vista-Controlador

División completa en tres capas especializadas.

## Controlador + Vista

Separación entre lógica de datos y presentación HTML.

## Con Abstracción de Datos

Independencia del gestor de base de datos específico.

Evolución del código

Ver apartado 4.3.2 de [https://iescelia.org/docs/dwes/\\_site/mvc/](https://iescelia.org/docs/dwes/_site/mvc/)

# Implementación Práctica



**index.php**

Punto de entrada que determina controlador y acción



**Controlador**

Invoca modelos y vistas en orden correcto



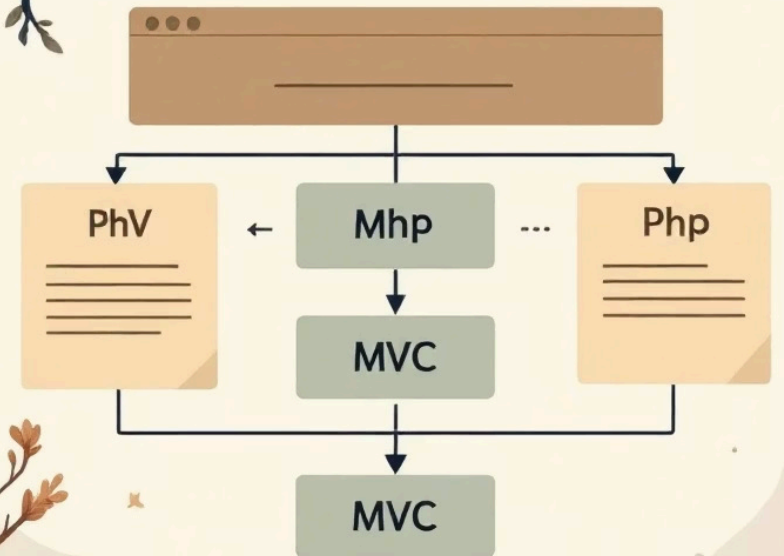
**Modelo**

Acceso a datos y lógica de negocio

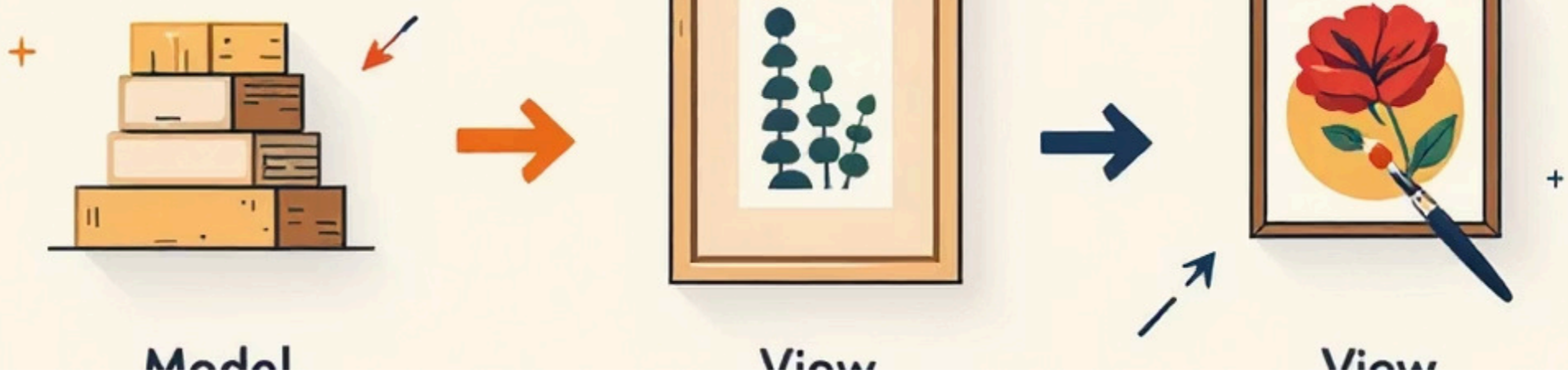


**Vista**

Generación de HTML y presentación







# Definición Teórica del MVC

## Modelos

**Lógica de negocio:** acceso a datos, filtros, algoritmos y restricciones del sistema. Un modelo por tabla maestra.

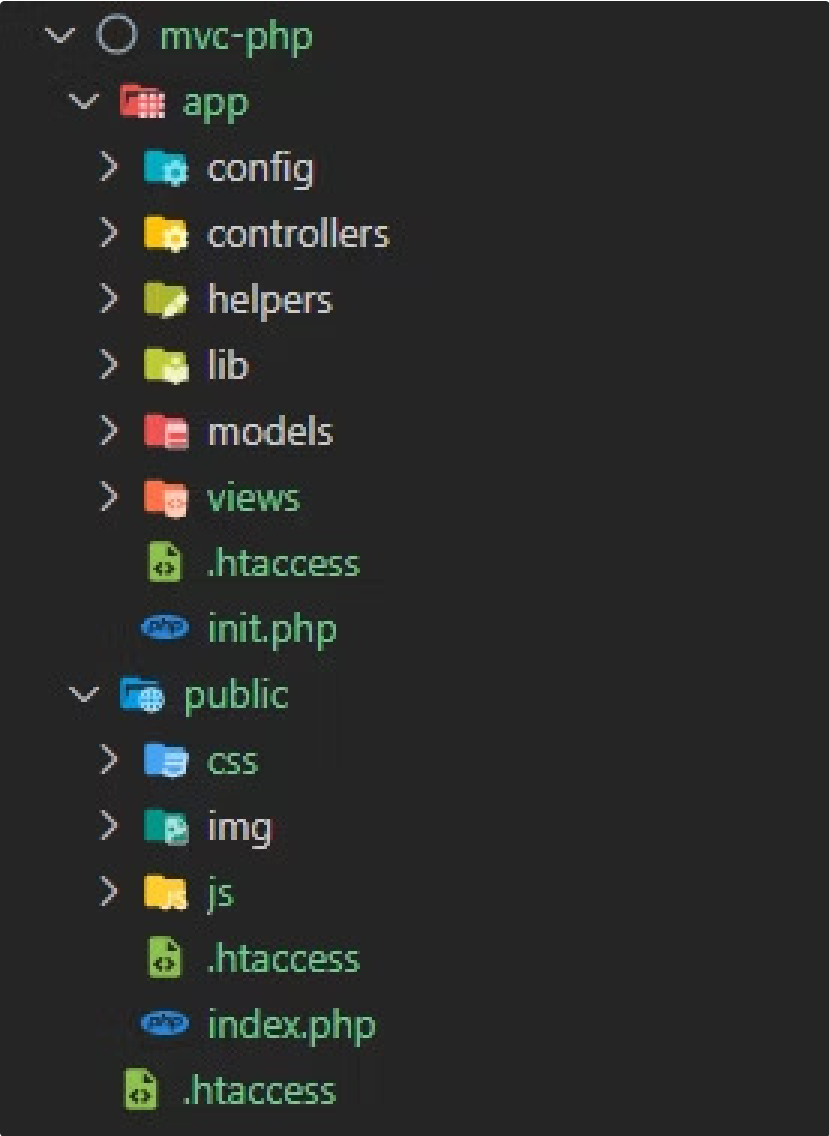
## Vistas

**Salidas HTML:** todo lo que el usuario ve e interactúa. Incluye JavaScript y CSS.

## Controladores

**Flujo de ejecución:** captura peticiones y dirige la ejecución. Un método por funcionalidad.

# Organización física del servidor



```

  v ○ mvc-php
    v 📁 app
      > ⚙️ config
      > 📁 controllers
      > 📁 helpers
      > 📁 lib
      > 📁 models
      > 📁 views
      📄 .htaccess
      📄 init.php
    v 🌐 public
      > 📁 css
      > 📁 img
      > 📁 js
      📄 .htaccess
      📄 index.php
      📄 .htaccess

```

Los **archivos estáticos** se guardarán en las **carpetas CSS, img, js**

La lógica de negocio y acceso a los datos en models

La representación del html en views

El controlador en controllers

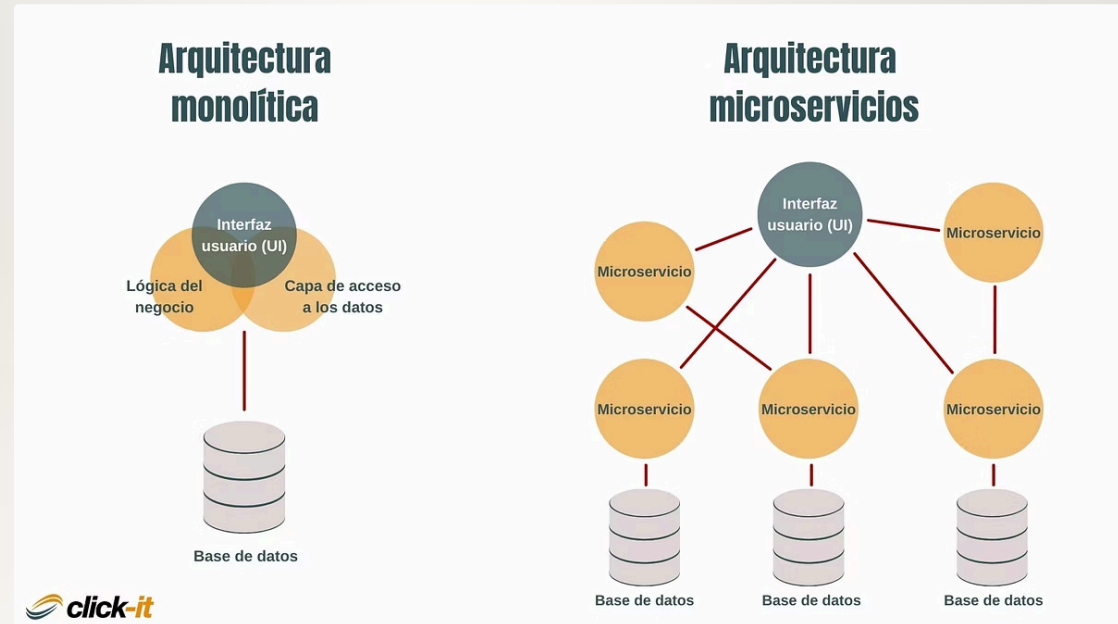
IMPORTANT

# Evolución de Multicapa a Microservicios

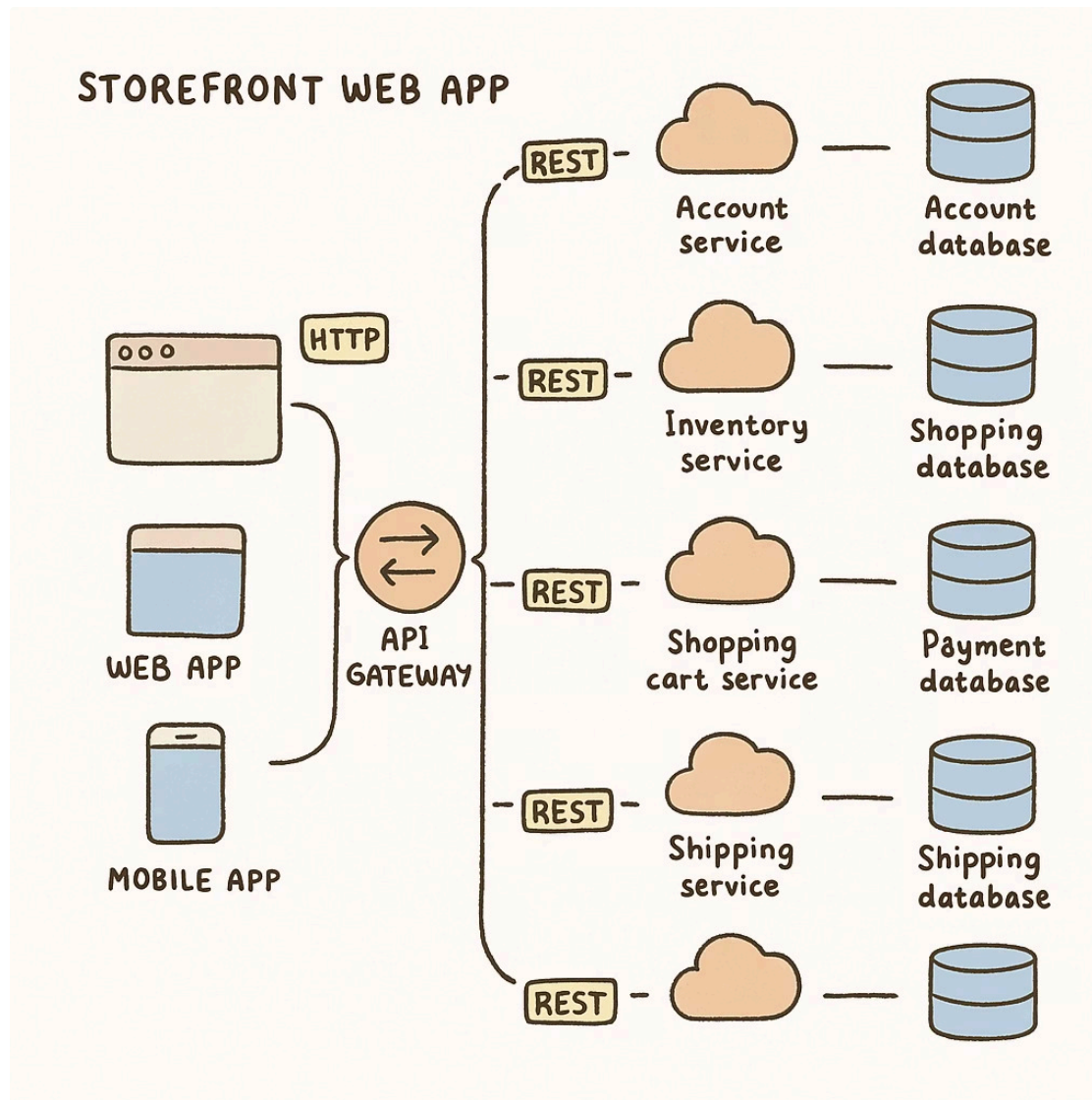
Es una arquitectura distribuida donde la aplicación se divide en servicios pequeños, independientes y autónomos, que se comunican entre sí (habitualmente vía API REST o mensajería).

Cada microservicio tiene su propia lógica de negocio y, muchas veces, su propia base de datos

Se pueden desarrollar, desplegar y escalar de forma independiente



# Ejemplo de arquitectura basada en microservicios



# Ventajas y Desventajas de Microservicios

## Ventajas:

- **Escalabilidad:** puedes aumentar los recursos solo en el servicio que lo necesite (ej. pagos en Black Friday)
- **Flexibilidad tecnológica:** cada servicio puede estar hecho en un lenguaje diferente si se desea
- **Despliegues más ágiles y frecuentes**

## Desventajas:

- **Más complejidad en la infraestructura** (orquestración, balanceo, seguridad, comunicación entre servicios)
- **Necesidad de herramientas adicionales** (Docker, Kubernetes, CI/CD, observabilidad, etc.)

