Arquitectura de Software

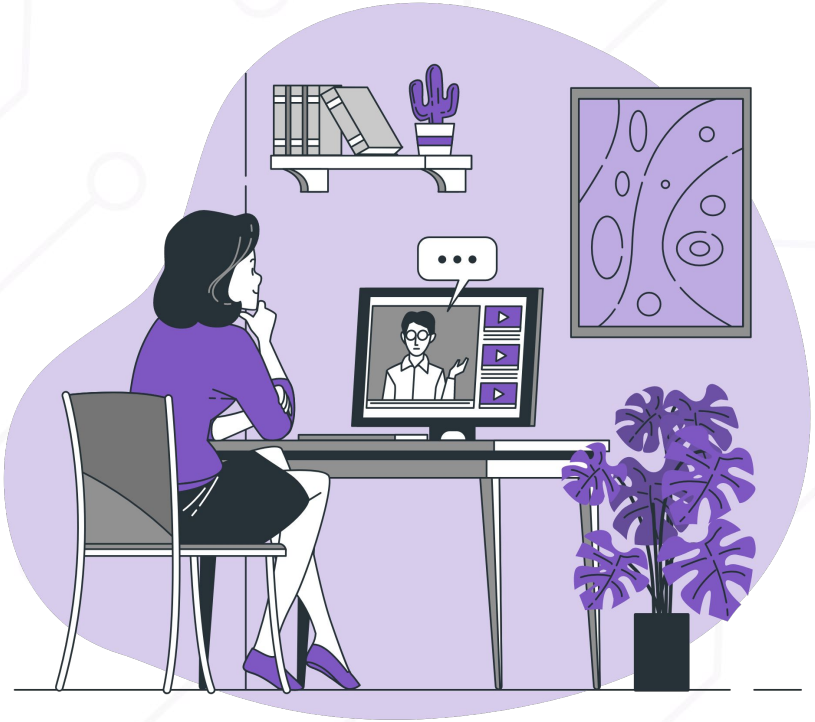
DEV.F::
DESARROLLAMOS(PERSONAS);

Objetivos de la sesión

- Hablaremos de la arquitectura Cliente-Servidor y de la web 2.0 y web 3.0

- Hablaremos de otras arquitecturas de software para implementar en la web.

- Recuerda que puedes ir tomando notas o apuntes en tu lap o en hojas, esto te ayudará a repasar y entender mejor el tema.

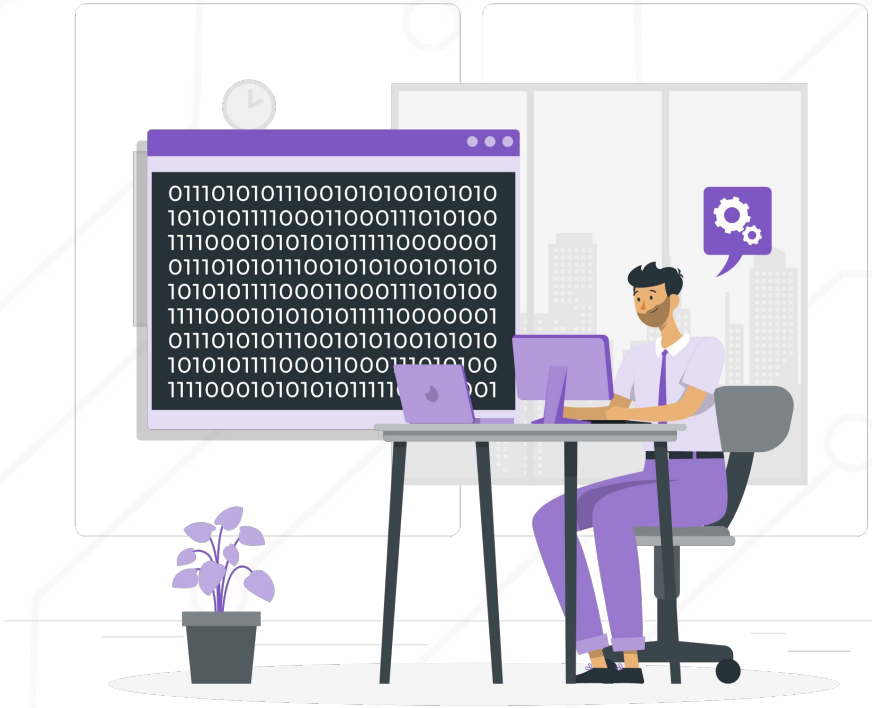


DEV.F

¿Qué es la Arq. de Software?

¿Qué entendemos por

Arquitectura de Software ?



¿Qué es la Arq. de Software?

La Arquitectura de Software es un método o estructura bien definida que se utiliza de base para el diseño de diferentes funcionalidades. De esta forma proporcionamos agilidad en el desarrollo además de obtener un performance óptimo

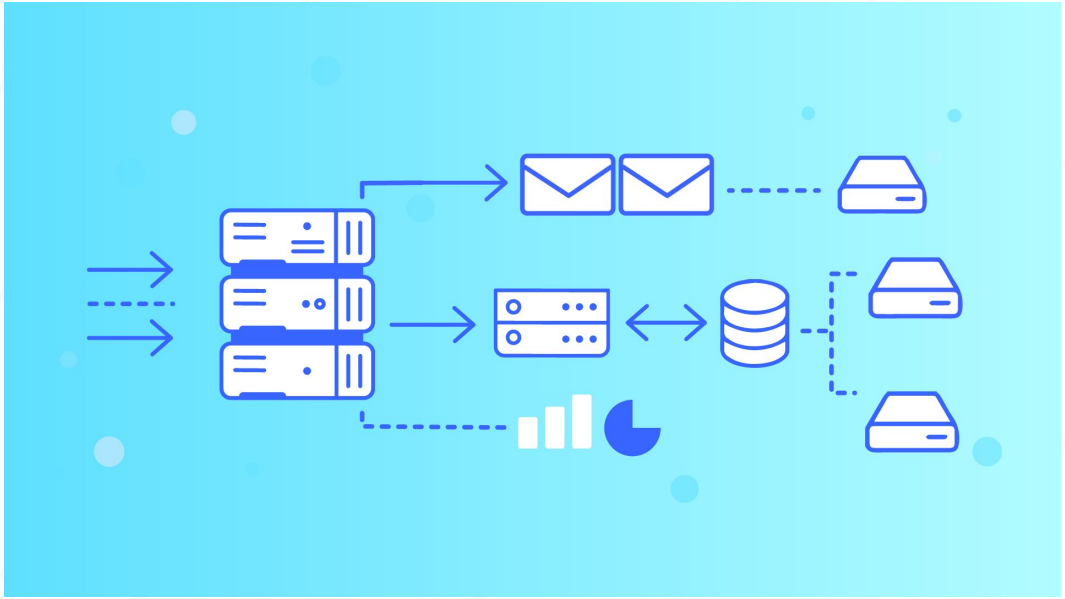
DEV.F

¿Por qué hablar de arquitectura?

A semejanza de los planos de un edificio o construcción.

¿Por qué hablar de arquitectura?

Estas arquitecturas indican la estructura, funcionamiento e interacción entre las partes del software.



Arq. de software

¿Alguna vez te has preguntado **cómo se diseñan los grandes sistemas empresariales**?

DEV.F

Arq. de software

Cuando hablamos de arquitecturas de software, no solo trabajamos bajo un esquema, modelo o patrón ya establecido, si no también podemos incluir otras características al proyecto, en este caso a nivel software, que puedan proporcionar una mejor calidad, servicio y productividad.

DEV.F

Arq. de software

Antes de que comiences un importante desarrollo de software, debemos elegir una arquitectura adecuada que nos proporcione la funcionalidad deseada y los atributos de calidad. Por lo tanto, debemos entender diferentes arquitecturas, antes de aplicarlas a nuestro diseño.

DEV.F

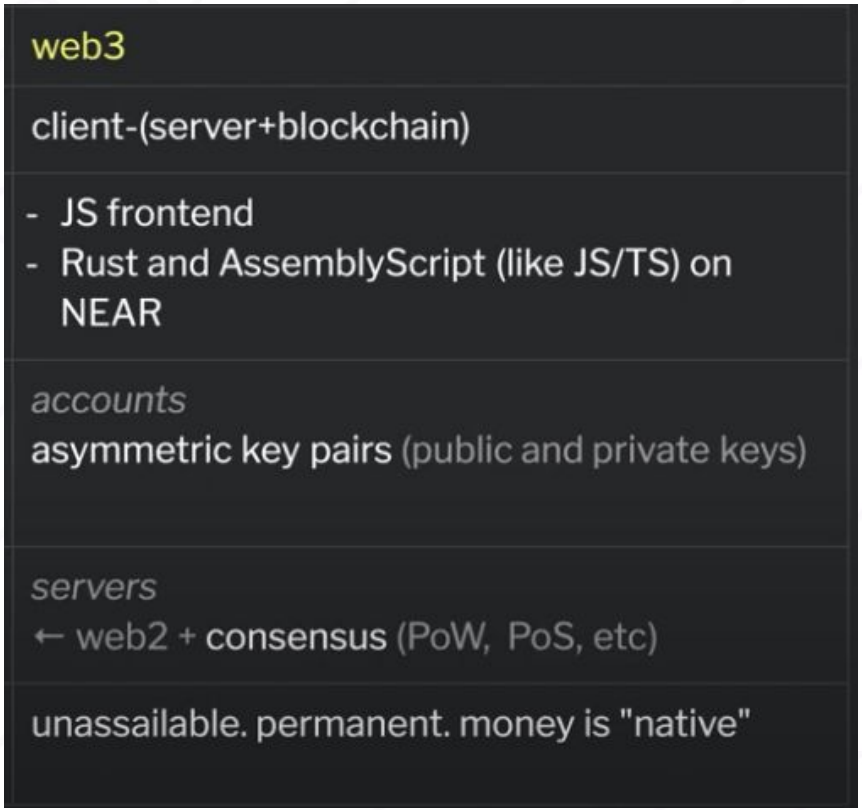
WEB2 -> WEB3

	web2	web3
topology	client-server	client-(server+blockchain)
app dev technology	- JS frontend - multiple backend techs	- JS frontend - Rust and AssemblyScript (like JS/TS) on NEAR
security	*accounts* - username / password - oAuth	*accounts* asymmetric key pairs (public and private keys)
	servers - AWS / GCP / Azure	*servers* ← web2 + consensus (PoW, PoS, etc)
key performance	fast. cheap. data is “native”	unassailable. permanent. money is "native"

moneda es la data para loguearme

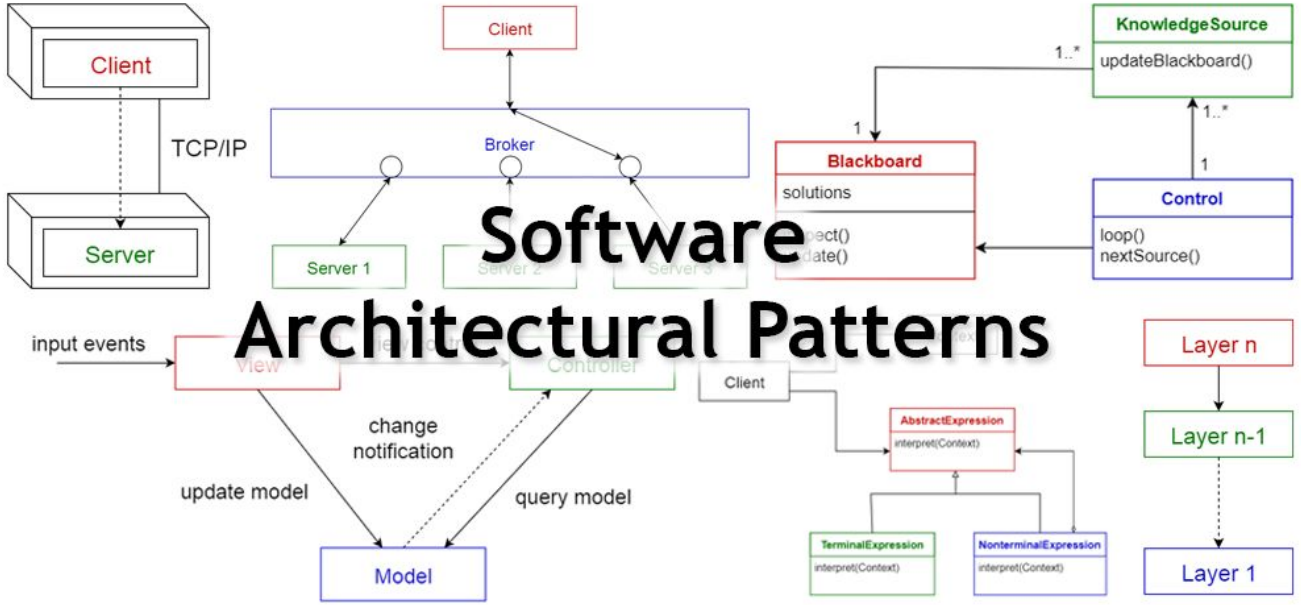
DEV.F::

WEB2 -> WEB3



La web 3 apunta a ser un sistema abierto centralizado donde cada usuario(cliente, persona, desarrollador, etc.) sea dueño de su propia información, donde la moneda digital (Ethereum, Bitcoins, Litecoin etc. será la data, "native")

Patrones de Arquitectura de Software

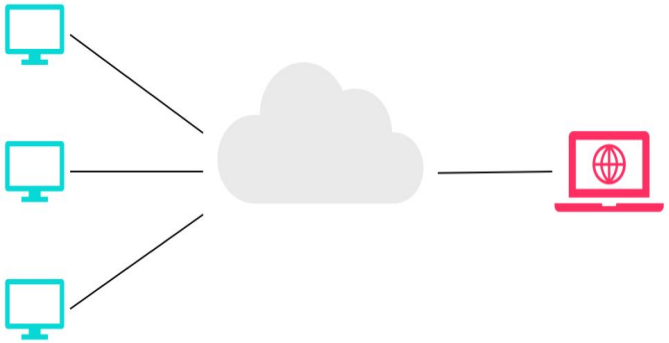


Cliente - Servidor

DEV.F::
DESARROLLAMOS(PERSONAS);

Cliente - Servidor

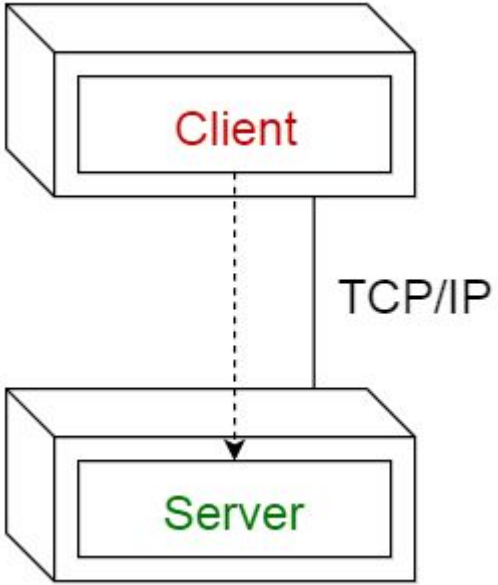
En la Arquitectura Cliente - Servidor existen recursos y servicios compartidos a los que desean acceder un gran número de clientes distribuidos y para los que se desea controlar el acceso o la calidad del servicio.



Un Servidor y Múltiples Clientes

Uso

- Podemos utilizar la arq. cliente-servidor para modelar una parte de un sistema que tiene muchos componentes que envían peticiones (clientes) a otro componente (servidor) que ofrece servicios: aplicaciones en línea como el correo electrónico, el intercambio de documentos y la banca.



Cliente - Servidor

Al gestionar un conjunto de recursos y servicios compartidos:

1. Podemos promover la modificabilidad y la reutilización, al factorizar los servicios comunes y tener que modificarlos en una sola ubicación, o en un número reducido de ellas.

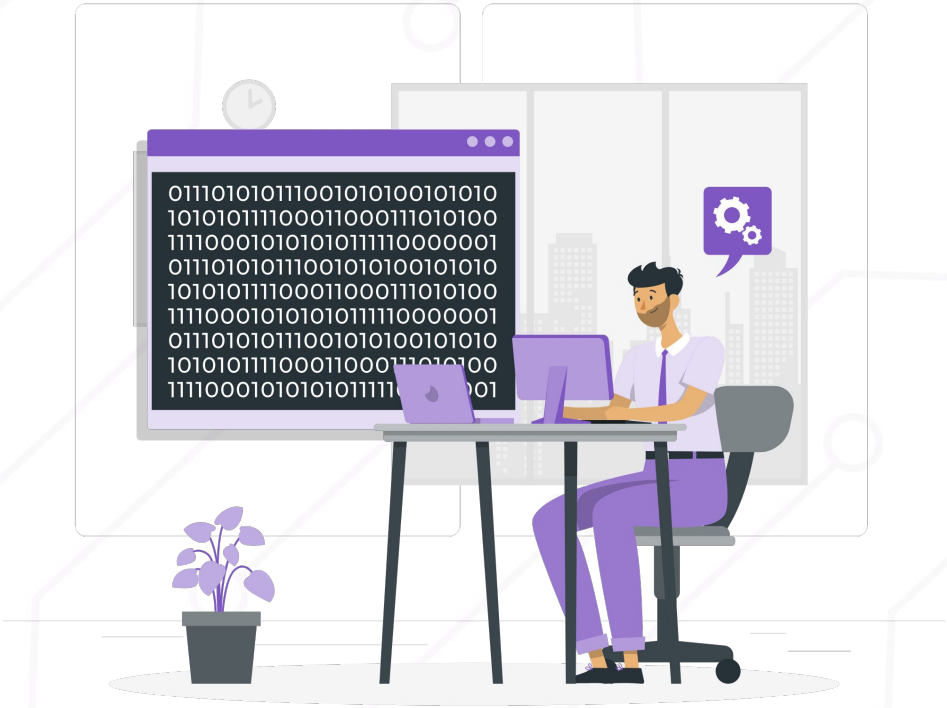
2. Queremos mejorar la escalabilidad y la disponibilidad centralizando el control de estos recursos y servicios y distribuyendo los propios recursos entre varios servidores físicos.

Tres Niveles

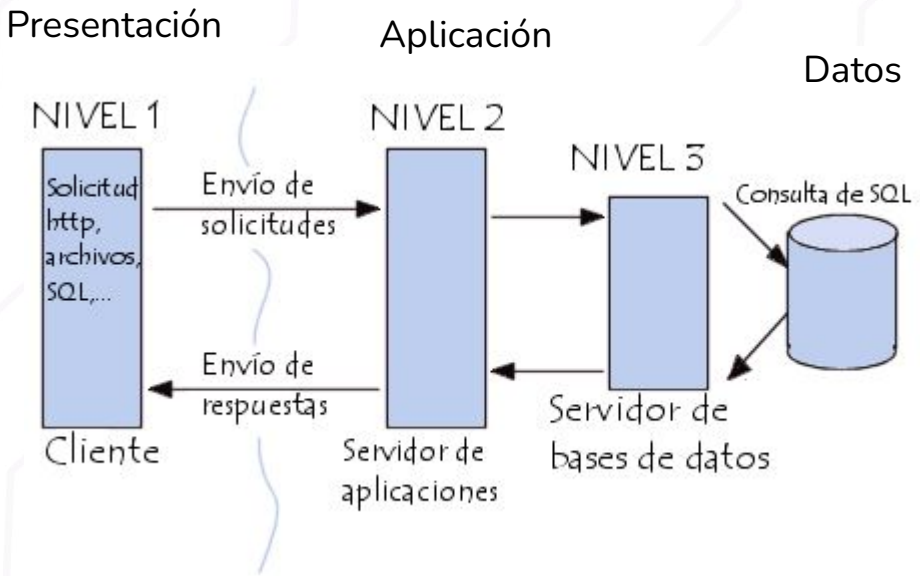
DEV.F::
DESARROLLAMOS(PERSONAS);

Arquitectura de tres niveles

La arquitectura de tres niveles, separa las aplicaciones en tres niveles de informática, lógica y física, es la arquitectura de software predominante para las aplicaciones de cliente-servidor tradicionales.



El nivel de presentación o la interfaz de usuario, el nivel de aplicación o donde se procesan los datos, y el nivel de datos donde se almacenan y gestionan los datos asociados con la aplicación.



En el desarrollo web, los niveles tienen nombres distintos pero realizan funciones similares.

Servidor web

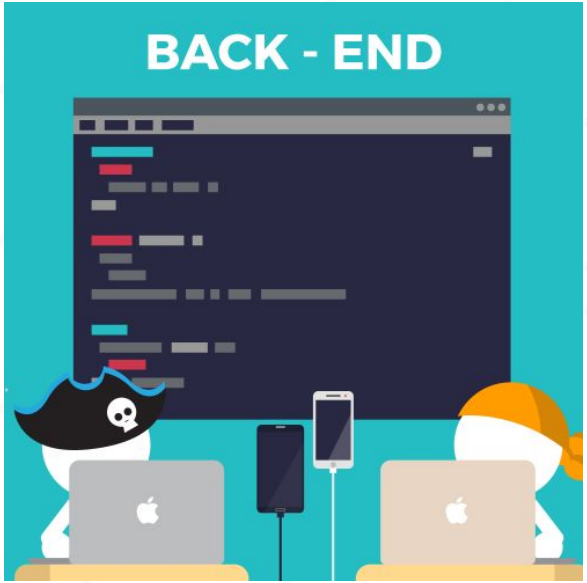
- El servidor web es el nivel de presentación y proporciona la interfaz de usuario. Este suele ser una página o sitio web, como un sitio de comercio electrónico en el que el usuario añade productos al carrito de compras, añade datos de pago o crea una cuenta. El contenido puede ser estático o dinámico y se suele desarrollar utilizando HTML, CSS y Javascript.

Servidor de Aplicaciones

- El servidor de aplicaciones corresponde al nivel medio, que aloja la lógica empresarial utilizada para procesar las entradas de usuario. Continuando con el ejemplo del comercio electrónico, este es el nivel que consulta la base de datos de inventario para informar la disponibilidad del producto o añadir detalles al perfil de un cliente. Esta capa a menudo se desarrolla utilizando Python, Javascript o PHP.

El servidor de base de datos

- El servidor de base de datos es el nivel de datos o backend de una aplicación web. Se ejecuta en un software de gestión de base de datos como MySQL, Oracle o PostgreSQL.



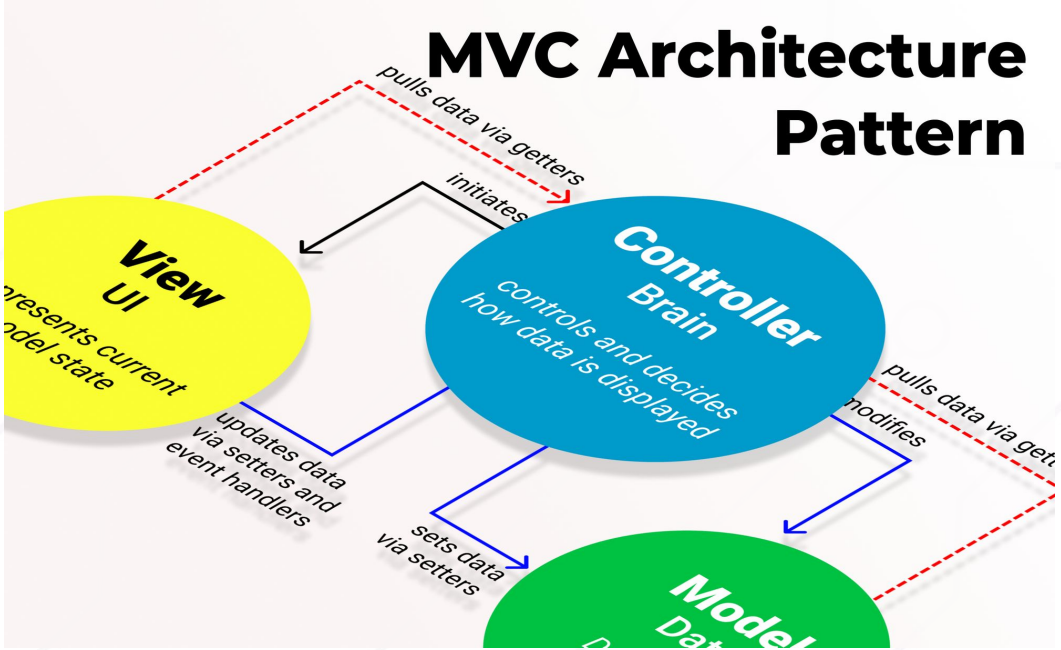
MVC

Modelo, Vista, Controlador

DEV.F:: DESARROLLAMOS(PERSONAS);

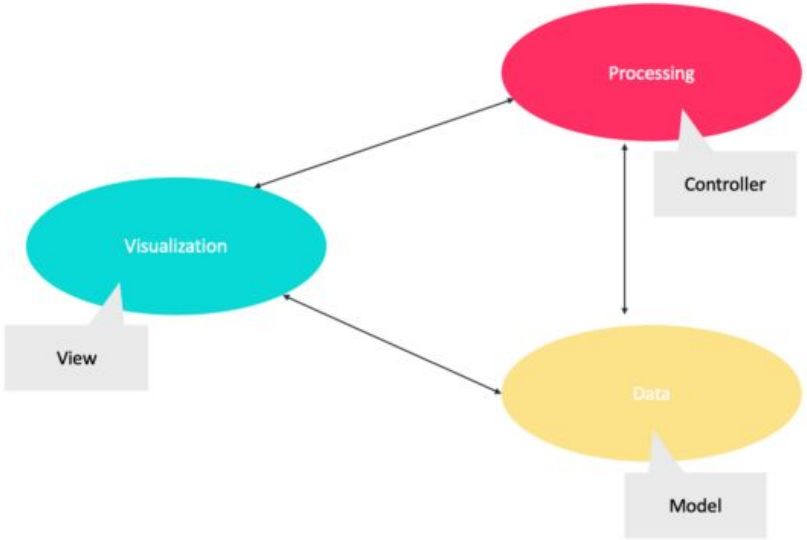
MVC

La interfaz de usuario suele ser la parte más modificada de una aplicación interactiva. Los usuarios suelen desear ver los datos desde diferentes perspectivas, como un gráfico de barras o un gráfico circular. Ambas representaciones deben reflejar el estado actual de los datos.



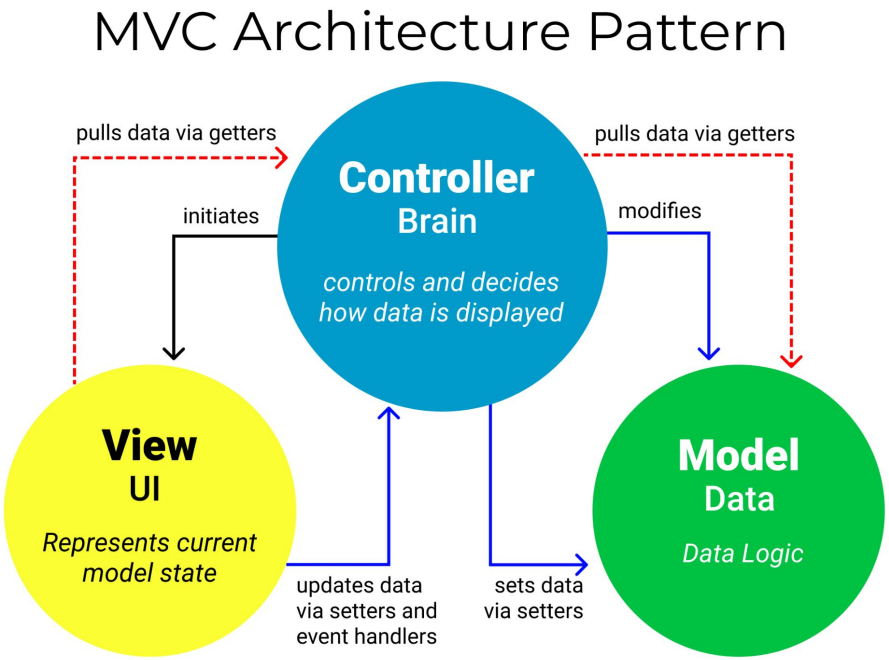
MVC

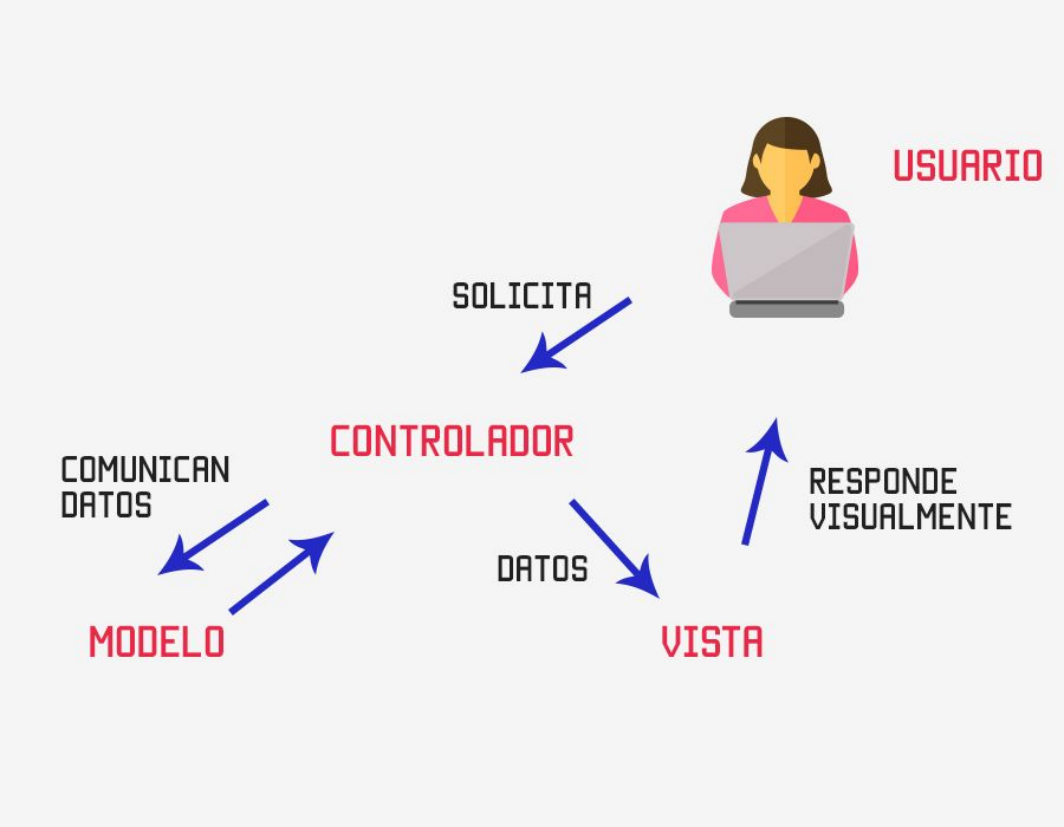
1. **modelo** — *contiene la funcionalidad y los datos básicos*
2. **vista** : *muestra la información al usuario (se puede definir más de una vista)*
3. **controlador** : *maneja la entrada del usuario*



Uso

- Arquitectura para aplicaciones World Wide Web en los principales lenguajes de programación.




USUARIO
SOLICITA
CONTROLADOR
COMUNICAN
DATOS
RESPONDE
VISUALMENTE
DATOS
MODELO
VISTA

BREAK

Break de 10 minutos

DEV.F::

Dirigida por Eventos

DEV.F::
DESARROLLAMOS(PERSONAS);

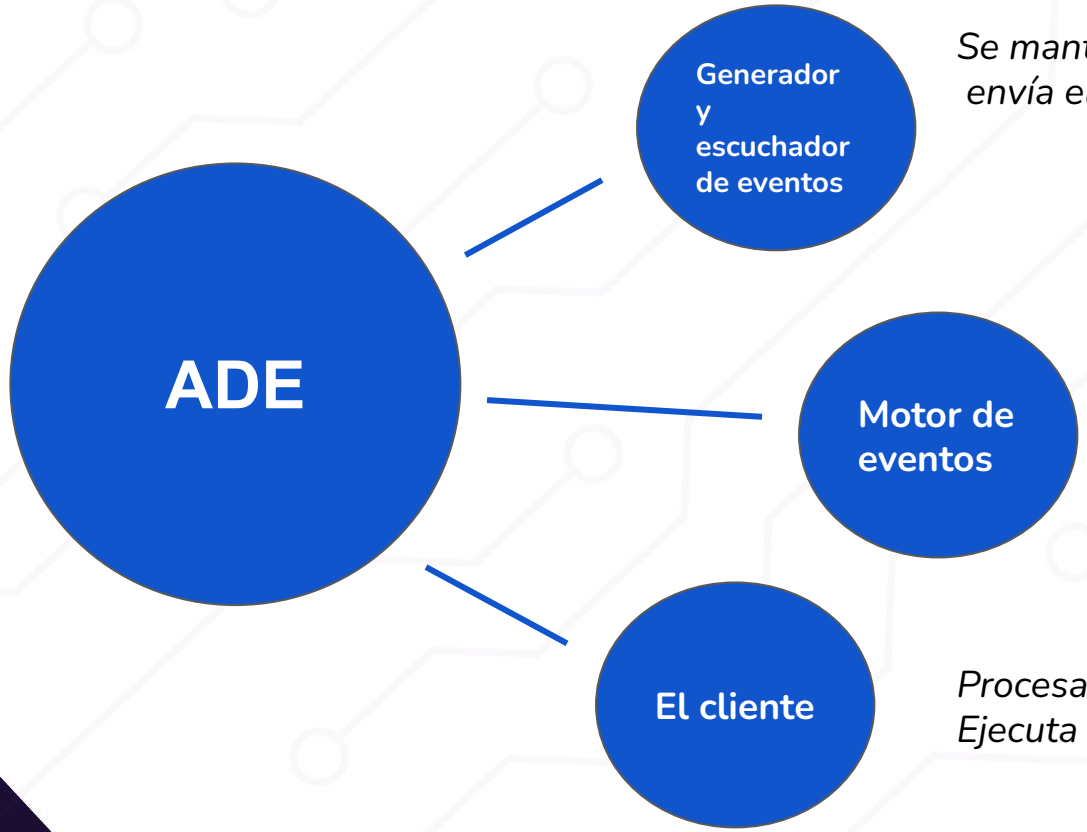
ADE (Arquitectura Dirigida por Eventos)

1. Promueve la producción, detección consumición y reacción en base a eventos.

2. Un evento puede ser lanzado para indicar un cambio de estado significante

Cuando un modelo está salvando (guardando) datos sobre la base de datos, tendremos eventos antes y después de realizar la operación

ADE (Arquitectura Dirigida por Eventos)

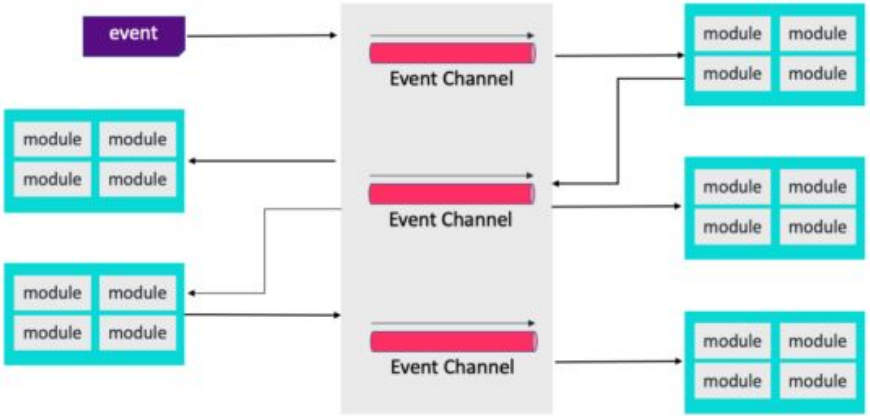


Se mantiene a la escucha,
envía el evento al motor de eventos.

Recibe, Gestiona el evento, envía al cliente de evento indicado para procesarlo.

Procesa el evento,
Ejecuta la función indicada.

Un programador extrae los eventos de la cola y los distribuye al gestor de eventos apropiado basándose en una política de programación.



Aplicación de comercio electrónico

El Servicio de Pedidos crea un Pedido en estado pendiente y publica un evento OrderCreated.

El Servicio de Atención al Cliente recibe el evento e intenta reservar crédito para ese Pedido. A continuación, publica un evento Credit Reserved o un evento CreditLimitExceeded.

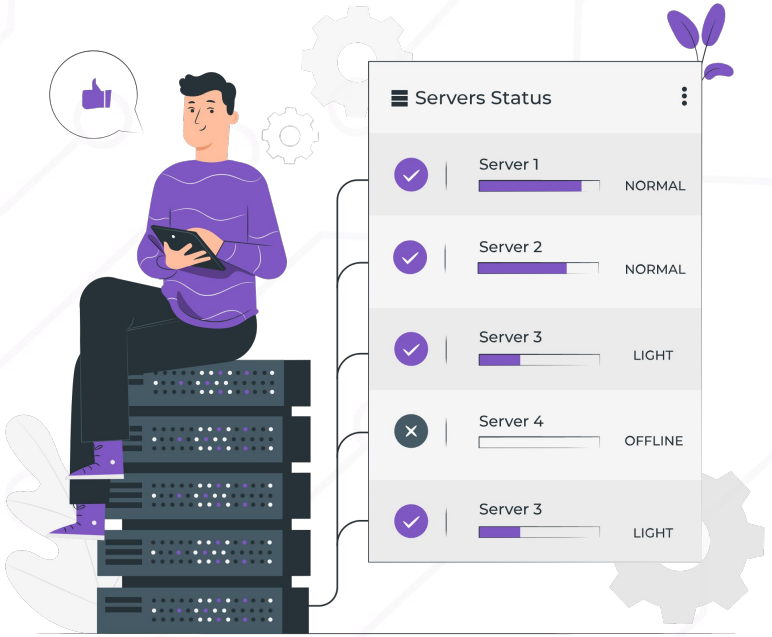
El Servicio de Pedidos recibe el evento del Servicio de Atención al Cliente y cambia el estado del pedido a aprobado o cancelado

Microservicios

DEV.F:
DESARROLLAMOS(PERSONAS);

En contexto

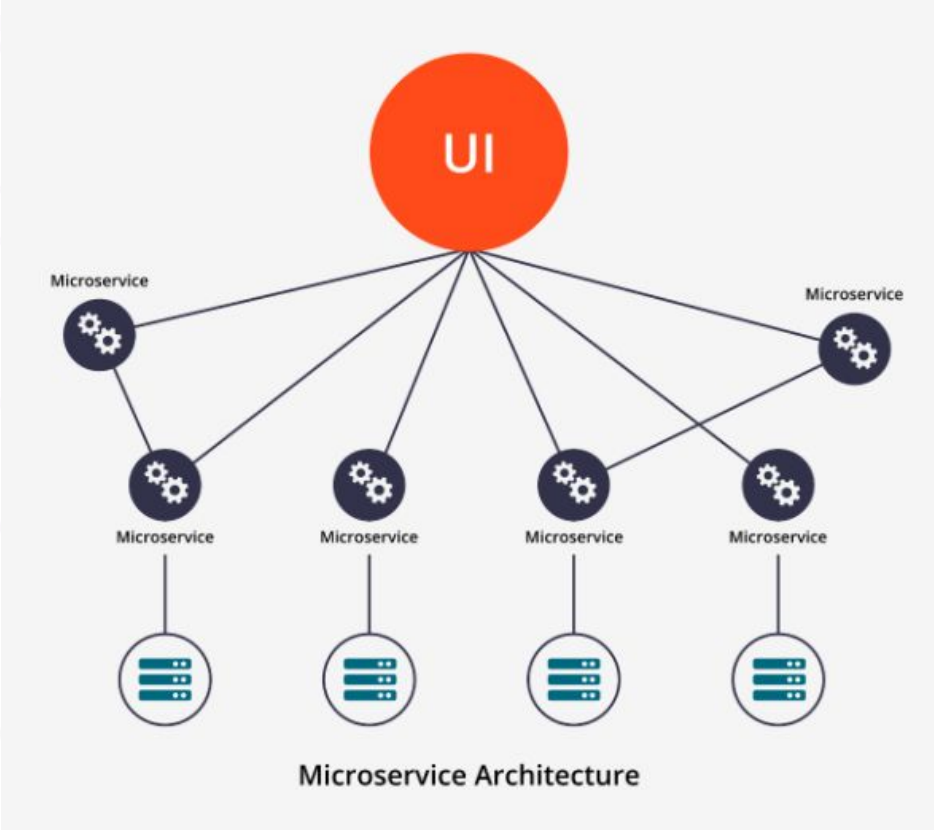
Detectada la necesidad por parte de las empresas de realizar cambios en el software e implementarlos de forma fácil y rápida, nacen los microservicios. La idea era dividir los sistemas en partes individuales, permitiendo que se puedan tratar y abordar los problemas de manera independiente sin afectar al resto.



Microservicios

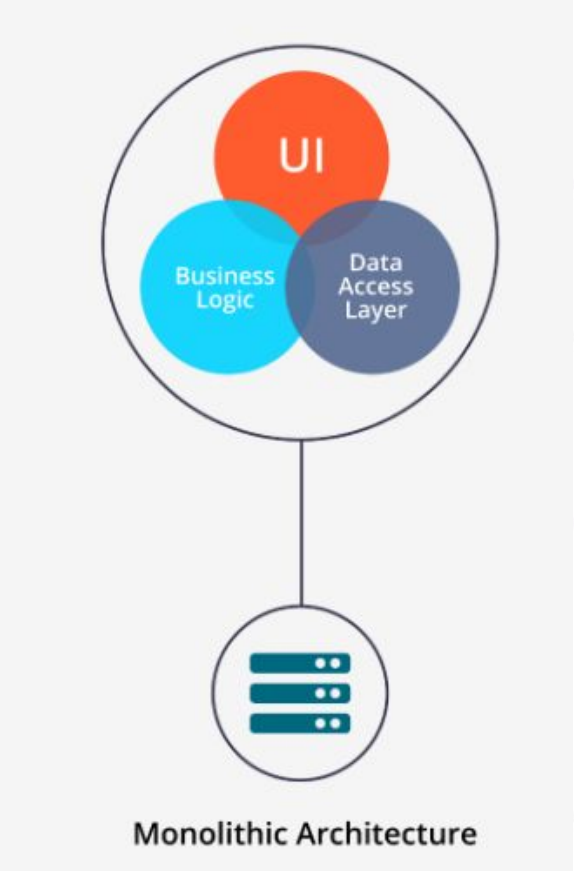
1. La arquitectura de microservicios es un método de desarrollo de aplicaciones software que funciona como un conjunto de pequeños servicios que se ejecutan de manera independiente y autónoma, proporcionando una funcionalidad de negocio completa.

2. En ella, cada microservicio es un código que puede estar en un lenguaje de programación diferente, y que desempeña una función específica. Los microservicios se comunican entre sí a través de APIs, y cuentan con sistemas de almacenamiento propios, lo que evita la sobrecarga y caída de la aplicación.



Los microservicios han creado infraestructuras IT más adaptables y flexibles. Porque si se quiere modificar solamente un servicio, no es necesario alterar el resto de la infraestructura. Cada uno de los servicios se puede desplegar y modificar sin que ello afecte a otros servicios o aspectos funcionales de la aplicación.

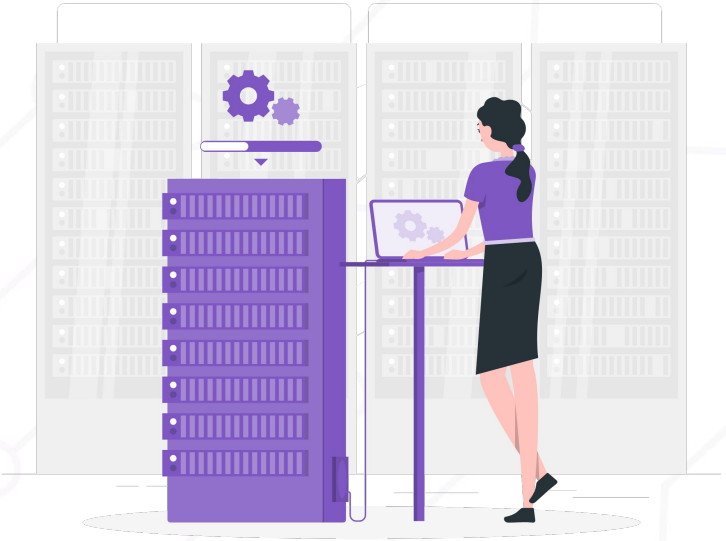
Actualmente muchos desarrolladores están dejando de utilizar arquitecturas monolíticas para pasarse a los microservicios, y muchas empresas líderes ya se han dado cuenta de que la arquitectura de microservicios es la mejor opción para sus organizaciones.



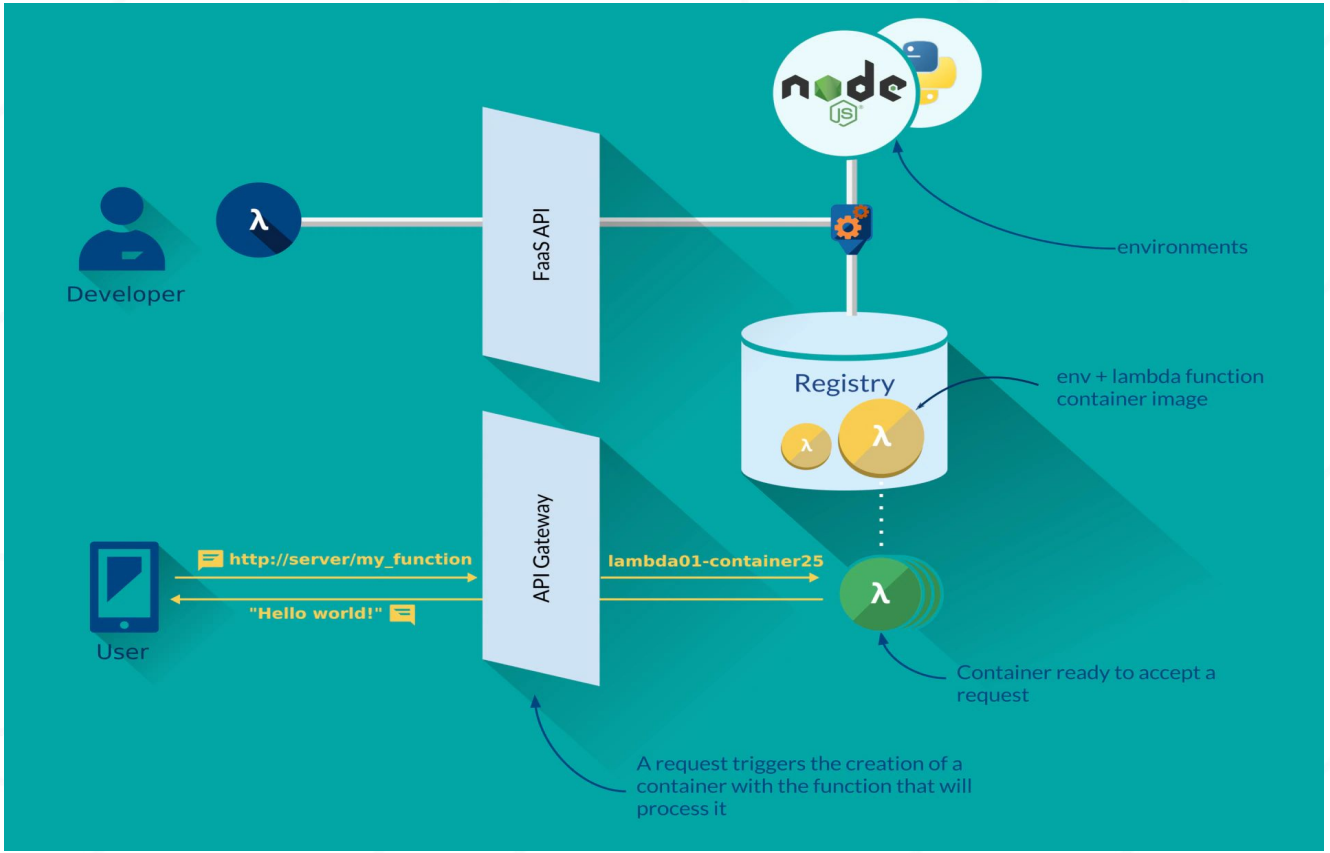
Serverless

DEV.F::
DESARROLLAMOS(PERSONAS);

La principal ventaja de una arquitectura *serverless* es la posibilidad de que el desarrollador se despreocupe de la gestión de la infraestructura sobre la que se ejecuta su servicio (*función*) y centrarse en la funcionalidad: el ciclo completo de desarrollo se simplifica.



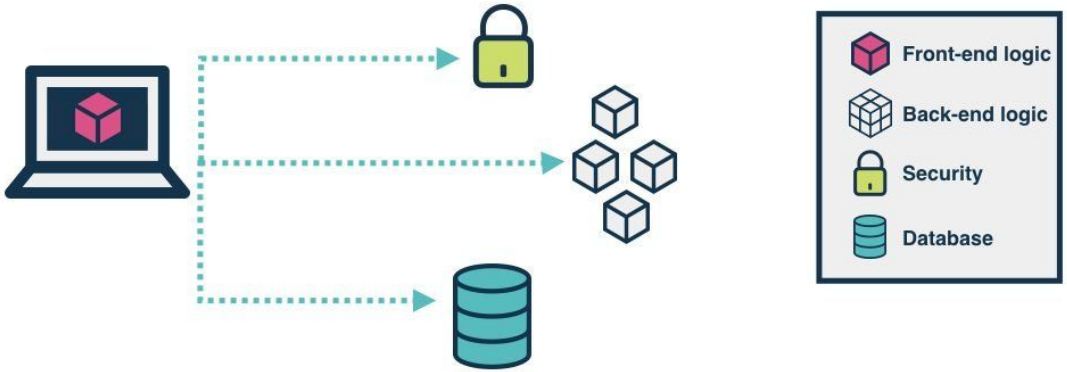
FaaS (Functions as a Service)



TRADITIONAL vs SERVERLESS

TRADITIONAL

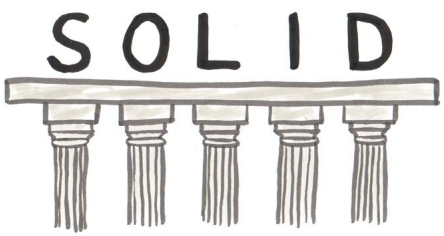
SERVERLESS
(using client-side logic and third-party services)



Principios en el diseño del software: KISS, DRY, SOLID YAGNI

DEV.F::
DESARROLLAMOS(PERSONAS);

Solid



Los Principios SOLID son cinco principios de diseño de clases orientadas a objetos. Son un conjunto de reglas y mejores prácticas a seguir mientras se diseña una estructura de clases.

Siguiendo el acrónimo SOLID, son:

The Single Responsibility Principle	(principio de responsabilidad única)
The Open-Closed Principle	(principio de apertura-cierre)
The Liskov Substitution Principle	(principio de sustitución de Liskov)
The Interface Segregation Principle	(principio de segregación de interfaces)
The Dependency Inversion Principle	(principio de inversión de la dependencia)

KISS - DRY - YAGNI

¿Qué es KISS?

Es un acrónimo de la frase inglesa “Keep it simple, Stupid!”

Este patrón lo que nos dice es que cualquier sistema va a funcionar mejor si se mantiene sencillo que si se vuelve complejo. Es decir que la sencillez tiene que ser una meta en el desarrollo y que la complejidad innecesaria debe ser eliminada.

¿Qué es DRY?

Quiere decir “Don’t repeat yourself”.

Cada pieza de funcionalidad debe tener una única, no ambigua y representativa identidad dentro del sistema.
Si se aplica este patrón de forma correcta un cambio en cualquier parte de la funcionalidad de un programa no incluye cambios en partes que no tengan una relación lógica con la funcionalidad cambiada.

YAGNI

- YAGNI significa “*You Aren't Gonna Need It*” (No vas a necesitarlo) Es uno de los principios de la programación extrema que indica que un programador no debe agregar funcionalidades extras hasta que no sea necesario.

- Así evitamos sobre diseñar sistemas, una tendencia que retrasa y entorpece el alcance inicial de la tarea que estamos desarrollando.

"Aplicar siempre las cosas cuando realmente los necesita, no cuando lo que prevén que los necesita." - ***Ron Jeffries***