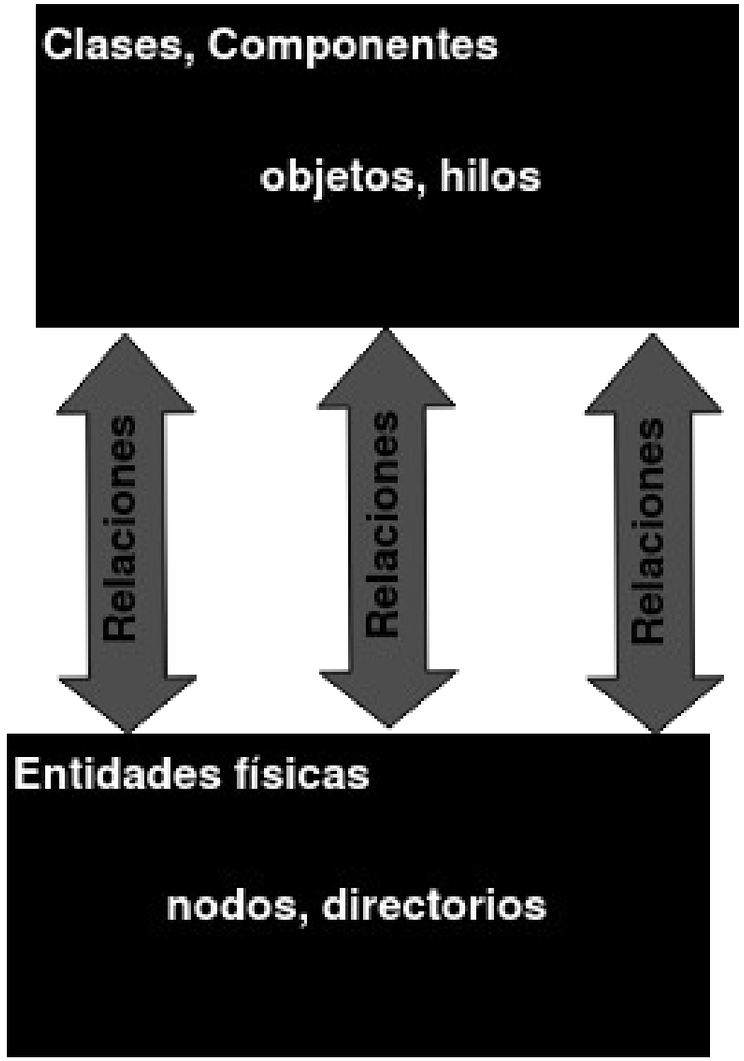
Arquitectura de Software

- **QUE ES UN A. S.**
  "las estructuras de un sistema, compuestas de elementos con propiedades visibles de forma externa y las relaciones que existen entre ellos".

  Los **elementos** pueden ser entidades que existen en tiempo de ejecución (**objetos, hilos**), entidades lógicas que existen en tiempo de desarrollo (**clases, componentes**) y entidades físicas (**nodos, directorios**).

  Las **relaciones** entre elementos dependen de propiedades visibles (públicas) de los elementos, quedando ocultos los detalles de implementación (Encapsulamiento). Finalmente, cada conjunto de elementos relacionados de un **tipo particular** corresponde a una estructura distinta, de ahí que la arquitectura esta compuesta por distintas estructuras.



Arquitectura de Software

- **Importancia**
  La arquitectura de software es de especial importancia ya que la manera en que se estructura un sistema tiene un impacto directo sobre la capacidad de este para satisfacer lo que se conoce como los atributos de calidad del sistema.

  Los atributos de calidad son parte de los requerimientos (no funcionales) del sistema y son características que deben expresarse de forma cuantitativa.

  Finalmente, los diseños arquitectónicos que se crean en una organización pueden ser reutilizados para crear sistemas distintos. Esto permite reducir costos y aumentar la calidad, sobre todo si dichos diseños han resultado previamente en sistemas exitosos.

- **El ciclo de desarrollo de la arquitectura**
  Desarrollo, que precede a la construcción del sistema, esta dividido en las siguientes etapas:

  Requerimientos: La etapa de requerimientos se enfoca en la captura, documentación y priorización de requerimientos que influencian la arquitectura.
  Diseño: La etapa de diseño es la etapa central en relación con la arquitectura y probablemente la más compleja. Durante esta etapa se definen las estructuras que componen la arquitectura.
  Documentación: Una vez creado el diseño de la arquitectura, es necesario poder comunicarlo a otros involucrados dentro del desarrollo.
  Evaluación. Dado que la arquitectura de software juega un papel crítico en el desarrollo, es conveniente evaluar el diseño una vez que este ha sido documentado con el fin de identificar posibles problemas y riesgos.

Arquitectura de Software

- **Importancia**
  La arquitectura de software es de especial importancia ya que la manera en que se estructura un sistema tiene un impacto directo sobre la capacidad de este para satisfacer lo que se conoce como los atributos de calidad del sistema.

  Los atributos de calidad son parte de los requerimientos (no funcionales) del sistema y son características que deben expresarse de forma cuantitativa.

  Finalmente, los diseños arquitectónicos que se crean en una organización pueden ser reutilizados para crear sistemas distintos. Esto permite reducir costos y aumentar la calidad, sobre todo si dichos diseños han resultado previamente en sistemas exitosos.

- **El ciclo de desarrollo de la arquitectura**
  Desarrollo, que precede a la construcción del sistema, esta dividido en las siguientes etapas:

  Requerimientos: La etapa de requerimientos se enfoca en la captura, documentación y priorización de requerimientos que influencian la arquitectura.
  Diseño: La etapa de diseño es la etapa central en relación con la arquitectura y probablemente la más compleja. Durante esta etapa se definen las estructuras que componen la arquitectura.
  Documentación: Una vez creado el diseño de la arquitectura, es necesario poder comunicarlo a otros involucrados dentro del desarrollo.
  Evaluación. Dado que la arquitectura de software juega un papel crítico en el desarrollo, es conveniente evaluar el diseño una vez que este ha sido documentado con el fin de identificar posibles problemas y riesgos.

RIA

- **RIA (rich Internet applications")**
  Surgen como una combinación de las ventajas que ofrecen las aplicaciones web y las aplicaciones tradicionales. Buscan mejorar la experiencia del usuario.

  RIA introduce un nuevo modelo de programación de aplicaciones que combina las ventajas de los dos modelos predominantes hasta el momento: el de las aplicaciones cliente-servidor y el del modelo multi-capa utilizado por las aplicaciones web, con un claro objetivo: mejorar la experiencia del usuario.

RIA

Ventajas

- Agilidad en la respuesta.
- Cálculos rápidos, controles prediseñados y funciones gráficas, interactivas y multimedia avanzadas.
- En muchos casos no requieren de instalación en el equipo del usuario (es suficiente con disponer de un navegador web), por lo que no es necesario pensar en distribuciones de software.
- Uso desde cualquier ordenador con acceso a Internet.

Retos

- Las RIA introducen cambios en los hábitos de navegación y en el uso de las aplicaciones web, y el usuario tardará un tiempo en digerirlos. Además, se dan ciertas complicaciones para el cumplimiento de los niveles de accesibilidad.
- Algunas de las tecnologías RIA que hacen uso del navegador web deberán superar algunos aspectos no resueltos aún, como la posibilidad de introducir "Favoritos" o la de utilizar el botón "Atrás" del navegador web.
- Las RIA deberán considerar la optimización de los motores de búsqueda o la capacidad de los sistemas de análisis para monitorizar sitios web construidos con esta tecnología.
- La incidencia de estas aplicaciones sobre aspectos relacionados con la seguridad deberá estudiarse en su globalidad a la hora de definir la arquitectura de sistemas y aplicaciones de la organización.

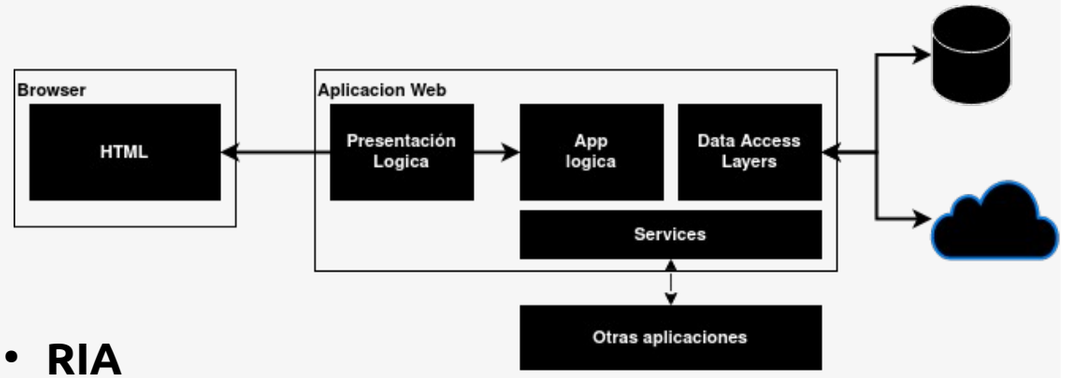
RIA

RIA Services

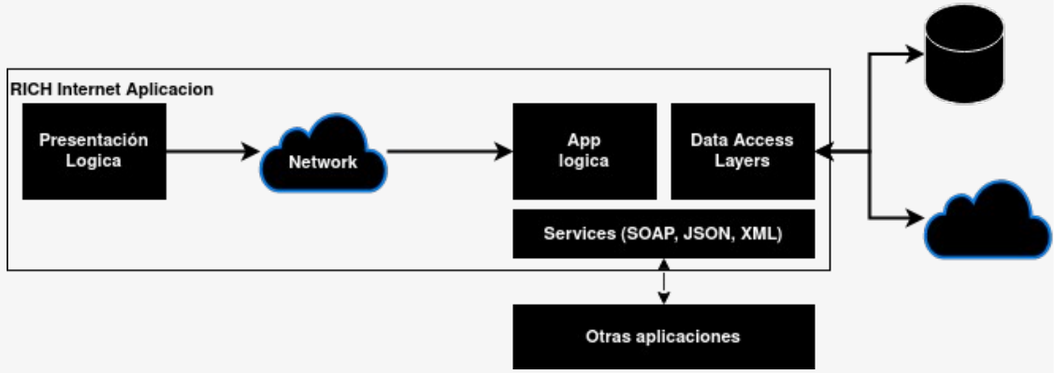
RIA desplaza las capas lógicas de la aplicación, la capa de presentación, a la parte cliente, convirtiéndole a éste en un "cliente pesado"…En el cliente tenemos parte de la aplicación, no sólo HTML y se complican las comunicaciones entre la lógica de presentación y la lógica de negocio.

- Si entre ambas capas esta **internet**, la comunicación ya no se puede hacer de manera directa y tendremos que pensar en desarrollar una capa de servicios en el servidor que el cliente pueda consumir, lo que complica el desarrollo, aumentando los tiempos y problemas con los que nos podemos encontrar: Crear la capa de servicios, exponer los métodos que necesita, crear proxys, validación, autenticación…

- **Tradicional**



- **RIA**



SPA Single Page App

Que es

- Rich Internet Applications (RIA) tiene una visión acerca de llevar la mecánica de una aplicación de escritorio a la web, convirtiendo el navegador en un entorno de escritorio, con el nacimiento de nuevas tecnologías da inicio con el mismo concepto a SPA.

  Una **Single Page Application SPA**, es un sitio web que recarga y muestra su contenido en respuesta a acciones propias de la navegación (enviar un formulario, clickar en un enlace, acceder a una sección interna…) sin tener que realizar peticiones al servidor para volver a cargar más código HTML. Las aplicaciones web básicas lo hacían por completo, AJAX lo intentaba resolver parcialmente y ahora SPA intenta reducirlo al mínimo necesario.

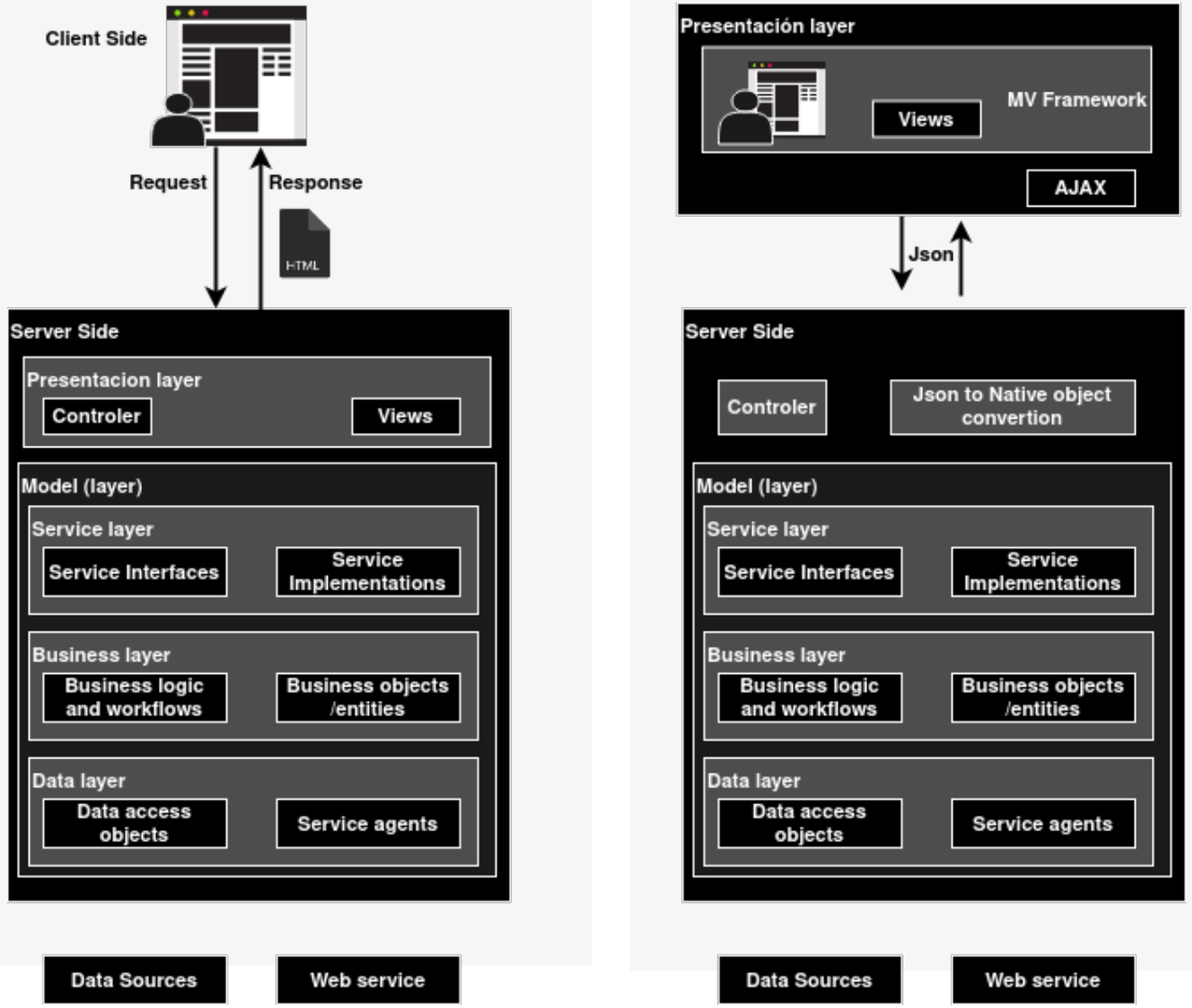
-

La filosofía de SPA es ofrecer una experiencia de usuario similar a la de una aplicación de escritorio pero desde un navegador, manejando datos, flujos y estructuras que dependen de una conectividad a Internet en menor medida que el resto de aplicaciones web tradicionales, de una manera ágil y grata, ofreciendo una experiencia de usuario muy cómoda, rápida y agradable.

Características de una SPA:

- Punto de entrada central: Un punto de entrada único, un que se va transformando y adaptando mediante acciones.
- Página fija, Vistas cambiantes, marco Unico
- Página fija, no URL fija
- Viajar ligera de equipaje: Las peticiones cliente — servidor tienden a ser más liviana y además muchos procesos quedan del lado del navegador web del cliente y los datos en formato Json

SPA Single Page App

- 

Web service

Que es

- Componente de Software que utiliza un conjunto de protocolos y estándares para intercambiar datos entre aplicaciones sobre una red. (OASIS y W3C)

  Los Servicios Web suelen ser considerados como APIs Web que pueden ser accedidos dentro de una red (principalmente Internet) y ejecutados en el sistema que los aloja.

- Son interoperables
- Superan las barreras geográficas
- Por naturaleza son flexibles
- Se basan en el protocolo HTTP

Qué es API

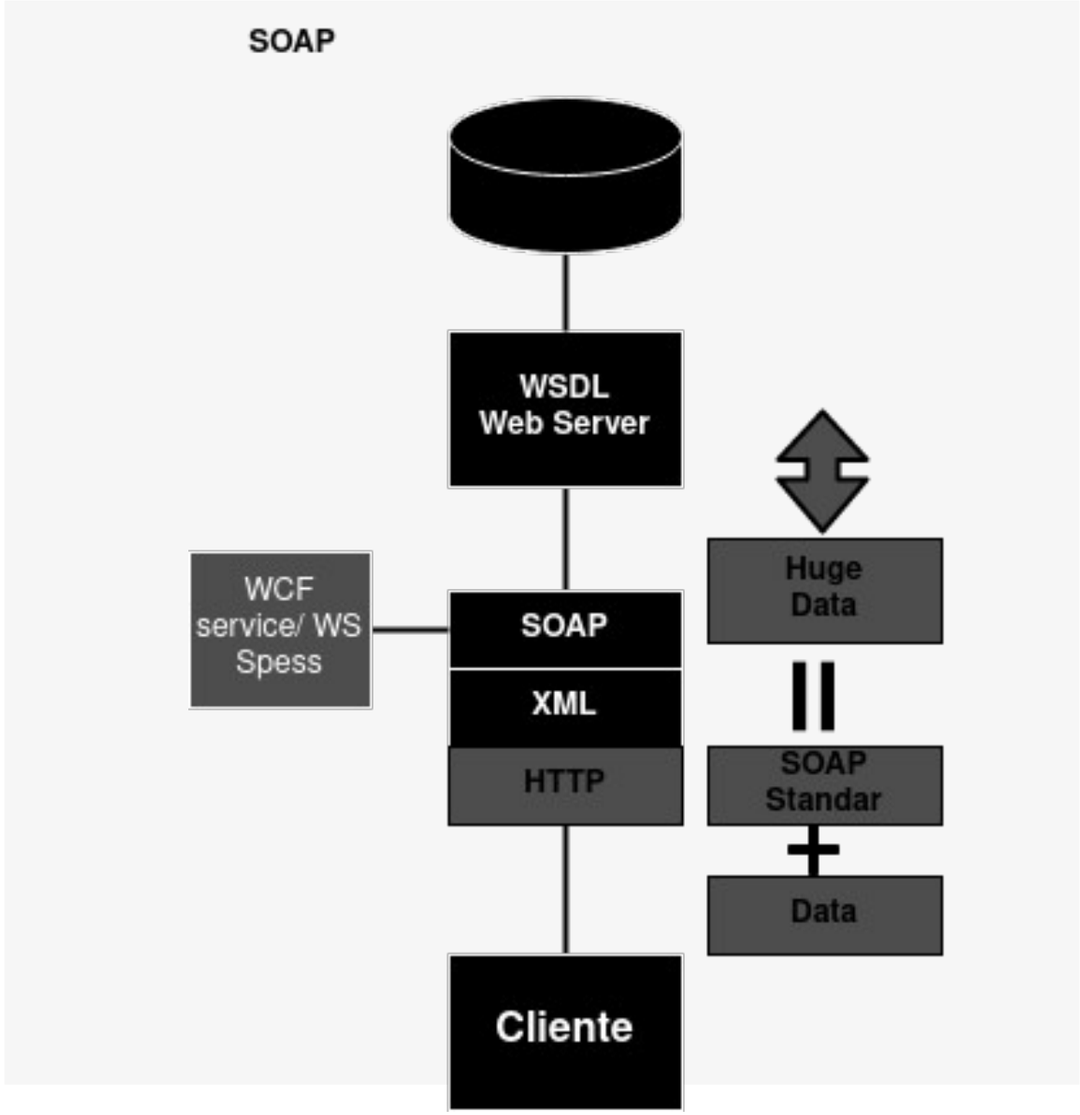
La palabra viene de Application Programming Interface, y no es más que un programa que permite que otros programas se comuniquen con un programa en específico

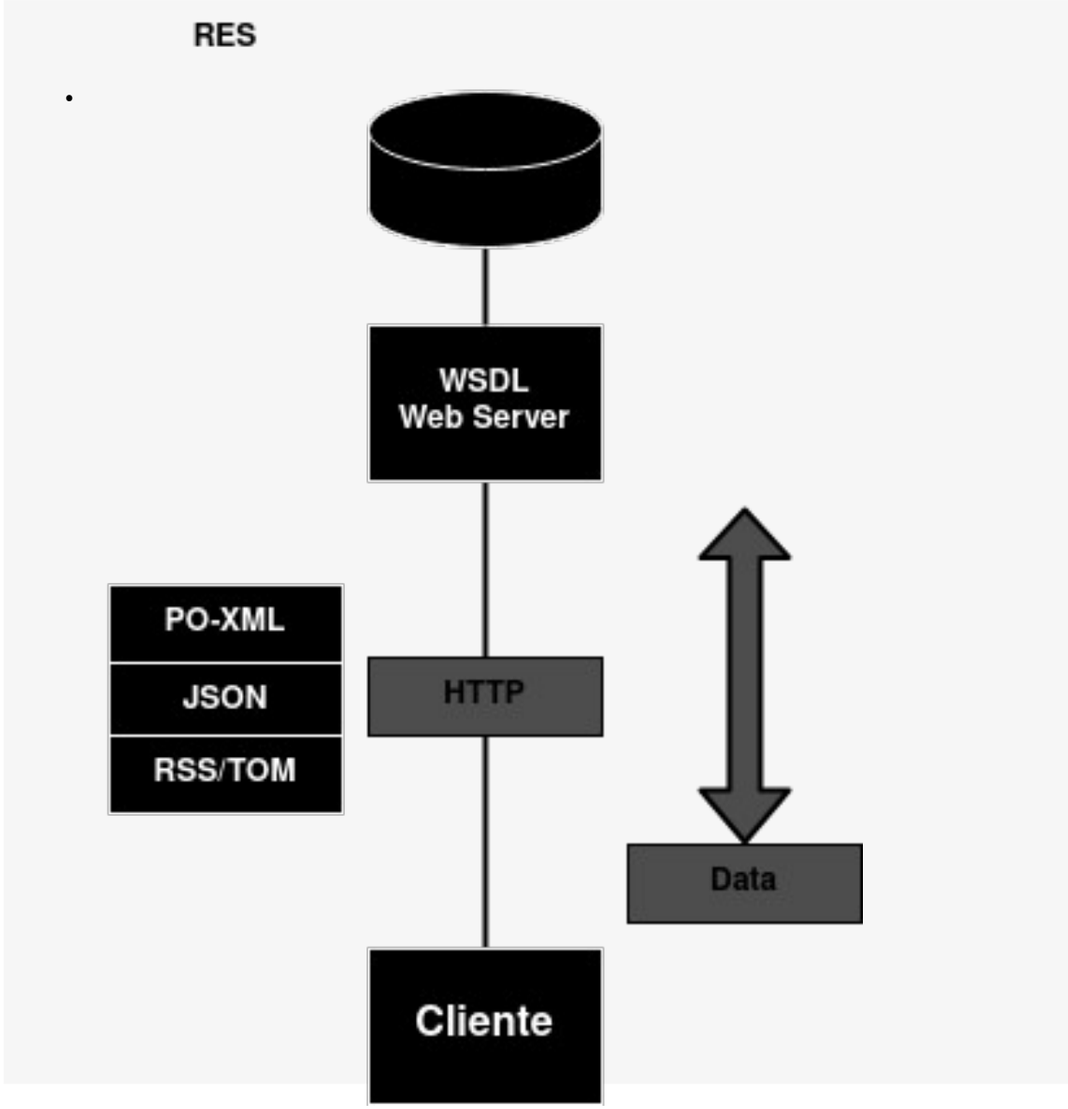
Tecnologias

- **SOAP** Simple Object Access Protocol es un protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML
  Entornos Aislados, Completo entornos Aislados, consume muchos recursos, Mayor seguridad.

- **RES** Resentational State Transfer arquitectura software para sistemas hipermedia distribuidos como la World Wide Web
- Dimensiones de la www, sencillo de consumir, aprovecha el ancho de banda (liviano), Mayor interoperabilidad.

Web service


SOAP
WSDL
Web Server
WCF
service/ WS
Spess
SOAP
XML
HTTP
Huge
Data
SOAP
Standar
Data
Cliente

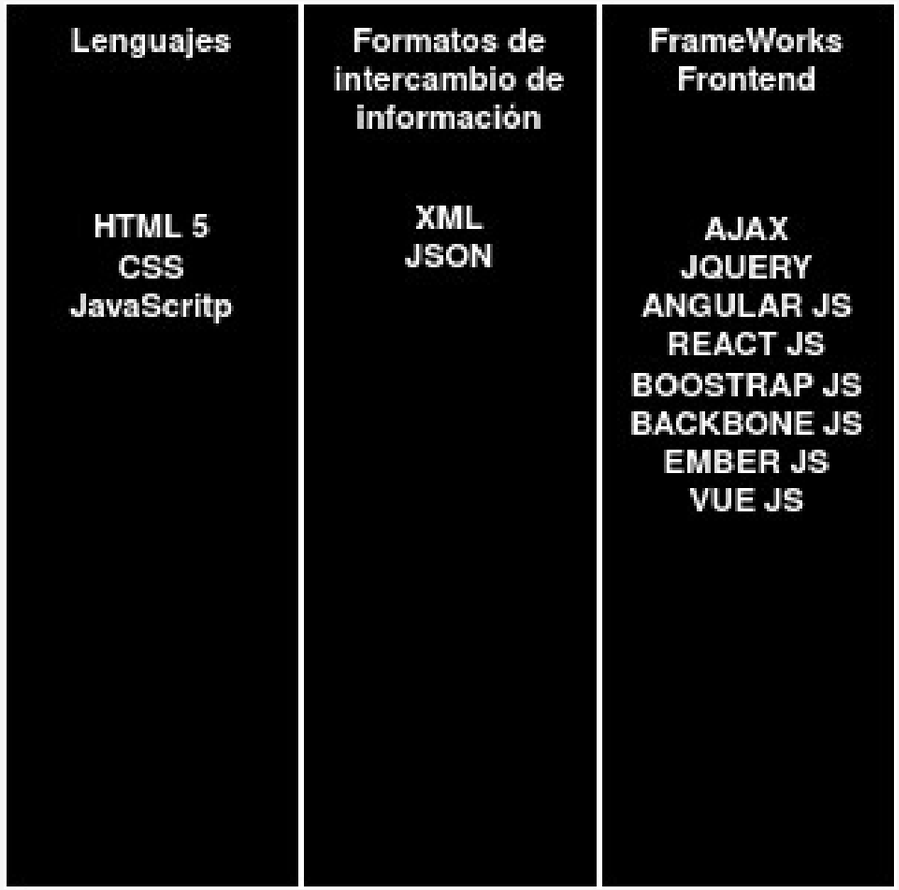

RES
WSDL
Web Server
PO-XML
JSON
RSS/TOM
HTTP
Data
Cliente

SPA Single Page App

- **Front-end**

  Se refiriere a todo lo relacionado con la interfaz gráfica de usuario. Es decir, lo que el usuario ve y con lo que interactúa. También es llamado client-side (lado cliente).

- Es así como la parte front- end tiene responsabilidades definidas, arquitectura, lenguajes, tecnologías y frameworks, tal y como se presenta en el gráfico que acompaña esta pantalla.



SPA Single Page App

Back-end

corresponde a los componentes de la aplicación que implementan los procesos de negocio y el almacenamiento de la información. También es llamado server-side (lado servidor).

Así como el front-end, el back-end tiene la responsabilidad de implementar la lógica de negocio y de interactuar con los módulos de almacenamiento de datos, para lo cual también hay diferentes arquitecturas, lenguajes y frameworks.

Arquitecturas	Lenguajes/ Técnologias	FrameWorks Frontend
Componentes Orietada a servicios (SOAP REST) MicroServicios (REST)	JAVA .NET Ruby Python	Spring Rubby on Rails Nodejs Django ASP.net Laravel ORACLE DB ORACLE middleware SQL server MySQL PostgreSQL MongoDB

SPA Single Page App

Vista completamente desacoplada

MVC clásico las páginas (JSP, PHP, ASP.NET…) que componían el interfaz del usuario ejecutaban código en lenguajes de programación propios del back-end. Como Java, PHP o ASP.NET. Esto hace que sea acoplada

EN **SPA** separamos completamente el back-end del front-end.

- El **back-end** se encarga de gestionar los datos y la lógica de negocio. Se ejecuta en el servidor y expone al front una API.
- El **front-end** constituye el interfaz de usuario. Se encarga de todo el dinamismo y la interacción. Se ejecuta (típicamente) en el navegador. Se comunica con la API mediante el intercambio de ficheros ligeros de datos (habitualmente JSON).

A nivel teórico es también una modalidad de MVC.

- El patrón SPA “obliga” a los desarrolladores de manera más consciente a separar front y back. Que es al fin y al cabo el objetivo último de ambas arquitecturas.

Que es Acoplamiento
Cuando la unidades hacen su trabajo totalmente independiente..

SPA Single Page App

Ventajas Desacoplar front-end y el back-end

- Probablemente la principal ventaja es que Dota a las interfaces web de la misma rapidez e interactividad fluida que las aplicaciones de escritorio.
- El back-end puede ser utilizado por otras interfaces web y por aplicaciones móviles. Es decir, los datos y la lógica pueden estar centralizadas y expuestas a través de una misma API que dé servicio a multitud de interfaces de usuario.
- Salvo en la carga inicial de la aplicación, el intercambio de información cliente y servidor es extremadamente ligero porque en cada petición y respuesta entre cliente y servidor lo que se intercambian son archivos JSON.
- Las SPA envían una solicitud al servidor, almacenan los datos localmente y conservan la capacidad de usar esos datos sin tener que realizar otra solicitud, trabaja si pierde conexión.
- Los frameworks de front-end están muy orientados a las componentización y la encapsulación. De modo que permiten crear grandes aplicaciones a partir de infinidad de piezas pequeñas independientes entre sí.
- Los desarrolladores back-end no necesitan lidiar con HTML, CSS y JS.

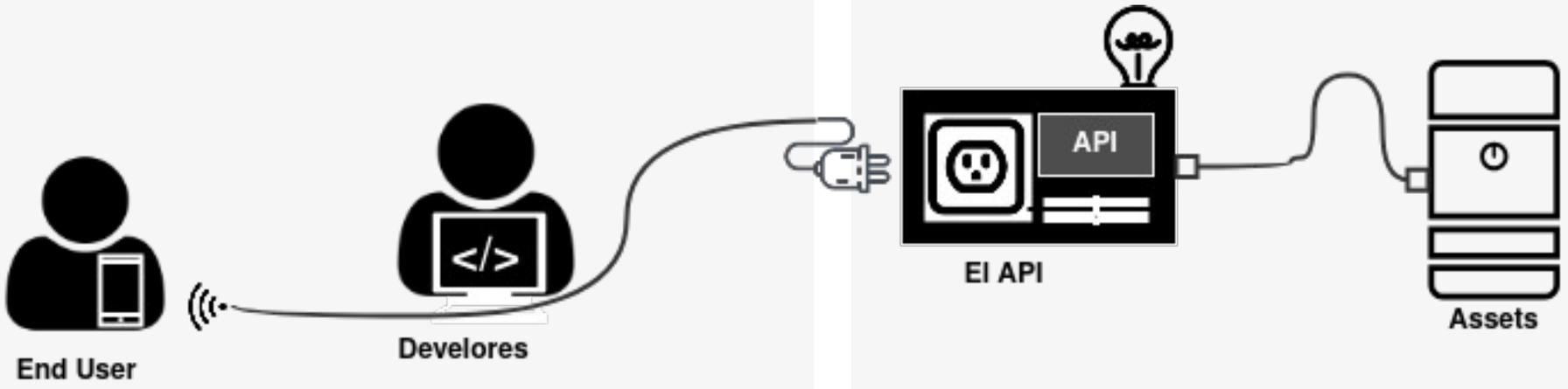
BackEnd

Qué es backend

- El backend es básicamente la arquitectura del lado del servidor de la web.

  Cualquier aplicación web, puede interactuar con el frontend, pero el procesamiento de su solicitud se realiza en el backend

- El Backend, también conocido como CMS o Backoffice, es la parte de la app que el usuario final no puede ver. Su función es acceder a la información que se solicita, a través de la app, para luego combinarla y devolverla al usuario final.



BackEnd

Componentes de un backend

- **Servidor**: es el lugar donde se almacena y ejecuta el **backend**. Estas son computadoras de alta potencia que proporcionan los recursos que necesita el **backend**, es decir, espacio de almacenamiento de archivos, potencia de procesamiento, seguridad y cifrado, bases de datos y otros servicios web..

- **Base de datos**: es el cerebro de cualquier sitio web que lo hace dinámico. La función de una base de datos es tomar la consulta y obtener los datos necesarios para el usuario.

- **Middleware**: es cualquier software (del lado del servidor) que facilita la conexión entre el **frontend** y el **backend**. Actúa como un medio que toma solicitudes del usuario y las proporciona al **backend** y luego facilita al usuario una respuesta dada por el **backend**.

- **Lenguajes de programación y FrameWorks**: hay una variedad de lenguajes disponibles en los que se puede codificar el **backend**, sin embargo, el lenguaje se elige en función del uso debido a la diferencia en su rendimiento, uso de memoria, compatibilidad, etc

BackEnd

Las funciones del Backend:

Acceder a la información que se pide, a través de la app: cuando usamos una aplicación móvil pedimos información. Esto implica que una parte de la app (el Backend), tiene que ser capaz de encontrar y acceder a la información que solicitemos. El proceso de búsqueda de datos a diferentes fuentes lo hace el Backend que debe ser capaz no sólo de encontrar la información precisa que el usuario requiere, sino también de acceder a ella de manera segura.

Combinar la información encontrada y transformarla: Una vez encontrada, el Backend combina la información para que resulte útil al usuario. la gran cantidad de información con la que el Backend trabaja, hace que su diseño deba ser sumamente preciso, ya que debe ser capaz de encontrar y filtrar lo que es relevante de lo que no, para luego combinarlo de manera útil.

Devolver la información al usuario: Finalmente, el Backend envía la información relevada de vuelta al usuario para que la consuma el **Frontend**.

BackEnd

Las funciones del Backend:

- Las APIs son las herramientas encargadas de transportar la información desde el Backend hasta el Frontend.

  Este proceso es hecho por en Frontend en dos fases, que tienen lugar en dos subcapas que conforman su estructura:

  La subcapa lógica, relacionada con el lenguaje específico, en el que la app ha sido desarrollada. En este punto, la subcapa lógica del Frontend se encarga de hacer una primera traducción del lenguaje en el que las APIs envían la información al lenguaje específico de cada sistema operativo. Este es un proceso que, a pesar de ocurrir en el Frontend de nuestra app, el usuario final no ve. Podríamos llamarlo el “Backend del Fronted”.

La subcapa visual, relacionada con el diseño estético de la app, en la que se encuentran todos los elementos que el usuario final puede ver. La capa visual del Frontend es donde, finalmente, se produce la conversión de la información a los elementos con el que el usuario final se relaciona, y a los que hacíamos referencia al comenzar este artículo.

BackEnd

Técnicas usadas en Backend y recomendaciones finales:

Las técnicas utilizadas para el desarrollo del Backend en término de lenguajes de programación debe ser claro e intuitivo.

Los lenguajes más populares de Backend se encuentran Ruby, Python, SQL, PHP y Java. Todos tienen ventajas e inconvenientes, por lo que dependerá de las necesidades específicas de nuestra app el escoger deU+0020acuerdo con los siguientes criterios:

- **Escalabilidad**: en Backend, flexibilidad del mismo para integrar nuevas estructuras y códigos. Este es un aspecto esencial, especialmente si nuestra app ha sido diseñada para utilizarse a largo plazo. En un contexto tan dinámico, la flexibilidad de nuestra app y su Backend es clave.
- **Seguridad**: debido a la constante interacción del Backend con las bases de datos, desarrollar el código siguiendo prácticas seguras es sumamente importante, más aún si nuestra app está diseñada para manejar información sensible, como datos personales, financieros o médicos. Para el desarrollo de un Backend seguro.
- **Robustez**: Capacidad para funcionar en cualquier contexto. Un Backend desarrollado de manera robusta nos asegura que, ante este tipo de situaciones, nuestra app siga funcionando, evitando los famosos **crashes**, que dan lugar a malas experiencias de uso y por lo tanto a la temida desinstalación de la app.

BackEnd

REST API

- REST Representational State Transfer (Transferencia de Estado Representacional). Es un conjunto de protocolos / estándares que describen cómo debe tener lugar la comunicación entre las computadoras y otras aplicaciones a través de la red.

- Expliquemos con un ejemplo.

  Suponga que quiere conducir un automóvil para saber que tiene que usar el acelerador, el embrague, etc. para conducirlo.

  De la misma manera, suponga que una aplicación web desea comunicarse con un servidor web.

- Entonces, una API Rest utiliza métodos GET, POST, PUT, DELETE para comunicarse.

- Por lo tanto, podemos concluir que **REST** es un estilo arquitectónico para diseñar aplicaciones de red. Utiliza métodos HTTP simples para comunicarse entre clientes y servidores. **Pero debe tenerse en cuenta que HTTP y REST no son lo mismo.**

- **REST API** es un conjunto de reglas que los desarrolladores siguen cuando crean su API. Una de esas reglas dice que deberíamos poder obtener una especificación cuando enlazamos a una URL específica. Esta URL es un localizador que cuando se busca genera una solicitud y la especificación que obtiene se llama respuesta.

BackEnd

REST API

- request consta de 4 partes:
- El **endpoint:** El end point o la ruta es la URL que hemos solicitado se estructura de la siguiente manera:

  root- endpoint /?

  Este suele ser el punto de partida de la API a la que estamos solicitando y la ruta correspondiente determina el recurso que está solicitando. Por ejemplo:

  https://api.github.com
  https://api.twitter.com

  Queremos obtener una lista de repositorios de una cuenta de Github de un usuario, a través de la API de Github.

Github nos pide que usemos la siguiente ruta:

/ users / : username/ repos

Los dos puntos (:) en una ruta denotan una variable, estas variables se reemplazan con sus valores. P.ej. : nombre de usuario: el nombre de usuario se reemplaza con el nombre de usuario de mi cuenta de Github Desarrollador Abhirupa.

Entonces, para aprovechar todos mis repositorios de github, el punto final / ruta será:

https://api.github.com/users/
DeveloperAbhirupa/repos

La parte final de un punto final es el parámetro de consulta, que nos permite modificar nuestra solicitud con pares clave-valor. Empiezan con un "?" Y cada par de parámetros se separa con un "&".

BackEnd

REST API

- El **method**: El método define el tipo de solicitud que se envía al servidor. Hay 5 tipos de métodos:

  GET: la solicitud GET se utiliza para obtener un recurso de un servidor. Al realizar una solicitud GET, el servidor busca los datos solicitados y nos los envía

  POST: Una solicitud POST crea una nueva entrada en la base de datos y vuelve a nosotros para decirnos si la reacción de la entrada ha sido exitosa o no.

  PUT / PATCH: estas solicitudes se utilizan para actualizar recursos en el servidor.

  DELETE: esta solicitud elimina un recurso del servidor.

BackEnd

REST API

El **body**: Los datos o el cuerpo consisten en el mensaje que queremos enviar al servidor.

Revisar HTTP

BackEnd

Arquitectura Orientada a Servicios SOA

- La arquitectura orientada a servicios (SOA) es el nexo que une las metas de negocio con el sistema de software. Su papel es el de aportar flexibilidad, desde la automatización de las infraestructura y herramientas necesarias consiguiendo, al mismo tiempo, reducir los costes de integración. SOA se ocupa del diseño y desarrollo de sistemas distribuidos y es un potente aliado a la hora de llevar a cabo la gestión de grandes volúmenes de datos, datos en la nube y jerarquías de datos.

- SOA es un estilo arquitectónico para la construcción de aplicaciones de software en base a servicios disponibles.

- Principales características destacan:

- Su flexibilidad, que permite la reutilización.

- Su versatilidad, que hace posible que los servicios puedan ser consumidos por los clientes en aplicaciones o procesos de negocio distintos.

- Sus posibilidades, que optimizan el trabajo con datos y su coordinación.

  SOA permite la reutilización de activos existentes para nuevos servicios que se pueden crear a partir de una infraestructura de TI que ya se había diseñado. además, conlleva ventajas de interoperabilidad entre las aplicaciones y tecnologías heterogéneas.

BackEnd

Arquitectura Orientada a Servicios SOA

La arquitectura orientada a servicios es fuente de ventaja competitiva ya que, por su configuración:

- Aumenta la eficiencia en los procesos.
- Amortiza la inversión realizada en sistemas.
- Reduce costes de mantenimiento.
- Fomenta la innovación orientada al desarrollo de servicios.
- Simplifica el diseño, optimizando la capacidad de organización.

Que busca SOAP

- Integración con los sistemas heredados.
- Reordenamiento de responsabilidades a través de reorganizaciones empresariales.
- Modernización de los sistemas obsoletos por razones económicas, funcionales o técnicas.
- Adquisición o decomiso de productos de software.

BackEnd

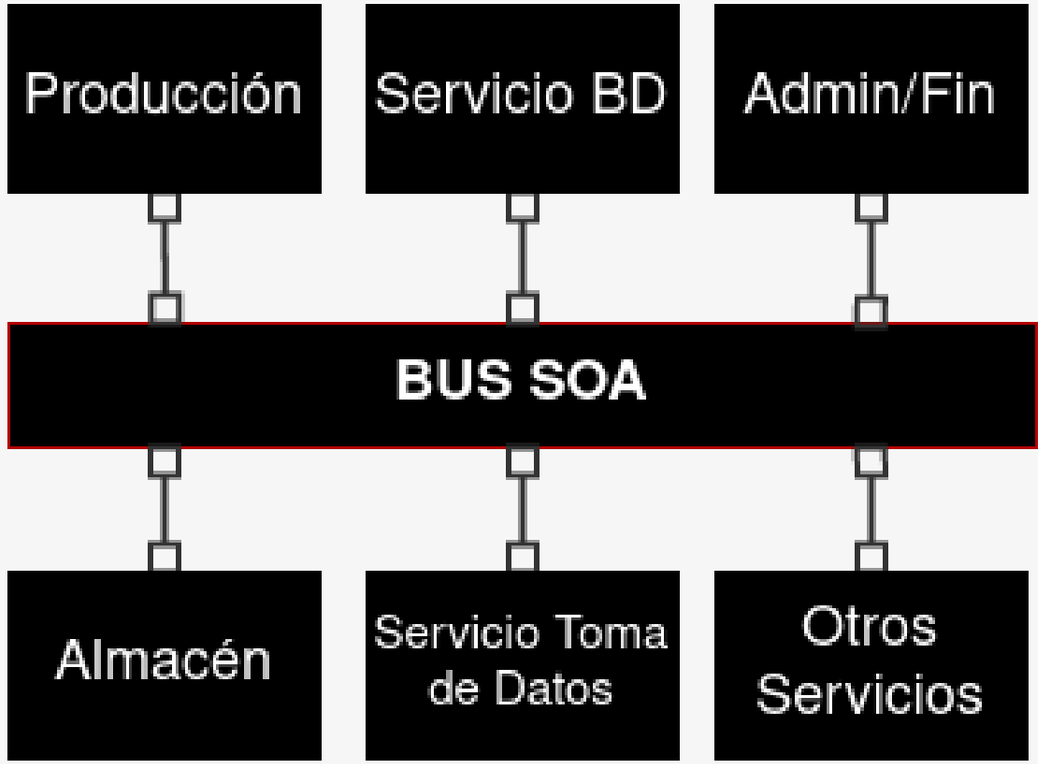
Arquitectura Orientada a Servicios SOA

Enfoques arquitectónicos de SOA
El diseño de los sistemas distribuidos se basaba en comunicaciones de red, seguridad, gestión transaccional, glosario y ubicación, con la arquitectura orientada a servicios es distinto, las preocupaciones se centran en dos aspectos:

- Comunicación.
- Integración de servicios.

A la hora de evaluar SOA:

- Capacidad de modificación.
- Rendimiento.
- Fiabilidad.
- Disponibilidad.
- Seguridad.



BackEnd

Arquitectura de Microservicios

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. 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.

BackEnd

Arquitectura de Microservicios

Ventajas

- **Modularidad**: al tratarse de servicios autónomos, se pueden desarrollar y desplegar de forma independiente. Además un error en un servicio no debería afectar la capacidad de otros servicios para seguir trabajando según lo previsto.
- **Escalabilidad**: como es una aplicación modular, se puede escalar horizontalmente cada parte según sea necesario, aumentando el escalado de los módulos que tengan un procesamiento más intensivo.
- **Versatilidad**: se pueden usar diferentes tecnologías y lenguajes de programación. Lo que permite adaptar cada funcionalidad a la tecnología más adecuada y rentable.
- **Rapidez de actuación**: el reducido tamaño de los microservicios permite un desarrollo menos costoso, así como el uso de "contenedores de software" permite que el despliegue de la aplicación se pueda llevar a cabo rápidamente.
- **Mantenimiento simple y barato**: al poder hacerse mejoras de un solo módulo y no tener que intervenir en toda la estructura, el mantenimiento es más sencillo y barato que en otras arquitecturas.
- **Agilidad**: se pueden utilizar funcionalidades típicas (autenticación, trazabilidad, etc.) que ya han sido desarrolladas por terceros, no hace falta que el desarrollador las cree de nuevo.

BackEnd

Arquitectura de Microservicios

Desventajas

- **Alto consumo de memoria:** al tener cada microservicio sus propios recursos y bases de datos, consumen más memoria y CPU.
- **Inversión de tiempo inicial:** al crear la arquitectura, se necesita más tiempo para poder fragmentar los distintos microservicios e implementar la comunicación entre ellos.
- **Complejidad en la gestión**: si contamos con un gran número de microservicios, será más complicado controlar la gestión e integración de los mismos. Es necesario disponer de una centralización de trazas y herramientas avanzadas de procesamiento de información que permitan tener una visión general de todos los microservicios y orquesten el sistema.
- **Perfil de desarrollador**: los microservicios requieren desarrolladores experimentados. Además de conocimiento sobre solución de problemas como latencia en la red o balanceo de cargas.
- **No uniformidad**: aunque disponer de un equipo tecnológico diferente para cada uno de los servicios tiene sus ventajas, si no se gestiona correctamente, conducirá a un diseño y arquitectura de aplicación poco uniforme.
- **Dificultad en la realización de pruebas:** debido a que los componentes de la aplicación están distribuidos, las pruebas y test globales son más complicados de realizar.
- **Coste de implantación alto:** una arquitectura de microservicio se supone un costo alt,o debido a costes de infraestructura y pruebas distribuidas.

BackEnd

Arquitectura de Microservicios

Actualmente gracias a los contenedores como Docker y orquestadores como Kubernetes, se han podido cambiar los servidores web tradicionales por contenedores virtuales mucho más pequeños y adaptables.

- Teniendo en cuenta las ventajas e inconvenientes de la implantación de una arquitectura de microservicios, lo más importante antes de decidir qué arquitectura elegir, será determinar qué solución es la que mejor se adapta a las necesidades del proyecto o la organización y qué ayudará a conseguir los objetivos propuestos.