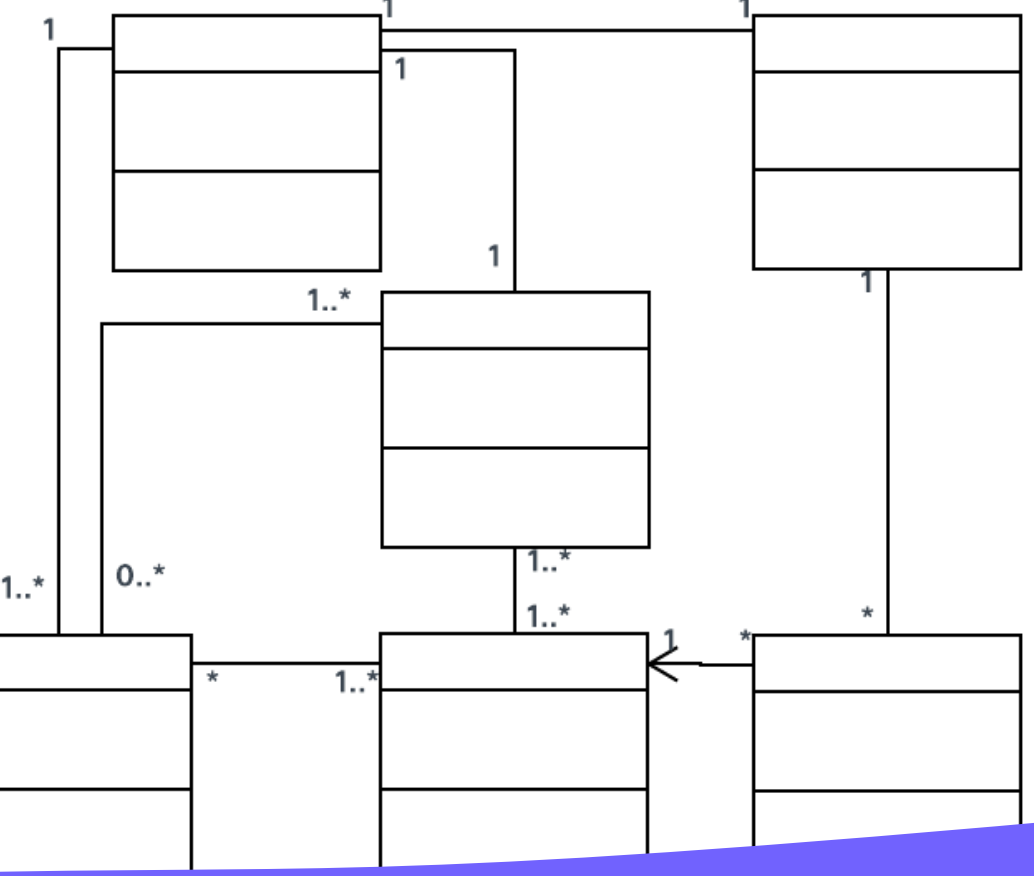
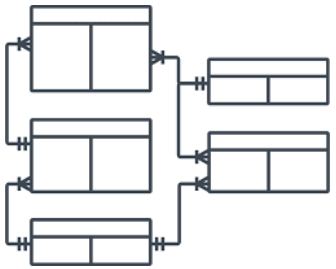




Diseño de Sistemas

UTN.BA
DPTO. INGENIERÍA EN SISTEMAS DE INFORMACIÓN
CÁTEDRA DISEÑO DE SISTEMAS

Agenda

- Sesión
- Cookies
- Stateless vs Stateful
- Recursos en Sistemas Web y su protección
- Patrones Arquitectónicos de Interacción
- Clientes Pesados y Livianos
- Single Page Application (SPA)
- Diagramas de Componentes y Despliegue
- Atributos de calidad (Arquitectónicos) y sus tradeoffs
- Escalabilidad

Sesión

- Las sesiones sirven para identificar al usuario que está intentando acceder a un recurso o ejecutar determinada funcionalidad sobre Sistema.
- Una sesión *puede* almacenar temporalmente información relacionada con las actividades del usuario mientras éste está conectado.
- No siempre es necesario contar con el concepto de sesión ya que depende del tipo de arquitectura que se haya establecido para el aplicativo.
- Para comprender cómo funcionan las sesiones es necesario conocer cómo se manejan las cookies…

Cookies

“Un archivo pequeño que se guarda en las computadoras de las personas para ayudar a almacenar las preferencias y demás información que se utiliza en las páginas web que visitan.

Las cookies pueden guardar la configuración de las personas en algunos sitios web y, a veces, se pueden utilizar para realizar un seguimiento de cómo los visitantes acceden a los sitios web e interactúan con ellos.”

[Google]

Cookies

- *Una cookie es un pequeño archivo de datos creado por el Servidor pero almacenado en el cliente.*
- *Las cookies viajan al Servidor que las creó en cada request que el cliente le realice.*
- *En su sentido más amplio, sirven generalmente para que el Servidor pueda identificar y reconocer al usuario que está realizando la request.*

Stateless vs Stateful (Web)

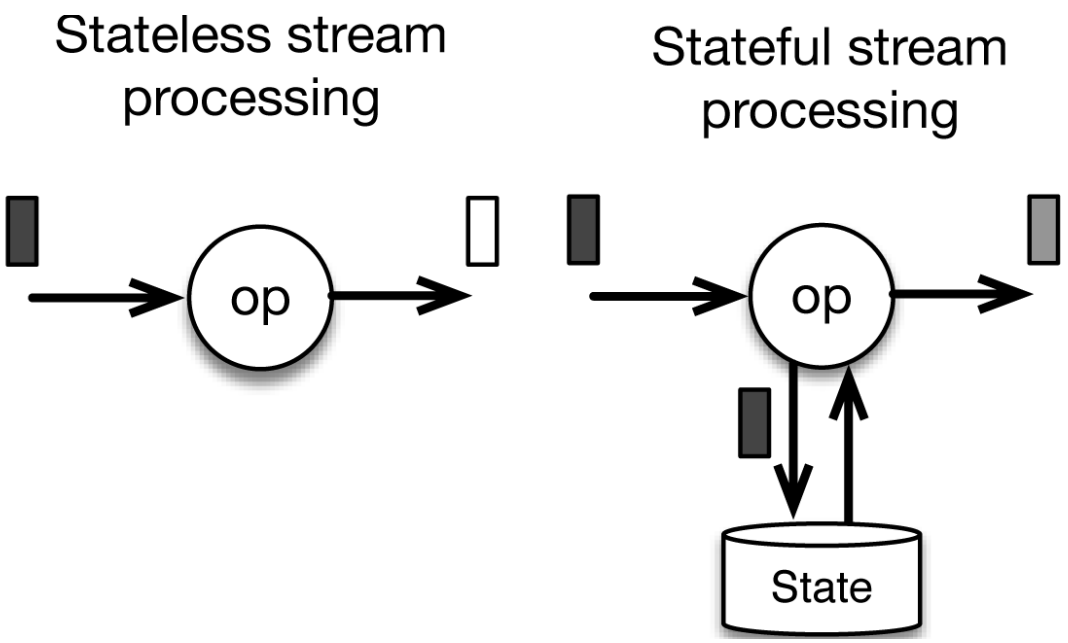
Stateless

- Un proceso, una aplicación o, genéricamente, un componente sin estado se refiere a los casos en que éstos están aislados.
- No se almacena información sobre las operaciones anteriores ni se hace referencia a ellas.
- Cada operación se lleva a cabo desde cero, como si fuera la primera vez.

Stateful

- Las aplicaciones, los procesos o, genéricamente, los componentes con estado son aquellos a los que se puede volver una y otra vez y éstos recuerdan "quiénes somos" y "qué hicimos anteriormente".
- Se realizan con el contexto de las operaciones anteriores, y la operación actual puede verse afectada por lo que ocurrió previamente.

Stateless vs Stateful (Web)



Stateless vs Stateful (Web)

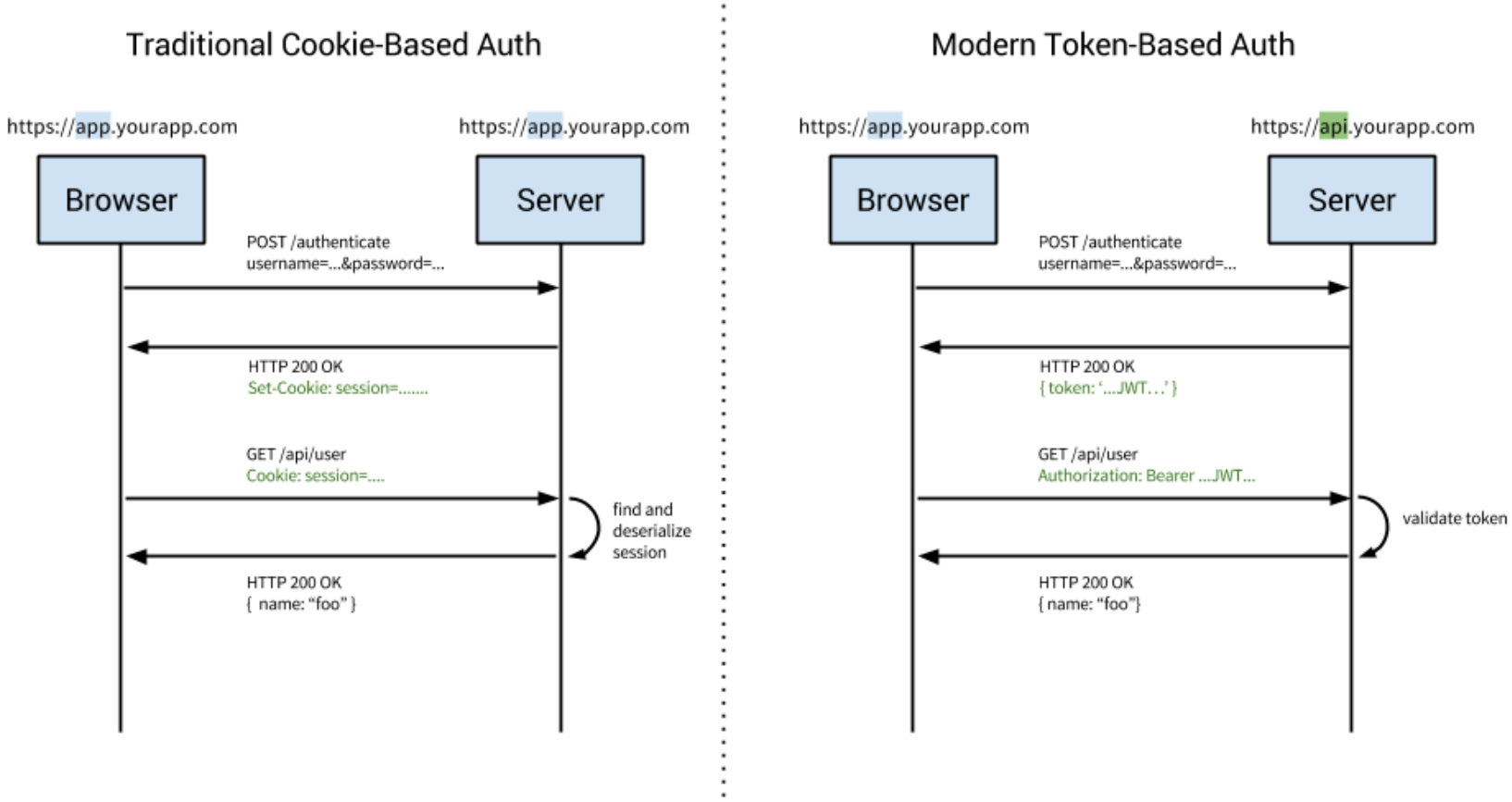
Stateless

- No se almacena sesión del lado del servidor.
- El usuario se identifica en cada solicitud que realiza.
- Para identificarse, el usuario envía un token al servidor. El servidor decodifica este token y descubre quién es el usuario que está realizando la solicitud.

Stateful

- Se almacena sesión en el servidor.
- El usuario se identifica una sola vez, no en cada solicitud realizada al servidor.
- En la solicitud donde el usuario inicia sesión (se validan sus credenciales), el servidor genera un espacio para él (en memoria/archivos/etc.), otorgándole un id. Este id es seteado en la cookie del usuario.
- En posteriores solicitudes, el servidor revisa la cookie del usuario (que llega en la request) y recupera la sesión por el id.

Stateless vs Stateful (Web)



Recursos en Sistemas Web

- Cuando diseñamos un Sistema con arquitectura Web debemos pensar en qué recursos vamos a permitir manipular.
- Podemos decir que un Recurso es una entidad que puede ser manipulada: puede ser dada de alta, modificada, eliminada o listada (buscada).
- Para poder identificar qué recursos debe exponer un Sistema se deben tener en cuenta los casos de uso.

Recursos en Sistemas Web

Recursos independientes/autónomos

- Recursos que no dependen de nadie, que tienen sentido de existencia propio.

Recursos dependientes/anidados

- Recursos que no son autónomos, que dependen de la existencia de otros.
- No tiene sentido listar todos estos recursos sin que sean condicionados por su recurso contenedor.
- A nivel objetos, podríamos pensar que estos recursos están involucrados en una relación de Composición.

Protección de Recursos

A menudo interesa proteger los recursos que expone un Sistema Web para que éstos no sean accedidos libremente por todos los tipos de usuarios existentes.

En ese sentido, existen al menos tres niveles de protección:

- Autenticación
- Autorización basada en roles
- Autorización basada en roles y permisos

Protección de Recursos

Autenticación

- Decimos que los recursos que están protegidos por autenticación exigen que el usuario se identifique antes de acceder a ellos.

- Si pensamos en un Sistema Web con capa de presentación visual, podríamos redirigir al usuario a una pantalla de login en el caso que esté queriendo acceder a una ruta protegida sin haber iniciado sesión.

- Si pensamos en un Sistema Web con capa de presentación de datos REST, podríamos devolverle al usuario un código de estado 401 (no autorizado) en el caso de que no esté autenticado.

Protección de Recursos

Roles

- Este tipo de protección depende del tipo de protección de Autenticación (es requerido que el usuario se identifique).
- Los recursos protegidos por roles exigen que el usuario, previamente identificado, cuente con un rol en particular para poder acceder a ellos.
- Este tipo de protección permite que no todos los usuarios manipulen los mismos recursos.

Protección de Recursos

Roles y Permisos

- Este tipo de protección depende del tipo de protección de Autenticación (es requerido que el usuario se identifique).
- Los recursos protegidos por roles y permisos exigen que el usuario, previamente identificado, cuente con un rol en particular y con uno o varios permisos para poder acceder a ellos.
- Este tipo de protección permite que no todos los usuarios con mismos roles accedan a los mismos recursos.
- También permite que usuarios con distintos roles cuenten con el mismo permiso.

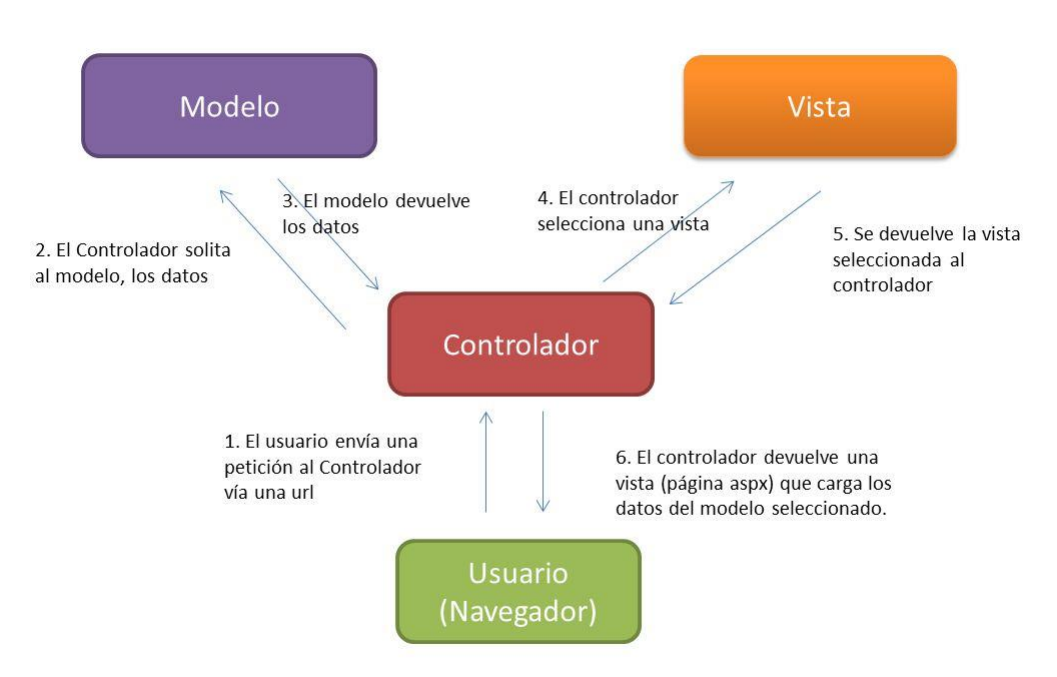
Protección de Recursos

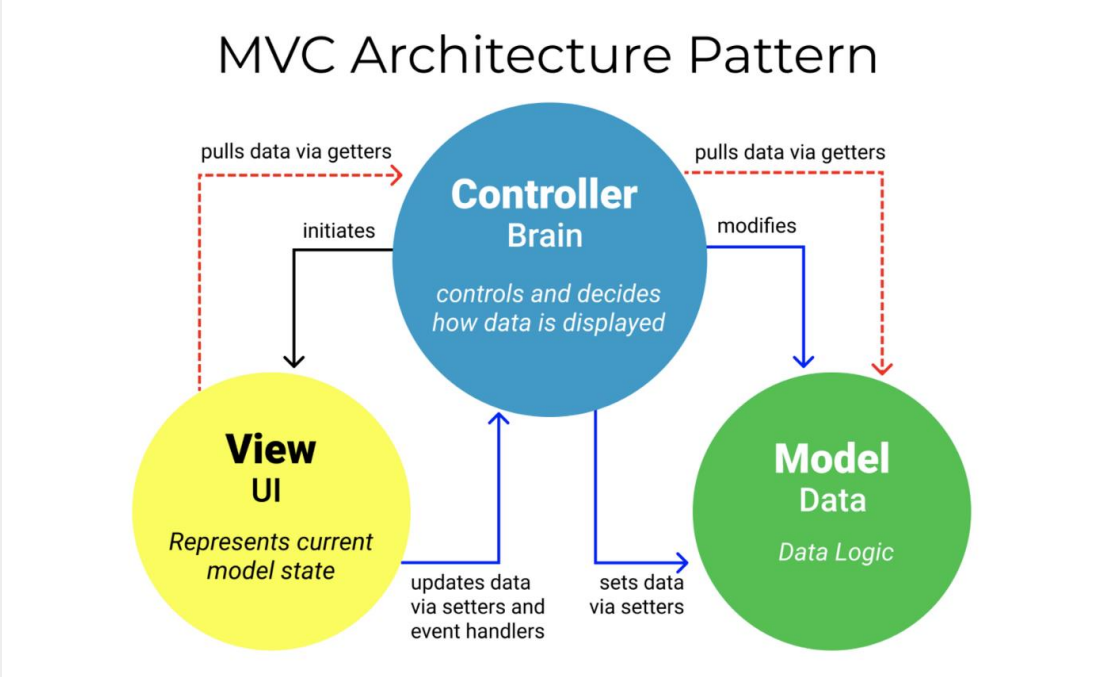
Roles y Permisos

- A veces es necesario, también, pensar en los diferentes alcances que se puede tener sobre los recursos.
- Por ejemplo: un rol "***Encargado***" podría llegar a ***ver*** el recurso ***empleados***, pero solamente los que tiene a su cargo, no todos.
- Los alcances podrían ser: propio, propios, todos.

Patrones Arquitectónicos de Interacción (Web)

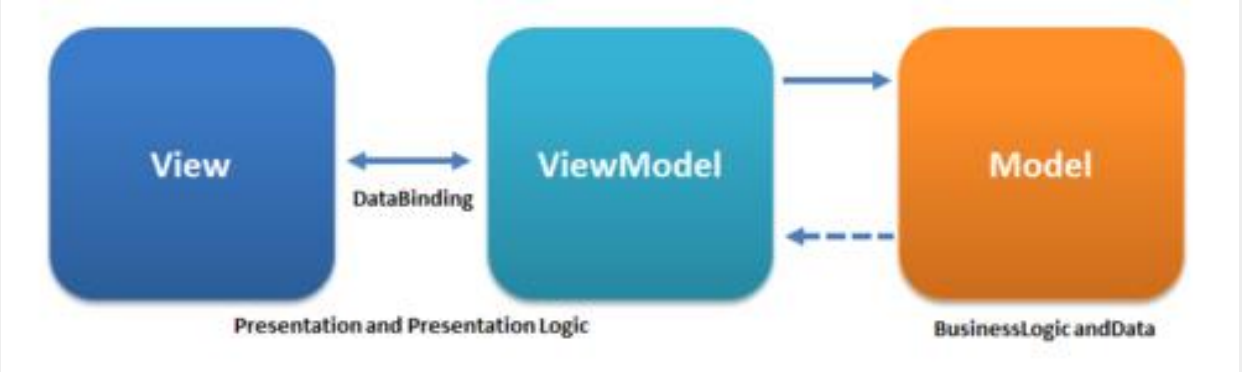
MVC: Modelo Vista Controlador





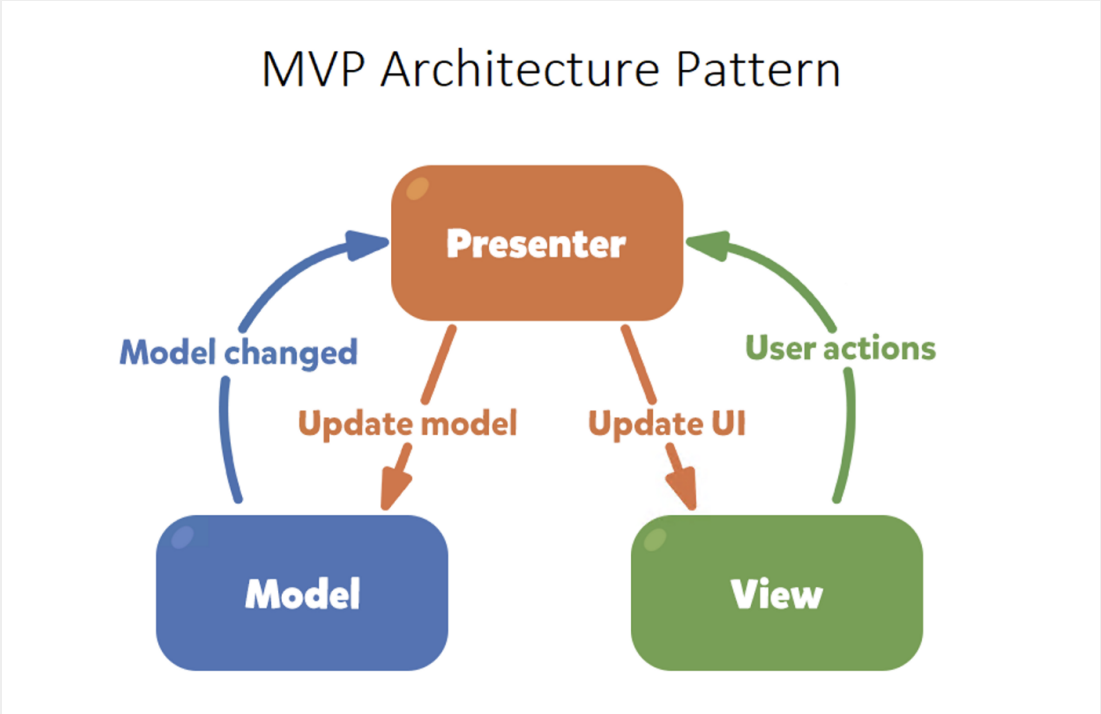
Patrones Arquitectónicos de Interacción (Web)

MVVM: Modelo Vista VistaModelo



Patrones Arquitectónicos de Interacción (Web)

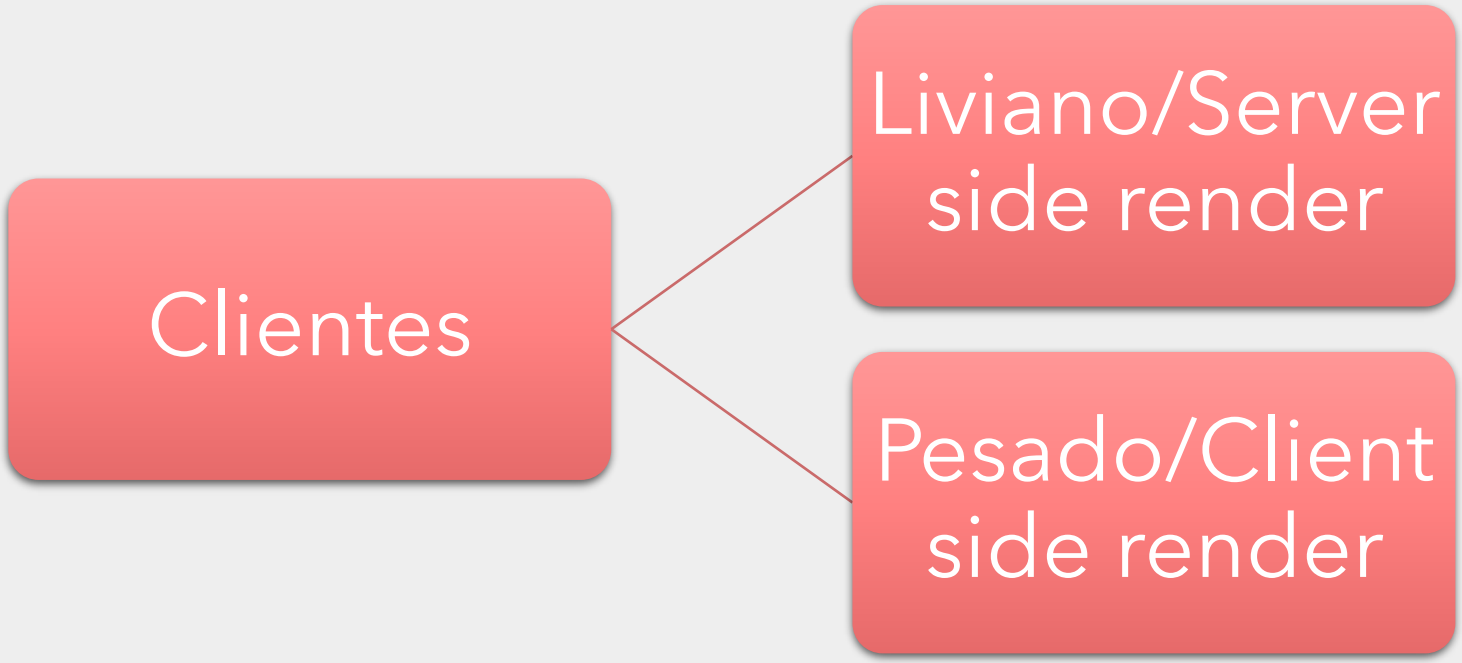
MVP: Modelo Vista Presentador



Tipos de Clientes

Cuando se tiene un Sistema diseñado bajo una Arquitectura Web y se necesita contar con capa de presentación visual, surge la necesidad de definir quién tendrá la responsabilidad de generar las vistas.

Tipos de Clientes



Tipos de Clientes

Liviano/Server side render

- Las vistas son generadas del lado del Servidor.
- Las respuestas a las solicitudes son –casi- siempre HTML.
- El navegador se encarga solamente de mostrar el contenido HTML.
- En ciertos casos, se puede contener lógica escrita en JavaScript del lado del Cliente.

Pesado/Client side render

- Las vistas son generadas del lado del Cliente.
- Las respuestas a las solicitudes son siempre JSON (o algún otro tipo de formato de intercambio similar).
- El Cliente posee toda la lógica necesaria para presentar los datos recibidos, mediante código JavaScript.

Tipos de Clientes

	Cliente Liviano	**Cliente Pesado**
Generación de la Vista	Servidor	Cliente
Stream (dato que se transmite)	HTML	JSON
Procesamiento	• Mayor en Servidor • Menor en Cliente	• Mayor en Cliente • Menor en Servidor
Uso de Memoria	• Mayor en Servidor • Menor en Cliente	• Mayor en Cliente • Menor en Servidor
Lógica de Interacción	Servidor	Cliente
Usabilidad (*)	-	+
Escalabilidad (*)	-	+
Mantenibilidad (*)	-	+

SPA: Single Page Application

"En pocas palabras, SPA son las siglas de Single Page Application. Es un tipo de aplicación web donde todas las pantallas las muestra en la misma página, sin recargar el navegador."

"Técnicamente, una SPA es un sitio donde existe un único punto de entrada, generalmente el archivo index.html. En la aplicación no hay ningún otro archivo HTML al que se pueda acceder de manera separada y que nos muestre un contenido o parte de la aplicación, toda la acción se produce dentro del mismo index.html."

SPA: Single Page Application

"Aunque solo tengamos una página, lo que sí tenemos en la aplicación son varias vistas, entendiendo por vista algo como lo que sería una pantalla en una aplicación de escritorio. En la misma página, por tanto, se irán intercambiando vistas distintas, produciendo el efecto de que tienes varias páginas, cuando realmente todo es la misma, intercambiando vistas."

SPA: Single Page Application

“El efecto de las SPA es que cargan muy rápido sus pantallas. De hecho aunque parezcan páginas distintas, realmente es la misma página, por eso la respuesta es muchas veces instantánea para pasar de una página a otra.”

SPA: Single Page Application

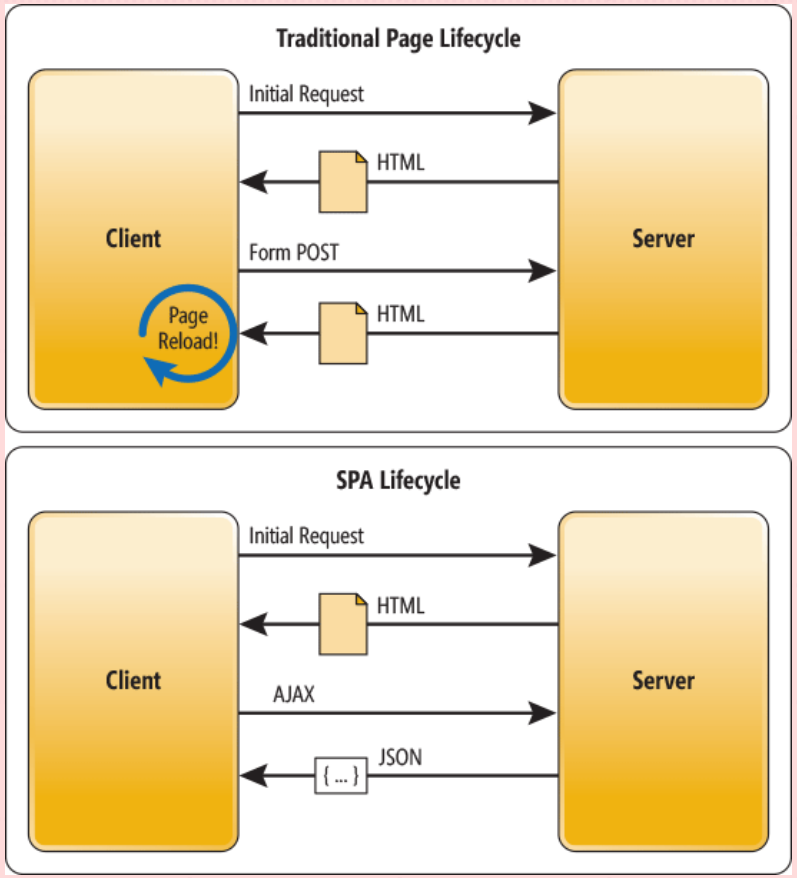
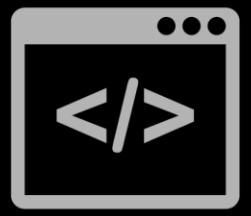
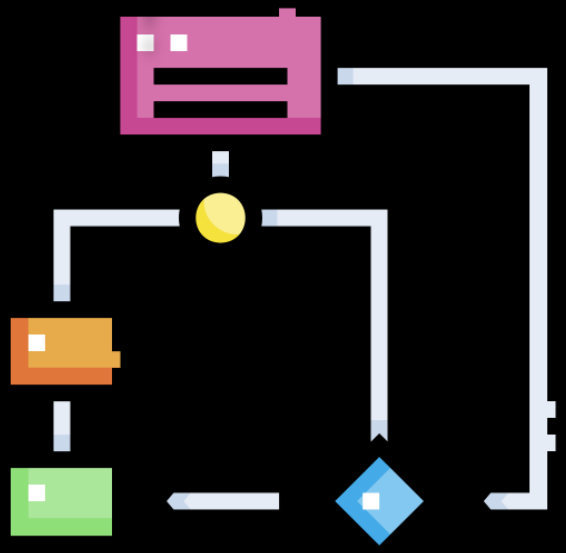


Diagrama de Despliegue y Componentes





UNIFIED MODELING LANGUAGE™

Diagrama de Componentes

- El Diagrama de Componentes es un Diagrama de Arquitectura UML que presenta un nivel de abstracción más alto que un Diagrama de Clases, por ejemplo.

- Es un diagrama estático que busca presentar todos los componentes de software existentes en el Sistema y sus relaciones.

Diagrama de Componentes

Un componente se representa como un rectángulo, con dos pequeños rectángulos superpuestos perpendicularmente en el lado izquierdo.

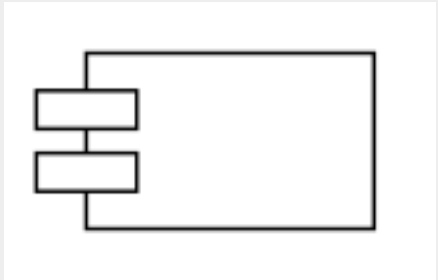


Diagrama de Componentes

- Para que dos componentes de Software interactúen entre sí, al menos uno de ellos debe exponer una interface.
- Esta interface, siendo más precisos, podría materializarse en una API REST, por ejemplo; o simplemente en un conjunto de funciones.
- Para dejar asentado gráficamente que un componente expone una interface se utiliza un “círculo”.
- Para dejar asentado gráficamente que un componente usa los servicios expuestos por el otro, se utiliza una “semicircunferencia”.

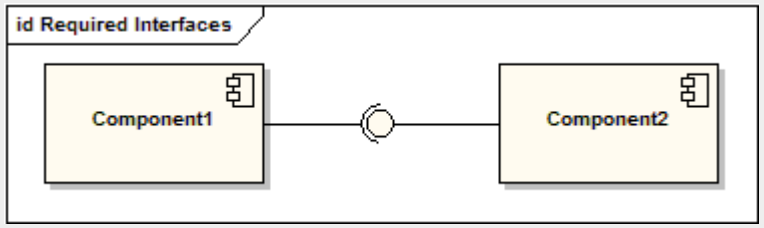


Diagrama de Componentes

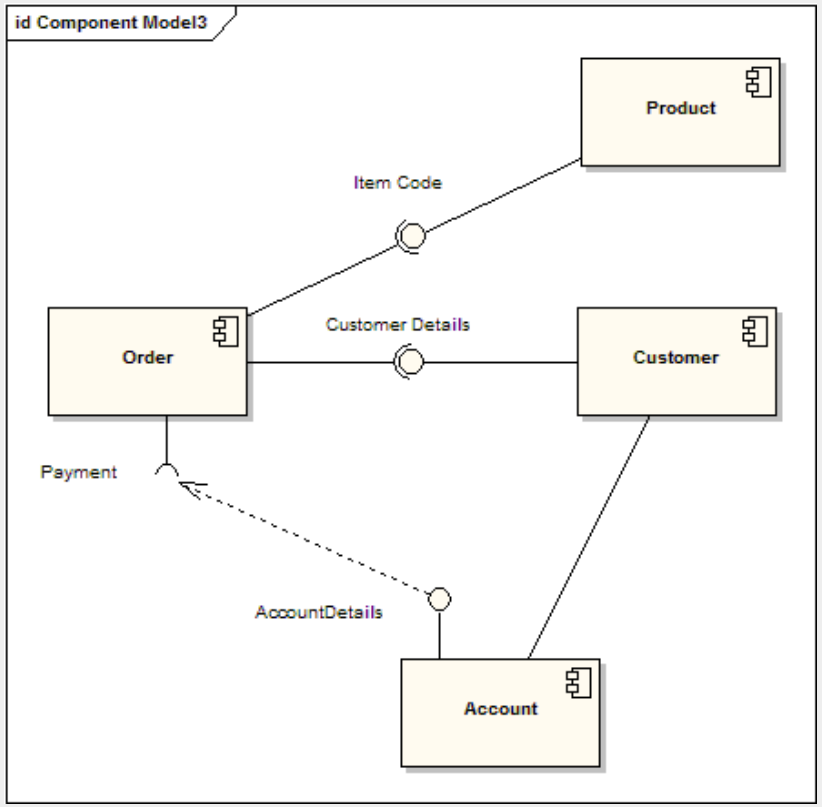


Diagrama de Despliegue

- El Diagrama de Despliegue es es un Diagrama de Arquitectura UML.
- Es un diagrama estático – físico, que busca presentar todos los nodos existentes en la arquitectura de un Sistema y los componentes de Software que éstos contienen.

Diagrama de Despliegue

Elementos

Artefacto: Un producto desarrollado por el software, representado por un rectángulo con el nombre y la palabra "artefacto" encerrado por flechas dobles.

Asociación: Una línea que indica un mensaje u otro tipo de comunicación entre nodos.

Componente: Un rectángulo con dos pestañas que indica un elemento de software.

Dependencia: Una línea discontinua que termina en una flecha, que indica que un nodo o componente depende de otro.

Interfaz: Un círculo que indica una relación contractual. Aquellos objetos que se dan cuenta de que la interfaz debe completar cierto tipo de obligación.

Diagrama de Despliegue

Elementos

Nodo: Un objeto de hardware o software, mostrado por un cuadro tridimensional.

Nodo como contenedor: Un nodo que contiene otro nodo dentro de sí, como en el ejemplo siguiente, en el que los nodos contienen componentes.

Estereotipo: Un dispositivo contenido dentro del nodo, presentado en la parte superior del nodo, con el nombre entre flechas dobles a manera de corchetes.

Diagrama de Despliegue

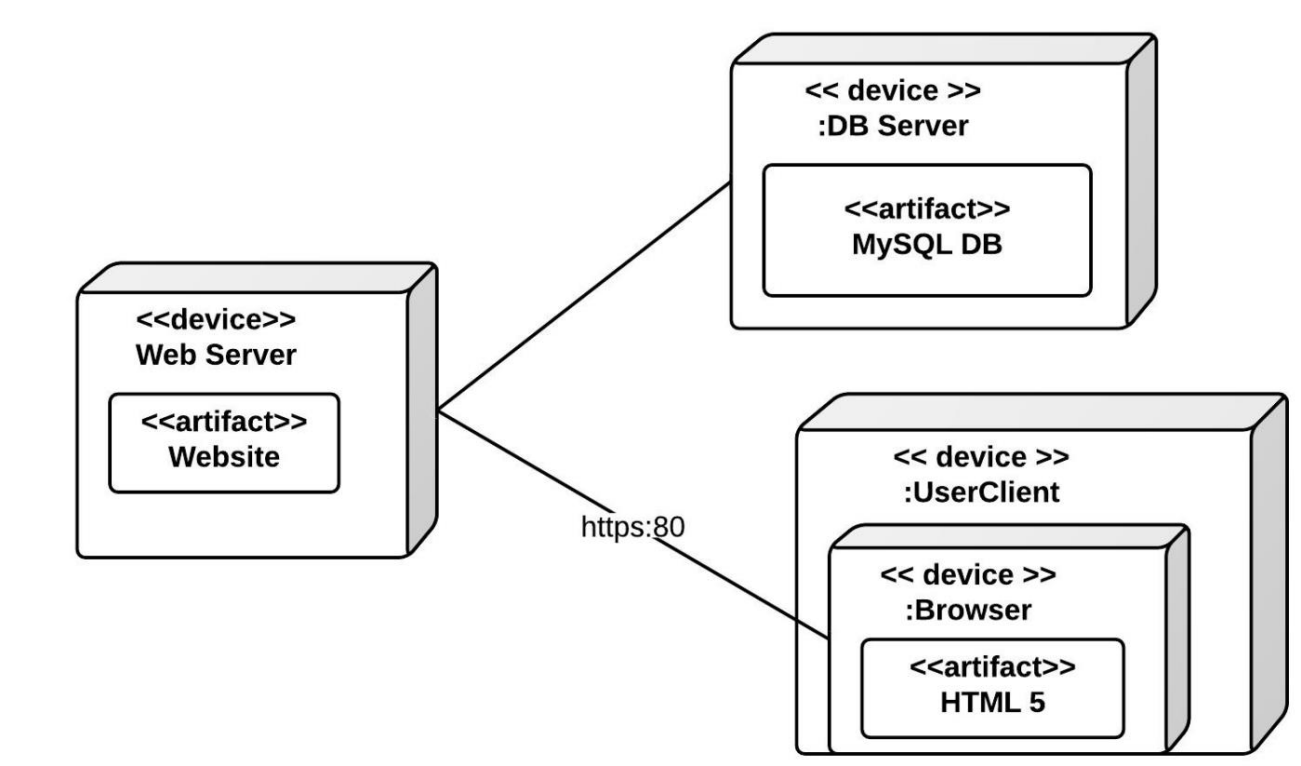


Diagrama de Despliegue

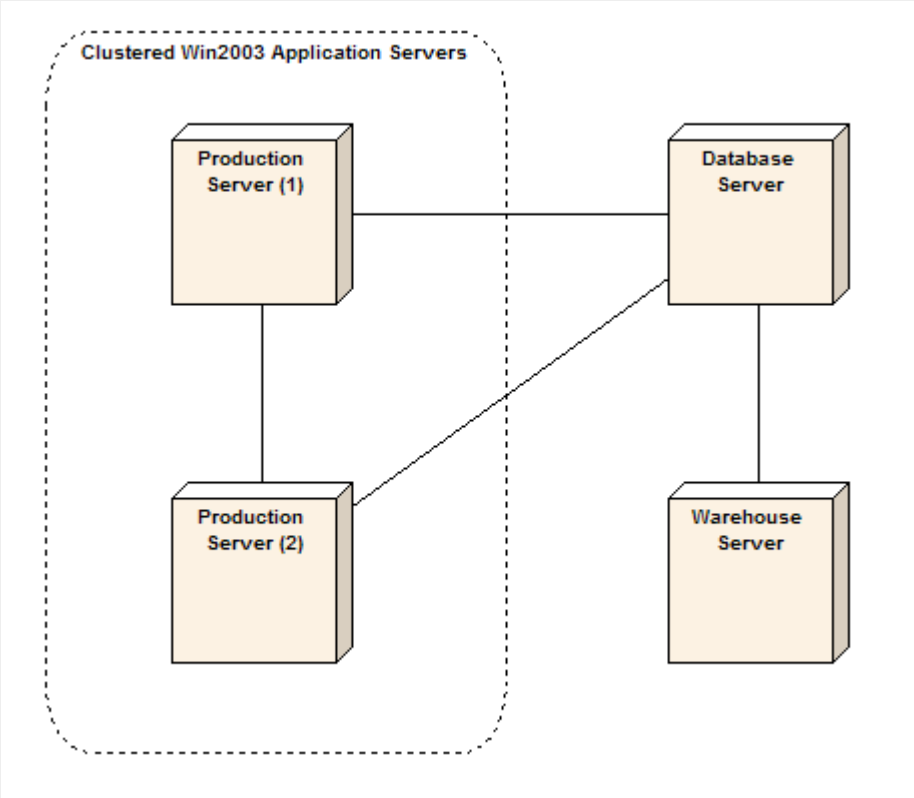


Diagrama de Despliegue + Componentes

¿Qué ***Componentes*** podemos encontrar en una Arquitectura?

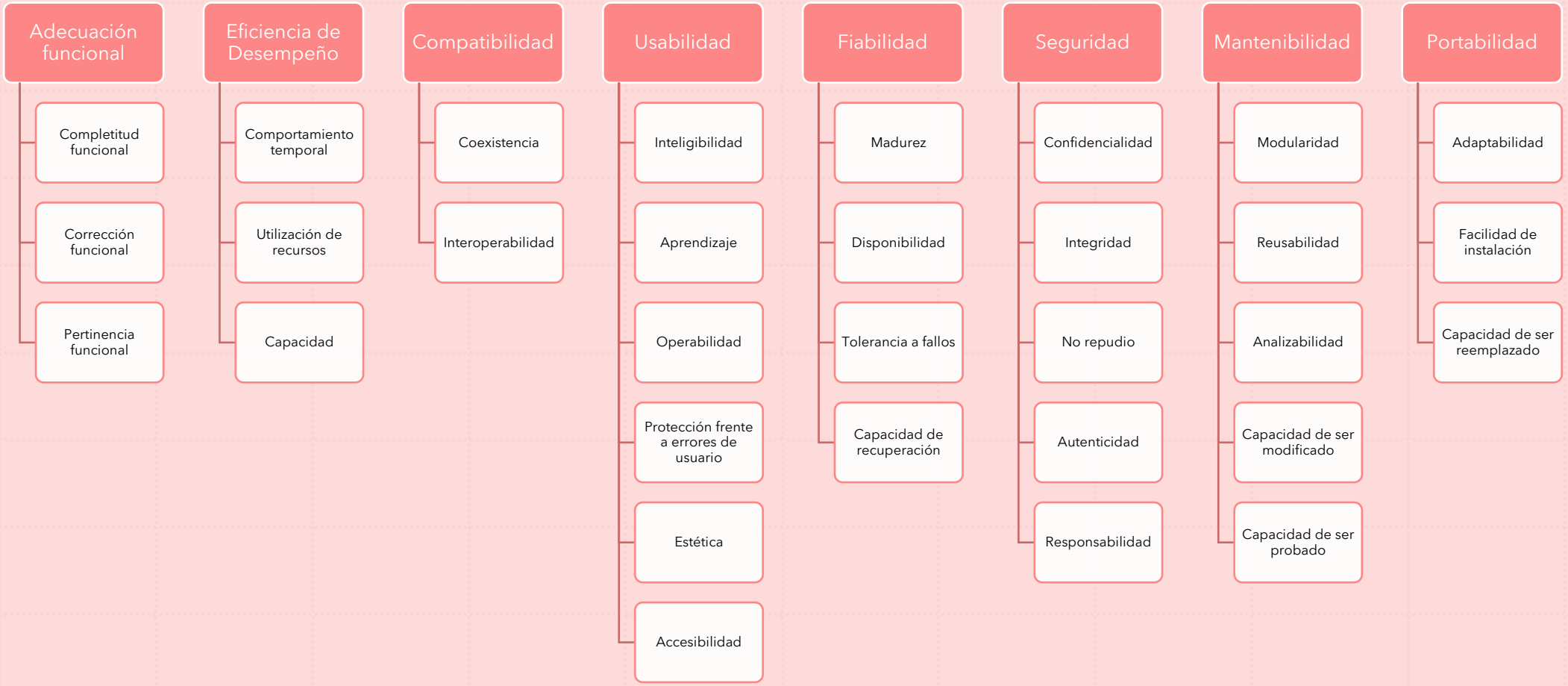
- Navegador Web
- Aplicación Móvil
- Motor de Base de Datos
- Load Balancer
- Cache
- Biblioteca
- ….

Diagrama de Despliegue + Componentes

¿Qué ***Nodos*** podemos encontrar en una Arquitectura?

- Servidor de Base de Datos
- Servidor de Aplicación
- Servidor de Archivos
- Dispositivo Móvil
- Computadora Personal
- Dispositivo Embebido
- ...

Atributos de Calidad (Arquitectónicos)



Atributos de Calidad (Arquitectónicos)

Aquellos atributos de calidad que –casi- siempre podemos evaluar son:

- Eficiencia de Desempeño – Utilización de recursos (“Performance”)
- Fiabilidad – Disponibilidad
- Fiabilidad – Tolerancia a fallos
- Seguridad (en general)

Tradeoffs entre Atributos de Calidad

Los **Atributos de Calidad** pueden **entrar en conflicto** unos con otros:

- Performance vs. Seguridad
- Seguridad vs. Disponibilidad (aunque están vinculados)
- Performance vs. Modificabilidad

Se deben **evaluar** los múltiples **atributos de calidad** con el objetivo de **Diseñar** un **Sistema** “Good Enough” para los **Stakeholders**.

Escalabilidad

¿Qué sucedería si un único Servidor tuviera que atender solicitudes de un número muy elevado de clientes a la vez? ¿Podría hacerlo solo?

Si pensamos en este escenario, notamos que el Servidor podría ser un cuello de botella:

- La performance podría degradarse sustancialmente (tiempos de respuesta muy elevados)
- La disponibilidad podría verse afectada ya que si dicho Servidor se “cae”, todos los clientes quedarían sin poder realizar sus solicitudes.

Escalabilidad

Dos son las formas por las que se puede optar para resolver este problema:

- Crecer verticalmente
- Crecer horizontalmente

Escalabilidad

Escalabilidad Vertical

- La escalabilidad vertical implica agregarle más recursos al Servidor físicamente (memoria, disco, procesamiento) para que éste pueda atender más solicitudes de forma concurrente.
- Existe una limitación física ya que llegará un punto en el cual no se podrán agregar más recursos al equipo Servidor.
- Este tipo de escalabilidad no soluciona el problema de Disponibilidad, sino solamente pretende resolver el problema de Performance.

Escalabilidad

Escalabilidad Horizontal

- La escalabilidad horizontal implica agregar más equipos Servidores para atender las solicitudes de los clientes, distribuyendo la carga entre ellos.
- No existe limitación física para este tipo de escalabilidad (se pueden agregar tantos equipos como se dispongan).
- Posee una complejidad extra ya que se deben distribuir las solicitudes entre todos los equipos Servidores existentes. La distribución puede realizarse por algún algoritmo: Sticky Session, Round Robin, etc.
- Es necesario contar con un Balanceador de Cargas, componente que se encargará de la distribución de las solicitudes.
- Este tipo de escalabilidad pretende solucionar el problema de Disponibilidad y de Performance.

Gracias