



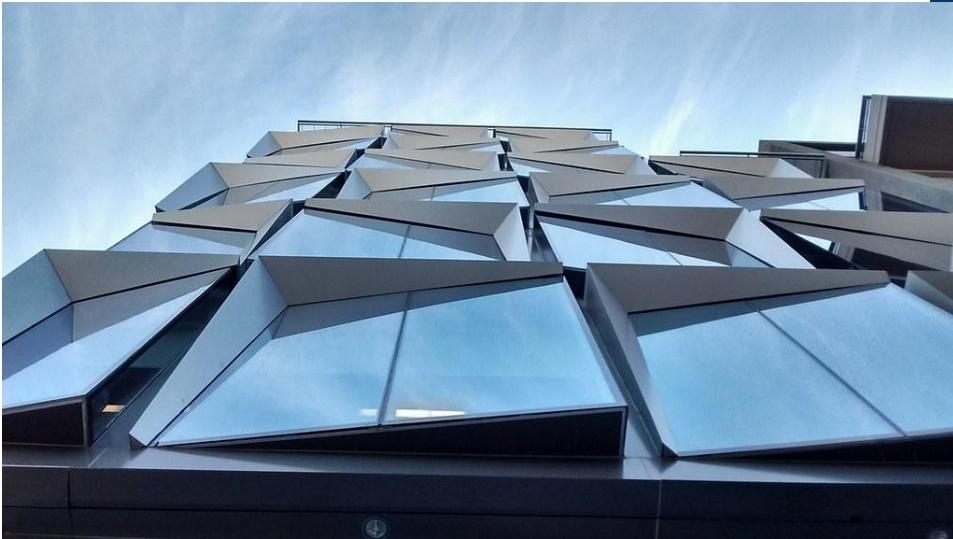
Arquitectura de sistemas, comportamiento y optimización

Week 6

M.Sc. Juan Luis Flores Pineda

Agenda

- Modelo y Diseño de Arquitectura
 - ◆ Diseño
 - ◆ Validación
 - ◆ Prototipos
- Laboratorio



Diseñando la
arquitectura

Diseñando la arquitectura

Para diseñar una arquitectura que satisfaga todos los requerimientos de calidad, funcionales y de negocio vamos a hacer uso de un método llamado *Attribute-Driven Design* (ADD).

Este método toma como base el conjunto de escenarios de atributos de calidad y emplea el conocimiento sobre la relación entre el logro de atributos de calidad y la arquitectura. Este es una extensión de otro métodos conocidos como Rational Unified Process.

Attribute-Driven Design

Es un enfoque para definir una arquitectura que basa el proceso de descomposición en los atributos de calidad que el software debe cumplir.

Es decir un proceso de descomposición recursivo donde, en cada etapa, se eligen tácticas y patrones arquitectónicos para satisfacer un conjunto de escenarios de calidad y luego se asigna funcionalidad para instanciar los tipos de módulos proporcionados por el patrón.

ADD se posiciona en el ciclo de vida después del análisis de requisitos y, como hemos dicho, puede comenzar cuando los controladores arquitectónicos se conocen con cierta confianza.



Attribute-Driven Design

La salida de ADD son los primeros niveles de una vista de descomposición del módulo de una arquitectura y otras vistas, según corresponda.

No todos los detalles de las vistas son el resultado de una aplicación de ADD; el sistema se describe como un conjunto de contenedores para la funcionalidad y las interacciones entre ellos.

Esta es la primera articulación de la arquitectura durante el proceso de diseño y por lo tanto, es necesariamente de refinamiento a alto nivel.



Attribute-Driven Design

La diferencia entre una arquitectura resultante de ADD y una lista para la implementación se basa en las decisiones de diseño más detalladas que deben tomarse.

ADD nos permite aplicar detalle a los escenarios y tácticas generales, nos permite fragmentar el trabajo y lo que creamos es la esencia del proceso de diseño.



Pasos de ADD

Los pasos para este método son:

0. Contar de entrada con los casos de calidad.
1. Elija el módulo para descomponer.
2. Refinar el módulo (se detallan los pasos).
3. Repita los pasos anteriores para cada módulo que necesite descomposición adicional.

0. Contar de entrada con los casos de calidad.

Se debe asegurar que cuenta con la información suficiente antes de iniciar el ADD. Para ello debe validar que se hayan priorizado los requerimientos acorde con los objetivos del negocio y la misión.

Es importante que usted como arquitecto priorice los requerimientos y determine en qué elementos del sistema debe centrarse durante la fase de diseño. Debe considerar los requisitos y el impacto que estos tienen en la arquitectura en orden descendente de importancia para los stakeholders.

Devuelva los requerimientos no clasificados a las partes interesadas para su priorización.

0. Contar de entrada con los casos de calidad.

Además de validar que tenga la información suficiente cada escenario de calidad debe estar expresado en forma de *estímulo-respuesta*. Cada escenario de calidad debe contar con las siguientes partes:

- Estimulo
- Fuente del estímulo
- Ambiente
- Artefacto
- Respuesta
- Medida de la respuesta

Quality Attribute Scenario 1: Quick Recovery

Element	Statement
Stimulus	A Track Manager software or hardware component fails.
Stimulus source	A fault occurs in a Track Manager software or hardware component.
Environment	Many software clients are using this service. At the time of failure, the component may be servicing a number of clients concurrently with other queued requests.
Artifact	Track Manager
Response	All query requests made by clients before and during the failure must be honored. Update service requests can be ignored for up to two seconds without noticeable loss of accuracy.
Response measure	The secondary replica must be promoted to primary and start processing update requests within two seconds of the occurrence of a fault. Any query responses that are underway (or made near the failure time) must be responded to within three seconds of additional time (on average).

1- Elija el módulo para descomponer

El módulo para comenzar suele ser todo el sistema. Todas las entradas requeridas para este módulo deben estar disponibles (restricciones, requisitos funcionales, requisitos de calidad).

Los siguientes son todos los módulos: sistema, subsistema y submódulo.

La descomposición generalmente comienza con el sistema, que luego se descompone en subsistemas, que luego se descomponen en submódulos.

2- Refinar el módulo

Para ello se deben seguir los siguientes pasos:

- A. Elija los impulsores arquitectónicos del conjunto de escenarios de calidad concretos y requisitos funcionales. Este paso determina lo que es importante para esta descomposición.
- B. Elija un patrón arquitectónico que satisfaga los impulsores arquitectónicos. Cree (o seleccione) el patrón en función de las tácticas que se pueden utilizar para lograr los controladores. Identifique los módulos secundarios necesarios para implementar las tácticas.
- C. Instale módulos y asigne funcionalidad a partir de los casos de uso y represente utilizando múltiples vistas.



2- Refinar el módulo

- D. Definir interfaces de los módulos secundarios. La descomposición proporciona módulos y restricciones sobre los tipos de interacciones de módulos. Documente esta información en el documento de interfaz para cada módulo.
- E. Verifique y refine casos de uso y escenarios de calidad y conviértelos en restricciones para los módulos secundarios. Este paso verifica que no se haya olvidado nada importante y prepara los módulos secundarios para su posterior descomposición o implementación.

	Pattern 1		Pattern 2		...	Pattern n	
	Pros	Cons	Pros	Cons		Pros	Cons
<i>Architectural driver 1</i>							
<i>Architectural driver 2</i>							
...							
<i>Architectural driver n</i>							

EJEMPLO DE ADD

0. Contar de entrada con los casos de calidad.

Escenario de Calidad #1	Respuesta del usuario a encender y apagar la luz por medio del app del fabricante
Elemento	Instrucción
Estímulo	Encender o apagar la luz por medio del app del fabricante
Fuente del estímulo	Solicitud del usuario de encender y apagar la luz por medio del app propietaria del fabricante
Ambiente	La bombilla se encuentra colocada en cualquier lugar de la casa del usuario
Respuesta	La bombilla se ilumina o se oscurece dependiendo de la solicitud del cliente
Medida de respuesta	La aplicación emite una notificación por medio de la interfaz de usuario en el que notifica del estado actual de la bombilla

Escenario de calidad #5	Cambiar de color de la bombilla por medio de una instrucción al asistente de Amazon (alexa)
Elemento	Instrucción
Estímulo	Cambiar de color de la bombilla por medio de una instrucción al asistente de Amazon (alexa)
Fuente del estímulo	El usuario le dará la instrucción al asistente de Amazon por medio de su aplicación.
Ambiente	La bombilla deberá estar conectada a la casa, deberá estar configurado el sistema de conexión con la bombilla con el asistente de amazon. El usuario deberá tener la aplicación instalada, de manera que únicamente ingrese a la aplicación y dar la instrucción.
Respuesta	Alexa responderá confirmando la instrucción, posteriormente la bombilla cambiará de color.
Medida de respuesta	La bombilla procedera a cambiar de color en base a la solicitud del usuario.

Escenario de Calidad #3	Encender o apagar la bombilla por medio de una instrucción con el asistente inteligente de Amazon (Alexa)
Elemento	Instrucción
Estímulo	Encender o apagar la bombilla por medio del asistente inteligente Alexa
Fuente del estímulo	El usuario le da a Alexa la instrucción "Alexa, enciende la luz" o en caso contrario "Alexa, apaga la luz"
Ambiente	La bombilla se encuentra en el hogar del usuario y con corriente disponible, además debe contar con el apareamiento con el entorno del asistente Alexa previamente establecido para poder llevar a cabo la instrucción
Respuesta	Alexa le responde al usuario "Muy bien"
Medida de respuesta	La bombilla se enciende o se apaga dependiendo de la solicitud específica del usuario

Escenario de calidad #4	Emparejar la bombilla con el servicio "Cloud" para poder utilizarse de forma remota fuera del hogar
Elemento	Instrucción
Estímulo	Emparejar la bombilla con el servicio "Cloud" para poder utilizarse de forma remota fuera del hogar
Fuente del estímulo	El usuario realizará la solicitud de conexión para emparejamiento de la bombilla con el servicio.
Ambiente	La bombilla deberá estar conectada a la casa, deberá estar configurado el sistema de conexión con el servicio, todo estará listo para que el usuario pueda realizar la petición del emparejamiento.
Respuesta	Se deberá mostrar un mensaje de emparejamiento exitoso, y el usuario podrá tener interacción con la bombilla de forma remota.
Medida de respuesta	La bombilla dará dos parpadeos para confirmar el emparejamiento., posterior a esto el usuario tendrá la capacidad de controlar funcionalidades de la bombilla de forma remota, tales como encender, apagar, etc.

Paso 1 - Elija el módulo a descomponer

Sistema de una lámpara inteligente. Las restricciones serán en la interoperabilidad que debe haber entre la bombilla con el asistente de amazon y el servicio cloud para utilizarse de forma remota.

- La bombilla debe conectarse en 30 segundos con la app del fabricante una vez esté conectada a la red del usuario.
- Una vez realizada la configuración y conexión del sistema con Alexa, debe encender y apagar la luz en 0.5 segundos.
- La arquitectura debe reflejar el protocolo de comunicación que debe existir para poder utilizar el bombillo de forma remota.
- La conexión con el asistente de amazon, proporcionará funcionalidades al usuario, tales como poder cambiar de color el bombillo. todas estas solicitudes deben realizarse en menos de 5 segs.



Paso 2 Refinar el módulo

Paso 2a. Elegir los controladores arquitectónicos

- La bombilla será capaz de conectarse a la red del usuario, de manera que esto suceda a través de una app del fabricante.
- Comunicación eficiente con el sistema en la nube y con el asistente de amazon.
- Podrá admitir diferentes configuraciones para conectarse con diferentes asistentes/programas (Alexa, Google Home, Apple HomeKit).
- Conexión en tiempo real, de tal modo que cuando el usuario realice una instrucción desde cualquier sistema o asistente, debe reflejarse de manera inmediata.



Nota

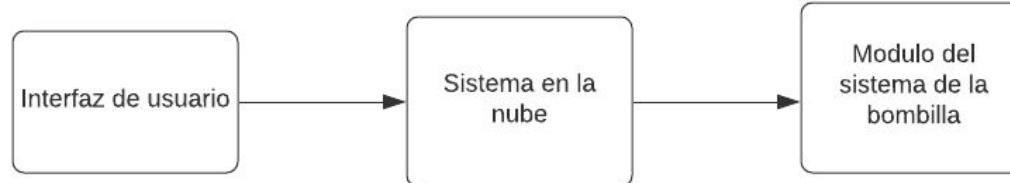
A veces es necesario realizar un prototipo o prueba de concepto de una parte del sistema.

Paso 2b. Elegir un patrón arquitectónico - Tácticas

Tener diferenciados la interfaz de usuario, la comunicación con el bombillo, y el sistema del propio bombillo.

Desarrollo correcto del sistema en la nube buscando tener un rendimiento de calidad en la conexión con el bombillo.

Desarrollo de los protocolos de comunicación a utilizar con el bombillo, además de invertir tiempo en verificar la compatibilidad con diferentes asistentes.



Paso 2b. Elegir un patrón arquitectónico - Tácticas

	Pattern 1		Pattern 2		...	Pattern n	
	Pros	Cons	Pros	Cons		Pros	Cons
Architectural driver 1							
Architectural driver 2							
...							
Architectural driver n							

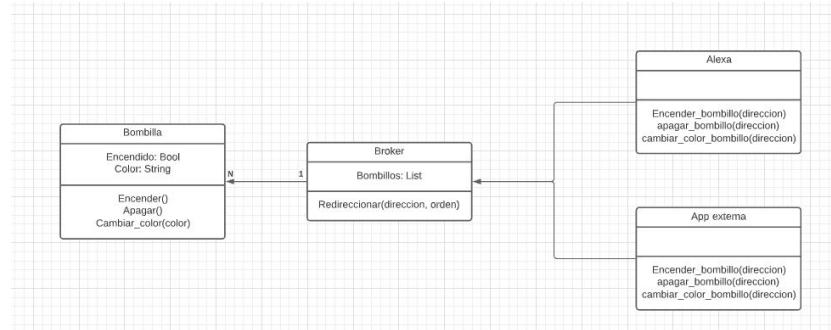


Nota

Se sugiere hacer un cuadro comparativo de los patrones que considera que puede utilizar en esta iteración y elegir el que más se ajuste.

Patrón arquitectónico: Broker.

Este patrón permite reenviar las órdenes a N cantidad de bombillas, pasando todas por un sistema central.



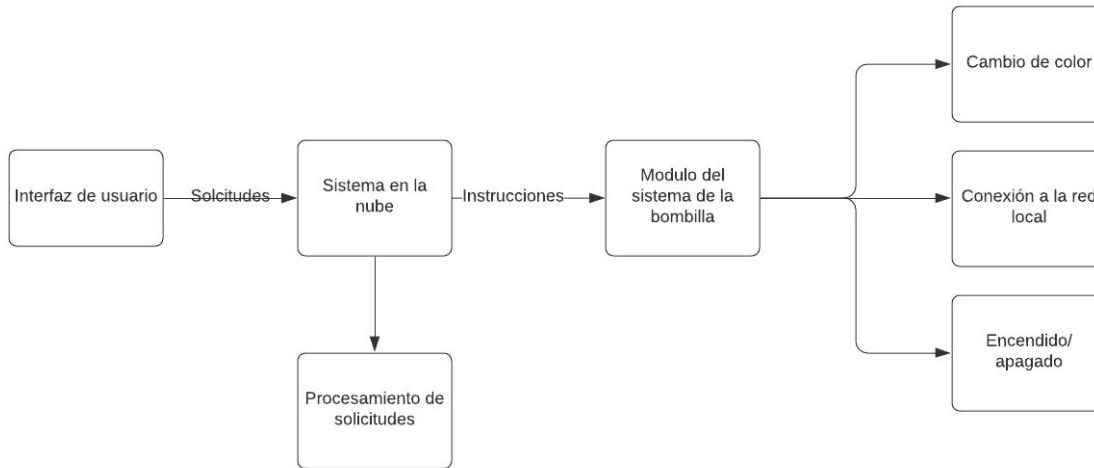
Nota

Normalmente se hace un diagrama en donde se ve plasmado el patrón arquitectónico

Paso 2c. Instanciar módulos y asignar funcionalidades usando múltiples vistas

- La interfaz de usuario incluye la conexión con los asistentes. Además de mandar las instrucciones al sistema desarrollado en la nube.
- Las responsabilidades del sistema en la nube será:
 - Mantener la comunicación con el bombillo el cual está conectado con la red de la casa.
 - Recolectar datos tanto de la app del usuario como del sistema de la bombilla.
- Módulo de la bombilla tiene las siguientes responsabilidades:
 - Ejecutar las instrucciones recibidas por parte de la aplicación o sistema. tales como encender, apagar, cambiar de color, etc.

Paso 2c. Instanciar módulos y asignar funcionalidades usando múltiples vistas



Nota

Es recomendable hacer un diagrama sencillo para que otras personas puedan entender los módulos y las funcionalidades

Paso 2d. Definir las interfaces de los módulos secundarios

El sistema en la nube debe tener los siguientes módulos secundarios:

A. Procesamiento de las peticiones

- Funcionales:
 - El bombillo procesa las peticiones de los otros módulos.
- De calidad:
 - Se tardará menos de 2 seg para tener una respuesta.
- De negocio:
 - Correcta instalación y configuración con el resto de módulos.

B. Envío de instrucciones al bombillo

- Funcionales:
 - Los módulos enviaran instrucciones al bombillo, para que las realice.
- De calidad:
 - Se tardará menos de 0,5 seg en enviarlo
- De negocio:
 - Correcta instalación y configuración con el resto de módulos.

Paso 2d. Definir las interfaces de los módulos secundarios

C. Almacenamiento de datos

- Funcionales:
 - Los datos recolectados en cualquier momento, deben ser almacenados.
- De calidad:
 - Se tendrá un backup en caso de un fallo
- De negocio:
 - Correcta instalación y configuración con el resto de módulos.

D. Monitoreo del bombillo

- Funcionales:
 - El sistema debe verificar que el bombillo siga en funcionamiento
- De calidad:
 - Si el bombillo se desconecta el sistema debe verificarlo
- De negocio:
 - Correcta instalación y configuración con el resto de módulos.

Paso 2d. Definir las interfaces de los módulos secundarios

La interfaz de usuario debe tener los siguientes módulos secundarios:

- A. Interfaz amigable con el usuario
 - Funcionales:
 - La app debe ser intuitiva con el cliente
 - De calidad:
 - La app debe seguir una paleta de colores
- B. Envío de solicitudes
 - Funcionales:
 - La aplicación debe enviar solicitudes tanto a los asistentes como al bombillo.
 - De calidad:
 - Se tardará menos de 2 seg para tener una respuesta.
 - De negocio:
 - Correcta instalación y configuración con el resto de módulos, además de compatibilidad de la aplicación con el dispositivo.

Paso 2d. Definir las interfaces de los módulos secundarios

Módulo del sistema de la bombilla los siguientes módulos secundarios:

A. Recepción de instrucciones

- Funcionales:
 - El bombillo recibirá, procesa y ejecuta las instrucciones recibidas.
- De calidad:
 - Se tardará menos de 2 seg para ejecutar la acción o tener una respuesta.
- De negocio:
 - Correcta instalación y configuración con el resto de módulos.

B. Conexión a la red

- Funcionales:
 - El sistema podrá conectarse a la red, una vez esté configurado e instalado.
- De calidad:
 - El sistema debe estar accesible siempre que esté encendida la aplicación o sistema
- De negocio:
 - Correcta instalación y configuración en la red.

Paso 2d. Definir las interfaces de los módulos secundarios

C. Conexión remota

- Funcionales:
 - El sistema podrá conectarse con una aplicación de forma remota.
- De calidad:
 - Se tardará menos de 3 seg para tener una respuesta cuando sea de manera remota.
- De negocio:
 - Correcta instalación y configuración con sistema.

D. Desarrollo de las instrucciones

- Funcionales:
 - El bombillo procesa las peticiones de los otros módulos.
- De calidad:
 - Se tardará menos de 2 seg para tener una respuesta.
- De negocio:
 - Correcta instalación y configuración con el resto de módulos.

Paso 2e. Verificar y refinar casos de uso y escenarios de calidad como restricciones para los módulos secundarios

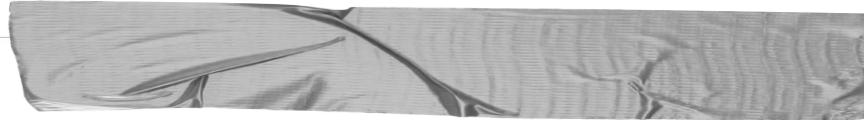
- Se debe delegar un módulo en caso de fallo de alguno de los módulos ya especificados.
- En dado caso de perder comunicación con alguno de los módulos, se deberá tener un procedimiento de recolección.
- El proceso de configuración con diferentes asistentes podría variar en su configuración pero no en su funcionalidad.



Nota

Cada módulo secundario tiene responsabilidades que derivan parcialmente de considerar la descomposición de los requisitos funcionales. Esas responsabilidades pueden traducirse en casos de uso para el módulo.

Paso 2e. Verificar y refinar casos de uso y escenarios de calidad como restricciones para los módulos secundarios

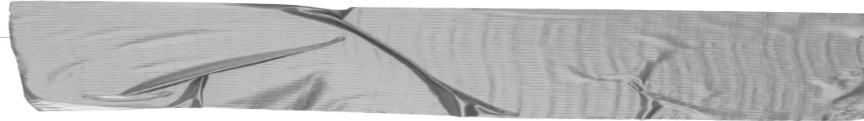


Nota

Los escenarios de calidad también tienen que ser refinados y asignados a los módulos secundarios:

- Un escenario de calidad puede quedar completamente satisfecho por la descomposición sin ningún impacto adicional. Luego se puede marcar como satisfecho.
- Un escenario de calidad puede ser satisfecho por la descomposición actual con restricciones en los módulos secundarios. Por ejemplo, el uso de capas puede satisfacer un escenario de modificabilidad específico, que a su vez limitará el patrón de uso de los hijos.
- La descomposición puede ser neutral con respecto a un escenario de calidad. Por ejemplo, un escenario de usabilidad se refiere a partes de la interfaz de usuario que aún no forman parte de la descomposición. Este escenario debe asignarse a uno de los módulos secundarios.
- Un escenario de calidad puede no ser satisfactorio con la descomposición actual. Si es importante, se debe reconsiderar la descomposición. De otra manera, se debe registrar la justificación de la descomposición que no admite este escenario. Esto suele ser el resultado de una compensación con otros escenarios, quizás de mayor prioridad.

Paso 2e. Verificar y refinar casos de uso y escenarios de calidad como restricciones para los módulos secundarios



Nota

Las restricciones del módulo principal se pueden satisfacer de una de las siguientes maneras:

La descomposición satisface la restricción. Por ejemplo, la restricción de usar un determinado sistema operativo puede satisfacerse definiendo el sistema operativo como un módulo secundario. La restricción ha sido satisfecha y no se necesita hacer nada más. La restricción es satisfecha por un solo módulo hijo: La restricción ha sido designada como un hijo. Si está satisfecho o no depende de lo que ocurra con la descomposición del hijo.

La restricción es satisfecha por múltiples módulos secundarios. Por ejemplo, el uso de la Web requiere dos módulos (cliente y servidor) para implementar los protocolos necesarios. La satisfacción de la restricción depende de la descomposición y coordinación de los hijos a los que se les ha asignado la restricción.

Ejemplo: Paso 2- Refinar el módulo

Al final de este paso tenemos una descomposición de un módulo en sus elementos secundarios, donde cada módulo secundario tiene una colección de responsabilidades; un conjunto de casos de uso, una interfaz, escenarios de calidad y una colección de restricciones. Esto es suficiente para comenzar la próxima iteración de descomposición.

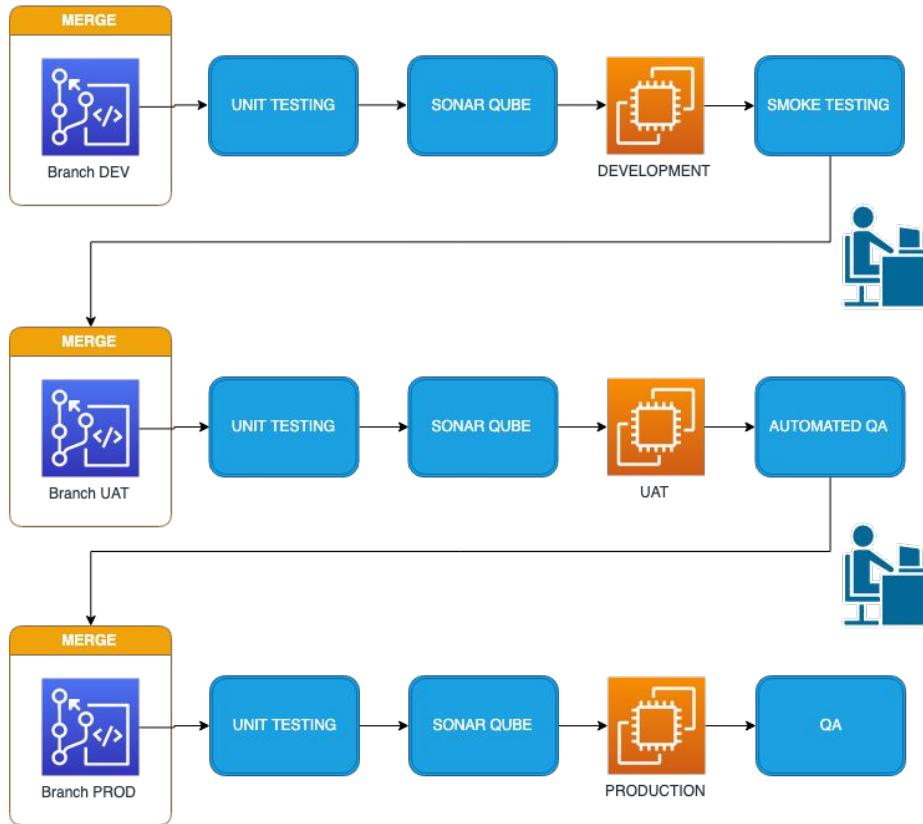
Observe en el ejemplo cuánto progreso (o poco) se logra en una sola iteración: tenemos un vocabulario de módulos y sus responsabilidades; Hemos considerado una variedad de casos de uso y escenarios de calidad y entendemos algunas de sus ramificaciones.

Hemos decidido las necesidades de información de los módulos y sus interacciones. Esta información debe ser capturada en la lógica del diseño, como discutimos, documentación de arquitecturas de software.



Paso 3 - Repita los pasos anteriores para cada módulo que necesite descomposición adicional.

Flujo Normal de Desarrollo



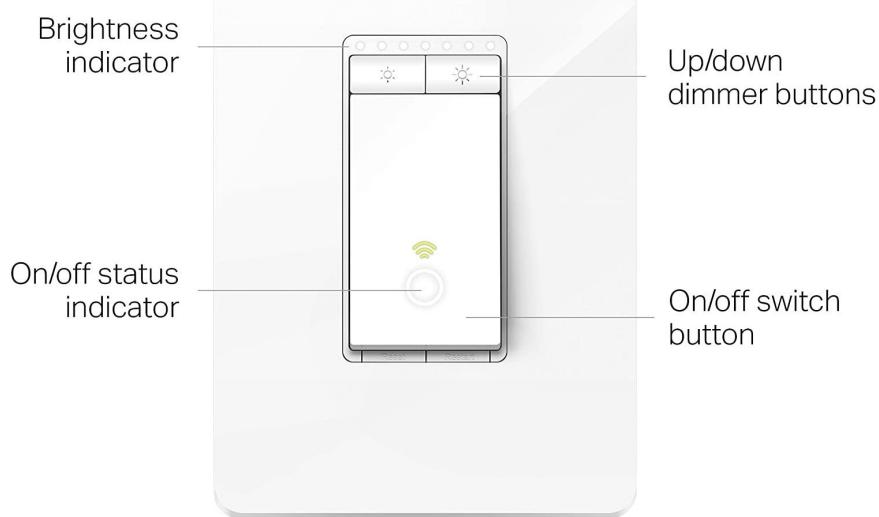
Ejercicio

Realice tres Escenario de
Calidad del proyecto Final del
Curso

Escenario de Calidad #1:

Elemento	Instrucción
Estímulo	
Fuente del estímulo	
Ambiente	
Respuesta	
Medida de respuesta	

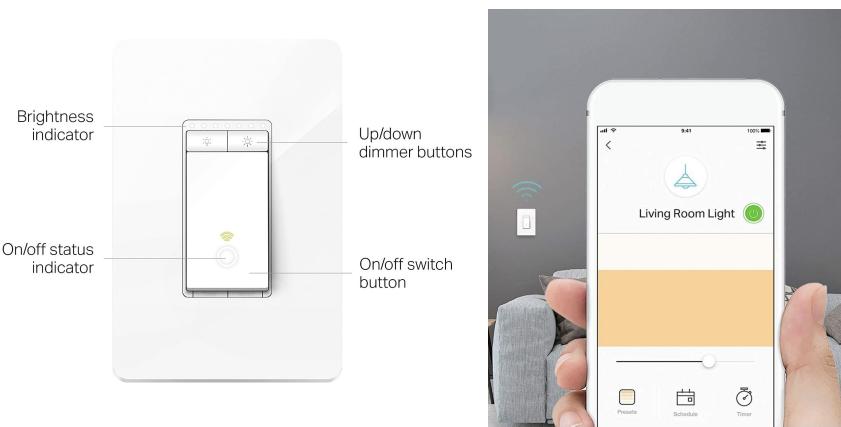
Ejercicio



Realice el Análisis ADD de un switch Inteligente con Dimmer.

Realice los pasos de ADD

Satisfaga todos los escenarios de calidad



Modelo y Diseño de Arquitectura

Validación



¿Qué es
validación en
software y/o
arquitectura?

Validación en software y/o arquitectura:

Es la que nos permite asegurar la calidad del software o la arquitectura desarrollada en las distintas etapas y certificar el buen funcionamiento de las distintas funcionalidades esperadas.

Esto se puede asegurar por medio de:

- En software: aplicación de los diferentes test, test escenarios.
- Arquitectura: por medio de la verificación de los escenarios de calidad, criterios de aceptación, restricciones y/o limitaciones.

En software ...

Tipo	¿Qué es?	¿Cuando?	¿Comó?
Unit Testing	La unidad del código fuente se prueba para determinar si es apta para el uso. Esta prueba es normalmente escrita y ejecutada por el desarrollador de software	Al principio del ciclo de desarrollo. Se recomienda crear pruebas unitarias antes de escribir el código es el mejor, si es posible.	Se ejecuta en el entorno local en la máquina de desarrollo. Comúnmente automatizado en el pipeline, pero puede realizarse manualmente (no tan recomendable).
Smoke Testing	Revisión rápida de las características del producto en un tiempo limitado. Si las funciones clave no funcionan o si los errores clave aún no se han solucionado, QC Analyst no comenzará a realizar los demás test.	Se realiza después de que la prueba de la unidad se ha completado y realizado con éxito. Se prepara el ambiente para realizarlo.	Pruebas manuales en instancias de desarrollo. Se ejecuta un subconjunto de casos de prueba para cubrir la funcionalidad más importante de un componente para determinar si las funciones cruciales de la misma funcionan correctamente.
Functional Testing	Pruebe la estrategia para verificar que una característica funcione como se espera, siguiendo un requisito proporcionado. Las pruebas funcionales usualmente describen lo que hace el sistema, se realiza una prueba del componente aislado.	Se realiza después de que la prueba de humo se ha completado y realizado con éxito. Se prepara el ambiente para realizarlo.	Pruebas manuales en instancia de desarrollo: crear y publicar casos de prueba de diseño de QC Analyst mientras se completa el desarrollo Ejecutar casos de prueba diseñados manualmente.

Tipo	¿Qué es?	¿Cuando?	¿Comó?
Regression Testing	<p>Pruebas de software para buscar nuevos errores de software descubiertos en áreas funcionales y no funcionales existentes después de que se hayan realizado cambios como mejoras, corrección de errores o nuevos desarrollos.</p> <p>Pruebe todas las funciones completas que tienen estabilidad y se aprobaron en Demo / UAT.</p>	<p>Se realizan después de que las pruebas funcionales han finalizado. La corrección de errores se ha completado, probado y aprobado.</p> <p>Fin del sprint, un sprint detrás del resto del equipo.</p>	<p>Automatizado</p> <p>Debe revisarse cada sprint para actualizarse de acuerdo con el desarrollo del software.</p>
Integration testing	<p>El nivel de prueba se usa para probar las interacciones entre los componentes del software y se realiza después de la prueba de los componentes. Todos los componentes desarrollados se integraron a una plantilla y con los componentes que estarán interactuando.</p>	<p>Se realiza después que se han completado las pruebas funcionales para cada componente. También después de realizar pruebas de regresión y no han surgido nuevos errores críticos</p>	<p>Se crean estructuras automatizadas.</p> <p>Se unen los componentes que conviven juntos para validar que funcionen correctamente en conjunto.</p>
Security Test	<p>Es validar que la información de los usuarios y/o software estén protegidos y que los usuarios tengan acceso únicamente a la información autorizada para ellos.</p>	<p>En todo el ciclo del software</p>	<p>Validando el acceso a la aplicación, protección de la información, SQL Injection, cross site scripting, Manejo de sesiones, acceso a third party services, etc. Existen varias herramientas.</p>

Arquitectura ...

¿Se acuerdan de los escenarios de calidad?

Quality Attribute Scenario 1: Quick Recovery

Element	Statement
Stimulus	A Track Manager software or hardware component fails.
Stimulus source	A fault occurs in a Track Manager software or hardware component.
Environment	Many software clients are using this service. At the time of failure, the component may be servicing a number of clients concurrently with other queued requests.
Artifact	Track Manager
Response	All query requests made by clients before and during the failure must be honored. Update service requests can be ignored for up to two seconds without noticeable loss of accuracy.
Response measure	The secondary replica must be promoted to primary and start processing update requests within two seconds of the occurrence of a fault. Any query responses that are underway (or made near the failure time) must be responded to within three seconds of additional time (on average).



Por medio de ADD - ¿Recuerdan este paso?

2e. Verificar y refinar casos de uso y escenarios de calidad como restricciones para los módulos secundarios:

- Requerimientos funcionales: Cada módulo secundario tiene responsabilidades que derivan parcialmente de considerar la descomposición de los requisitos funcionales. Esas responsabilidades pueden traducirse en casos de uso para el módulo.



Por medio de ADD - ¿Recuerdan este paso?

Los escenarios de calidad también tienen que ser refinados y asignados a los módulos secundarios:

- Un escenario de calidad puede quedar completamente satisfecho por la descomposición sin ningún impacto adicional. Luego se puede marcar como satisfecho.
- Un escenario de calidad puede ser satisfecho por la descomposición actual con restricciones en los módulos secundarios.
- La descomposición puede ser neutral con respecto a un escenario de calidad. Si aún no forman parte de la descomposición asignarse a uno de los módulos secundarios.
- Un escenario de calidad puede no ser satisfactorio con la descomposición actual. Si es importante, se debe reconsiderar la descomposición. De otra manera, se debe registrar la justificación de la descomposición que no admite este escenario. Esto suele ser el resultado de una compensación con otros escenarios, quizás de mayor prioridad.

Por medio de ADD - ¿Recuerdan este paso?

Las restricciones del módulo principal se pueden satisfacer de una de las siguientes maneras:

- La descomposición satisface la restricción.
- La restricción es satisfecha por un solo módulo hijo.
- La restricción es satisfecha por múltiples módulos secundarios.

Conocer qué módulo o submódulos satisfacen la restricción nos permite validarla específicamente después de su desarrollo.



Por medio de ADD - ¿Recuerdan este paso?

Este paso nos permite determinar los escenarios de calidad que aplican para cada módulos y/o submódulos.

También debemos verificar las responsabilidades, restricciones de cada módulo que surja de la descomposición utilizando casos de uso.

Todos estos escenarios de calidad, responsabilidades, restricciones debemos documentarlas



Modelo y Diseño de Arquitectura

Prototipo

¿Qué es un prototipo?

Es una representación concreta de parte o de todo un sistema. Este es un artefacto tangible y no una descripción abstracta que requiere interpretación. Estos permiten a los usuarios, clientes y desarrolladores visualizar y reflexionar sobre el sistema final.

Este enfoque es intuitivo, más orientado al descubrimiento y generación de nuevas ideas que a la evaluación de ideas existentes.

¿Qué es un prototipo?

En un prototipo podemos limitar la cantidad de información que se puede manejar, interfaz, etc.

La creación de prototipos es principalmente una actividad de diseño, aunque utilizamos ingeniería de software para asegurar que los prototipos de software evolucionan en sistemas técnicamente sólidos y usamos métodos científicos para estudiar la efectividad de diseños particulares.

¿Qué es un prototipo?

Los ingenieros y arquitectos de software a menudo crean prototipos para estudiar la viabilidad de un proceso técnico.

Los prototipos de software exitosos evolucionan hacia el producto final y luego a nuevas versiones del producto.

Podemos ver los prototipos como artefactos concretos en sí mismos o como componentes importantes del proceso de diseño.

Prototipos como Artefactos de diseño

Vistos como artefactos, los prototipos exitosos tienen varias características:

- Apoyan la creatividad, ayudando al desarrollador a capturar y generar ideas.
- Facilita la exploración de un diseño espacio, y a descubrir información relevante sobre los usuarios y sus prácticas.
- Fomentan la comunicación, ayudando a ingenieros, gerentes, clientes, desarrolladores de software y usuarios a discutir opciones e interactuar entre ellos.

Prototipos como Artefactos de diseño

- Permiten una evaluación temprana, porque pueden ser probados, incluyendo estudios de usabilidad tradicionales e informales.
- Comentarios de diferentes tipos de usuarios, durante todo el proceso de diseño.

Prototipos como Artefactos de diseño

Podemos analizar prototipos y técnicas de creación de prototipos a lo largo de cuatro dimensiones:

1. La representación describe la forma del prototipo (p. Ej., Conjuntos de bocetos en papel o simulaciones por computadora).
2. La precisión describe el nivel de detalle en el que el prototipo debe ser evaluado (por ejemplo, informal y relajado o altamente refinado).

Prototipos como Artefactos de diseño

- 
3. La interactividad describe la medida en que el usuario puede acceder e interactuar con el prototipo (por ejemplo, solo visual o completamente interactivo).
 4. La evolución describe el ciclo de vida esperado del prototipo. (por ejemplo, desecharle ó iterativo).

Prototipos como Artefactos de diseño

Ventajas	Desventajas
<ul style="list-style-type: none">• Tiempos y costos reducidos: permiten estimar mejor el próximo esfuerzo.• Mejora y mayor participación del usuario: aclaran las ideas falsas y expectativas y permiten recopilar feedback de las primeras fases de desarrollo.• Seguro de calidad: permite darnos cuenta de los descuidos, requisitos adicionales y limitaciones.• Innovación: permite explorar las ideas nuevas y experimentar nuevas opciones.	<ul style="list-style-type: none">• Análisis Insuficiente: La confianza del equipo en un prototipo puede hacer que el equipo no evalúe todos los detalles y características. se debe controlar mediante procesos concretos de requerimientos.• Confusión del usuario entre el prototipo y el sistema terminado: cuando el sistema final difiere mucho del prototipo.• Gastos de prototipos: siempre habrá cobros en la implementación de una fase de creación de prototipos.

Pasos

1. Comprender el Proyecto:

Debe comprender la tecnología existente y el panorama empresarial, así como la visión y estrategia futuras. Esta es una base crucial para desarrollar cualquier solución.



3. Investigación:

Determine si ya existe algo que cumpla con los requisitos. Descubra los detalles no mencionados en los requisitos básicos.



2. Identificar los requisitos básicos:

Se derivan de las ideas en cuestión, o de los problemas que intenta resolver. Los requisitos generalmente evolucionan a medida que aprende más.



4. Desarrollar un prototipo inicial:

Acerca de las características que son importantes y tienen un alto impacto. Esto depende del objetivo del prototipo. Si existen características complicadas con posibilidades desconocidas, aborde estas primero.



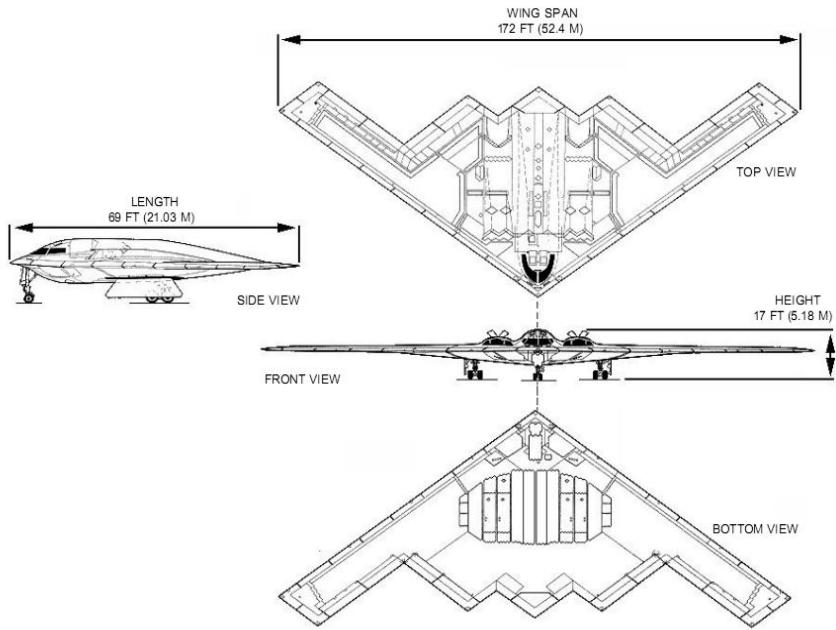
Pasos

5.evaluar y revisar el prototipo: Se debe revisar el prototipo desarrollado con el grupo de usuarios objetivo y las partes interesadas relevantes. El rendimiento de las características y la usabilidad deben evaluarse y medirse tanto cuantitativa como cualitativamente.



7. Repita: Si aún existen incógnitas, repita el proceso anterior. De hecho, incluso con soluciones en producción, este proceso puede aprovecharse para probar nuevas ideas para la solución.

6. Revisar y mejorar el prototipo:
Después de revisar los comentarios del prototipo, reúne información. Entonces puedes hacer mejoras y cambios



Dimensiones del Prototipo

Prototipos Horizontales

El objetivo es proporcionar una visión amplia de toda la solución. Habrá poca complejidad en las características individuales.

Este enfoque es bueno para los sitios web y las instancias en las que necesita lograr una idea general del producto.

Por lo general, se trata de soluciones dirigidas al público o soluciones que requieren pruebas de usabilidad intensivas.

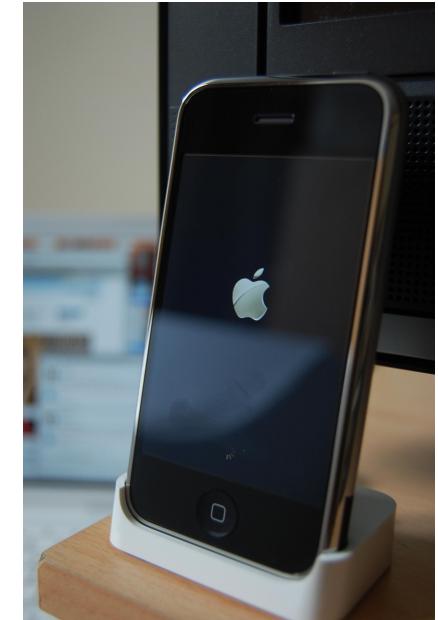


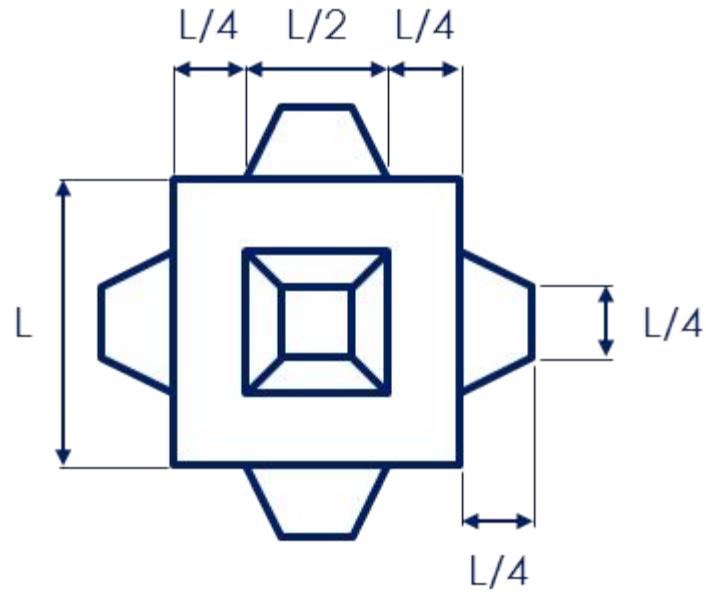
Prototipos Verticales

El prototipo se centrará en un pequeño conjunto de características, a veces incluso en una o dos. Las características elegidas se exploran e investigan por completo.

Este enfoque es bueno para soluciones en las que utiliza algoritmos oscuros o complejos, o cuando intenta algo inusual o poco ortodoxo.

Esto es útil para experimentar con nuevas tecnologías, nuevos enfoques y, por lo general, cuando desea "disruption."



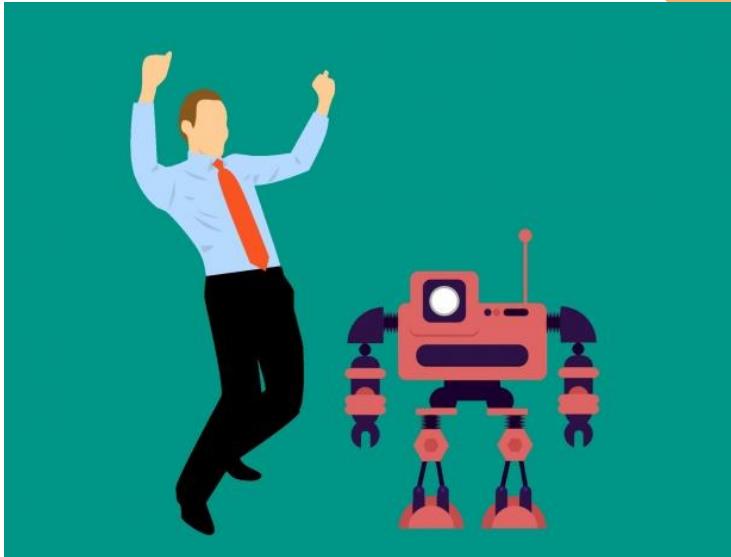


Tipos de Prototipos

Prototipos Desechables

Esto también se conoce como prototipos cerrados. La creación rápida de prototipos implica la creación de un modelo de trabajo de partes del sistema, en una etapa temprana de desarrollo, después de una investigación relativamente corta.

Este tipo de creación de prototipos muestra a las personas cómo se verá la característica. Pero el código base o proyecto no se usa necesariamente para la versión de producción de la aplicación.



Prototipos evolutivos

El objetivo principal aquí es construir un prototipo muy robusto de manera estructurada y refinarlo constantemente.

El prototipo forma el corazón de la aplicación de producción, y se le agregan características adicionales.



Prototipos incrementales

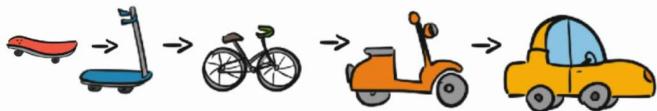
En la creación de prototipos incrementales, partes del sistema se desarrollan como prototipos separados y se conectan entre sí para formar una aplicación completa.

Es importante desarrollar las interfaces para los componentes separados temprano, ya que la integración puede resultar una pesadilla.



MÓDELO ITERATIVO

MÓDELO ITERATIVO E INCREMENTAL



Prototipos Extremos

Extreme Prototyping se emplea principalmente para aplicaciones web y generalmente en tres fases:

1. Se crea HTML / CSS / JS estático: esto brinda a los usuarios una sensación tangible instantánea del producto.
2. A partir de entonces, la capa de servicio se simula, esto incluye reglas y lógica de negocios.
3. Se desarrolla la capa de servicio real; esto implica integrarse con sistemas del mundo real y conectarlo a las vistas frontales HTML/CSS/JS.

Esto brinda a los usuarios una vista temprana de la aplicación sin tener una funcionalidad real detrás de ella.

El backend se unirá gradualmente a medida que avanza el proceso.



→ Easy Retro Tool



Escenarios de calidad

Entre los Escenarios de Calidad tenemos:

Escenario de Calidad #1	Respuesta del usuario a encender y apagar la luz por medio del app del fabricante
Elemento	Instrucción
Estímulo	Encender o apagar la luz por medio del app del fabricante
Fuente del estímulo	Solicitud del usuario de encender y apagar la luz por medio del app propietaria del fabricante
Ambiente	La bombilla se encuentra colocada en cualquier lugar de la casa del usuario
Respuesta	La bombilla se ilumina o se oscurece dependiendo de la solicitud del cliente
Medida de respuesta	La aplicación emite una notificación por medio de la interfaz de usuario en el que notifica del estado actual de la bombilla

Escenario de Calidad #2	Conectividad de la bombilla con la red local del usuario
Elemento	Instrucción
Estímulo	Conexión de la bombilla a la red local del usuario
Fuente del estímulo	Solicitud del usuario de conectar la bombilla a la red local por medio del app propietaria del fabricante
Ambiente	La bombilla se coloca conectada y con corriente en cualquier lugar de la casa, adicionalmente debe estar cerca del router para poder realizar exitosamente la conexión
Respuesta	La bombilla se conecta exitosamente a la red local del usuario
Medida de respuesta	La bombilla emite un parpadeo repetitivo un total de 5 veces para delimitar qué se ha conectado exitosamente a la red

Escenario de Calidad #3	Encender o apagar la bombilla por medio de una instrucción con el asistente inteligente de Amazon (Alexa)
Elemento	Instrucción
Estímulo	Encender o apagar la bombilla por medio del asistente inteligente Alexa
Fuente del estímulo	El usuario le da a Alexa la instrucción “Alexa, enciende la luz” o en caso contrario “Alexa, apaga la luz”
Ambiente	La bombilla se encuentra en el hogar del usuario y con corriente disponible, además debe contar con el apareamiento con el entorno del asistente Alexa previamente establecido para poder llevar a cabo la instrucción
Respuesta	Alexa le responde al usuario “Muy bien”
Medida de respuesta	La bombilla se enciende o se apaga dependiendo de la solicitud específica del usuario

Escenario de calidad #4	Emparejar la bombilla con el servicio “Cloud” para poder utilizarse de forma remota fuera del hogar
Elemento	Instrucción
Estímulo	Emparejar la bombilla con el servicio “Cloud” para poder utilizarse de forma remota fuera del hogar
Fuente del estímulo	El usuario realizará la solicitud de conexión para emparejamiento de la bombilla con el servicio.
Ambiente	La bombilla deberá estar conectada a la casa, deberá estar configurado el sistema de conexión con el servicio, todo estará listo para que el usuario pueda realizar la petición del emparejamiento.
Respuesta	Se deberá mostrar un mensaje de emparejamiento exitoso, y el usuario podrá tener interacción con la bombilla de forma remota.
Medida de respuesta	La bombilla dará dos parpadeos para confirmar el emparejamiento., posterior a esto el usuario tendrá la capacidad de controlar funcionalidades de la bombilla de forma remota, tales como encender, apagar, etc.

Escenario de calidad #5	Cambiar de color de la bombilla por medio de una instrucción al asistente de Amazon (alexa)
Elemento	Instrucción
Estímulo	Cambiar de color de la bombilla por medio de una instrucción al asistente de Amazon (alexa)
Fuente del estímulo	El usuario le dará la instrucción al asistente de Amazon por medio de su aplicación.
Ambiente	La bombilla deberá estar conectada a la casa, deberá estar configurado el sistema de conexión con la bombilla con el asistente de Amazon. El usuario deberá tener la aplicación instalada, de manera que únicamente ingrese a la aplicación y dar la instrucción.
Respuesta	Alexa responderá confirmando la instrucción, posteriormente la bombilla cambiará de color.
Medida de respuesta	La bombilla procederá a cambiar de color en base a la solicitud del usuario.

ADD luz inteligente

Paso 1:

Sistema de una lámpara inteligente. Las restricciones serán en la interoperabilidad que debe haber entre la bombilla con el asistente de Amazon y el servicio cloud para utilizarse de forma remota.

- La bombilla debe conectarse en 30 segundos con la app del fabricante una vez esté conectada a la red del usuario.
- Una vez realizada la configuración y conexión del sistema con Alexa, debe encender y apagar la luz en 0.5 segundos.
- La arquitectura debe reflejar el protocolo de comunicación que debe existir para poder utilizar el bombillo de forma remota.
- La conexión con el asistente de Amazon proporcionará funcionalidades al usuario, tales como poder cambiar de color el bombillo. Todas estas solicitudes deben realizarse en menos de 5 segundos.

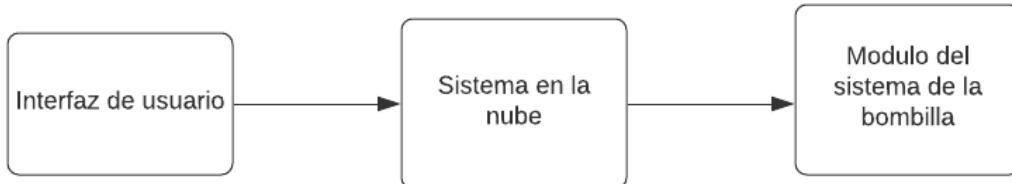
Paso 2:

2a. Elegir los controles arquitectónicos:

- La bombilla será capaz de conectarse a la red del usuario, de manera que esto suceda a través de una app del fabricante.
- Comunicación eficiente con el sistema en la nube y con el asistente de Amazon.
- Podrá admitir diferentes configuraciones para conectarse con diferentes asistentes/programas (Alexa, Google Home, Apple HomeKit).
- Conexión en tiempo real, de tal modo que cuando el usuario realice una instrucción desde cualquier sistema o asistente, debe reflejarse de manera inmediata.

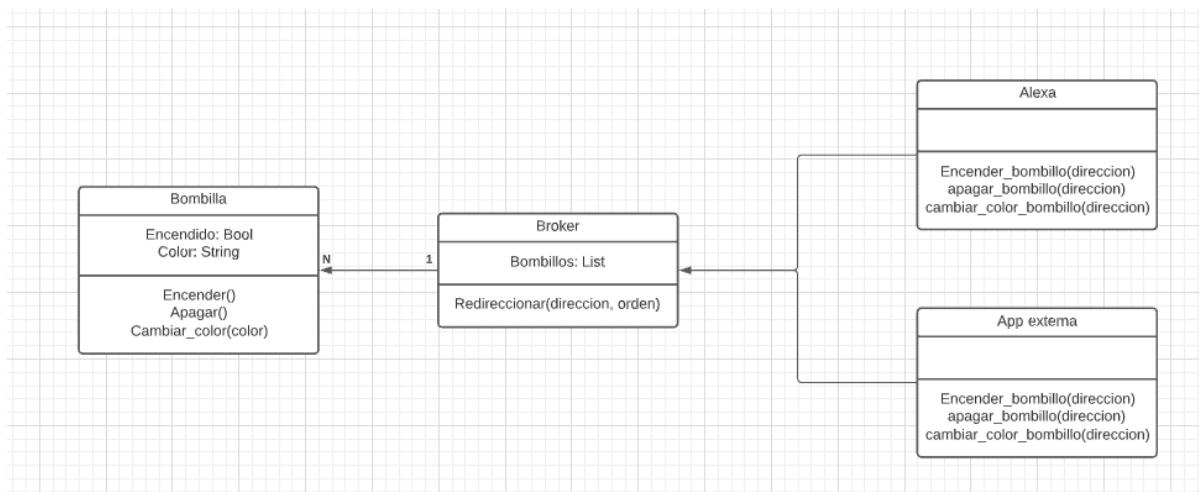
2b. Elegir un patrón arquitectónico - Tácticas:

- Tener diferenciados la interfaz de usuario, la comunicación con el bombillo, y el sistema del propio bombillo.
- Desarrollo correcto del sistema en la nube buscando tener un rendimiento de calidad en la conexión con el bombillo.
- Desarrollo de los protocolos de comunicación a utilizar con el bombillo, además de invertir tiempo en verificar la compatibilidad con diferentes asistentes.



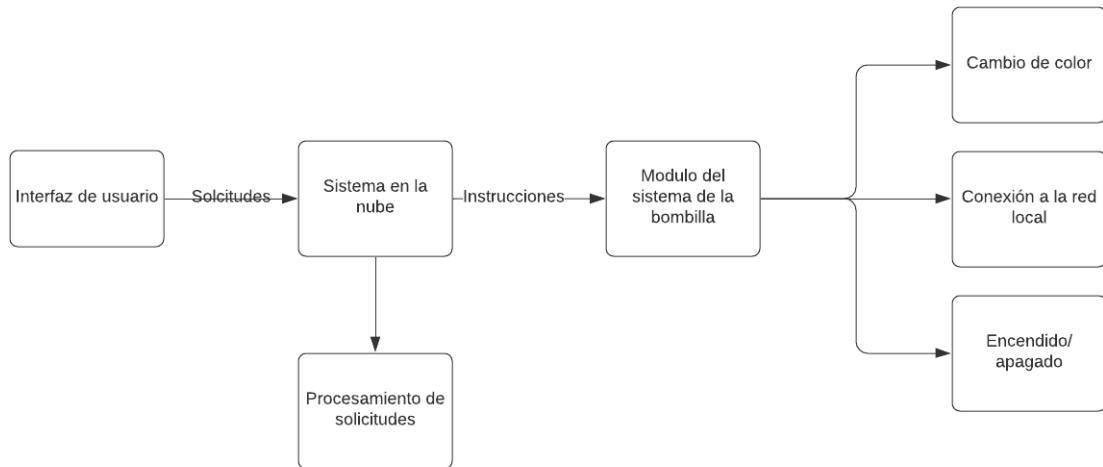
Patrón arquitectónico: Broker.

Este patrón permite reenviar las órdenes a N cantidad de bombillas, pasando todas por un sistema central.



2c Instanciar módulos y asignar funcionalidades usando vistas múltiples:

- La interfaz de usuario incluye la conexión con los asistentes. Además de mandar las instrucciones al sistema desarrollado en la nube.
- Las responsabilidades del sistema en la nube será:
 - Mantener la comunicación con el bombillo el cual está conectado con la red de la casa.
 - Recolectar datos tanto de la app del usuario como del sistema de la bombilla.
- Módulo de la bombilla tiene las siguientes responsabilidades:
 - ejecutar las instrucciones recibidas por parte de la aplicación o sistema. tales como encender, apagar, cambiar de color, etc.



2d. Definir las interfaces de módulos secundarios

El sistema en la nube debe tener los siguientes módulos secundarios:

- Procesamiento de las peticiones
 - Funcionales:
 - El bombillo procesa las peticiones de los otros módulos.
 - De calidad:
 - Se tardará menos de 2 seg para tener una respuesta.
 - De negocio:
 - Correcta instalación y configuración con el resto de módulos.
- Envío de instrucciones al bombillo
 - Funcionales:
 - Los módulos enviarán instrucciones al bombillo, para que las realice.
 - De calidad:
 - Se tardará menos de 0,5 seg en enviarlo
 - De negocio:
 - Correcta instalación y configuración con el resto de módulos.
- Almacenamiento de datos
 - Funcionales:
 - Los datos recolectados en cualquier momento, deben ser almacenados.
 - De calidad:
 - Se tendrá un backup en caso de un fallo
 - De negocio:
 - Correcta instalación y configuración con el resto de módulos.
- Monitoreo del bombillo
 - Funcionales:
 - El sistema debe verificar que el bombillo siga en funcionamiento
 - De calidad:
 - Si el bombillo se desconecta el sistema debe verificarlo
 - De negocio:
 - Correcta instalación y configuración con el resto de módulos.

La interfaz de usuario debe tener los siguientes módulos secundarios:

- Interfaz amigable con el usuario
 - Funcionales:

- La app debe ser intuitiva con el cliente
- De calidad:
 - La app debe seguir una paleta de colores
- Envío de solicitudes
 - Funcionales:
 - La aplicación debe enviar solicitudes tanto a los asistentes como al bombillo.
 - De calidad:
 - Se tardará menos de 2 seg para tener una respuesta.
 - De negocio:
 - Correcta instalación y configuración con el resto de módulos, además de compatibilidad de la aplicación con el dispositivo.

Módulo del sistema de la bombilla los siguientes módulos secundarios:

- Recepción de instrucciones
 - Funcionales:
 - El bombillo recibirá, procesa y ejecuta las instrucciones recibidas.
 - De calidad:
 - Se tardará menos de 2 seg para ejecutar la acción o tener una respuesta.
 - De negocio:
 - Correcta instalación y configuración con el resto de módulos.
- Conexión a la red
 - Funcionales:
 - El sistema podrá conectarse a la red, una vez esté configurado e instalado.
 - De calidad:
 - El sistema debe estar accesible siempre que esté encendida la aplicación o sistema
 - De negocio:
 - Correcta instalación y configuración en la red.
- Conexión remota
 - Funcionales:
 - El sistema podrá conectarse con una aplicación de forma remota.
 - De calidad:
 - Se tardará menos de 3 seg para tener una respuesta cuando sea de manera remota.
 - De negocio:
 - Correcta instalación y configuración con sistema.
- Desarrollo de las instrucciones
 - Funcionales:
 - El bombillo procesa las peticiones de los otros módulos.
 - De calidad:
 - Se tardará menos de 2 seg para tener una respuesta.
 - De negocio:
 - Correcta instalación y configuración con el resto de módulos.

2e. Verificar y refinar casos de uso y escenarios de calidad como restricciones para los módulos secundarios:

- Se debe delegar un módulo en caso de fallo de alguno de los módulos ya especificados.
- En dado caso de perder comunicación con alguno de los módulos, se deberá tener un procedimiento de recolección.

- El proceso de configuración con diferentes asistentes podría variar en su configuración pero no en su funcionalidad.



Arquitectura de sistemas, comportamiento y optimización

Week 9

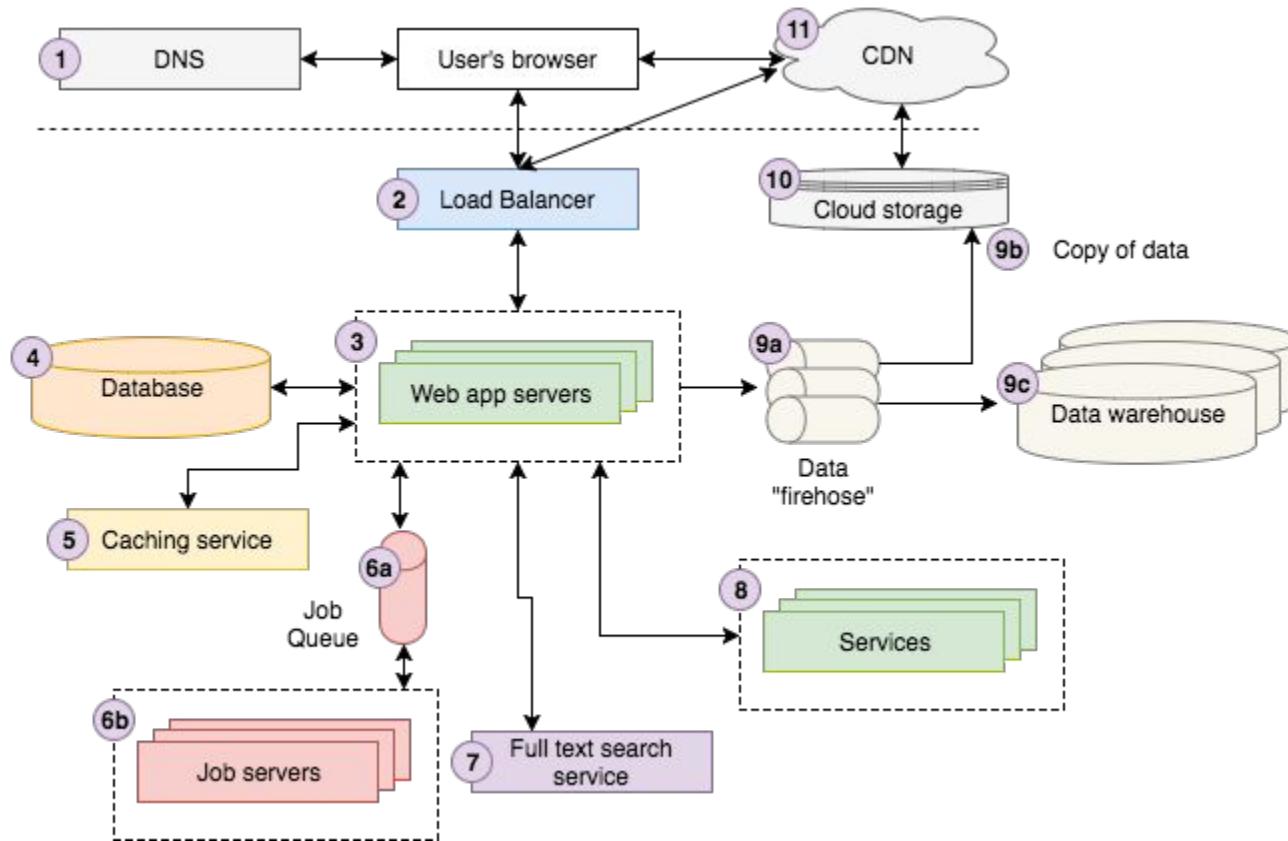
M.Sc. Juan Luis Flores Pineda

Agenda

- Arquitectura Web
 - ◆ Introducción
 - ◆ Conceptos básicos
 - ◆ Servicios REST
 - ◆ Application Programming Interface (API)
- Modelo y Diseño de Arquitectura - Móviles
 - ◆ Aplicaciones Híbridas
 - ◆ Cross Compiled / Cross Platform
- Modelo y Diseño de Arquitectura - Web
 - ◆ RIA
 - ◆ SPA (Single Page Applications)

Arquitectura Web

Introducción



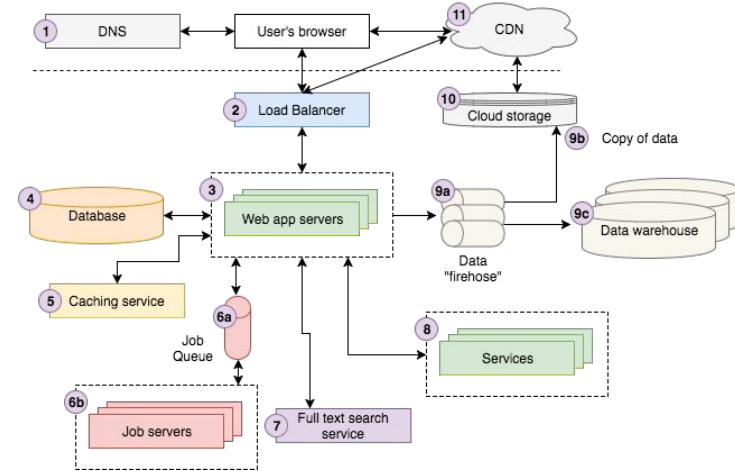


Conceptos Básicos

1 - DNS - Conceptos Básicos

Significa "Domain Name System" y es una tecnología troncal que hace posible la red mundial. En el nivel más básico, el DNS proporciona una búsqueda de clave / valor de un nombre de dominio a una dirección IP.

Por ejemplo, google.com a una dirección IP
74.125.224.72



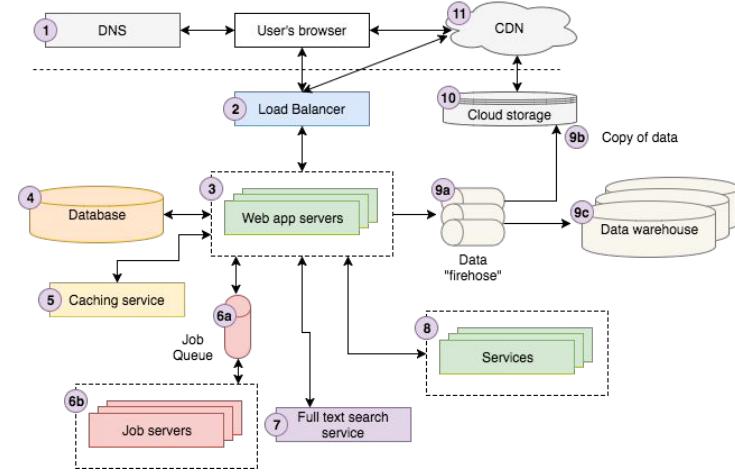
2 - Load Balancer - Conceptos Básicos

Existen dos tipos de escalado de recursos :

- Horizontal: se agregan más máquinas a su grupo de recursos.
- Vertical: Se agrega más potencia a una máquina existente (CPU, RAM, Disco, etc).

Normalmente en una arquitectura web se tiende a escalar de forma horizontal debido a:

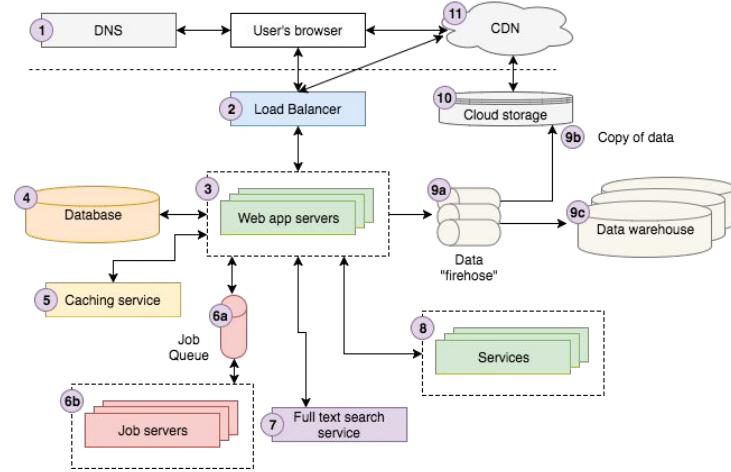
- El Escalado Vertical tiene un límite.
- Degradación de la red.
- Permite planificar interrupciones y que la aplicación continúe en la nube.
- Vuelve a la aplicación tolerante a fallas.



2 - Load Balancer - Conceptos Básicos.

Load Balancer o balanceador de carga permite equilibrar la carga entre los diferentes servidores (escalado Horizontal), de manera de tener un buen rendimiento.

Permiten escalar de forma horizontal, debido a que se encargan de enrutar las solicitudes entrantes a uno de los servidores de la aplicación y busca distribuir las solicitudes de manera eficiente para que ninguno de los servidores esté sobrecargado.

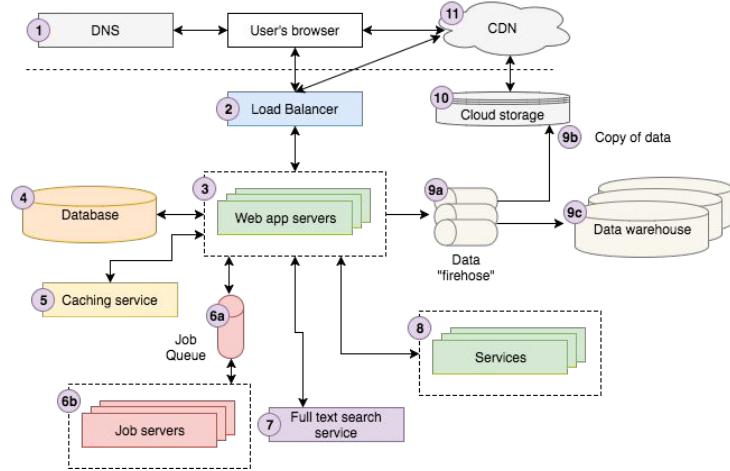


3- Servidores de aplicaciones Web -Conceptos Básicos

Ejecutan la lógica comercial central que maneja la solicitud de un usuario y envía de vuelta HTML al navegador del usuario.

Estas generalmente se comunican con una variedad de infraestructura como: bases de datos, capas de almacenamiento en caché, colas de trabajos, servicios de búsqueda, microservicios, data-login, etc.

Las implementaciones del servidor de aplicaciones requieren elegir un lenguaje de programación específico normalmente.

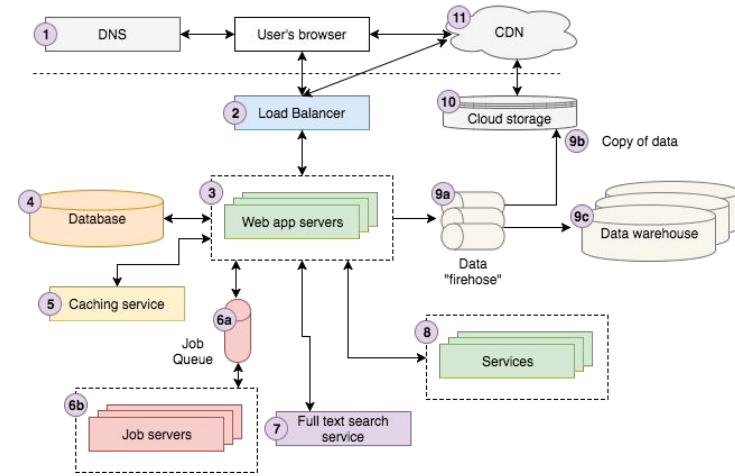


4 - Servidores de DB - Conceptos Básicos

Las bases de datos proporcionan formas de definir sus estructuras de datos, insertar datos nuevos, encontrar datos existentes, actualizar o eliminar datos existentes, realizar cálculos a través de los datos, etc.

En la mayoría de los casos, los servidores de aplicaciones web se comunican directamente con un servidor de base de datos.

También, cada servicio puede tener su propia base de datos aislada del resto de la aplicación.

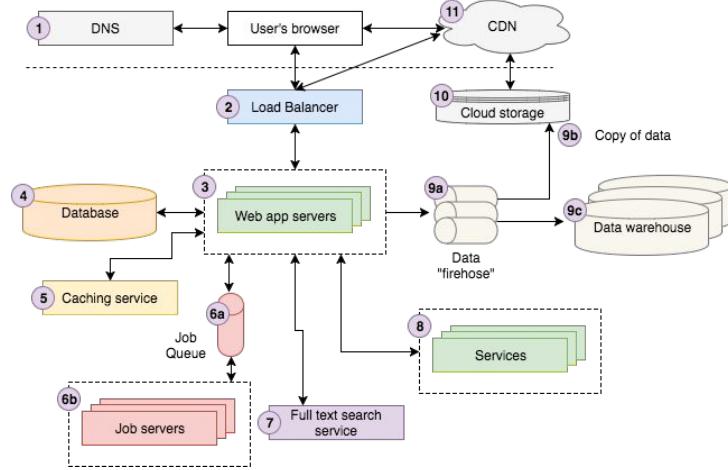


5 - Servicio de Caché - Conceptos Básicos.

Un servicio de almacenamiento en caché proporciona un almacén de datos clave, permitiendo guardar y buscar información en un tiempo menor.

Por lo general, las aplicaciones aprovechan los servicios de almacenamiento en caché para guardar los resultados de cálculos costosos para que sea posible recuperar los resultados de la memoria caché en lugar de volver a calcularlos la próxima vez que se necesiten.

Una aplicación puede almacenar en caché los resultados de una consulta de base de datos, llamadas a servicios externos, HTML para una URL determinada y muchos más.

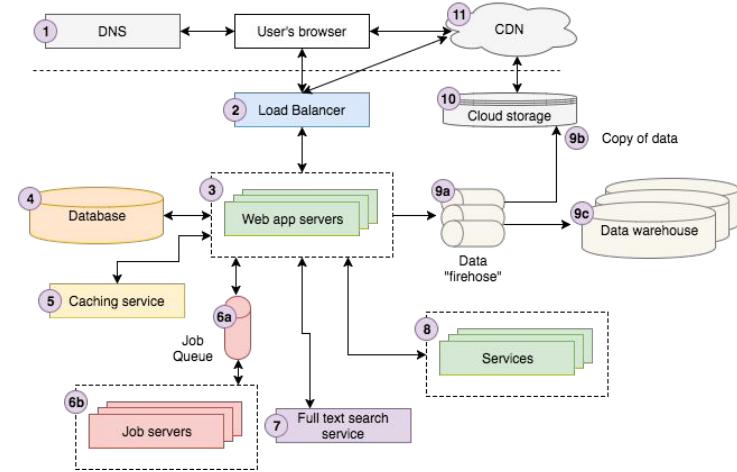


6 - Colas de Trabajo y Servidores -Conceptos Básicos.

La mayoría de las aplicaciones web necesitan hacer un trabajo asíncrono detrás de escena que no está directamente asociado con la respuesta a la solicitud de un usuario.

Si bien existen diferentes arquitecturas que permiten realizar trabajos asíncronos, la más ubicua es la arquitectura de cola de trabajos que consta de dos componentes:

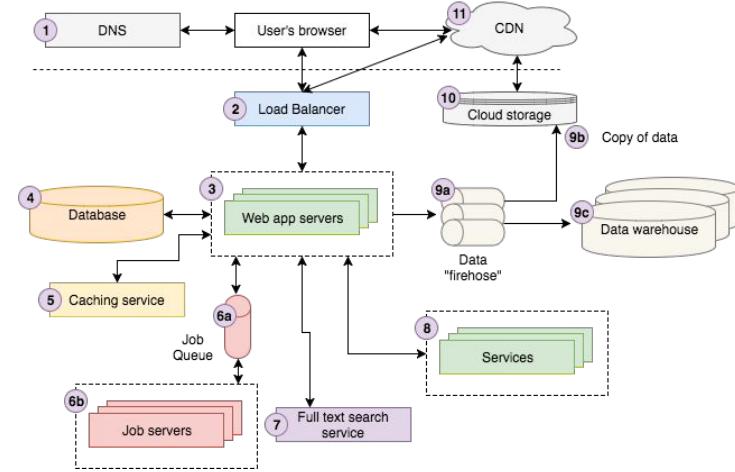
- una cola de "trabajos" que deben ejecutarse.
- Uno o más servidores de trabajo que ejecutan los trabajos en la cola.



7 - Full-Text Search Service - Conceptos Básicos.

La mayoría de las aplicaciones web admiten algún tipo de función de búsqueda en la que un usuario proporciona una entrada de texto y la aplicación devuelve los resultados más relevantes.

La tecnología que impulsa esta funcionalidad generalmente se conoce como "Full-Text", que aprovecha un índice invertido para buscar rápidamente documentos que contienen las palabras clave de consulta.



7 - Full-Text Search Service - Conceptos Básicos.

Documents (Photo titles)	
id	title
1	Man running in the mountains
2	Mountains with snow
3	Man running marathon



Inverted Index	
keyword	photo_ids
man	1, 3
running	1, 3
mountains	1, 2
snow	2
marathon	3

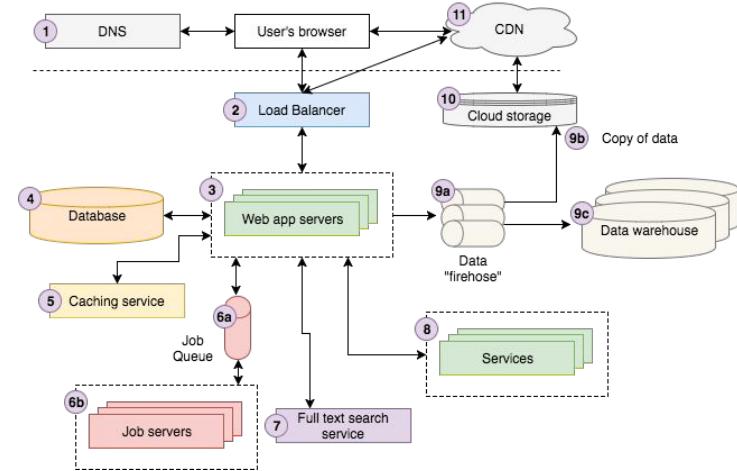
Lo común es utilizar un servicio de búsqueda separado que computa, almacena el índice invertido y proporciona una interfaz de consulta como: Elasticsearch (amazon), Apache Solr, Algolia y otros.

8 - Servicios - Conceptos Básicos.

Una vez que una aplicación alcanza cierta escala, es probable que haya ciertos "servicios" diseñados para ejecutarse como aplicaciones separadas.

Posiblemente estos no están expuestos externamente pero la aplicación interactúa con ellos.

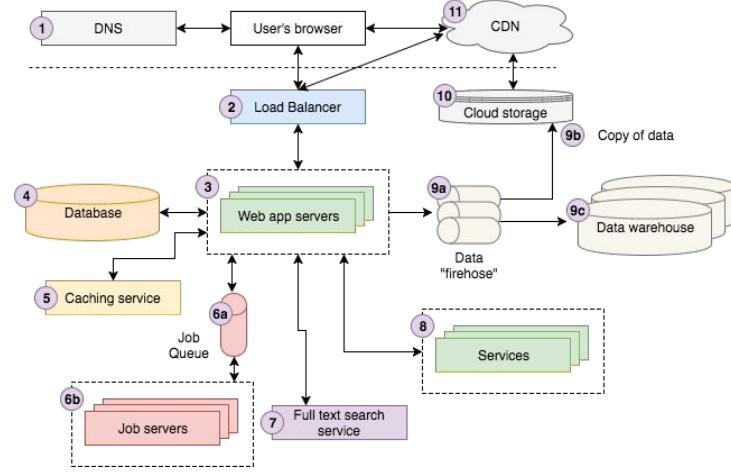
Ejemplo de estos son: servicios de autenticación de usuarios, control de usuarios(Cognito AWS), contenido (assets), servicios de pago, conversiones de datos, etc.



9 - Datos - Conceptos Básicos.

Casi todas las aplicaciones en estos días, una vez que alcanza una cierta escala, aprovechan una tubería de datos para garantizar que los datos se puedan recopilar, almacenar y analizar. Una tubería típica tiene tres etapas principales:

- A. Firehose: La aplicación envía datos, sobre las interacciones del usuario y proporciona una interfaz para procesar los datos.
- B. Warehouse: permiten hacer análisis e informes sobre datos estructurados y semiestructurados de diferentes fuentes.
- C. Almacenamiento de datos transformados para su uso en el futuro.

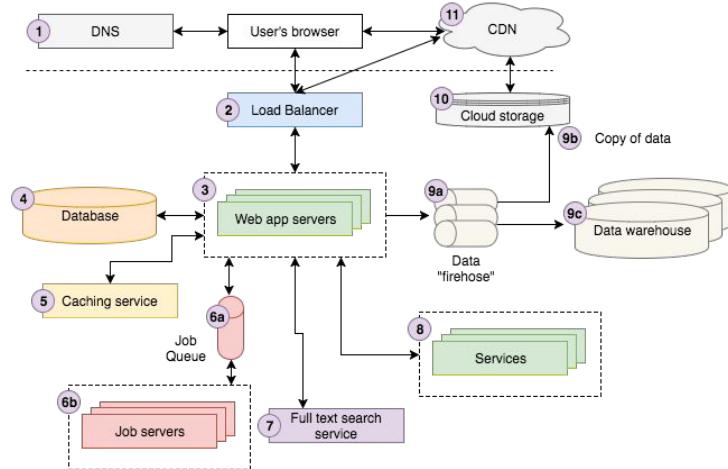


10 - Almacenamiento en la nube - Conceptos Básicos.

Es una forma simple y escalable de almacenar, acceder y compartir datos a través de Internet.

Puede usarlo para almacenar y acceder a más o menos cualquier cosa que almacene en un sistema de archivos local con los beneficios de poder interactuar con él a través de una API RESTful a través de HTTP.

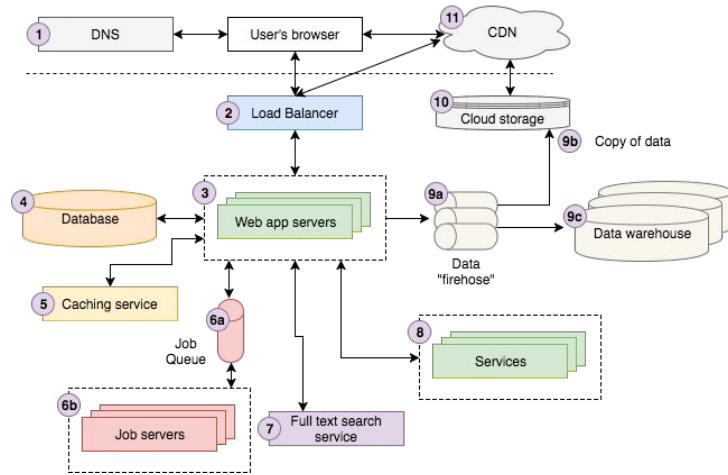
Amazon S3 es el almacenamiento en la nube más popular disponible en la actualidad.



11 - CDN Conceptos Básicos.

Significa "Content Delivery Network" y la tecnología proporciona una forma de servir activos como HTML estático, CSS, Javascript e imágenes en la web mucho más rápido que servirlos desde un único servidor de origen.

Funciona mediante la distribución del contenido en muchos servidores "perimetrales" de todo el mundo para que los usuarios terminen descargando activos de los servidores "perimetrales" en lugar del servidor de origen.



Arquitectura Web

Servicios REST



¿Qué es REST?

REpresentational State Transfer

Es un estilo arquitectónico, permite establecer estándares informáticos en la web, facilitando la comunicación entre sistemas.

Los sistemas RESTful, se caracterizan por manejar transacciones independientes y establecer la separación entre el cliente y servidor.

Cliente y Servidor

Debido a la naturaleza de la arquitectura la implementación del cliente y el servidor se pueden realizar de forma independiente. El código se puede cambiar en cualquiera de los dos sin afectar el funcionamiento del otro.

Todo esto se logra mientras cada uno de los lados sepa que formato de mensajes debe enviar al otro.

Esta arquitectura permite mejorar la interfaz entre plataformas y la escalabilidad, debido a que cada uno maneja sus tareas de forma independiente.

Cada componente puede evolucionar de forma independiente.

Comunicación Cliente

REST requiere que un cliente realice una solicitud al servidor para recuperar o modificar datos en el servidor.

Un Request generalmente consiste en:

- A. Verbo HTTP, que define qué tipo de operación realizar.
- B. Header, que permite al cliente transmitir información sobre la solicitud.
- C. Path del recurso.
- D. Cuerpo de mensaje (opcional) que contiene datos.

A. Verbos HTTP

Existen 4 verbos básicos HTTP, que son utilizados en los request para interactuar con los recursos en un sistema REST:

- GET: permite recuperar un recurso específico (por id) o una colección de recursos.
- POST: permite crear un nuevo recurso
- PUT: Actualizar un recurso específico por ID.
- DELETE: eliminar un recurso específico por ID.

B. Headers y Parámetros

En el header de la solicitud, el cliente envía el tipo de contenido que puede recibir del servidor. Esto se define en el *accept* y garantiza que el servidor, no envíe datos que el cliente no pueda comprender o procesar.

Las opciones para los tipos de contenido son MIME Types (Extensiones multipropósito de correo de Internet).

Los tipos MIME, utilizados para especificar los tipos de contenido en el campo Accept, consisten en un tipo y un subtipo. Están separados por una barra inclinada (/).

Nota: *accept* puede especificar más de un tipo de formato.

B. Headers y Parámetros

Los más comunes son:

- text/html: indica que es un archivo de texto que contiene HTML.
- text/css: indica que es un archivo de texto contiene CSS.
- text/plain: indica que es un archivo de texto genérico o plano.
- Formatos de imagen: **JPEG 2000, JPEG XR, WebP**, PNG, JPEG, GIF
- Formatos de audio: audio/wav, audio/mpeg
- Formatos de video: video/mp4, video/ogg
- Formatos de application: application/json, application/pdf, application/xml, application/octet-stream, application/xhtml.

Nota: si un cliente espera un tipo de formato y recibe otro no podrá reconocer el contenido.

C. Path del Recurso

Es importante que los request contengan una ruta al recurso al cual quieren realizar la operación.

En las API RESTful las rutas deben diseñarse de manera de ayudar al cliente a conocer qué está sucediendo. Por estández:

- La primera parte de la ruta debe ser la forma plural del recurso.
- Las rutas deben ser jerárquicas y descriptivas.
- Las rutas deben contener la información necesaria para ubicar un recurso con el grado de especificidad necesario.
- Cuando se hace referencia a una lista o colección de recursos no hace falta agregar un ID, eso solo se agrega si necesitamos identificar una ruta en específico.

D. Cuerpo del mensaje

Este es un field opcional, normalmente es utilizado cuando se crean nuevos recursos ejemplo de llamada utilizando POST:

Request:

```
POST http://www.myrestaurant.com/dishes/
```

Body -

```
{
  "dish": {
    "name": "Avocado Toast",
    "price": 8
  }
}
```

Comunicación Servidor

Response es una respuesta a un request, es una respuesta que da un servidor.

En los response tenemos dos partes importantes, adicionales:

- Tipo de contenido
- Códigos de respuestas

Tipo de Contenido

En los casos en que el servidor envía una carga de datos al cliente, el servidor debe incluir *content-type* en el encabezado de la respuesta. Este campo de encabezado de *content-type* alerta al cliente sobre el tipo de datos que está enviando en el cuerpo de respuesta.

Estos tipos de contenido son tipos MIME, tal como se encuentran en el campo *accept* del encabezado de la solicitud.

Nota: El tipo de contenido que el servidor devuelve en la respuesta debe ser una de las opciones que el cliente especificó en el campo *aceptar* de la solicitud.

Códigos de respuesta

Los responses del servidor contienen códigos de estado para alertar al cliente sobre información acerca del éxito de la operación. No necesitamos conocer todos los códigos de estado, solo los más comunes y cómo se usan:

200 (OK)	Esta es la respuesta estándar para solicitudes HTTP exitosas.
201 (CREATED)	Esta es la respuesta estándar para una solicitud HTTP que resultó en la creación exitosa de un elemento.
204 (NO CONTENT)	Esta es la respuesta estándar para solicitudes HTTP exitosas, donde no se devuelve nada en el cuerpo de la respuesta.
400 (BAD REQUEST)	La solicitud no se puede procesar debido a una sintaxis de solicitud incorrecta, un tamaño excesivo u otro error del cliente.
403 (FORBIDDEN)	El cliente no tiene permiso para acceder a este recurso.
404 (NOT FOUND)	El recurso no se pudo encontrar en este momento. Es posible que se haya eliminado o que aún no exista.
500 (INTERNAL SERVER ERROR)	La respuesta genérica para una falla inesperada si no hay más información específica disponible.

Restful

Tal como se define en la tesis de Roy Fielding, “Architectural Styles and the Design of Network-based Software Architectures”, las API son RESTful siempre que cumplan con las 6 limitaciones principales de un sistema RESTful:

1. **Arquitectura cliente-servidor:** la arquitectura REST está compuesta por clientes, servidores y recursos, y administra las solicitudes con HTTP.
2. **Sin estado:** el contenido de los clientes no se almacena en el servidor entre las solicitudes, sino que la información sobre el estado de la sesión se queda en el cliente. En su lugar, la información sobre el estado de la sesión está en posesión del cliente.

Restful

3. **Capacidad de caché:** el almacenamiento en caché puede eliminar la necesidad de algunas interacciones cliente-servidor.
4. **Sistema en capas:** las interacciones cliente-servidor pueden estar mediadas por capas adicionales, que pueden ofrecer otras funciones, como el equilibrio de carga, los cachés compartidos o la seguridad.
5. **Código de demanda (opcional):** los servidores pueden extender las funciones de un cliente transfiriendo código ejecutable.

RESTful - Interfaz Uniforme

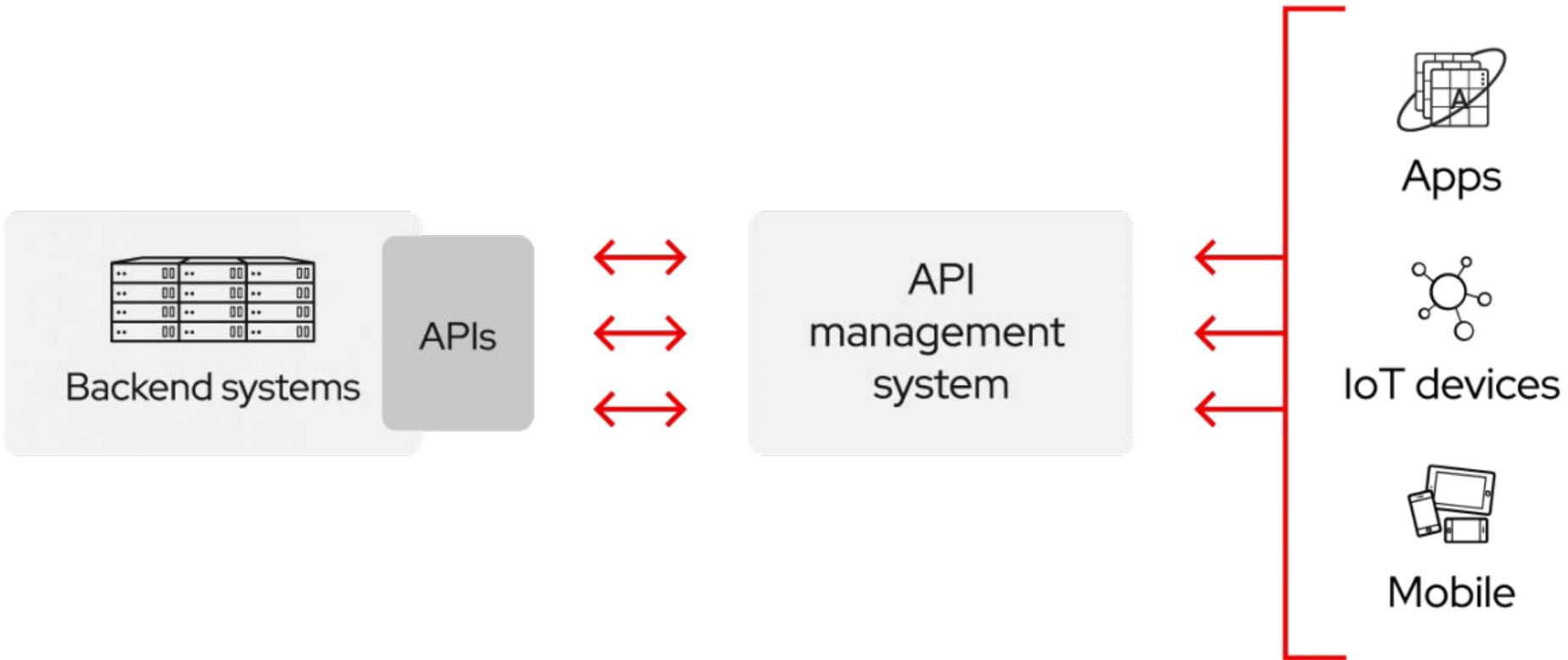
6. **Interfaz uniforme:** esta limitación es fundamental para el diseño de las API de RESTful e incluye 4 aspectos:
 1. Identificación de recursos en las solicitudes: los recursos se identifican en las solicitudes y se separan de las representaciones que se devuelven al cliente.
 2. Administración de recursos mediante representaciones: los clientes reciben archivos que representan los recursos. Estas representaciones deben tener la información suficiente como para poder ser modificadas o eliminadas.

RESTful - Interfaz Uniforme

3. Mensajes autodescriptivos: cada mensaje que se devuelve al cliente contiene la información suficiente para describir cómo debe procesar la información.
4. Hipermedios es el motor del estado de la aplicación: después de acceder a un recurso, el cliente REST debe ser capaz de descubrir mediante hipervínculos todas las otras acciones que están disponibles actualmente.

Arquitectura Web

Application Programming Interface (API)



Breve Historia

Las API surgieron los primeros días de la informática, mucho antes que la computadora personal.

En esa época, una API normalmente se usaba como biblioteca para los sistemas operativos. Casi siempre estaban habilitadas localmente en los sistemas en los que operaban, aunque a veces pasaban mensajes entre las computadoras centrales.

Después de casi 30 años, las API se expandieron más allá de los entornos locales. A principios del año 2000, ya eran una tecnología importante para la integración remota de datos.

API

Significa interfaz de programación de aplicaciones y es un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones.

Estas permiten que sus productos y servicios se comuniquen con otros, sin necesidad de saber cómo están implementados. Permiten:

- Flexibilidad
- Simplificar el diseño
- Administrar y usar de mejor manera de las aplicaciones
- Proporcionar oportunidades de innovación, lo cual es ideal al momento de diseñar herramientas y productos nuevos.

API

En las empresas las API se consideran como contratos, con documentación que representa un acuerdo entre las partes: si una de las partes envía una solicitud remota con cierta estructura en particular, esa misma estructura determinará cómo responderá el software de la otra parte.

Son un medio simplificado para conectar su infraestructura a través del desarrollo de aplicaciones nativas de la nube, pero también le permiten compartir sus datos con clientes y otros usuarios externos. Las API públicas representan un valor comercial único porque simplifican y amplían la forma en que se conecta con sus partners y, además, pueden rentabilizar sus datos (API's de Google).

API Remotas

Están diseñadas para interactuar en una red de comunicaciones. Remoto indica que los recursos que administran las API están, fuera de la computadora que solicita los recursos.

Debido a que la red de comunicaciones más usada es Internet, la mayoría de las API están diseñadas de acuerdo con los estándares web. No todas las API remotas son API web, pero se puede suponer que las API web son remotas.

Las API web normalmente usan HTTP para solicitar mensajes y proporcionar una definición de la estructura de los mensajes de respuesta.

Por lo general, estos mensajes de respuesta toman la forma de un archivo XML o JSON, que son los formatos preferidos porque presentan los datos en una manera fácil de manejar para otras aplicaciones.



API - Seguridad

Estas permiten habilitar el acceso a sus recursos y, al mismo tiempo, mantener la seguridad y el control.

La seguridad de las API tiene que ver con que se gestionen bien. Para conectarse a las API y crear aplicaciones que utilicen los datos o las funciones que estas ofrecen, se puede utilizar una plataforma de integración distribuida que conecte todos los elementos, incluidos los sistemas heredados y el Internet de las cosas (IoT).

API - Seguridad

Respecto a las políticas de privacidad existen tres enfoques:

- Privada: Las API solo se pueden usar internamente, así que las empresas tienen un mayor control sobre ellas. Esto le da a las empresas un mayor control sobre sus API.



- Pública: Todos tienen acceso a las API, así que otras empresas pueden desarrollar API que interactúen con las de usted y así convertirse en una fuente de innovaciones. Esto permite que terceros desarrollen aplicaciones que interactúen con su API, y puede ser un recurso para innovar.

De partners: Las API se comparten con partners empresariales específicos, lo cual puede ofrecer flujos de ingresos adicionales, sin comprometer la calidad. Esto puede proporcionar flujos de ingreso adicionales, sin comprometer la calidad.

SOAP Versus REST

Protocolo de Acceso a Objetos Simples	RESTful
<ul style="list-style-type: none">• Es un protocolo• Las API diseñadas con SOAP usan XML para el formato de sus mensajes y reciben solicitudes a través de HTTP o SMTP.• Permite que las aplicaciones que funcionan en entornos distintos o están escritas en diferentes lenguajes comparten información.• Cuenta con un estándar Oficial.	<ul style="list-style-type: none">• Es un estilo de arquitectura.• Normalmente usan JSON o XML para sus mensajes.• Permite compartir información entre sistemas sin importar como esta implementado cada sistema independiente.• Es más sencillo que un protocolo definido, no cuenta con un estándar oficial.• Actualmente es el más utilizado por su facilidad.

API Remotas - Arquitectura

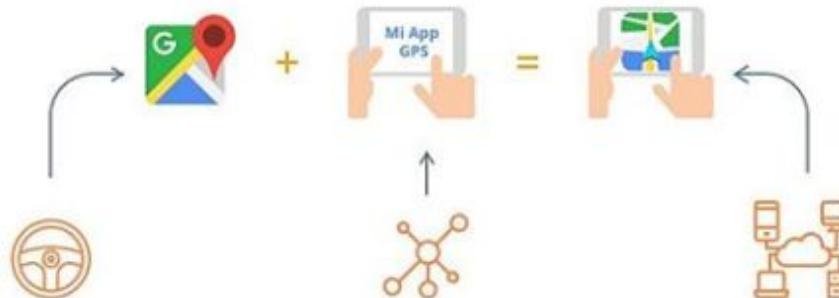
Los enfoques más utilizados son SOA y microservicios.

Como vimos en el curso SOA es una mejora a las aplicaciones monolíticas y permite usar varias aplicaciones para que realicen diferentes funciones todo gracias a la orquestación y coreografía. Puede ser un riesgo si las aplicaciones no se comprenden claramente.

Las arquitecturas de microservicios descomponen las arquitecturas en partes más pequeñas. Estas pueden utilizar un marco de mensajes comunes como API RESTful que no necesita integración de capas adicionales. Cada servicio es independiente, puede ser reemplazado, mejorado o abandonado. Esta arquitectura optimiza los recursos y admite la escalabilidad dinámica de los servicios individuales.

Application Programming Interface

Es una interfaz que permite compartir datos entre aplicaciones, podemos consumir los datos de otras Apps para nuestra App, por ej.



Interfaz

Permite interactuar con un sistema sin la necesidad de saber que es lo que esta pasando por debajo



REST

Es la arquitectura con la que se puede consultar información de otros sistemas.

Arquitectura de software

Es La forma en la que está diseñado un sistema, como están organizado sus componentes, que funciones cumplen



Json

Es el formato más usado para transferir información JavaScript Object Notation

Servicios Web

Es un sistema que permite la comunicación entre equipos, estos sistemas pueden usar, entre otros, el protocolo HTTP



Token

Es un objeto que contiene todos los datos de una autenticación para poder usar una API



Laboratorio

Sling

Recomendación usar
<https://www.postman.com/>

Laboratorio

Crear un catálogo de libros, debajo de cada libro crear editorial y un listado de autores.

Se debe guardar los siguientes datos por tipo de evento:

- Autor (nombre, edad)
- Libro (título, autor, género, año)
- Editorial (nombre y país)

Crear 5 items de cada uno

Editar los recursos:

- Autor (actualizar nombre)
- Libro (actualizar campo año).

Eliminar

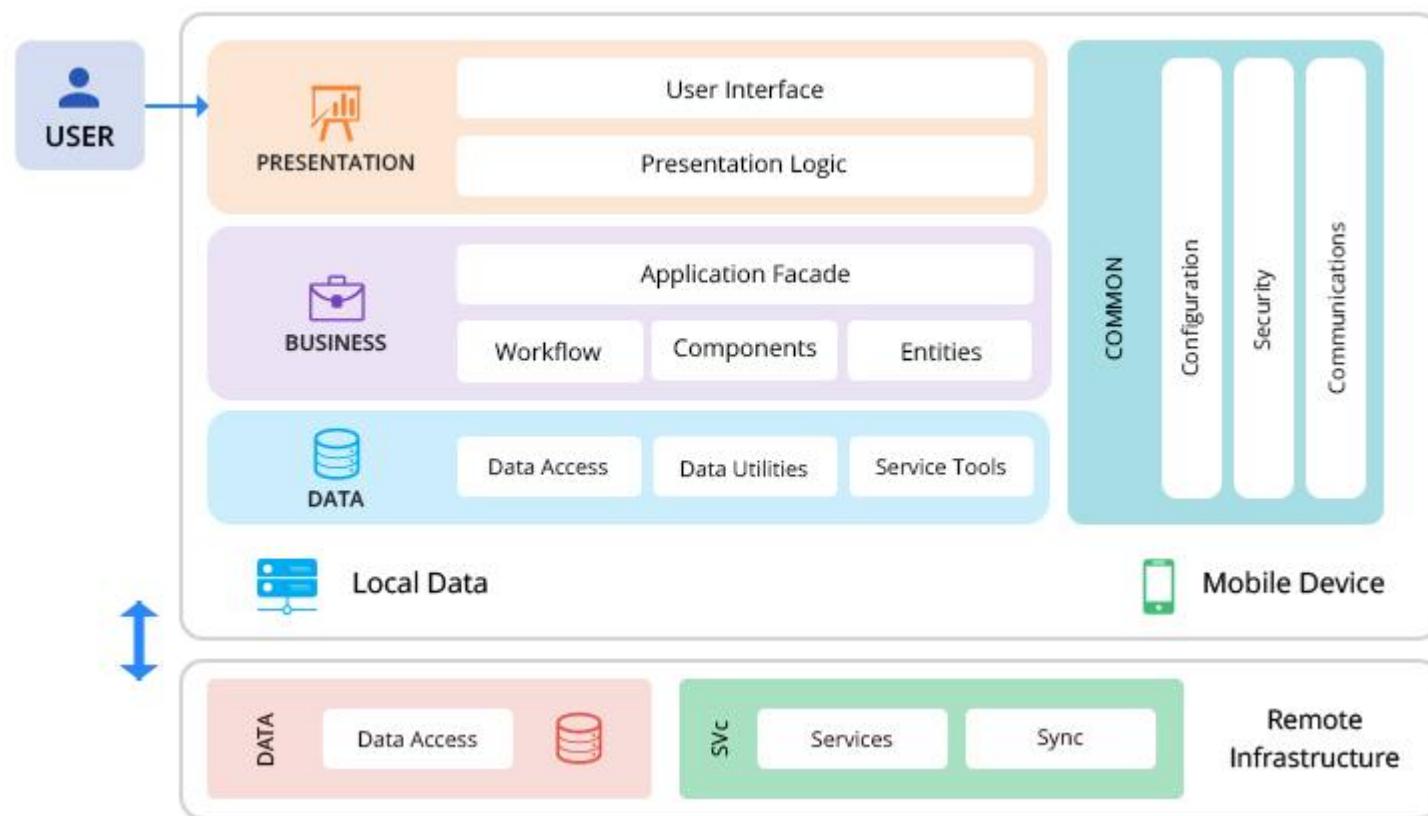
- Un autor



Arquitectura - Móviles

Aplicaciones Híbridas

Analicemos la arquitectura...



Aplicaciones Híbridas

Es una aplicación de software que combina elementos de aplicaciones nativas y aplicaciones web .

Las aplicaciones híbridas son esencialmente aplicaciones web que se han colocado en un shell de aplicaciones nativas. Una vez que se descargan de una tienda de aplicaciones y se instalan localmente, el shell puede conectarse a cualquier capacidad que la plataforma móvil brinde a través de un navegador que está integrado en la aplicación. El navegador y sus complementos se ejecutan en el back-end y son invisibles para el usuario final.

Aplicaciones Híbridas



Las aplicaciones híbridas son populares porque permiten a los desarrolladores escribir código para una aplicación móvil una vez y aún acomodar múltiples plataformas (Android, iOS).

Debido a que las aplicaciones híbridas agregan una capa adicional entre el código fuente y la plataforma de destino, pueden funcionar un poco más lento que las versiones nativas o web de la misma aplicación.

Aplicaciones Híbridas

Entre sus características principales están:

- La capacidad de funcionar independientemente de si el dispositivo está conectado o no.
- Integración con el sistema de archivos del dispositivo móvil.
- Integración con servicios web.
- Un navegador integrado para mejorar el acceso al contenido dinámico en línea.
- Opera en diferentes plataformas.

¿Comó Funcionan?

Estas funcionan de manera similar a las aplicaciones web, pero al igual que las aplicaciones nativas, se descargan en el dispositivo. Al igual que las aplicaciones web, las aplicaciones híbridas generalmente se escriben en HTML5 , CSS y JavaScript .

Las aplicaciones híbridas ejecutan código dentro de un contenedor. El motor del navegador del dispositivo se utiliza para representar HTML y JavaScript y API nativas para acceder al hardware específico del dispositivo.

¿Comó Funcionan?

Aunque una aplicación híbrida generalmente compartirá elementos de navegación similares a los de una aplicación web, si la aplicación puede funcionar sin conexión o no depende de sus funcionalidades.

Si una aplicación no necesita soporte de una base de datos, se puede hacer que funcione sin conexión.

Aplicaciones Híbridas - Ventajas

Las ventajas están:

- Operará en diferentes plataformas.
- Tiempos de compilación más rápidos en comparación con las aplicaciones nativas.
- Más barato de desarrollar en comparación con la construcción de dos versiones de una aplicación nativa para dos plataformas diferentes.
- Más fácil de iniciar parches y actualizaciones.
- Puede trabajar en línea y sin conexión
- Usan web views.

Aplicaciones Híbridas - Desventajas

Las desventajas están:

- Variaciones debidas al desarrollo inclinado en una plataforma.
- La apariencia de una aplicación puede variar de una plataforma a otra.
- La experiencia del usuario (UX) puede disminuir si la interfaz de usuario (UI) no es similar y está suficientemente diseñada para los navegadores a los que está acostumbrado el usuario.
- Con respecto al control de calidad se debe probar la aplicación en una variedad de dispositivos para garantizar un funcionamiento correcto.
- Más difícil de ser aceptado y publicado en App Store

Mobile App Technology Stacks

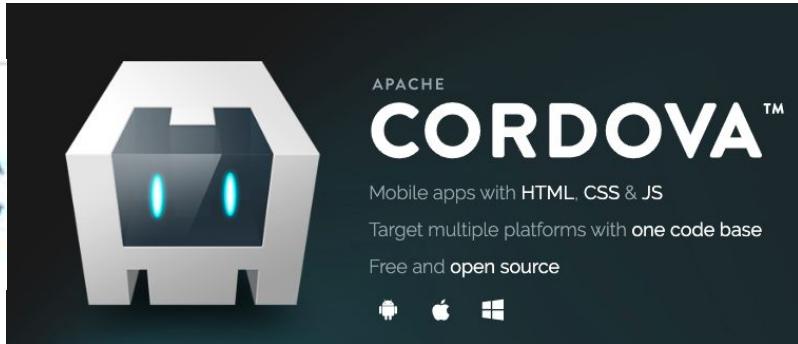


Híbrido Vs Nativo Vs Web

Híbrido	Nativo	Web
<ul style="list-style-type: none">• Se crea para ser utilizado en múltiples plataformas.• Solo pueden aprovechar parte del hardware de un dispositivo.• Escrito bajo utilizando lenguajes utilizados para crear aplicaciones web.• Se instala en el dispositivo.• Es más lenta y la experiencia de usuario es estándar.• Aparecen en el store/market.	<ul style="list-style-type: none">• Se crean específicas para una plataforma.• Pueden aprovechar el hardware de un dispositivo.• Escritas bajo el lenguaje de programación del sistema operativo.• Se instala en el dispositivo• Es más rápida y tiene una mejor experiencia de usuario.• Aparecen en el store/market.	<ul style="list-style-type: none">• Se crean para funcionar dentro de un browser.• Puede aprovechar sólo los recursos proporcionados al browser.• Escritas utilizando lenguajes como HTML, librerías de javascript, css, etc.• No se instala en el dispositivo.• Pueden ser lentas, menos intuitivas.• No aparecen en el store/market.

En Resumen: Las aplicaciones híbridas combinan aplicaciones nativas y web. Esta se instala y funciona de manera similar a una aplicación nativa, pero tiene el funcionamiento interno de una aplicación web.

Herramientas



Arquitectura - Móviles

Cross Compiled / Cross Platform

Cross Compiled / Cross Platform

Este es el nivel más cercano al desarrollo nativo, pero tiene la ventaja de usar una única base de código para apuntar a Android e iOS.

Con este tipo de herramientas se crea el código que luego es compilado hacia la plataforma nativa.

Generando una aplicación Nativa con algunos pequeños cambios.



Cross Compiled - Ventajas

Entre las ventajas están:

- Se obtiene una app para ambas plataformas con un solo lenguaje.
- Aproximadamente el mismo rendimiento que las aplicaciones nativas, ya que también tratan con componentes nativos de la interfaz de usuario

Nota: Flutter, proporcionará su propio conjunto de widgets junto con el código de su aplicación



Cross Compiled - Desventajas

Entre las desventajas están:

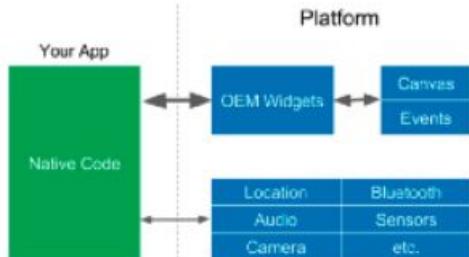
- Soporte ligeramente retrasado para las últimas actualizaciones de la plataforma.
- Código no compatible con desarrollo web.
- Incluso con una única base de código, el desarrollador debe conocer los componentes nativos.

Herramientas

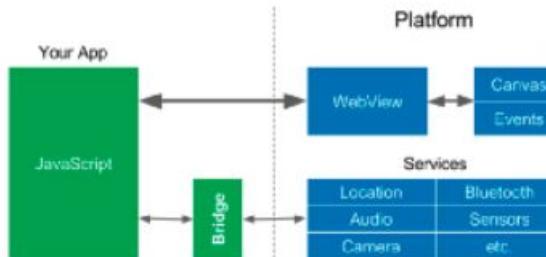




RUBYMOTION



NATIVE & CROSS COMPILED

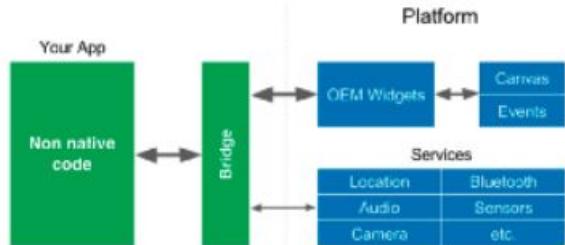


HYBRID APACHE CORDOVA™

PhoneGap

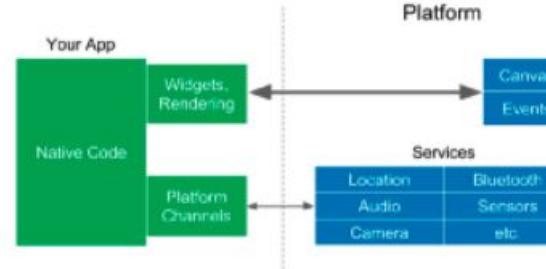
Xamarin

WEB NATIVE & CROSS COMPILED



FLUTTER

Flutter



Cuatro arquitecturas de tiempo de ejecución de aplicaciones móviles diferentes.

Imagen tomada del siguiente [link](#)

Laboratorio

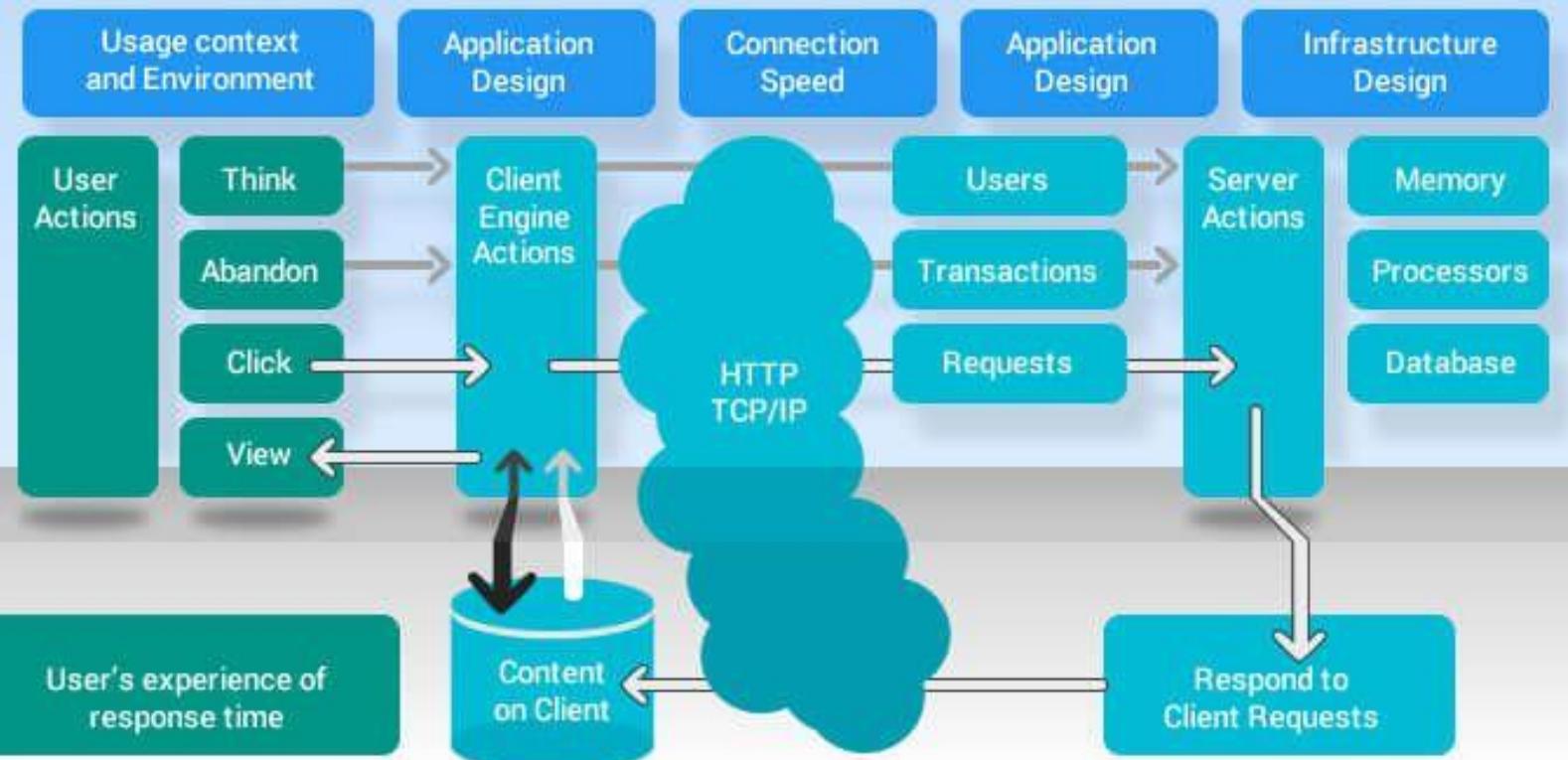


First Flutter App: Building
your first Flutter app |
Workshop

Modelo y Diseño de Arquitectura

- Web

Rich Internet Application (RIA)



The Rich Internet Application Behaviour Model

RIA (Rich Internet Application)

Las aplicaciones de Internet enriquecidas son aplicaciones web que incorporan la mayoría de las características de una aplicación de escritorio al tiempo que proporcionan una interfaz de usuario que responde mejor que una aplicación tradicional.

A diferencia de las aplicaciones tradicionales, estas permiten tener actualizada la aplicación, permitiendo al usuario tener los features.

Las aplicaciones de Internet enriquecidas se usan comúnmente para ofrecer funciones como videos, procesadores de texto y juegos en línea a los usuarios.

RIA (Rich Internet Application)

Las aplicaciones RIA permiten trasladar parte del cálculo al cliente. Los elementos del programa en el cliente están listos para recibir y ejecutar comandos asíncronos del servidor. Eliminando request adicionales innecesarios.

Normalmente en RIA la interfaz comprende una sola pagina que comprende subpáginas, administra las interacciones del usuario, como en una aplicación de escritorio. Este paradigma evita actualizaciones de página completa en cada interacción y permite que las aplicaciones carguen, muestren y actualicen elementos de página individuales de forma independiente.

RIA - Beneficios

Los RIA ofrecen a las organizaciones una forma comprobada y rentable de ofrecer altos niveles de funcionalidad y brindar beneficios comerciales reales:

- Ofrecen a los usuarios una experiencia más rica y atractiva y aumenta la productividad del usuario.
- Admite la visualización avanzada de datos, incluidos cuadros y presentaciones gráficas de datos.
- Permite tener la información en tiempo real.
- Heredan los beneficios de las aplicaciones web tales como accesibilidad, escalabilidad y portabilidad.

RIA - Ventajas

En comparación con las aplicaciones estándar basadas en HTML, los RIA potencian su negocio con soluciones que son más:

- **Consistente:** a diferencia de las aplicaciones HTML que a menudo necesitan recargar varias páginas para completar una acción, los RIA permiten transiciones de etapas sin interrupciones y controladas interactivamente que no distraen a los usuarios del objetivo final de una acción.
- **Orientado:** las tecnologías de RIA brindan una excelente experiencia para el desarrollo de interfaces de usuario atractivas que se centran por completo en sus necesidades y las de sus clientes, y tienen el diseño de la aplicación optimizado para el propósito que sirve.

RIA - Ventajas

En comparación con las aplicaciones estándar basadas en HTML, los RIA potencian su negocio con soluciones que son más:

- **Receptivo:** debido a las tecnologías específicas utilizadas, los RIA simplifica procesos complejos (como el registro o la compra), ahorran ancho de banda y funcionan considerablemente más rápido que las aplicaciones tradicionales.
- **Inteligente:** ajustando dinámicamente su comportamiento mediante la captura, el mantenimiento y el uso de información contextual, los RIA entregan de forma interactiva a los usuarios exactamente lo que necesitan y reducen su carga de trabajo, tiempo y esfuerzo dedicados al desempeño de acciones específicas.

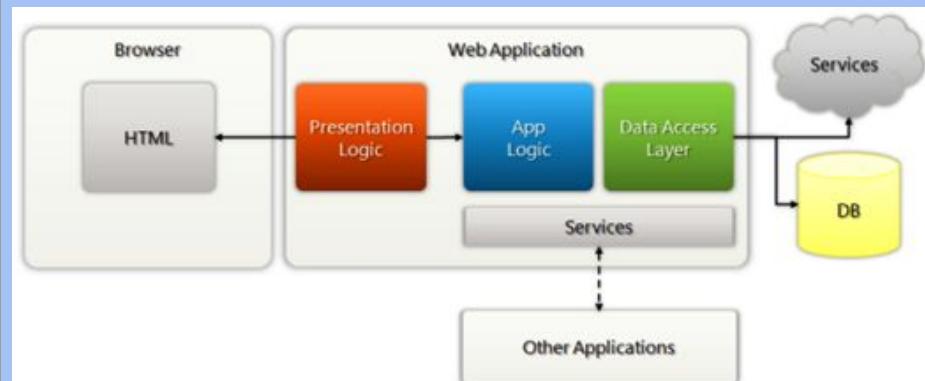
Aplicaciones Web Tradicional Versus RIA

Aplicaciones Web Tradicionales	RIA
<ul style="list-style-type: none">• Basada en Texto• Comunicación Sincrona: cada vez que se solicita algo se espera la respuesta del servidor.• Necesita recargar la página o cambiar de página.• La carga de trabajo la tiene el servidor (sesión de usuario, procesar los datos).• Cliente liviano regularmente.• La lógica de presentación y de negocio suelen estar dentro de la misma capa física, por lo que la comunicación entre ambas no supone ninguna complejidad.	<ul style="list-style-type: none">• Basados en la interacción del cliente.• Comunicación asíncrona: normalmente el cliente puede realizar varias peticiones sin esperar la respuesta.• Normalmente se actualiza solo la porción de información que se necesita.• Aprovecha también los recursos del cliente, ya que ejecuta la mayoría de funciones localmente.• El cliente es pesado, debido a que tenemos parte de la aplicación en el.• La lógica de presentación y negocio no están en la misma capa física, hay que considerar proxys, validación y autenticación.

Aplicaciones Web Tradicional Versus RIA

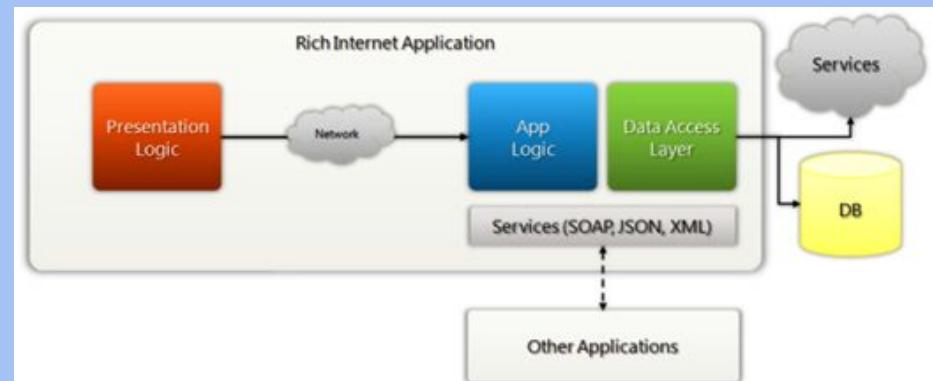
Aplicaciones Web Tradicionales

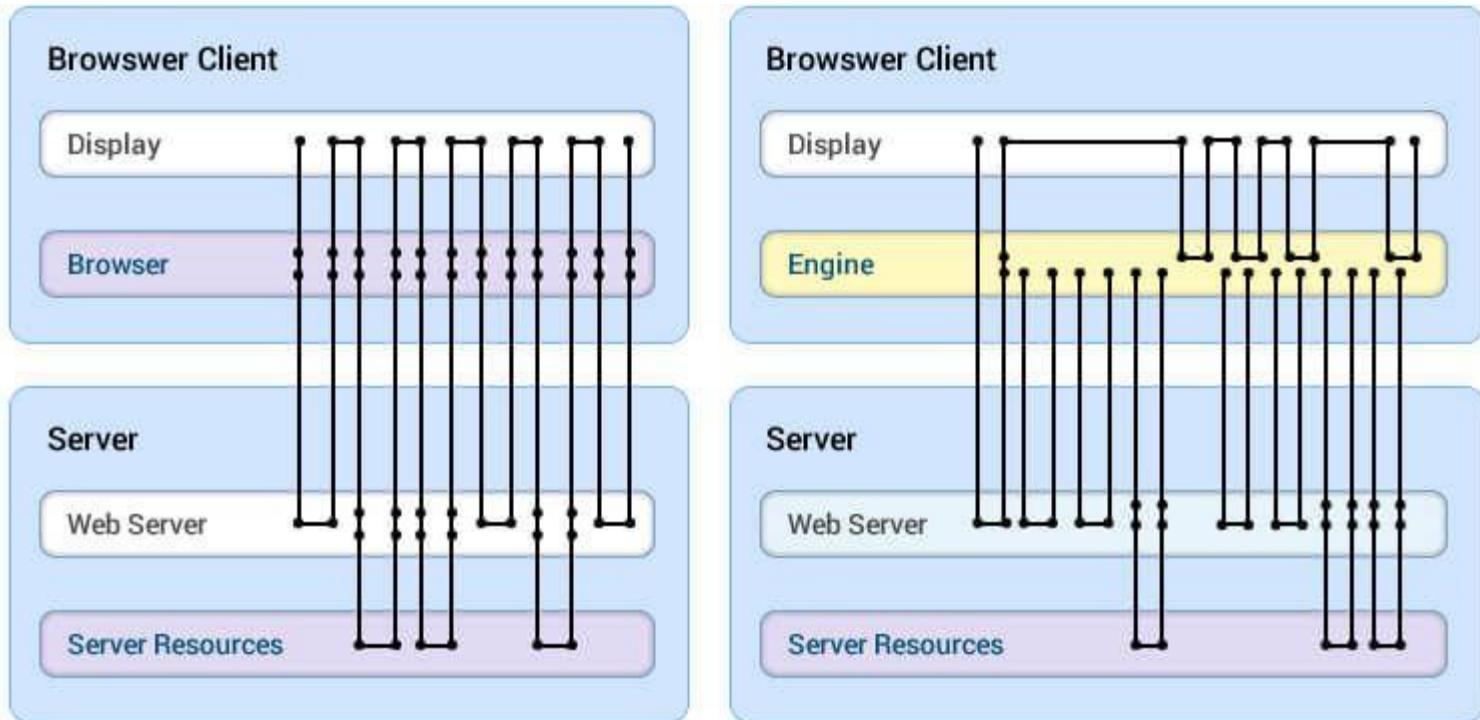
- Arquitectura base



RIA

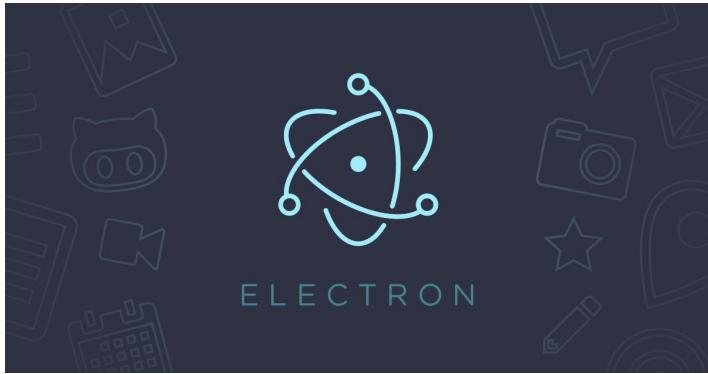
- Arquitectura base





The Communication Model: Traditional Web Application vs. Rich Internet Application

RIA Herramientas

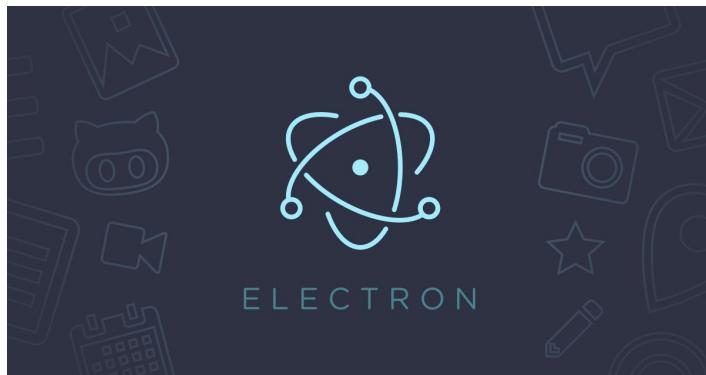


Algunas RIA



Basecamp 3

Laboratorio



Electron - Hello World :
[https://www.electronjs.org
/docs/tutorial/quick-start](https://www.electronjs.org/docs/tutorial/quick-start)

Modelo y Diseño de Arquitectura - Web Single Page Application (SPA)

SPA (Single Page Application)

Las aplicaciones de una sola página (SPA) son aplicaciones web que cargan una sola página HTML y actualizan dinámicamente esa página a medida que el usuario interactúa con la aplicación.

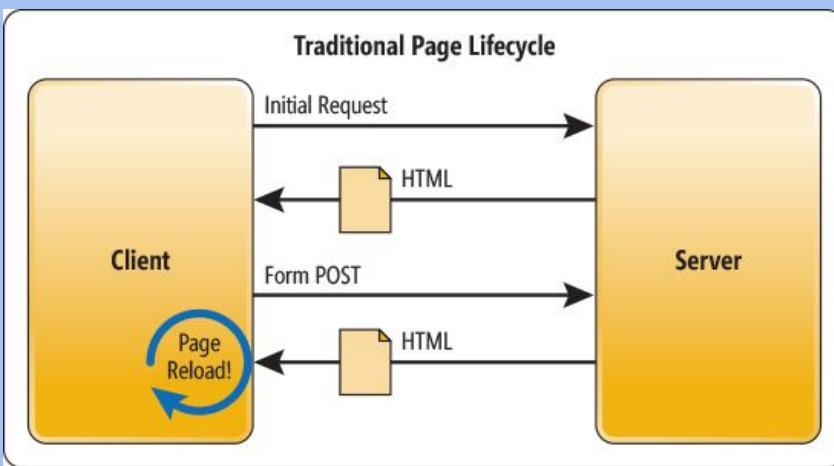
Los SPA utilizan AJAX y HTML5 para crear aplicaciones web fluidas y receptivas, sin recargas constantes de página. Sin embargo, esto significa que gran parte del trabajo ocurre en el lado del cliente, en JavaScript. Afortunadamente, hay muchos frameworks de JavaScript de que facilitan la creación de SPA.

Aplicaciones Web Tradicional Versus SPA

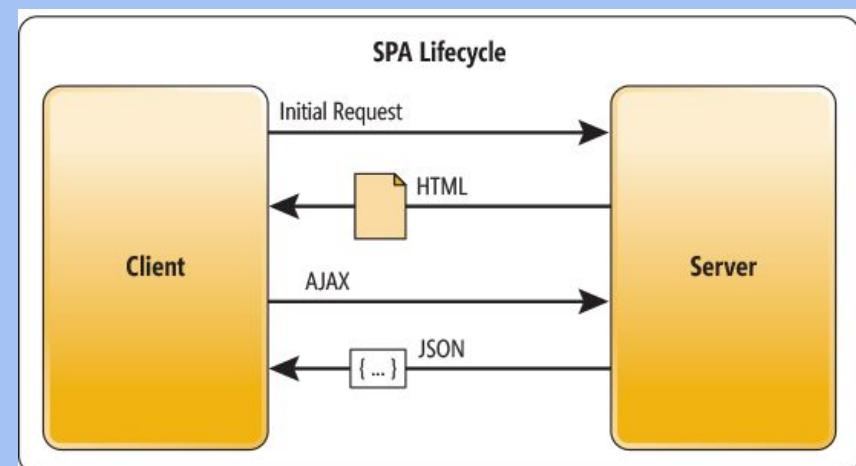
Aplicaciones Web Tradicionales	SPA
<ul style="list-style-type: none">• cada vez que la aplicación llama al servidor, el servidor presenta una nueva página HTML. Esto desencadena una actualización de la página en el navegador.• Actualización de la página se da por recarga de toda la página.• Aplicaciones no son tan fluidas.• Puede no hacerse la separación de presentación y lógica.• Se realizan muchas llamadas al servidor.• Cliente y servidor dependientes.	<ul style="list-style-type: none">• después de cargar la primera página, toda interacción con el servidor ocurre a través de llamadas AJAX. Estas llamadas AJAX devuelven datos, no marcas, generalmente en formato JSON.• Se actualiza solo un segmento de la página de forma dinámica.• Aplicaciones más fluidas y receptivas.• Separación entre presentación (HTML) y lógica (respuestas json).• Se optimiza el número de llamadas al servidor y los recursos.• Cliente y servidor independientes, el servidor actúa como capa de servicio tras la carga inicial de la página.

Aplicaciones Web Tradicional Versus SPA

Aplicaciones Web Tradicionales



SPA



SPA Herramientas

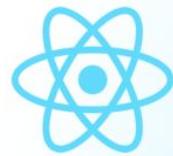


ReactJS



Vue.js

Laboratorio



ReactJS

React - Tutorial :

<https://reactjs.org/tutorial/tutorial.html>



Arquitectura de sistemas, comportamiento y optimización

Week 10

M.Sc. Juan Luis Flores Pineda

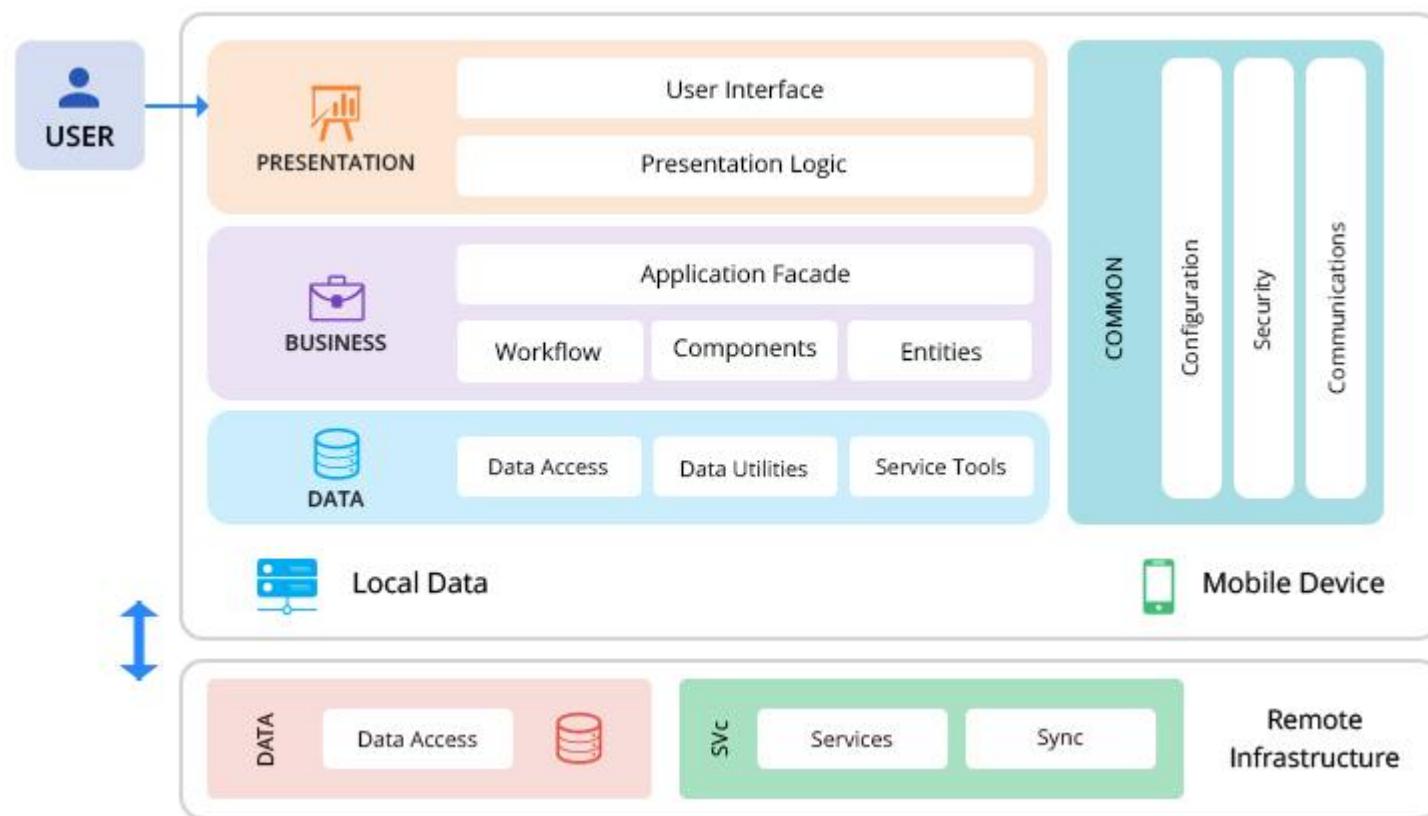
Agenda

- Arquitectura Web
 - ◆ Incidencias comunes.
 - ◆ Incidencias de aplicación
 - ◆ Incidencias de seguridad
 - ◆ Herramientas
 - ◆ Análisis de casos de arquitectura.
- Introducción a la Integración de software
 - ◆ Introducción.
 - ◆ Entrega y Implementación continua
 - ◆ Diferencias entre aplicaciones distribuidas y aplicaciones integradas.

Arquitectura Web

Incidencias Comunes

Analicemos la arquitectura...



Introducción

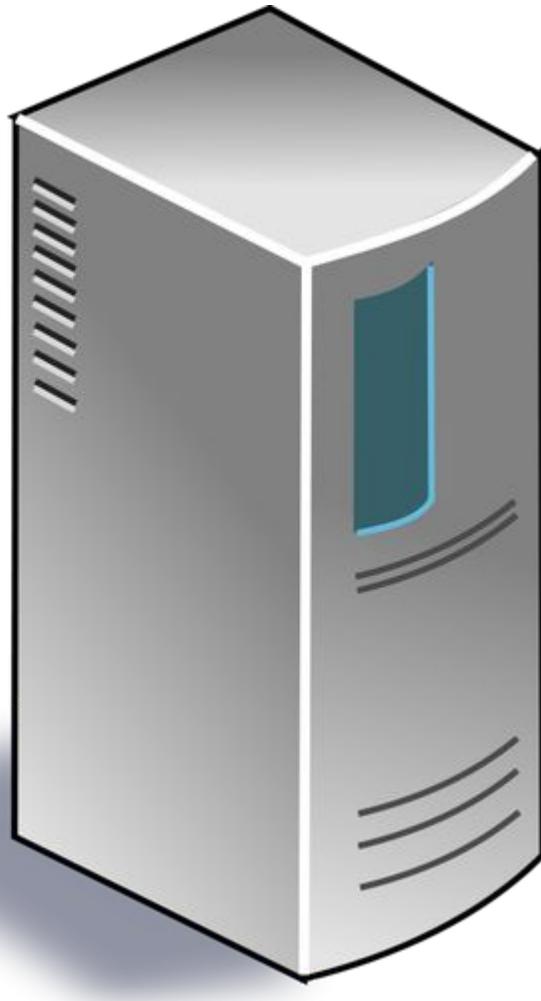
Una de las tareas principales para el arquitecto de soluciones es hacer que dicho sistema sea altamente independiente de: proveedor, base de datos, lenguaje de programación, marco, etc. Para lograrlo, necesita un gran esfuerzo para invertir. Pregúntese como ingeniero, ¿cuántas veces en su carrera migró el motor de la base de datos de su solución a otro? ¿Alguna vez reemplazó su base de datos MySQL con Oracle, por ejemplo?

Introducción

Para lograr la independencia, debe invertir mucho tiempo y dinero. Si es una startup que necesita probar su MVP en el mercado, nunca debe construirla teniendo en cuenta dicha arquitectura. Pero una vez que MVP se convierte en un producto que es utilizado por un número significativo de personas, la arquitectura de MVP debe ser reemplazada por una que sea escalable, flexible, independiente, etc. Esta trampa es muy común, por lo que puede diseñar demasiado su solución en el comienzo, lo que destruye su capacidad de ir al mercado rápidamente o nunca reemplaza su arquitectura MVP con la adecuada.



"Un buen arquitecto de soluciones tratará de comprender el futuro del sistema y, basándose en eso, elegirá el nivel correcto de arquitectura que debería estar en su lugar."



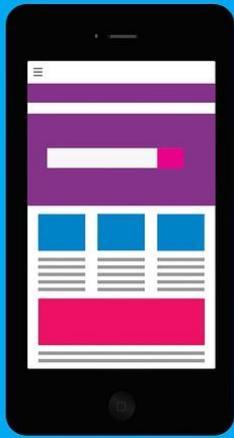
Incidencias de los Fisicas



Incidencias Fisicas

Son las incidencias que nacen debido a cuestiones físicas tales como:

- Capacidad del servidor
 - Disco
 - Memoria
 - Conexión
- Ubicación
 - Tiempo de respuesta
 - Conexión entre distintos servidores.
- Ancho de banda
 - Protocolos de Internet



Incidencias de aplicación

Escalabilidad



Debido al desarrollo de alta velocidad y a la idea de tener un MVP lo antes posible. Esto ha causado que la mayoría de los equipos den por sentada la escalabilidad.

Muchas veces seleccionar una base de datos escalable y un servidor web e intentar separar todas las capas de aplicaciones.

Debemos pensar que sucede al querer escalar la aplicación que problemas podemos encontrarnos, que sucede al poner un load balancer, como sincronizamos los distintos servidores.

Escalabilidad

Entre los problemas que podemos tener en escalabilidad están:

1. Sincronización de los distintos servidores.
2. Tiempos de respuesta entre servidores.
3. Diferentes Ubicaciones.
4. Módulos difíciles de escalar o adaptar.
5. Desarrollo no pensando en ser escalable.
6. Otros.

SEO incorrecto o faltante

Se debe pensar en SEO desde el inicio de la aplicación, en muchas ocasiones se piensa en esta actividad al final.

Ya que está relacionado con la buena configuración del contenido, palabras clave (keywords), etiquetas alternativas para las imágenes, mapa del sitio, estructura del contenido, tiempos de carga eficientes, enlaces inteligentes.

Solución: muchas veces reorganizar el contenido completo, agregar implementación.



Incidentes de Seguridad

Seguridad - Introducción

La seguridad es un problema que vemos cada vez más frecuente en todo tipo de arquitectura, se ha vuelto muy común que hackers roban datos de empresas importantes (Sony, etc).

Es importante siempre practicar medidas de seguridad y siempre esté preparado para protegerse y proteger el futuro de su empresa de un ataque del que nunca se recuperará.

La mejor manera de saber si su sitio web o servidor es vulnerable es realizar auditorías de seguridad periódicas.

Entre los casos más comunes están:

Seguridad - La inyección SQL

Es un tipo de vulnerabilidad de seguridad de la aplicación web en la que un atacante intenta usar el código de la aplicación para acceder o dañar el contenido de la base de datos. Si tiene éxito, esto permite al atacante crear, leer, actualizar, alterar o eliminar datos almacenados en la base de datos de fondo.

La inyección SQL es uno de los tipos más frecuentes de vulnerabilidades de seguridad de aplicaciones web .

Seguridad - Cross Site Scripting (XSS)

Las secuencias de comandos entre sitios (XSS) se dirigen a los usuarios de una aplicación inyectando código, generalmente una secuencia de comandos del lado del cliente como JavaScript, en la salida de una aplicación web.

El concepto de XSS es manipular los scripts del lado del cliente de una aplicación web para ejecutarlos de la manera deseada por el atacante. XSS permite a los atacantes ejecutar scripts en el navegador de la víctima que pueden secuestrar sesiones de usuario, desfigurar sitios web o redirigir al usuario a sitios maliciosos.



Seguridad - Mal Manejo de Autenticación

La autenticación y la administración de sesión interrumpidas abarcan varios problemas de seguridad, todos ellos relacionados con el mantenimiento de la identidad de un usuario.

Si las credenciales de autenticación y los identificadores de sesión no están protegidos en todo momento, un atacante puede secuestrar una sesión activa y asumir la identidad de un usuario.

Seguridad - Exposición de referencias

La referencia directa a objetos inseguros es cuando una aplicación web expone una referencia a un objeto de implementación interno.

Los objetos de implementación interna incluyen archivos, registros de bases de datos, directorios y claves de bases de datos.

Cuando una aplicación expone una referencia a uno de estos objetos en una URL, los hackers pueden manipularla para obtener acceso a los datos personales de un usuario.

Seguridad - Configuración incorrecta de Seguridad

La configuración incorrecta de seguridad abarca varios tipos de vulnerabilidades, todas centradas en la falta de mantenimiento o la falta de atención a la configuración de la aplicación web.

Se debe definir e implementar una configuración segura para la aplicación, los marcos, el servidor de aplicaciones, el servidor web, el servidor de bases de datos y la plataforma.

La configuración incorrecta de seguridad brinda a los hackers acceso a datos o funciones privadas y puede dar lugar a un compromiso completo del sistema.

Seguridad - Cross-Site Request Forgery (CSRF)

Cross-Site Request Forgery (CSRF) es un ataque malicioso en el que se engaña a un usuario para que realice una acción que no tenía la intención de hacer.

Un sitio web de terceros enviará una solicitud a una aplicación web con la que el usuario ya está autenticado (por ejemplo, su banco). El atacante puede acceder a la funcionalidad a través del navegador ya autenticado de la víctima. Los objetivos incluyen aplicaciones web como redes sociales, en clientes de correo electrónico del navegador, banca en línea e interfaces web para dispositivos de red.

Arquitectura Web

Técnica de Análisis

Root Cause Analysis



Análisis de Causa Raíz

El análisis de causa raíz (RCA) es una técnica popular y de uso frecuente que ayuda a las personas a responder la pregunta de por qué ocurrió el problema en primer lugar. Busca identificar el origen de un problema utilizando un conjunto específico de pasos, con herramientas asociadas, para encontrar la causa principal del problema, de modo que pueda:

1. Determina lo que pasó.
2. Determina por qué sucedió.
3. Averigüe qué hacer para reducir la probabilidad de que vuelva a suceder.

Análisis de Causa Raíz

RCA supone que los sistemas y eventos están interrelacionados. Una acción en un área desencadena una acción en otra, y en otra, y así sucesivamente.

Al rastrear estas acciones, puede descubrir dónde comenzó el problema y cómo se convirtió en el síntoma que enfrenta ahora.

RCA analiza los tres tipos de causas. Implica investigar los patrones de efectos negativos, encontrar fallas ocultas en el sistema y descubrir acciones específicas que contribuyeron al problema. Esto a menudo significa que RCA revela más de una causa raíz.

Análisis de Causa Raíz - Tipos de Causas

Por lo general, encontrará tres tipos básicos de causas:

1. Causas físicas : los elementos materiales tangibles fallaron de alguna manera.
2. Causas humanas : las personas hicieron algo mal o no hicieron algo que era necesario. Las causas humanas generalmente conducen a causas físicas.
3. Causas organizativas : un sistema, proceso o política que las personas usan para tomar decisiones o hacer su trabajo es defectuoso.

Análisis de Causa Raíz

Dificultad: Determinar qué tan lejos llegar en su investigación requiere buen juicio y sentido común. Teóricamente, podría seguir rastreando las causas fundamentales, pero el esfuerzo no serviría para nada.

Análisis de Causa Raíz - Paso 1

RCA tiene cinco pasos identificables:

1. Definir el problema:

- Responde a las preguntas
 - i. ¿Qué ves que pasa?
 - ii. ¿Cuáles son los síntomas específicos?

Análisis de Causa Raíz - Paso 2

2. Recopilar los datos:

a. Responde a las preguntas:

- i. ¿Qué prueba tienes de que el problema existe?
- ii. ¿Cuánto tiempo ha existido el problema?
- iii. ¿Cuál es el impacto del problema?

Debe analizar completamente una situación antes de poder seguir para ver los factores que contribuyeron al problema. Para maximizar la efectividad de su RCA, reúna a todos, expertos y los que están más familiarizadas con el problema.

Análisis de Causa Raíz - Paso 3

3. Identifique los posibles factores causales:
 - a. Responde a las preguntas
 - i. ¿Qué secuencia de eventos conduce al problema?
 - ii. ¿Qué condiciones permiten que ocurra el problema?
 - iii. ¿Qué otros problemas rodean la ocurrencia del problema central?

Durante esta etapa, identifique tantos factores causales como sea posible. Con demasiada frecuencia, las personas identifican uno o dos factores y luego se detienen, pero eso no es suficiente. Con RCA, no desea tratar simplemente las causas más obvias: desea profundizar.

Análisis de Causa Raíz - Paso 3

Use estas herramientas para ayudar a identificar factores causales:

- 5 porqués - ¿Pregunta porque?" hasta llegar a la raíz del problema.
- Profundizar - Divida un problema en partes pequeñas y detalladas para comprender mejor el panorama general.
- Apreciación - Usa los hechos y pregunta "¿Y qué?" para determinar todas las posibles consecuencias de un hecho.
- Diagramas de causa y efecto - Cree una tabla de todos los posibles factores causales, para ver dónde puede haber comenzado el problema.

Análisis de Causa Raíz - Paso 4

4. Identifique la (s) causa (s) raíz (s)
 - a. Responde a las preguntas:
 - i. ¿Por qué existe el factor causal?
 - ii. ¿Cuál es la verdadera razón por la que ocurrió el problema?

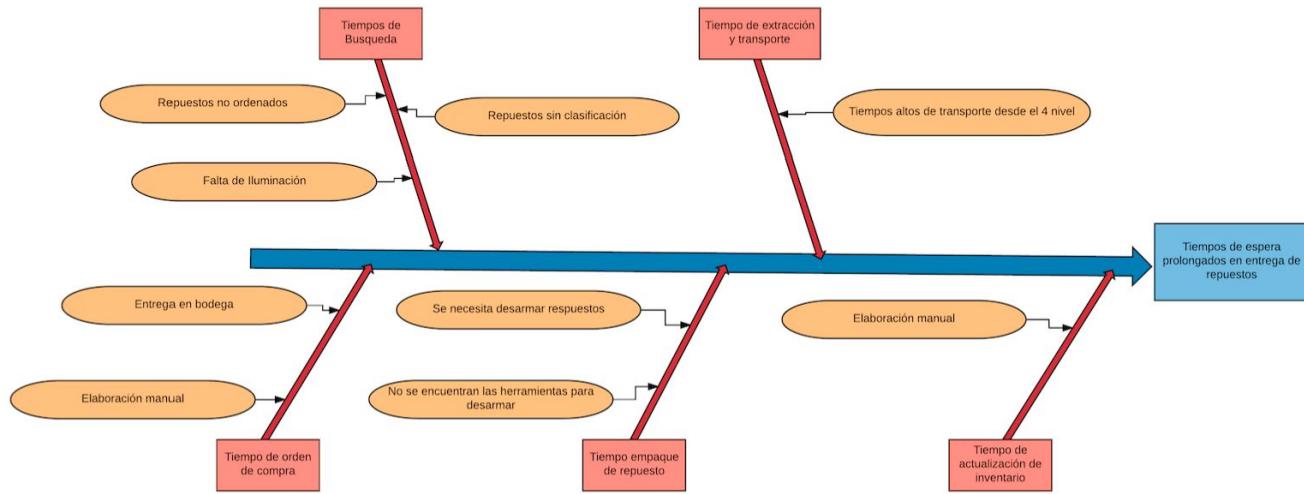
Use las mismas herramientas que usó para identificar los factores causales (*paso tres*) para observar las raíces de cada factor. Estas herramientas están diseñadas para alentarlo a profundizar en cada nivel de causa y efecto.

Análisis de Causa Raíz - Paso 5

5. Recomendación e implementar soluciones
 - a. Responde a las preguntas:
 - i. ¿Qué puede hacer para evitar que el problema vuelva a ocurrir?
 - ii. ¿Cómo se implementará la solución?
 - iii. ¿Quién será responsable de ello?
 - iv. ¿Cuáles son los riesgos de implementar la solución?

Analice su proceso de causa y efecto e identifique los cambios necesarios para varios sistemas. También es importante que planifique con anticipación para predecir los efectos de su solución. De esta manera, puede detectar posibles fallas antes de que sucedan.

Ejemplo:



Ejemplo:

Utilizando 5 Porqués

Arquitectura Web

Herramientas

Herramientas Velocidad - Página Web

The screenshot shows the homepage of PageSpeed Insights. At the top, there's a navigation bar with the PageSpeed Insights logo, a "HOME" button (which is highlighted in blue), and a "DOCS" button. Below the navigation is a large blue header section with the text "Mejora la velocidad de tus páginas web en todos los dispositivos". Underneath this, there's a search bar containing the placeholder "Escribe una URL de página web" and a blue "ANALIZAR" button.

Novedades

Read the latest [Google Webmaster posts](#) about performance & speed.

Enviar comentarios

¿Tienes alguna pregunta concreta y rápida sobre PageSpeed Insights? Hazla en [Stack Overflow](#). Si lo que quieres es enviar comentarios más generales, crea una conversación en nuestra [lista de distribución](#).

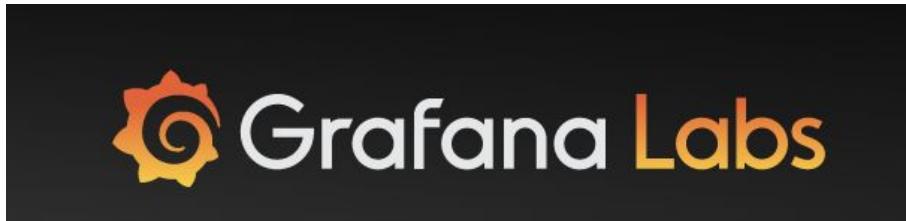
Rendimiento web

Más información sobre [herramientas de rendimiento web de Google](#).

Sobre PageSpeed Insights

PageSpeed Insights analiza el contenido de una página web y, a continuación, genera sugerencias para mejorar la velocidad de las páginas. [Más información](#).

Herramientas Monitoreo



splunk> IT SECURITY IoT BUSINESS ANALYTICS WHY SPLUNK? EXPLORE ▾

Security Risk Mitigation Security Analysis and Response Compliance Use Cases ▾

USE CASES

Security Monitoring

Never miss a gap in your security posture with Splunk's flexible out-of-the-box or customizable correlations, searches and visualizations of all your data. See your full environment with real-time monitoring and harness the power of a single truth.

Try the Sandbox Watch the Video

A diagram illustrating the shift from traditional security monitoring to expanded visibility. It features two overlapping semi-circles. The inner circle is labeled "TRADITIONAL" and contains icons for a shield, a magnifying glass, and a checkmark. The outer circle is labeled "EXPANDED VISIBILITY" and contains a wider range of icons including a triangle, a document, a gear, a computer monitor, and a cloud, symbolizing a more comprehensive monitoring scope.

Herramientas - Test de Stress



Arquitectura Web

Ejercicio - Análisis de casos

Pagina Web

Realice el RCA aplicado a la página de la Universidad utilizando el diagrama de causa y Efecto. Utilice page speed Insights.



Ejercicio

Instale JMeter
Pruebas de stress a google.com
100 usuarios (threads)

Introducción a la Integración de Software

Introducción

Integración Continua



Esta es una práctica a raíz de las metodologías ágiles, en donde el software se empezó a automatizar, debido a los tiempos de implementación de entrega continua. Esto dio nacimiento a la integración continua.

La integración continua es una práctica de desarrollo de software en la que los desarrolladores fusionan regularmente sus cambios de código en un repositorio central, después de lo cual se ejecutan compilaciones y pruebas automatizadas.

Integración Continua

La integración continua se refiere con mayor frecuencia a la etapa de compilación o integración del proceso de lanzamiento de software e implica tanto un componente de automatización como un componente cultural del equipo de desarrollo.

Los objetivos clave de la integración continua son:

- encontrar y abordar los errores más rápido,
- mejorar la calidad del software y
- reducir el tiempo que lleva validar y
- lanzar nuevas actualizaciones de software.

Integración Continua



En el pasado, los desarrolladores de un equipo podían trabajar de forma aislada durante un período prolongado de tiempo y solo fusionar sus cambios en la rama maestra una vez que se completaba su trabajo.

Esto hizo que los cambios de código de fusión fueran difíciles y llevaran mucho tiempo, y también resultó en la acumulación de errores durante mucho tiempo sin corrección.

Estos factores dificultaron la entrega rápida de actualizaciones a los clientes.

Integración Continua - ¿Comó Funciona?

Con una integración continua, los desarrolladores frecuentemente se comprometen a un repositorio compartido utilizando un sistema de control de versiones como Git.

Antes de cada confirmación, los desarrolladores pueden elegir ejecutar pruebas unitarias en su código como una capa de verificación adicional antes de la integración.

Un servicio de integración continua crea y ejecuta automáticamente pruebas unitarias en los nuevos cambios de código para detectar inmediatamente cualquier error.

Integración Continua - ¿Comó Funciona?

Con la entrega continua , los cambios de código se crean, prueban y preparan automáticamente para su lanzamiento a producción.

La entrega continua se expande tras la integración continua mediante la implementación de todos los cambios de código en un entorno de prueba y/o en un entorno de producción después de la etapa de desarrollo.

Integración continua - Beneficios

Buscar y solucionar errores más rápido:

Con pruebas más frecuentes, su equipo puede descubrir y abordar errores antes de que se conviertan en problemas más grandes más adelante.



Entregar actualizaciones más rápido:

La integración continua ayuda a su equipo a entregar actualizaciones a sus clientes de manera más rápida y frecuente.

Mejora la productividad del desarrollador: debido a que ayuda a su equipo a ser más productivo al liberar a los desarrolladores de las tareas manuales y al fomentar comportamientos que ayudan a reducir la cantidad de errores y errores lanzados a los clientes.

Introducción a la Integración de Software

Aplicaciones Distribuidas

Aplicaciones Distribuidas



Son aplicaciones o software que se ejecutan en varias computadoras dentro de una red al mismo tiempo y se pueden almacenar en servidores o con computación en la nube.

A diferencia de las aplicaciones tradicionales que se ejecutan en un solo sistema, las aplicaciones distribuidas se ejecutan en múltiples sistemas simultáneamente para una sola tarea o trabajo.

Aplicaciones Distribuidas

Estas pueden comunicarse con múltiples servidores o dispositivos en la misma red desde cualquier ubicación geográfica.

La naturaleza distribuida de las aplicaciones se refiere a la distribución de datos en más de una computadora en una red.

Las aplicaciones distribuidas se dividen en dos programas separados:

1. Software del cliente
2. Software del servidor.

Aplicaciones Distribuidas

La diferencia reside en que el software o la computadora del cliente accede a los datos desde el servidor o cloud, mientras que el servidor o cloud se encarga de procesar los datos.

Si un componente de aplicación distribuida se cae, puede conmutar por error a otro componente para continuar ejecutándose.

Las aplicaciones distribuidas permiten que múltiples usuarios accedan a las aplicaciones a la vez.

Aplicaciones Distribuidas

Muchos desarrolladores, profesionales de TI o empresas eligen almacenar aplicaciones distribuidas en la nube debido a la elasticidad y escalabilidad de la nube , así como a su capacidad para manejar grandes aplicaciones o cargas de trabajo.

Las empresas pueden optar por utilizar tecnología de contenedores, como Docker , para empaquetar e implementar aplicaciones distribuidas. Los contenedores pueden construir y ejecutar aplicaciones distribuidas, así como aplicaciones distribuidas separadas de otras aplicaciones en una nube o infraestructura compartida.

Aplicaciones Distribuidas

Existen seis características principales responsables de la utilidad de los sistemas distribuidos:

1. Compartición de recursos.
2. Apertura (openness).
3. Concurrencia.
4. Escalabilidad
5. Tolerancia de Fallos.
6. Transparencia.

Aplicaciones Distribuidas - Compartir recursos

Un sistema distribuido permite compartir recursos hardware y software (discos, impresoras, ficheros y compiladores) que se asocian con Computadores de una red.

El término recurso es el que mejor abarca toda la variedad de entidades que pueden compartirse en un sistema distribuido incluyendo componentes hardware como discos e impresoras hasta elementos software como ficheros, ventanas, bases de datos y otros objetos de datos.

La idea de compartición de recursos no es nueva ni aparece en el marco de los sistemas distribuidos.

Aplicaciones Distribuidas - Compartir recursos

Los sistemas multiusuario clásicos prevén compartir recursos entre sus usuarios aunque esta acción se hace de manera natural entre todos sus usuarios.

Los usuarios de estaciones de trabajo monousuario o computadoras personales dentro de un sistema distribuido no obtienen automáticamente los beneficios de la compartición de recursos.

Los recursos en un sistema distribuido están físicamente encapsulados en una de los Computadores y sólo pueden ser accedidos por otros equipos mediante las comunicaciones en red.

Para que la compartición de recursos sea efectiva debe ser manejada por un programa que ofrezca un interfaz de comunicación permitiendo que el recurso sea accedido, manipulado y actualizado de una manera fiable y consistente.

Todo esto lleva a la definición de gestor de recursos.

Aplicaciones Distribuidas - Apertura

Los sistemas distribuidos son normalmente sistemas abiertos lo que significa que se diseñan sistemas utilizando protocolos estándar que permiten combinar hardware y software de diferentes fabricantes.

Un sistema informático es abierto si el sistema puede ser extendido de diversas maneras. Un sistema puede ser abierto o cerrado con respecto a extensiones hardware (añadir periféricos, memoria o interfaces de comunicación, etc.) o con respecto a las extensiones software (añadir características al sistema operativo, protocolos de comunicación y servicios de compartición de recursos, etc.).

Aplicaciones Distribuidas - Apertura

Básicamente los sistemas distribuidos deben cumplir una serie de características:

1. Los interfaces software clave del sistema están claramente especificados y se ponen a disposición de los desarrolladores. Los interfaces se hacen públicos.
2. Los sistemas distribuidos abiertos se basan en la provisión de un mecanismo uniforme de comunicación entre procesos e interfaces publicados para acceder a recursos compartidos.
3. Los sistemas distribuidos abiertos pueden construirse a partir de hardware y software heterogéneo, posiblemente proveniente de vendedores diferentes.

La conformidad de cada componente con el estándar publicado debe ser cuidadosamente comprobada y certificada si se quiere evitar tener problemas de integración.

Aplicaciones Distribuidas - Conurrencia

Estos procesos pueden (aunque no es obligatorio) comunicarse con otros durante su funcionamiento normal. Cuando existen varios procesos en una única máquina decimos que se están ejecutando concurrentemente.

Si la computadora está equipada con un único procesador central la concurrencia tiene lugar entrelazando la ejecución de los distintos procesos. Si la computadora tiene N procesadores entonces se pueden estar ejecutando estrictamente a la vez hasta N procesos.

En los sistemas distribuidos hay muchas máquinas, cada una con uno o más procesadores centrales. Es decir, si hay M Computadoras en un sistema distribuido con un procesador central cada una entonces hasta M procesos estar ejecutándose en paralelo.

Aplicaciones Distribuidas - Conurrencia

En un sistema distribuido que está basado en el modelo de compartición de recursos, la posibilidad de ejecución paralela ocurre por dos razones:

1. Muchos usuarios interactúan simultáneamente con programas de aplicación: Este caso es el menos conflictivo porque normalmente las aplicaciones de interacción se ejecutan aisladamente en la estación de trabajo del usuario y no entran en conflicto con las aplicaciones ejecutadas en las estaciones de trabajo de otros usuarios.
2. Muchos procesos servidores se ejecutan concurrentemente, cada uno respondiendo a diferentes peticiones de los procesos clientes: Esta situación se produce por la existencia de uno o más servidores para cada tipo de recurso. Estos procesos servidores se ejecutan en distintas máquinas, de manera que se están ejecutando en paralelo diversos servidores, junto con diversos programas de aplicación .

Aplicaciones Distribuidas - Conurrencia

Las peticiones para acceder a los recursos de un servidor dado son puestas en las colas de trabajo del servidor y son procesadas secuencialmente o procesadas concurrentemente por múltiples instancias del gestor de recursos.

Cuando esto ocurre los procesos servidores deben sincronizar sus acciones para asegurarse de que no existen conflictos.

La sincronización debe ser cuidadosamente planeada para asegurar que no se pierden los beneficios de la concurrencia.

Aplicaciones Distribuidas - Escalabilidad

Los sistemas distribuidos son escalables en tanto que la capacidad del sistema puede incrementarse añadiendo nuevos recursos para cubrir nuevas demandas sobre el sistema.

Los sistemas distribuidos operan de manera efectiva y eficiente a muchas escalas diferentes. La escala más pequeña consiste en dos estaciones de trabajo y un servidor de archivos, mientras que un sistema distribuido construido alrededor de una red de área local simple podría contener varios cientos de estaciones de trabajo, varios servidores de archivos, servidores de impresión y otros servidores de propósito específico.

Aplicaciones Distribuidas - Escalabilidad

A menudo se conectan varias redes de área local para formar internetworks, y éstas podrían contener muchos miles de Computadores que forman un único sistema distribuido, permitiendo que los recursos sean compartidos entre todos ellos.

Tanto el software de sistema como el de aplicación no deberían cambiar cuando la escala del sistema se incrementa.

La necesidad de escalabilidad no es solo un problema de prestaciones de red o de hardwares que está íntimamente ligada con todos los aspectos del diseño de los sistemas distribuidos.

Aplicaciones Distribuidas - Escalabilidad

El diseño del sistema debe reconocer explícitamente la necesidad de escalabilidad o de lo contrario aparecerán serias limitaciones. La demanda de escalabilidad en los sistemas distribuidos ha conducido a una filosofía de diseño en que cualquier recurso simple -hardware o software- puede extenderse para proporcionar servicio a tantos usuarios como se quiera. Si la demanda de un recurso crece, debería ser posible extender el sistema para darla servicio.

El trabajo necesario para procesar una petición simple para acceder a un recurso compartido debería ser prácticamente independiente del tamaño de la red. Las técnicas necesarias para conseguir estos objetivos incluyen el uso de datos replicados, la técnica asociada de caching, y el uso de múltiples servidores para manejar ciertas tareas, aprovechando la concurrencia para permitir una mayor productividad.

Aplicaciones Distribuidas - Tolerancia a fallos

Los sistemas distribuidos pueden ser tolerantes a algunos fallos de funcionamiento del hardware y del software.

En la mayoría de los sistemas distribuidos se puede proporcionar un servicio degradado cuando ocurren fallos de funcionamiento. Realmente una completa pérdida de servicio sólo ocurre cuando existe un fallo de funcionamiento en la red.

Los sistemas informáticos a veces fallan. Cuando se producen fallos en el software o en el hardware, los programas podrían producir resultados incorrectos o podrían pararse antes de terminar la operación que estaban realizando.

Aplicaciones Distribuidas - Tolerancia a fallos

El diseño de sistemas tolerantes a fallos se basa en dos soluciones:

1. Redundancia hardware: uso de componentes redundantes.
2. Recuperación del software: diseño de programas que sean capaces de recuperarse de los fallos.

En los sistemas distribuidos pueden replicarse los servidores individuales que son esenciales para la operación continuada de aplicaciones críticas.

Aplicaciones Distribuidas - Tolerancia a fallos

La recuperación del software tiene relación con el diseño de software que sea capaz de recuperar (roll-back) el estado de los datos permanentes antes de que se produjera el fallo.

Los sistemas distribuidos también proveen un alto grado de disponibilidad en la vertiente de fallos hardware.

La disponibilidad de un sistema es una medida de la proporción de tiempo que está disponible para su uso.

Aplicaciones Distribuidas - Transparencia

La transparencia se define como ocultar al usuario y al programador de aplicaciones de la separación de los componentes de un sistema distribuido, de manera que el sistema se percibe como un todo, en vez de una colección de componentes independientes.

La transparencia ejerce una gran influencia en el diseño del software de sistema. Estas proveen un resumen útil de la motivación y metas de los sistemas distribuidos.

Aplicaciones Distribuidas - Transparencia

Los tipos de transparencias definidas son:

1. **Transparencia de Acceso:** Permite el acceso a los objetos de información remotos de la misma forma que a los objetos de información locales.
2. **Transparencia de Localización:** Permite el acceso a los objetos de información sin conocimiento de su localización
3. **Transparencia de Concurrencia:** Permite que varios procesos operen concurrentemente utilizando objetos de información compartidos y de forma que no exista interferencia entre ellos.

Aplicaciones Distribuidas - Transparencia

- 
- 4. Transparencia de Replicación:** Permite utilizar múltiples instancias de los objetos de información para incrementar la fiabilidad y las prestaciones sin que los usuarios o los programas de aplicación tengan que conocer la existencia de las réplicas.
 - 5. Transparencia de Fallos:** Permite a los usuarios y programas de aplicación completar sus tareas a pesar de la ocurrencia de fallos en el hardware o en el software.
 - 6. Transparencia de Migración:** Permite el movimiento de objetos de información dentro de un sistema sin afectar a los usuarios o a los programas de aplicación.

Aplicaciones Distribuidas - Transparencia

7. Transparencia de Prestaciones: Permite que el sistema sea reconfigurado para mejorar las prestaciones mientras la carga varía.

8. Transparencia de Escalado: Permite la expansión del sistema y de las aplicaciones sin cambiar la estructura del sistema o los algoritmos de la aplicación.

Las dos más importantes son las transparencias de acceso y de localización; su presencia o ausencia afecta fuertemente a la utilización de los recursos distribuidos.

A menudo se las denomina a ambas transparencias de red. La transparencia de red provee un grado similar de anonimato en los recursos al que se encuentra en los sistemas centralizados.

Aplicaciones Distribuidas - Seguridad

Puede accederse al sistema desde varias computadoras diferentes, y el tráfico en la red puede estar sujeto a hackers indeseables..

Esto hace más difícil el asegurar que la integridad de los datos en el sistema se mantenga y que los servicios del sistema no se degraden por ataques de denegación de servicio.

Aplicaciones Distribuidas - Manejabilidad

Las computadoras en un sistema pueden ser de diferentes tipos y pueden ejecutar versiones diferentes de sistemas operativos.

Los defectos en una máquina pueden propagarse a otras máquinas con consecuencias inesperadas.

Esto significa que se requiere más esfuerzo para gestionar y mantener el funcionamiento del sistema.



Ejercicio

Instale Alguna herramienta para monitorear contenedores de Docker

Introducción a la Integración de Software

Aplicaciones Integradas



Aplicaciones Integradas

Los sistemas de aplicaciones integrados son sistemas de infraestructura pre-integrados con bases de datos, software de aplicaciones o ambos, que proporcionan una funcionalidad similar a un dispositivo para soluciones de Big Data, plataformas analíticas o demandas similares.



Aplicaciones Integradas

Existen 4 razones clave para considerar la entrega de sus soluciones en forma de sistemas de aplicaciones integradas:

Mayor coherencia de la solución: en un sistema de aplicación integrado, integra el hardware y el software para su producto, de modo que controla el entorno que lo respalda. Eso garantiza que sus aplicaciones y análisis tengan todos los recursos de procesamiento, almacenamiento y memoria que necesitan para ofrecer un rendimiento óptimo en trabajos de computación intensiva. En resumen, su aplicación se ejecuta de la forma en que fue diseñada, por lo que se optimiza la satisfacción del cliente.



Aplicaciones Integradas

Mejor seguridad del sistema: los análisis y otros sistemas, con frecuencia manejan información financiera, médica u otra información patentada. Al entregar su producto como un sistema de aplicación integrado, puede incorporar las herramientas de seguridad necesarias para evitar el acceso de usuarios no autorizados, piratas informáticos u otros intrusos. Su aplicación es más segura, por lo que sus clientes ganan confianza en sus productos.



Aplicaciones Integradas

Ciclo más rápido desde el desarrollo de la solución hasta la implementación: con una aplicación tradicional, entregada solo como software, sus ingenieros pasan mucho tiempo considerando posibles escenarios de implementación del sistema de análisis y descubriendo cómo maximizar el rendimiento del sistema en cada uno. Eso puede retrasar que su producto llegue a los clientes por meses o más. Pero con un sistema de aplicación integrado, puede especificar el entorno una vez y llevar su producto al campo sin demoras innecesarias.



Aplicaciones Integradas

Reduce la carga de soporte para la solución: un beneficio adicional de la coherencia de la solución es la facilidad de soporte para sus soluciones y análisis. Cuando vende software solo, su personal de soporte nunca puede estar seguro exactamente de qué hardware se está ejecutando, qué más se está ejecutando en la plataforma y una gran cantidad de otras variables. Con un sistema de aplicación integrado, toda esa información está bajo su control, por lo que replicar y resolver problemas se vuelve simple y directo.



Ejercicio

Realice una tabla de comparación entre aplicaciones distribuidas y integradas