Ingeniería del Software 2

Arquitectura de Software

Definición

- Se conoce como la disciplina que estudia el armado de aplicaciones de software.
- Permite diseñar y estudiar al software más allá de los algoritmos y estructuras de datos que lo componen.
- Estudia todos los componentes de una aplicación y la forma en que ellos se combinan.
- Existe el rol de Arquitecto de Software, el cual es el encargado de velar por un buen diseño y una buena arquitectura de la o las aplicaciones.

Definición

• Antiguamente los equipos de desarrollo se componían de uno o varios desarrolladores trabajando juntos.

• A medida que el número de desarrolladores aumentó, se hizo necesaria la presencia de un líder técnico en el equipo.

• A medida que se cuenta con varios equipos de desarrollo, se hace necesaria la presencia de un rol que vele por la integridad técnica y las buenas prácticas de desarrollo.

• Este rol se conoce como ***Arquitecto de software*** en sus responsabilidades se encuentran:
  - Asegurar que la estrategia de IT y de Negocio de la compañía se encuentran alineadas.
  - Participar de forma activa en el diseño, mantenimiento y mejora continua del desarrollo de software.
  - Desarrollar y mantener políticas y estándares dentro de la compañía.
  - Identificar y gestionar riesgos técnicos.
  - Identificar y analizar tecnología que permita el crecimiento de la empresa.

Requerimientos de Negocio

- Son necesidades que la aplicación debe cumplir para sus interesados.

- Generalmente son separados en 3 tipos:
  - **De Visión:** relacionados al futuro de la aplicación.
  - **Funcionales:** definen lo que la aplicación debe ser capaz de hacer.
  - **No Funcionales:** no son definidos por el usuario, pero son necesarios para que la aplicación funcione.

- Para identificarlos es necesario:
  - Definir los objetos principales del negocio.
  - Mantener una buena comunicación con los interesados en la aplicación.
  - Aprender a priorizarlos.
  - Analizando competidores con productos o servicios similares.
  - Tener en cuenta legislaciones y normas vigentes (PCI-DSS por ejemplo).

Atributos de Calidad

- Son necesarios para que la aplicación funcione.

- Algunos pueden deducirse de forma directa, ya que son especificados por el cliente, mientras que otros deben analizarse en más detalle.

- ***Relacionados al negocio***:
  - *Adaptabilidad:* el sistema debe poder adaptarse y reaccionar a los cambios solicitados.
  - *Extensibilidad:* la aplicación debe soportar de una forma simple añadir nuevos componentes.
  - *Compatibilidad:* debe ser compatible con regulaciones relacionadas al negocio.
  - *Usabilidad:* relacionada a la facilidad de uso del sistema.

- ***Relacionados a la Seguridad:***
  - *Confidencialidad:* la información sensible no debe ser accedida por usuarios no autorizados.
  - *Integridad:* la información debe ser consistente en todo momento en la aplicación.
  - *Disponibilidad:* la información debe estar disponible siempre que se considere necesario.
  - *Responsabilidad:* deben registrarse las interacciones con la información, como así también los usuarios intervinientes en los cambios de la misma.

Atributos de Calidad

- ***Relacionados a la Performance***:
  - *Escalabilidad:* habilidad de la aplicación para soportar más cargas de trabajo con un uso de recursos proporcionado. Debe ser fácil añadir y quitar recursos.
  - *Responsividad:* la aplicación debe responder a las interacciones de usuarios de una furma fluida y sin demoras.
  - *Throughput:* generalmente se mide como un número de usuarios, operaciones o acciones que el sistema debe soportar sin degradar su performance.

- ***Relacionados a la Información y Configuración:***
  - *Interoperabilidad:* define con que facilidad el sistema interactúa con otros sistemas.
  - *Durabilidad:* define la capacidad de que cualquier información crítica sea almacenada bajo cualquier circunstancia.

- ***Otros atributos de Calidad***:
  - *Monitoreo:* define que tan fácil es medir el estado de salud de una aplicación y su consumo de recursos, generalmente en tiempo real.
  - *Testeabilidad:* define la habilidad de probar la aplicación en varios niveles, generalmente incluyendo pruebas automatizadas.

Atributos de Calidad

Aplicación Bancaria:

- Seguridad
- Performance
- Monitoreo

Un blog o sitio de contenido:

- Usabilidad
- Responsividad
- Troughput

Dinámicas de Equipo

- ***Con el equipo de desarrollo:***
  - Revisar decisiones de arquitectura con el equipo y estar seguro de sus motivaciones.
  - Ser asertivo, no tener miedo de discusiones y conflictos y aprender a manejarlos.
  - Ser abierto a críticas y considerar las opiniones y aportes técnicos del equipo.
  - Mantener una buena relación, no ser considerado como un problema.
  - Delegar las tareas en la medida en que sea posible.

- ***Con los interesados o dueños del producto:***
  - Comunicarse de una forma abierta sin importar el nivel técnico.
  - Saber presentar y comunicar ideas de forma sencilla.
  - Contar con habilidades de negociación para las fases de desarrollo, plazos y costos.
  - Mantener un enfoque orientado a resultados.

Patrones de Arquitectura de Software

• Son conocidos como soluciones reusables a problemas que ocurren frecuentemente en arquitecturas de software.

• Son agnóstico a la tecnología, es decir que pueden aplicarse utilizando diferente tecnologías.

• Son considerados como una abstracción generalizada de los patrones de diseño (que veremos posteriormente).

• En una aplicación puede utilizarse más de un patrón.

• A continuación veremos algunos patrones de arquitectura, en conjunto con sus beneficios y donde deben ser utilizados.

Arquitectura en Capas

- Describe una aplicación basada en capas, en donde cada una de ellas cumple un rol en particular y tiene una responsabilidad bien definida.

- Generalmente se implementa un modelo de tres capas:
  - ***Capa de Presentación***
  - ***Capa de Negocio***
  - ***Capa de Acceso a Datos***

- A medida que aumenta la complejidad de un sistema, pueden agregarse capas.

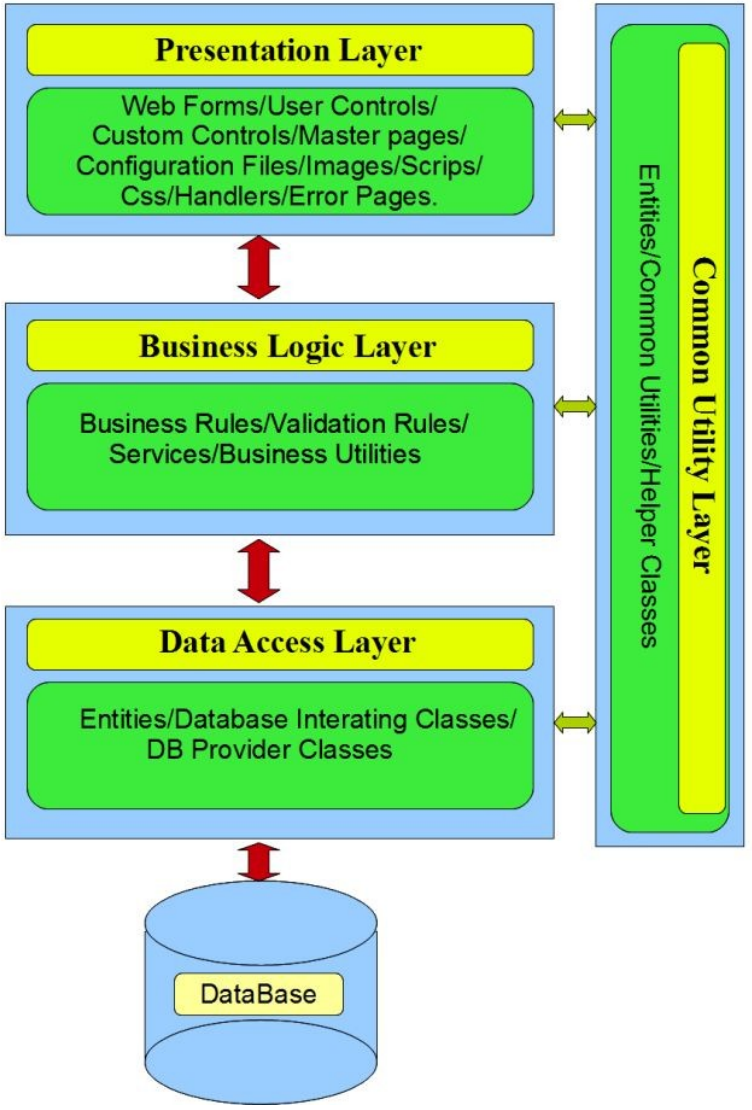
- La responsabilidad de un componente es clara, de acuerdo en que capa se ubica.

- Cada capa deben ser accedidas en un orden en particular y se encuentran separadas y encapsuladas del resto.

- Ventajas:
  - Permite definir claramente la responsabilidad de cada componente.
  - Son fáciles de desarrollar y probar.
  - Es un patrón de buen uso para propósitos generales.

Arquitectura en Capas

- Desventajas:
  - El contar con muchas capas agrega complejidad.
  - Generalmente conllevan a desarrollar aplicaciones monolíticas.
  - El acoplamiento de las capas conlleva a problemas de dependencia y escalabilidad.

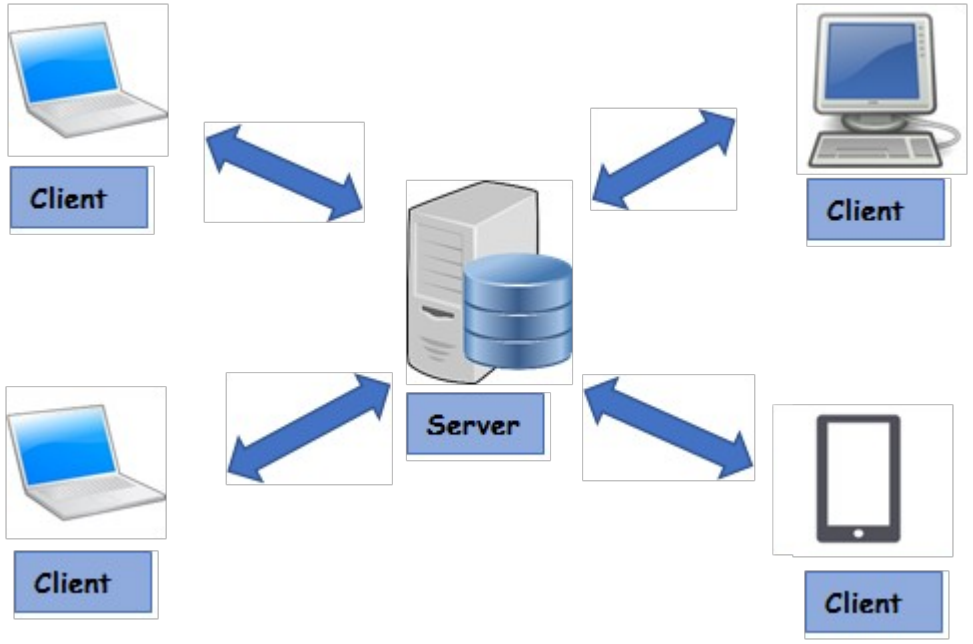


Arquitectura Cliente Servidor

- Es una arquitectura centralizada, en donde hay servidores que ofrecen servicios que los clientes utilizan.
- Generalmente los clientes residen en un lugar distinto que el servidor.
- Es un patrón muy utilizado por las aplicaciones web y móviles.
- Generalmente el servidor cuenta con un mayor procesamiento que los clientes.
- Pueden contarse clientes livianos o pesados según el nivel de procesamiento que provean.
- Ventajas:
  - Promueve la seguridad centralizada.
  - Simplifica los procesamientos de backup y restauración de la información.
  - Permite una administración centralizada.
  - Facilita la actualización y la escalabilidad de las aplicaciones.

Arquitectura Cliente Servidor

- Desventajas:
  - Contienen un simple punto de falla. Esto puede mitigarse pero requiere un costo de implementación mayor.
  - Traen problemas de infraestructura asociados, como problemas de conectividad.



Arquitectura Modelo Vista Controlador (MVC)

• Se basa en dividir la aplicación en tres partes: ***Modelo, Vista y Controlador***.

• Es una arquitectura basada puramente en la interfaz de usuario.

• ***Modelo:*** maneja los datos y la lógica de la aplicación.

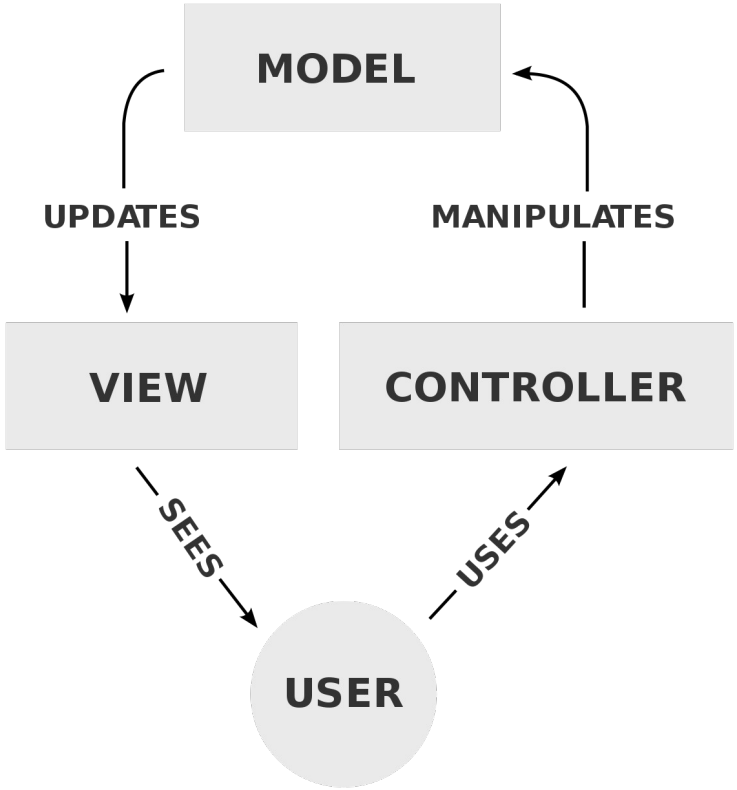
• ***Vista:*** maneja la representación visual de la información y gestiona los eventos del usuario. Puede existir más de una vista para la aplicación.

• ***Controlador:*** reacciona ante eventos en la vista o en el modelo y actualiza la parte consecuente.

• Ventajas:
- Permite bajo acoplamiento y alta cohesión entre componentes.
- Permite el desarrollo en paralelo.
- Permite la reutilización de código.

Arquitectura Modelo Vista Controlador (MVC)

- Desventajas:
  - Al principio presentan una curva de aprendizaje muy lenta, sobre todo para desarrolladores no muy experimentados.
  - No es recomendable para aplicaciones de gran tamaño, salvo que sólo se aplique para la capa de presentación de una aplicación.

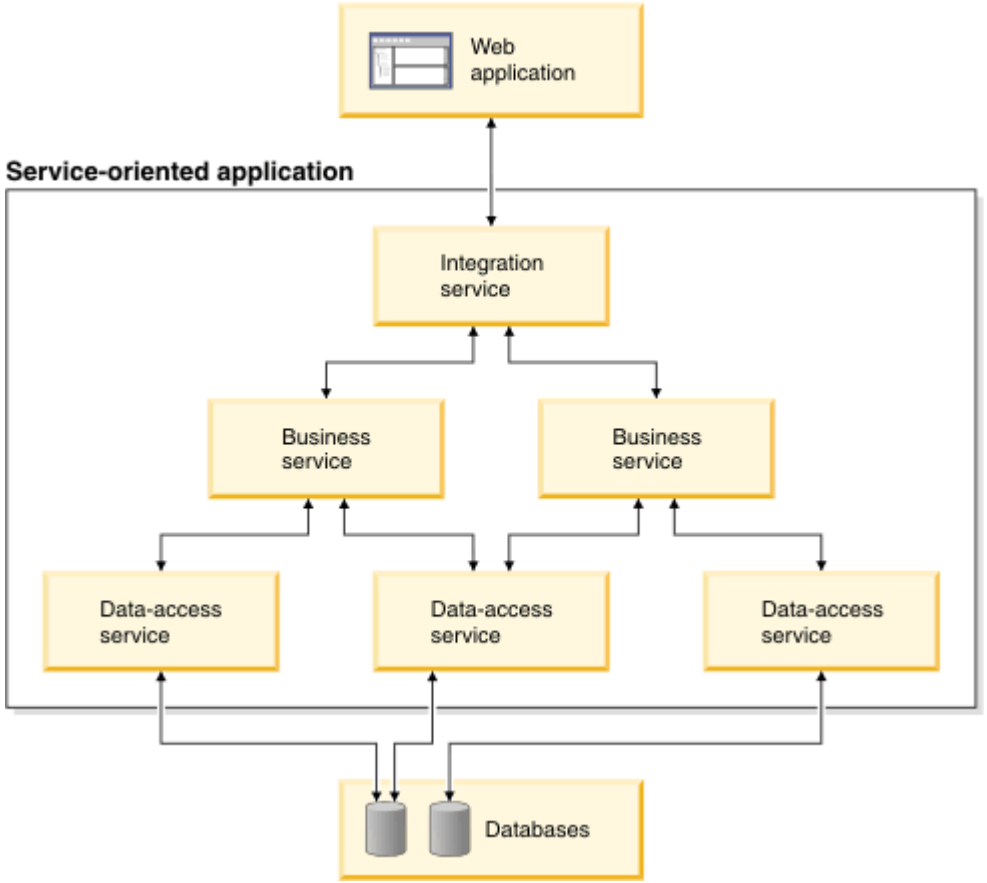


Arquitectura Orientada a Servicios (SOA)

- Se basa en los servicios de una aplicación y en su comunicación.
- Estudia a una aplicación como a un conjunto de servicios muy poco acoplados.
- Un servicio puede ser implementado en diferentes tecnologías, siendo la más popular los Web Services o Servicios Web.
- Cada servicio representa una actividad de negocio en el sistema, con una salida particular.
- Cada servicio se presenta bajo el esquema de caja negra y pueden dividirse en más servicios.
- Ventajas:
  - Permite bajo acoplamiento entre servicios.
  - Permite la reutilización de código y de servicios.
  - Son fácilmente mantenibles y testeables.
  - Escalan fácilmente.
  - Promueven el desarrollo en paralelo.

Arquitectura Orientada a Servicios (SOA)

- Desventajas:
  - La gestión de servicios suele tornarse compleja.
  - El despliegue de nuevos servicios suele tornarse una tarea compleja y costosa.
  - La comunicación entre servicios puede generar una sobrecarga en el sistema.



Arquitectura de Microservicios

- Es una variación de la arquitectura de SOA, en donde una aplicación se conforma de muchos servicios de poco tamaño.

- Cada servicio tiene un objetivo y alcance muy bien definido y son mucho más livianos.

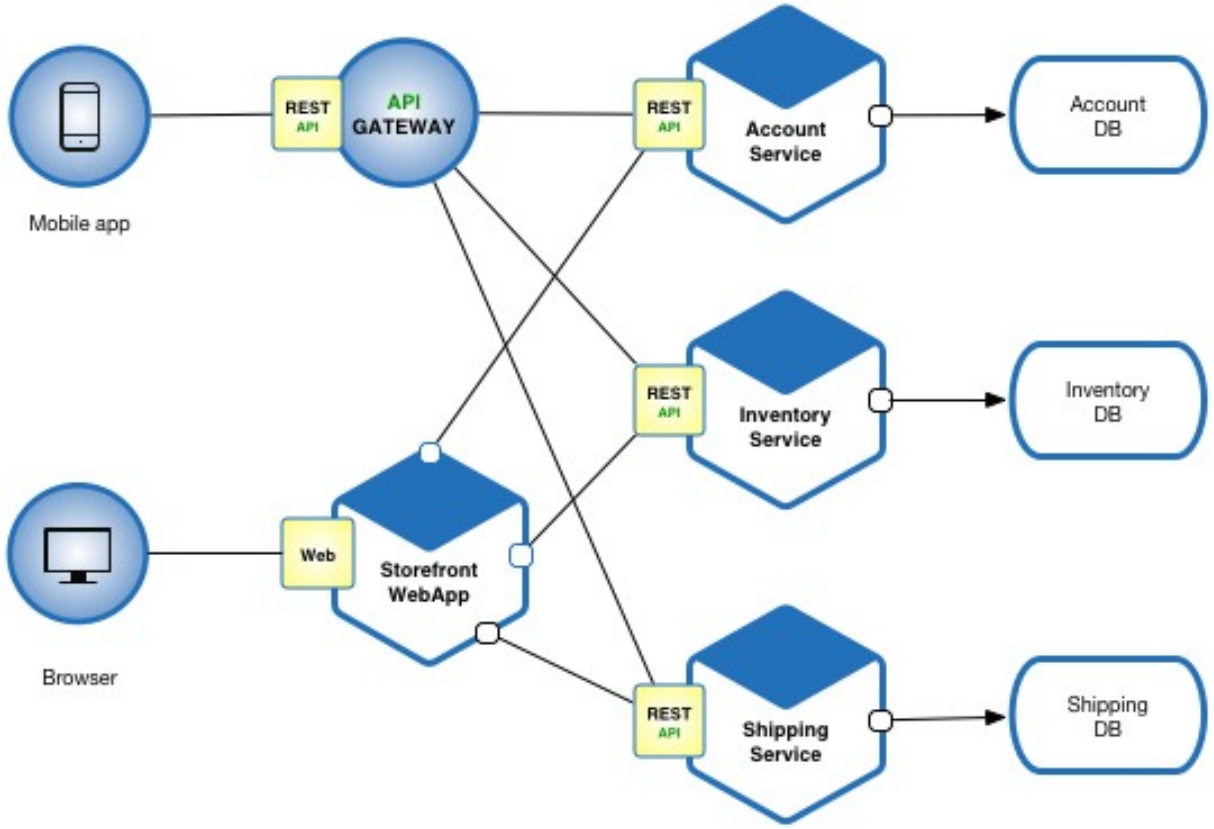
- Los microservicios pueden comunicarse de distintas formas, ya sea utilizando una API Web o usando memoria compartida o un servicio de Mensajería.

- Es una estilo de arquitectura que se ha vuelto muy popular y recomendado para aplicaciones de mediano o gran tamaño.

- Ventajas:
  - Bajo acoplamiento.
  - Fomentan la modularidad de la aplicación.
  - Escalan fácilmente.
  - Promueven el desarrollo en paralelo.
  - Fomentan la disponibilidad.

Arquitectura de Microservicios

- Desventajas:
  - Suelen traer costos más altos de infraestructura.
  - Las pruebas de integración suelen ser más complejas.
  - La gestión de microservicios suele ser una tarea tediosa y requerir personal adicional.



Arquitectura Basada en Eventos (Event Driven)

- En este caso la arquitectura se basa en los distintos eventos que pueden ocurrir en una aplicación.

- Un evento puede ser definido como un cambio de estado en el sistema, como una compra, el la creación de una cuenta de usuario, entre otros.

- Se definen como ***producers*** a objetos que generalmente generan eventos y como ***consumers*** que esperan a que dichos eventos ocurran y reaccionan.

- Se fomenta el bajo acomplamiento entre producers y consumers, los primeros no tienen idea de que consumidores están esperando sus eventos.

- Generalmente se implementan en modelos de Publish / Subscribe o de Log Writting – Event Stream.

- Ventajas:
  - Funcionan bien en procesamiento en tiempo real de la información.
  - Escalan fácilmente.
  - Promueven el bajo acoplamiento.
  - Puede funcionar como una arquitectura distribuída.

Arquitectura Basada en Eventos (Event Driven)

- Desventajas:
  - Se hace complicado gestionar el orden de ejecución y procesamiento de los eventos.
  - Implementar cambios en los ***Producers*** impacta en todos sus ***Consumers***.