La información brindada en los siguientes puntos es de carácter escaso, por lo cual se espera que el postulante responda teniendo una respuesta general como base e incluyendo detalles que entienda necesarios para acercar una respuesta que tenga información de carácter importante y/o valioso.

Cuando las preguntas o consideraciones le resulten ambiguas puede aclarar el criterio adoptado.

Las respuestas pueden incluir consideraciones de tecnologías especificas a utilizar desde la plataforma al código en cada componente o Sistema.

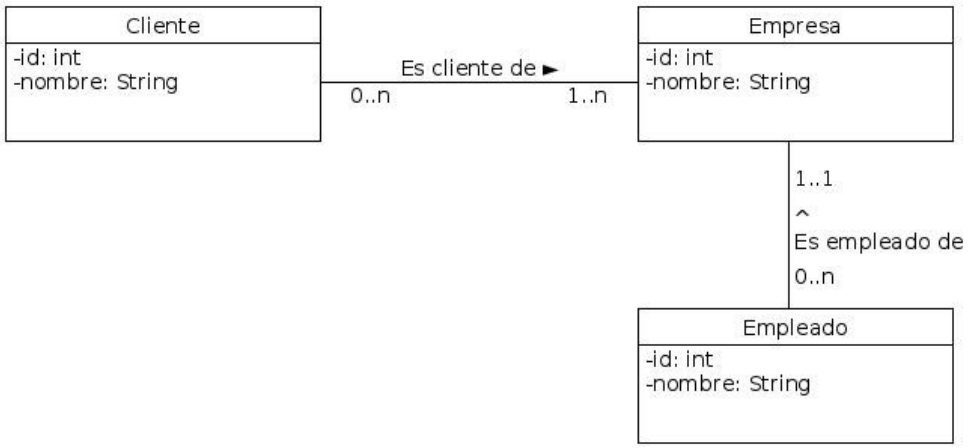
1) Se necesita modelar las interacciones entre los actores y los objetos de un sistema. ¿Qué diagrama utilizaría para dicho fin?

   Utilizaría el diagrama de secuencia de UML, ya que me permite incluir en un mismo diagrama tanto a los actores (usuarios u otros sistemas) y los componentes (objetos o clases) del sistema en cuestión. Una vez incluidos elementos del diagrama, se pueden graficar las interacciones de forma secuencial, pudiendo hacer más de un diagrama con las distintas alternativas, siempre tomando como base el mismo diagrama con los mismos elementos.

2) Realice un diagrama de clases para una aplicación que necesita almacenar información sobre empresas, sus empleados y sus clientes. Considere a su criterio para cada uno de estos ítems los diferentes atributos.

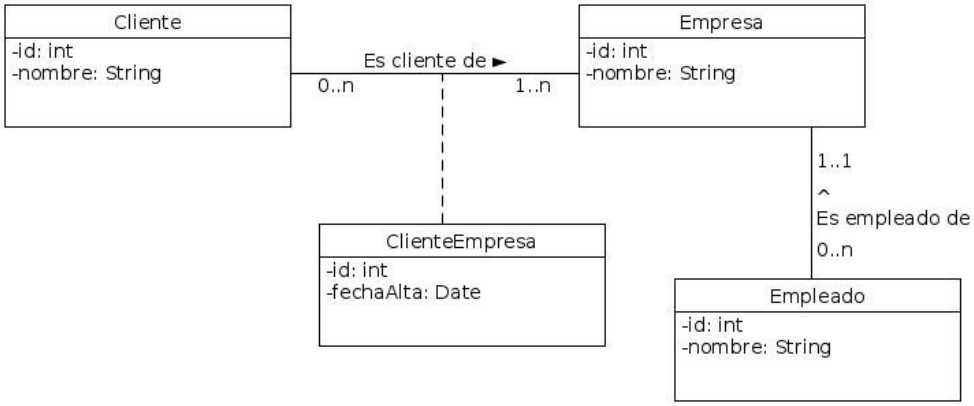
   Se asume que se solicita el diagrama de clases de UML haciendo referencia a las clases persistentes, es decir, a las clases que se relacionan directamente con los objetos en las bases de datos. Solo se incluyen las 3 clases solicitadas, con los atributos minimos (id y nombre) y sus relaciones. No se incluyen metodos ya que no son necesarios para el almacenamiento de los datos.

   Tambien se asume que cada empleado pertenece a una unica empresa, pero que cada cliente puede contratar a varias empresas. La relacion entre los clientes y los empleados es unicamente a traves de las empresas, no se contempla un ejecutivo de cuenta ni nada que relacione a un cliente puntualmente con un empleado.



   La relacion muchos a muchos se da entre Empresa (1..n), que puede estar creada sin tener clientes, y Cliente (0..n), que solo puede generarse cuando el cliente tiene contratada al menos una empresa.

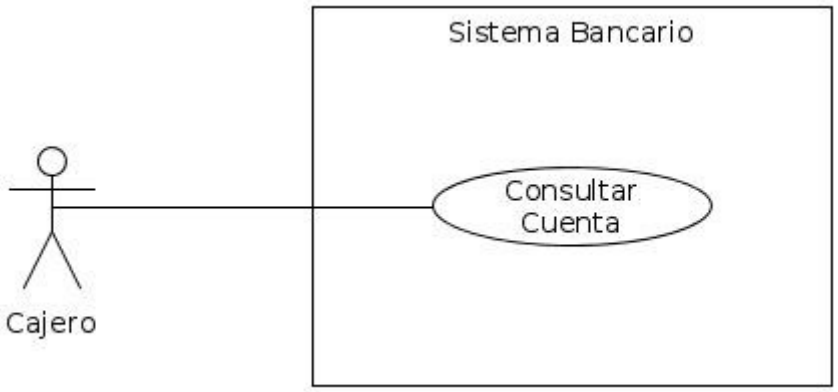
   La relacion uno a muchos se da entre Empresa (1..1), que puede no tener empleados cuando se crea, y Empleado (0..n), ya que varios empleados pueden pertenecer a una empresa.



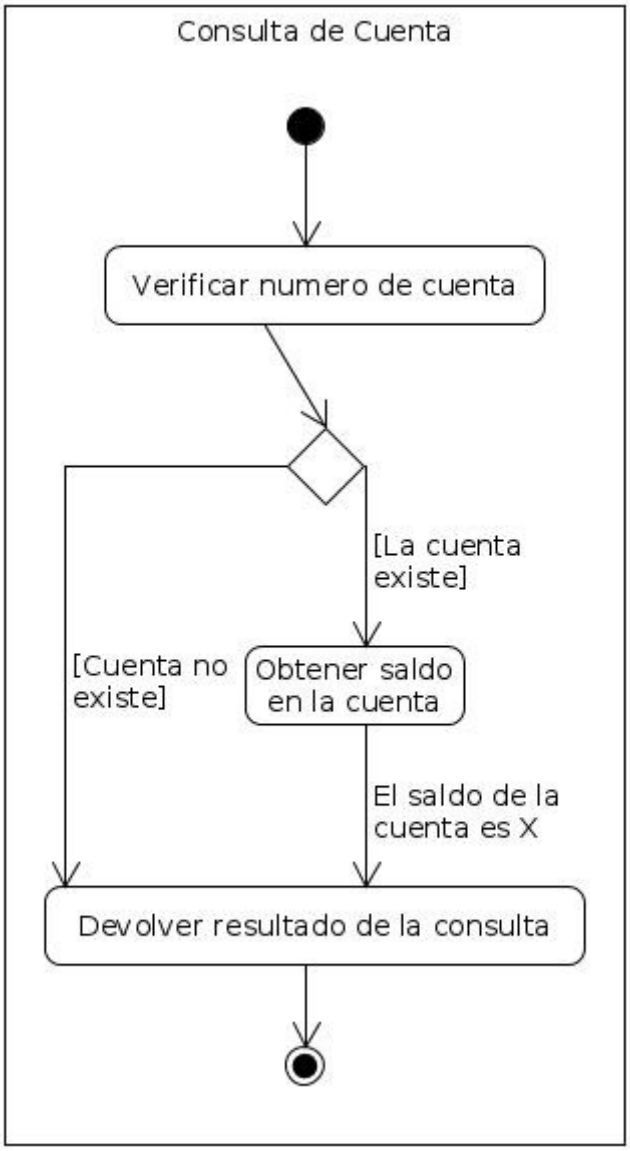
Como alternativa se incluye en el segundo diagrama, una clase extra, que contiene información de la relacion entre el cliente y la empresa. En cualquiera de los dos casos, la entidad tendra lugar en la base de datos, pero depende del negocio, y la relevancia de esa información, que la incluyamos en el diagrama de clases. En este caso, se usa una dato basico, la fecha de alta del cliente con esa empresa, como atributo para tener en cuenta de esa relacion.

3) Explique con ejemplos de diagramas UML lo que serían diagramas de secuencias, diagrama de actividades y diagramas de caso de uso para un Sistema dado.

   El sistema que se usa de ejemplo es el de un banco. En el cual tenemos el caso de uso de un cajero consultando un numero de cuenta. Para ese caso de unos, tenemos los diagramas de secuencia y actividades. Para el diagrama de secuencia, tendremos que hacer 2 casos, el primero en que la cuenta existe, y el alternativo en que la cuenta no existe. Para el diagrama de actividad, podemos hacer un unico diagrama que muestre ambos caminos.



   En el diagrama de casos de uso, tenemos al cajero que ejecuta la accion de consultar una cuenta a traves de su numero.



En el diagrama de actividad de la consulta de cuenta, se muestran los 2 posibles caminos en cuanto a las actividades, pero no que objetos las realizan.

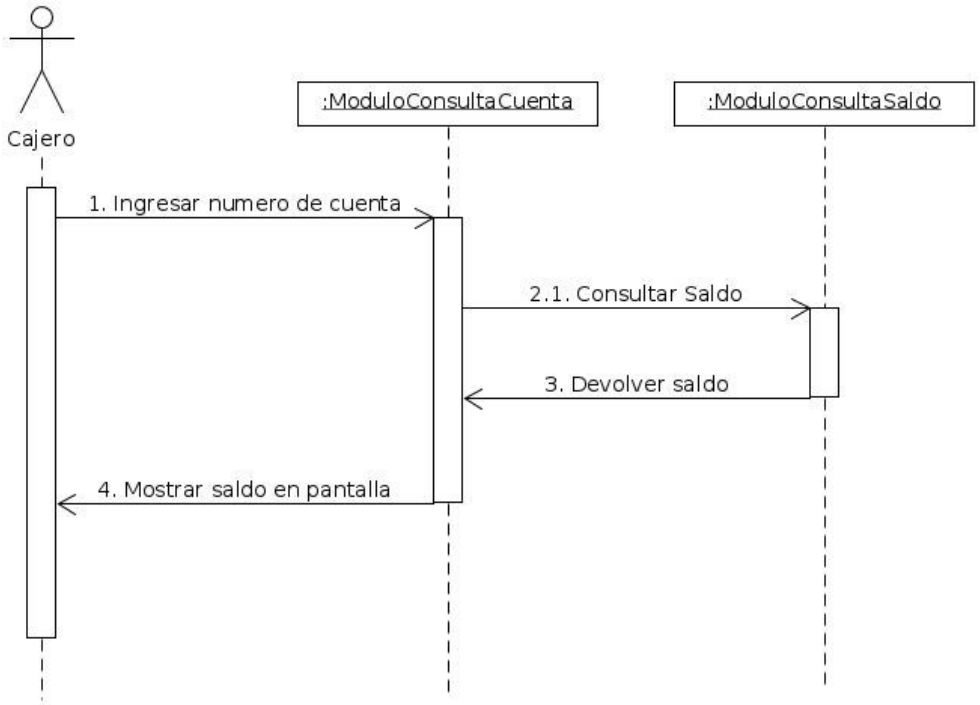


Diagrama de secuencia para cuando la cuenta existe, se muestra el saldo en pantalla.

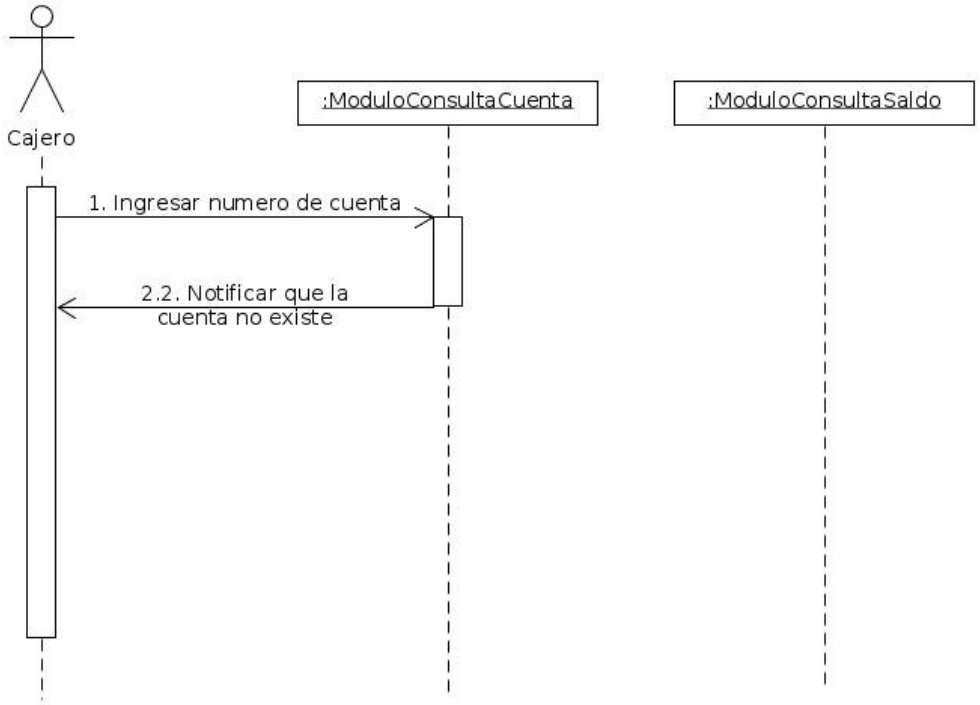


Diagrama de secuencia para cuando la cuenta no existe.

4) ¿Qué diagrama utilizaría para representar la estructura de un Software (aplicación)?

   Utilizaría el diagrama de componentes, ya que permite graficar los componentes que forman parte de la aplicación, asi como las relaciones entre ellos e información adicional. En este caso se entiende que no tenemos en cuenta la implementación, en cuyo caso deberiamos aplicar el diagrama de despliegue.

5) ¿Cuál es la forma adecuada para describir la arquitectura de un sistema durante el proceso de diseño?

   Durante el proceso de diseño de un sistema se usan varios diagramas y documentos para describirlo. A grandes rasgos, la arquitectura se basa en mencionar los componentes que forman parte del sistema, las relaciones entre componentes y las definiciones necesarias para que ese sistema logre su objetivo.

   Depende enteramente del tamaño y tipo de sistema que estemos diseñando para saber que tipo de diagrama y que documentos usaremos para describir el sistema.

   Esencialmente, los dos componentes mas importantes son el diagrama general del sistema (puede ser un diagrama de componentes o un diagrama de despliegue, dependiendo del espectro que abarque el sistema) y el documento de requisitos no funcionales, que incluye las características que debe cumplir el sistema más alla de sus funciones principales, las cuales le van a permitir ser escalable en un futuro.

   En base al progreso del proyecto, se pueden ir agregando diagramas y documentos, en base a lo que sea necesario para esa arquitectura. Al ser cada arquitectura muy diferente, es dificil precisar con exactitud una forma unica para describirla.

   En el caso puntual del diagrama de la arquitectura, es necesario contemplar todos los componentes importantes, separandolos si es posible en capas, funciones o subsistemas, indicando tambien las relaciones entre ellos. Lo más importante es que se pueda tener, a simple vista, una vision general de los componentes y como se van a relacionar.

6) Para una aplicación destinada a la venta y reproducción de películas a demanda por internet. ¿Qué patrón arquitectónico se debería implementar para dicha aplicación?

   Teniendo en cuenta que es una misma aplicación, se deberia implementar la arquitectura de microservicios.

   La mayor ventaja es que cada modulo funciona de forma independiente, por lo que no tenemos el problema de que si no podemos vender peliculas, tampoco podemos ver las que ya compramos, y viceversa. Lo mismo para las distintas funcionalidades dentro de la aplicación, como ver el catalogo o agregar peliculas a una lista personalizada.

   En algún caso puntual, se puede reemplazar el microservicio por un servicio mas grande y/o complejo, para poder atender esa necesidad especificamente.

   La flexibilidad de esta arquitectura permite que hagamos un diseño general del sistema, teniendo en cuenta las distintas funcionalidades, y a su vez que incorporemos componentes externos o de orquestracion para las distintas tareas, como validar una tarjeta de credito antes de procesar una venta.

7) Realice un diagrama de componentes de una Arquitectura cliente-servidor para una aplicación que debe tener 24/7 de disponibilidad donde el cliente que corre tanto en browser (web app) como “mobile” (native app) se registra(subscribe) y/o loguea para poder usar la aplicación. Considere el caso de usar SSO (Single Sign On) con cuenta de Google como alternativa de login.

    a) El FrontEnd está replicado en al menos 3 servidores para atender la demanda y disponibilidad, y consta de webservices de cliente y del web server.
    b) El Backend está replicado en al menos 2 servidores por disponibilidad y se conecta a una DDBB donde residen los datos de los clientes y la información que utiliza la aplicación.
    c) Una minima parte de la lógica de negocios existe en la parte cliente de la aplicación

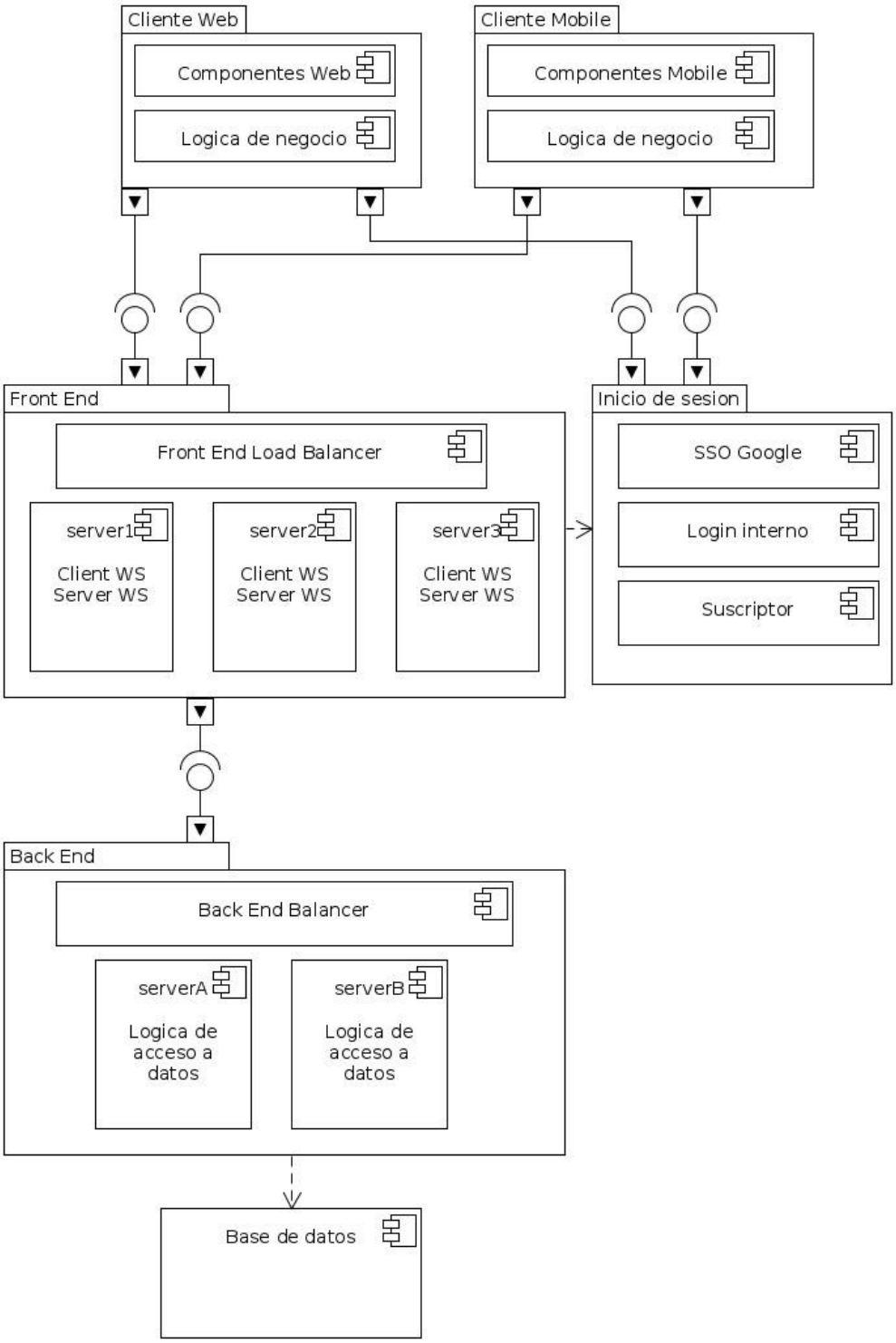
En el diagrama se distinguen los distintos tipos de componentes, agrupandolos por paquetes.

Los clientes se componen de los componentes propios del tipo de cliente asi como de la parte de la logica de negocio.

El paquete de inicio de sesion tiene los componentes para suscripcion, login y SSO de google, y posee las interfaces para ser consumido via web o via mobile.

El paquete de front end tiene un load balancer, para balancear la carga entre los 3 servidores, y los 3 servidores en si, los cuales poseen los servicios de cliente y del servidor web. Además, el front end se comunica con el inicio de sesion para validar que las peticiones provienen de un cliente autenticado. También posee las interfaces para ser consumido via web o via mobile.

El paquete back end tiene el load balancer y dos servidores que funcionan para aplicar la logica de acceso a datos. El mismo tiene una interfaz para ser consumido por los servidores de front end, y a su vez depende de la base de datos para funcionar.


Cliente Web
Componentes Web
Logica de negocio
Cliente Mobile
Componentes Mobile
Logica de negocio
Front End
Front End Load Balancer
server1
Client WS
Server WS
server2
Client WS
Server WS
server3
Client WS
Server WS
Inicio de sesion
SSO Google
Login interno
Suscriptor
Back End
Back End Balancer
serverA
Logica de
acceso a
datos
serverB
Logica de
acceso a
datos
Base de datos

8) Que consideraciones realizaría a la Arquitectura del punto 1 si la cantidad de usuarios de la aplicación crece de manera constante.

   No se plantea ninguna arquitectura en el punto 1, por lo que se procede a trabajar la arquitectura del punto 7 y de manera general.

   Cuando se tiene un crecimiento constante de usuarios, se debe ampliar la capacidad de procesamiento. Tambien puede ser necesario aumentar la capacidad de almacenamiento.

   En cuanto a la capacidad de procesamiento, se debe crecer en procesamiento, es decir, que sera necesario contar con mayor cantidad de servicios que puedan atender a las solicitudes de los usuarios. En caso de trabajar en cloud, se deben definir los parametros para que de forma automatica se vayan agregando los recursos a medida que crece el consumo. En caso de trabajar on premise, se debe considerar una revision regular de los recursos para detectar prematuramente cuando los recursos van a ser insuficientes para atender la demanda. Tambien se pueden aplicar ciertas tecnicas, como tener componentes para atender las solicitudes de solo lectura, para bajar la carga de los servidores, aunque esto solo colabore con bajar la carga del servidor, pero no cubre la demanda completa a futuro. Además, quizas sea necesario generar un cluster para la base de datos y para el inicio de sesion, para atender esas solicitudes de una forma mas optima.

   En cuanto al almacenamiento, aplica lo mismo que al procesamiento. En este caso, contamos con datos de la aplicación y datos de usuarios. El espacio donde se almacenan los datos de usuarios, debe contemplar un crecimiento regular, ya que los mismos crecen constantemente. El espacio donde se almacenan los datos de la aplicación depende exclusivamente de como se generan esos datos. Si los datos los generan los usuarios que crecen constantemente, es necesario contemplar el crecimiento tambien. Si los datos son generados por usuarios internos, en intervalos irregulares, el espacio y su crecimiento se analizan de forma separada.

9) Que consideraciones realizaría a la Arquitectura del punto 1 si la aplicación tiene alta concurrencia/simultaneidad de login en determinados momentos.

   No se plantea ninguna arquitectura en el punto 1, por lo que se procede a trabajar la arquitectura del punto 7 y de manera general.

   Para este caso, se debe agregar un balanceador de carga para el login, el cual permita gestionar mejor esa carga. En caso de estar ese modulo en un sistema cloud, es necesario definir los parametros para que la capacidad de procesamiento de ese componente crezca en cuanto hay una mayor demanda o en ciertos horarios. En caso de tener ese modulo on premise, se debe considerar un sistema de orquestracion que permita disponer de mayor cantidad de servidores en el cluster de inicio de sesion, en los momentos determinados para esa alta concurrencia.

10) Que consideraciones realizaría a la Arquitectura del punto 1 si la aplicación tiene alta concurrencia/simultaneidad de login en no determinados momentos.

No se plantea ninguna arquitectura en el punto 1, por lo que se procede a trabajar la arquitectura del punto 7 y de manera general.

Para este caso, es completamente necesario disponer de un sistema que permita agregar servidores al cluster en base a la demanda generada, en el momento que se suceda, para poder asi aumentar la capacidad de procesamiento y no tener demoras en el login del sitio o de la app.
Otra alternativa es disponer de la capacidad maxima que se va a necesitar en cualquier momento, pero esto genera dos problemas, el no aprovechar bien los recursos, ya que tendran mucho tiempo ocioso, y el hecho de que siempre puede aparecer un pico de consumo no previsto.

11) Que consideraciones realizaría a la Arquitectura del punto 1 si la aplicación tiene alta concurrencia/simultaneidad de uso en determinados momentos.

No se plantea ninguna arquitectura en el punto 1, por lo que se procede a trabajar la arquitectura del punto 7 y de manera general.

Para este caso, aplica la misma solucion que usamos para el login, cuando los momentos son determinados. Debemos disponer de mayor cantidad de recursos para procesar tanto el front end, como el back end y la base de datos. En este ultimo caso es recomendable agregar redundancia, para poder seguir sumando recursos en base a la demanda que se vaya generando.

12) Que consideraciones realizaría a la Arquitectura del punto 1 si la aplicación tiene alta concurrencia/simultaneidad de uso en no determinados momentos.

No se plantea ninguna arquitectura en el punto 1, por lo que se procede a trabajar la arquitectura del punto 7 y de manera general.

En este caso, aplica la misma solucion que aplicamos al login, cuando no podemos determinar los momentos en los cuales tendremos los picos de consumo. Es necesario disponer de mayor procesamiento en cuanto la demanda de servicios empieza a crecer, tanto en el front end, como en el back end y en la base de datos.

13) Que consideraciones realizaría para que la información actualizada en la BBDD y que es utilizada por la parte de lógica de negocios que corre en la parte cliente de la aplicación se sincronice.

Para lograr esta sincronizacion, es necesario que llegue la nueva informacion, o una actualizacion completa de la misma, desde la base de datos al cliente. El metodo a utilizar, depende 100% del tipo de informacion, asi como su volumen y sensibilidad.

Las dos formas de hacer esto son que el cliente inicie la solicitud, cada una determinada cantidad de tiempo, o que cada vez que se actualice la base de datos, se envie la actualizacion al cliente.

En primer caso, basta con poner un acceso periodico a la base de datos del cliente. La ventaja de esto es la simplicidad de desarrollo, pero puede rendundar en multiples accesos a la base de datos sin sentido, generando mayor consumo. Tambien se puede generar un hash para saber la ultima version de los datos y reducir esos accesos.

En el segundo caso, es necesario disponer de un sistema de suscripcion, en el cual cada cliente que ingresa al sistema, se suscribe a un servicio, y cada vez que se actualiza la base de datos, las novedades llegan al cliente. Esta metodologia es mas costosa de implementar, pero es la mejor para sistemas de gran escala. Es la tecnica que se aplica para generar arquitecturas manejadas por eventos.

14) Mucha de la información que la aplicación muestra a los diferentes usuarios resulta ser la misma, que consideraciones realizaría para cierto grupo de esta información que cambia ocasionalmente y para cierto grupo de dicha información que cambia seguido.

En cuanto a la informacion que cambia ocasionalmente, lo mejor es tener un cache de esa informacion, de forma tal que se pueda acceder de forma rapida, generando la menor cantidad de accesos. Lo importante es tener un proceso, o evento, que genere la actualizacion de esa informacion, cuando haya un cambio en la misma.

En cuanto a la informacion que cambia seguido, tambien conviene tener un cache, pero no conviene que se actualice cada vez que cambia, ya que generaria mucha carga. En esos casos, lo mejor es hacer actualizaciones cada un intervalo de tiempo determinado, que sea corto, para mantener la informacion actualizada, pero que tambien evite que las multiples consultas de la misma informacion durante ese intervalo se traduzca en mas accesos a la base de datos.

15) Que consideraciones realizaría para que la aplicación soporte “Disaster recovery” (ej lugar de hosting queda fuera de servicio)

Para poder realizar un disaster recovery se necesita tener resguardos regulares de la informacion, asi como los pasos necesarios para que esa informacion y el sistema que la consume, puedan ser restaurados en el menor tiempo posible.

Dependiendo de las plataformas que se usen para ejecutar el sistema, el resguardo y restauracion del sistema puede demorar mas o menos. Por ende, siempre es conveniente usar componentes que permitan la automatizacion de despliegue de la informacion, ya sea a través de scripts o imagenes de resguardo.

Para el ejemplo de que el lugar de hosting queda fuera de servicio, puede ser tan simple como tener el resguardo generado cada un cierto intervalo de tiempo, generar los sistemas en el nuevo hosting (o tenerlo siempre disponible para un eventual problema) y restaurar la informacion en el nuevo hosting, asi como hacer los cambios en el entorno para que el las peticiones de los clientes sean enviadas al nuevo hosting.

16) La aplicación debe mostrar una pantalla de información al Usuario, está información es construida por la unión (o merge) de diferentes respuestas a peticiones desde la parte cliente de la aplicación. Que consideraciones realizaría cuando la cantidad de peticiones necesarias son 10 o más para que la presentación sea consistente.

Como primer medida, es necesario que el componente que va a la mostrar el merge de informacion este separado del resto de la aplicación, ya que puede tener mayor demoras y no es positivo que la aplicación se detenga para esperar a la resolucion de esa solicitud.

Una vez que tenemos el componente por separado, es conveniente manejar a traves de eventos la resolucion de las 10 o mas peticiones que va a generar el cliente. Manejar esas peticiones con eventos nos permite que esas peticiones se vayan resolviendo una a una, sin bloquear a las demas. Hay dos estrategias posibles, que se hagan en paralelo o que se hagan de forma serializada, todo depende del detalle y la naturaleza de las peticiones.

A medida que se van resolviendo las peticiones, se va procesando la informacion. Considerando el peor caso (que necesitemos todas las peticiones para generar el minimo resultado), un ultimo evento se dispara para consolidar la informacion de todas las peticiones, cargar la informacion en el componente y actualizarlo para que el usuario pueda disponer de la informacion.

En caso de que cada una de las peticiones obtenga informacion de diferentes componentes de la pantalla, cada peticion debe hacerse por separado, generando un evento cuando haya terminado y que ese evento dispare la actualizacion de la informacion.

17) Que consideraciones realizaría si la aplicación puede ser explotado por diferentes proveedores del servicio dado por la aplicación, es decir, dada una aplicación A, la parte cliente tendrá el branding de los diferentes proveedores de cara a los usuarios (concepto de Multitenant)

Para este caso, el sistema debe contar con cierta parte que sea personalizable en base al proveedor que desea ofrecer el servicio. Esto quiere decir que aplicacion debe poseer parametrizacion para los componentes, asi como estandarizacion para la informacion que provee.

Tambien se debe contar con una forma de acceso a la aplicación (API), para poder simplificar el acceso de los distintos sistemas que la exploten.

Ademas, se puede agregar cierta parte de las ejecuciones que permita que quienes explotan la aplicación, puedan aplicar su propia logica de negocio a ciertas partes del proceso.

18) Que consideraciones realizaría para que la aplicación sea distribuida geográficamente para mejorar la calidad de experiencia y disponibilidad.

Para tener mejor disponibilidad en una mayor area de cobertura, a nivel geografico, es conveniente disponer de servidores, o nodos, en las distintas regiones, lo que genere menor latencia en cada solicitud de la aplicación.

Para poder lograr esto, es simple a nivel aplicativo, pero un poco mas dificil a nivel informacion. En cuanto a la aplicación, bastara con desplegarla en distintos host, que se encuentren en diferentes regiones.

El desafio con los datos, reside en que no se actualizan de forma unilateral, por lo que hay que mantener sincronizada la informacion que se genera en los distintos nodos. Esto puede hacerse aplicando tecnicas de replicacion de las bases de datos directamente, o manejando eventos que notifiquen los cambios en una base de datos a las demas.

19) Que consideraría para que la aplicación sea agnóstica a la infraestructura

Para lograr esto, se necesita contar con contenedores que aislen a la aplicación del entorno en el cual estan ejecutandose. El más popular de estos es Docker, que permite generar un contenedor que correra de igual forma, sin importar en que infraestructura lo implementemos.

20) La aplicación resulta de una constante evolución (mejoras, agregado de funcionalidades, resolución de bugs, etc), que consideraría en la cadena desde el desarrollo hasta el deployment para tener una aplicación de estas características

Para esto se necesitan aplicar las tecnicas de DevOps, esto incluye varios componentes como un respositorio de codigo fuente, para tener centralizado el codigo de la aplicación. Tambien incluye un sistema de construccion, para que la construccion de los componentes, a partir del codigo fuente sea siempre igual. Estas construcciones son ejecutadas por un sistema de integracion continua, que aplica una serie de pruebas luego de la construccion y despliegue del componente.

Este concepto de integracion continua y despliegue continuo, permite que las mejoras y agregado de funcionalidades se logren en el menor tiempo posible. Tambien permite detectar los errores en el momento que se desarrollan, ya que se dispone de las pruebas que detectan esos errores y sabemos que version fue la que genero el error.

21) Debido a como los Usuarios usan la aplicación las APIs son usadas de maneras diferentes y la implementación de las mismas pueden requerir cómputo, acceso múltiple a DDBB, etc , pudiendo existir grupos de APIs que son afines entre sí y otras que no. Que consideraciones realizaría en la aplicación para atender cierta divergencia en el comportamiento de las APIs.

Cuando tenemos APIs que realizan distintos tipos de operaciones (procesamiento, I/O, distinto volumen, integracion de informacion), es conveniente tener un componente que nos permita gestionar donde y como se ejecutan esas APIs. Ese componente debe permitirnos asignar recursos diferentes, separarlas por grupos, los cuales van a tener esa disponibilidad de recursos.