**Requisito I : Suscripciones:**

* Necesidad de usuarios concurrentes
* No es necesaria consistencia fuerte
* Máxima disponibilidad

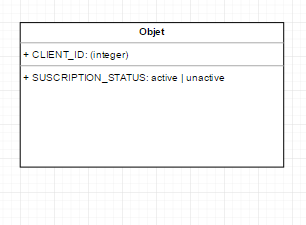
Opción de implementación: Riak

Pros de la selección:

* Es una bbdd con máxima disponibilidad, con gran concurrencia de usuarios. En la parte de AP del teorema CAP.
* Como es una NoSQL clave valor,siempre se va a consultar por ID de usuario y el único valor es un boleano para saber si tiene las suscripción activa o no.

Porque se han descartado otras opciones:

* Podría haberse seleccionado Cassandra porque es la otra NoSQL AP del teorema CAP, pero no es necesario almacenar otros valores que no sea el estado de la suscripción, por lo que por simplicidad y escalabilidad se dedice Riak. Además, en Cassandra el balanceo es manual, por lo que igualmente Riak es la mejor opción.
* NoSQL documentales se descartan por la necesidad de maxima disponibilidad y concurrencia de usuarios, por lo que las BBDD CP se descartan de la opción.
* Idem, para los modelos relacionales que no son escalables.



**Requisito II : Catálogo**:

* Se requiere que solo los usuarios con suscripción activa vean el catálogo
* Se requiere integridad
* El listado de series y países solo disponibles para ciertos países
* necesario también el listado de actores
* Necesario identificar las relaciones entre series/películas y actores
* Se necesita identificar para series/películas aquellas similares por categoría y año
* No se necesita gran escalabilidad
* Solo es necesario metadatos

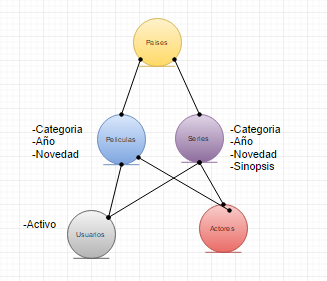
Opción de implementación: Neo4j

Pros de la selección:

* Los motivos técnicos son que no se necesita una gran escalabilidad y sin embargo una consistencia, por lo que es necesario buscar una BBDD en la zona CP o AC.
* Puesto que no es necesario almacenar datos, sino únicamente metadatos, descartamos un modelo relacional AC.
* Puesto que es necesario identificar varios tipos de relaciones (Actores-peliculas/series, Paises-Peliculas/series, Similares-Peliculas/series) la mejor opcion es sin duda usar una bbdd NoSQL de grafo, que permita explotar las realciones, que es lo verdaderamente importante es este requisito.

Porque se han descartado otras opciones:

* En el punto anterior se argumenta porque se descarta el modelo relacional.
* Los modelos AP se descarta puesto que se indica que se necesita consistencia, por lo tanto priman las soluciones CP
* Dentro de las soluciones CP, se descartan las BBDD documentales,puesto que es más difícil explotar las relaciones entre las diferentes entidades, aparte de no ser necesario el almacenamiento de datos.



**Requisito III : Navegación de usuario:**

* Se requiere almacenar datos de la navegación (URL de página,hora de la conexión y tiempo que permanece en esa página)
* Se necesita acceder a todas las conexiones de un usuario para hacer analítica, por tanto la clave de acceso será el id del usuario.
* Se necesita escalabilidad y disponibilidad de la solución

Opción de implementación: Cassandra

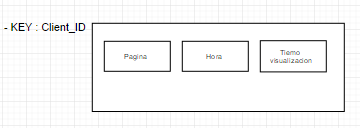
Pros de la selección:

* Por los requisitos técnicos de escalabilidad y disponibilidad se descartan soluciones AC, CP por lo que lo ideal es una solución de almacenamiento noSQL AP.
* Puesto que hay familias de atributos que almacenar, relacionados con un identificacion de usuario, y además series temporales de navegación, lo ideal es usar una BBDD NoSQL de formato columnar (o de familia de columnas) como Cassandra.

Porque se han descartado otras opciones:

* Se descartan las bases de datos documentales y de grafos (CP) por temas de escalabilidad y disponibilidad. Idem para las BBDD relacionales
* Podría darse el caso de seleccionar una BBDD de clave valor como Riak, pero en este caso, es necesario almacenar varias variables que además pueden ser utilizadas para consultar y hacer analítica (por ejemplo, podría quererse consultar todas las páginas navegadas en una ventana temporal para identificar a los usuarios que acceden en esa franja). Este tipo de consultas serían mas dificultosas de hacer en una BBDD clave valor.

.



**Requisito IV : Sistema de recomendación**:

* Se requiere la valoración de 1-10 de cada película/serie que vea cada usuario así como los actores de manera independiente.
* Se requiere que periódicamente se actualicen las películas recomendadas para el usuario
* También se requiere que se puedan votar a los usuarios en las últimas películas.
* Es necesario almacenar también las preferencias de cada usuario.
* Será necesario recuperar para cada usuario las pelis que ha visto, sus votaciones y las películas que el sistema recomendador le recomienda.

Opción de implementación: MongoDB

Pros de la selección:

* Los requisitos no estipulan factores técnicos de disponibilidad o transaccionalidad, Sin embargo, las votaciones que los clientes hacen a varias películas, va a requerir cierto nivel de transaccionalidad por cliente.
* Posiblemente se necesiten distinto tipos de consulta por agregados (ya sea información de usuarios, información de películas/series (ya sean votadas o recomendadas). Este nivel de agregación, es demasiado desestructurado para hacer familias de columnas, ya que puede haber de 1..N peliculas y series que un usuario haya visto / votado o tenga recomendado.
* Como los niveles de agregación están bastante claros, y son transaccionales por usuario, la mejor opción es MongoDB. Además permite la opcion de correr map-reduce para el calculo de métricas sobre las colecciones.

Porque se han descartado otras opciones:

* Se descartan bases de datos de grafos porque no se van a analizar relaciones, sino informacion de determinadas coleciones.
* Se descartan opciones de claves valor, porque hay varios tipos de colecciones y agregaciones de datos que es necesario manejar, no unicamente consultas por id.
* Las bases de datos columnares (o de familias de columnas) podrían ser una opcion, pero sin dudael nivel de concurrencia de una votación será poco elevado comparado con otros casos de uso, y sin embargo si se necesitará cierto nivel de transaccionabilidad, con lo que este tipo de BBDD las maneja peor.