**Decisiones de Diseño y Justificación**

**Modelo arquitectónico**

Se decidió por la implementación del patrón arquitectónico “MVC” para poder dividir las responsabilidades de la VISTA, CONTROLADOR y MODELO, que facilite la mantenibilidad y al mismo tiempo nos de flexibilidad para poder adaptar la aplicación a los requerimientos que se fueran sumando.

El uso del framework GRAILS, fue una decisión en conjunto que se tomó dado que es un framework específico para este tipo de modelo arquitectónico. Grails pretende ser un marco de trabajo altamente productivo siguiendo paradigmas tales como [convención sobre configuración](https://es.wikipedia.org/wiki/Convenci%C3%B3n_sobre_Configuraci%C3%B3n) o [no te repitas (DRY)](https://es.wikipedia.org/wiki/No_te_repitas), proporcionando un entorno de desarrollo estandarizado y ocultando gran parte de los detalles de configuración al programador. La idea es que nos centremos en el paradigma de objetos, utilizando el patrón MVC y abstraernos de la configuración, así como también tener la suficiente flexibilidad y rapidez ante los cambios requeridos.

**JDBC vs ORM**

La utilización de JDBC por sobre una ORM se debió a varios factores, entre ellos:

* Performance
* Control total sobre querys para poder optimizarlas lo más posible
* Experiencia del equipo
* Flexibilidad. La posibilidad de normalizar la base a gusto, nos daba un escenario más conveniente para adaptar la aplicación a un posible crecimiento que implicarían relaciones más complejas, difíciles de manejar con una ORM.

Sin embargo al implementar SpringSecurity, ya viene por default con una ORM, la cual se decidió conservar, descartando la autenticación por medio de base de datos porque nos parecía poco seguro.

Aunque somos conscientes de que un correcto logueo a la aplicación requeriría la re implementación de un servicio, es decir un sistema de autenticación; lo cual significa una inversión de tiempo que preferimos invertir en otros hitos de la aplicación y dejar para futuros releases.

**Uso de Cache**

Ante los requerimientos cambiantes y la potencial escalabilidad futura de la aplicación con nuevos usuarios, lo cual significa múltiples consultas y procesamiento, surgió la idea de la implementación de una Cache Redis, que mejore la performance considerablemente, permitiéndonos poder guardar consultas ya hechas, guardas las cuentas calculadas, etc..

Encontramos mayor facilidad de levantamiento de clusters en Redis por sobre MySQL.

**Mongo - base documental**

Para la persistencia de los Indicadores y Metodologías, debido a su complejidad, se decidió llevarlas a una base documental, con menos estaticidad en las tablas que un modelo relacional, donde tenemos la posibilidad de guardar directamente los objetos tal cual son definidos, simplificando también las querys de obtención.