Cómo usar repositorios de datos en un proyecto software
Por Rafael Gómez Blanes
Artículo publicado en www.rafablanes.com **– 09/2016**

Me llama mucho la atención cómo a lo largo de los años puedo haber trabajado en aplicaciones con .NET en C#, con javascript utilizando Angular y NodeJS, y también con PHP cuando estaba más centrado a hacer módulos en Drupal, y cómo con todos esos lenguajes con sus respectivos entornos de desarrollo se pueden aplicar las mismas **buenas prácticas**.

Hay veces que éstas se refieren a procedimientos de trabajo o al modo de estructurar un proyecto, otras hacen referencia a aplicar patrones de diseño e incluso *antipatrones* para detectar precisamente lo que no se hace bien. En otras ocasiones, son sencillamente prácticas habituales que hace la mayoría de la gente que usa una tecnología en particular (de ahí, por ejemplo, los proyectos de *scaffolding*).

Un buen proyecto software debe *oler* a buenas prácticas, de principio a fin, desde la arquitectura del mismo hasta la evolución de las funcionalidades que se le van añadiendo poco a poco.

Nada peor que una aplicación, proyecto o sistema en el que no se pueda ver claramente cómo están implementadas esas buenas prácticas.

Y es que además tienen algo en común: la mayoría de esas buenas prácticas son fáciles de implementar.

Y como me gusta escribir... pues voy a ir hablando de ellas, de aquellas con las que yo directamente he visto que tienen más impacto en la mejora de la calidad de un proyecto software.

Un aviso a navegantes: en muchas ocasiones los beneficios de estas recomendaciones no se

ven claramente, ni siquiera en el corto plazo, y su impacto positivo a medio y largo plazo es más una cuestión sutil que sólo se aprende cuando has cometido todos los errores posibles y entonces se hace la luz y dices algo como, "ahora sí que lo entiendo".

Para mí la *implementación de repositorios de datos* es una de las prácticas más útiles en cualquier proyecto software.

El principio general de implementar un repositorio de datos vendría a ser el siguiente:

> *"Todos los accesos a los datos que utiliza una aplicación (vengan de bases de datos, entidades de datos del mundo cloud, datos en caché, etc.) deben realizarse desde un* ***único punto****."*

Dependiendo de la tecnología y del tipo de aplicación que se esté desarrollando, ese *único punto* se entiende por un módulo con sus clases, una librería, una API, un servicio REST, etc. Lo importante es que todos esos accesos estén *centralizados* y bien localizados en el proyecto.

¿Qué beneficios se obtiene de esto? Son muchos, muchísimos, algunos de ellos no fáciles de ver en un principio.

Por resumir, centralizando el acceso a los datos conseguimos:

- Cumplimos con el principio **SoC**, (*separation of concerns* o *separación se intereses*), básico en el desarrollo ágil. De este modo tenemos una parte muy concreta y localizada del proyecto especializada en acceder a los datos que utiliza.

- Se evitan **duplicar código** repitiendo los mismos accesos a los mismos datos a lo largo de la solución. En lugar de ejecutar algo como un "select * from dbo.users where userId = 20" cada vez que se necesita la información de un usuario, tenemos algo mucho más general como "data.getUserById(20)" (esto es un sencillo ejemplo en pseudocódigo, eh!).

- Se **simplifica la solución** y se reduce el número de líneas de código. Al estar encapsulado el acceso a los datos en métodos muy concretos, no necesitamos *ensuciar* u ofuscar la lógica de negocio de la aplicación con detalles de acceso a los datos (tales como abrir una conexión a la base de datos, cerrarla, etc.), y ni hablar cuando esos accesos con más complejos y necesitamos transacciones o *joins* anidados.

- Es más fácil ejecutar un análisis sencillo de ***profiling***: tan sólo repasando en el objeto repositorio a qué datos se accede y cómo, podemos determinar qué índices hacen falta, en el caso de bases de datos relacionales e implementar tests de rendimiento que se puedan ejecutar continuamente.

- Conseguimos **desacoplar** la solución de cómo ésta accede a los datos. ¿Y si mañana cambiamos de motor de base de datos? ¿Y si por escalabilidad se decide distribuir los datos en diversas bases de datos incluso con diversas tecnologías? Nada peor que empañar todas las partes de la solución con los detalles de acceso a los datos. Puesto que todo está *encapsulado* en el repositorio, tan sólo hay que tocar éste. El resto de la aplicación, ni se entera del cambio.

- Con la implementación del repositorio, al estar éste desacoplado, se pueden crear **tests**específicos, unitarios y de rendimiento.

- Es trivial también crear ***mocks***. ¿Qué pasará cuando la tabla "usuarios" tenga cien mil registros? ¿Aguantará el *front-end*? Con un repositorio de datos es fácil simular diversas situaciones en previsión de problemas de escalabilidad pero sobre todo, que esas situaciones pueden estar respaldadas por pruebas.

- Al estar todos los accesos centralizados, es más fácil para la persona encargada del desarrollo de la base de datos y su acceso trabajar en esa área especializada.

- Al no estar dispersos por la solución todos los accesos al repositorio de datos, las **migraciones** del mismo no terminan en pesadilla...

Hace un tiempo se hablaba en ciertas tecnologías de la *capa DAL* (*data access layer*), que vendría a ser el mismo concepto pero más ligado a bases de datos relacionales.

Si se utiliza algún tipo de ORM que crea objetos alrededor de las entidades de datos, ¿deberían esos objetos *viajar* a lo largo de la lógica de negocio? Esta pregunta me la encuentro recurrentemente, y mi opinión es que no, puesto que de ese modo la aplicacíon tiene una dependencia clara de la tecnología ORM utilizada. Los objetos que se deberían distribuir a lo largo de la lógica de negocio serán similares pero no ligados a la tecnología ORM usada (son los que se llaman los *data transfers objects* o DTOs).

A ver, al final hay que aplicar el sentido común (con un poco de experiencia y hasta de suerte...): si trabajamos en un proyecto cuyo ciclo de vida es muy claro y sabemos que se terminará y que no va a evolucionar, pues nos podemos tomar cierto tipo de licencias, pero si tenemos entre manos un producto o proyecto que crecerá y evolucionará en funcionalidad a lo largo de muchos años, entonces sí hay que cuidar mucho este tipo de decisiones y dependencias, así como aplicar con mayor rigor aún todas las buenas prácticas posibles.

Sí, ya sé, es que a veces no hay tiempo, el proyecto viene muy cerrado en horas, etc. De acuerdo, razón de más para aplicar este tipo de buenas prácticas desde el principio, porque sólo así se podrá avanzar en el proyecto con productividad y mayor rapidez.

Todo esto es más fácil de implementar que de describir; sin embargo, un buen proyecto software que necesita de cualquier mecanismo para persistir datos, tiene que acceder a ellos a través de un objeto / módulo / librería que implemente esta buena práctica.

Algunos enlaces de interés:
[The Repository Pattern]
[Repository Pattern Done Right]

Rafael Gómez Blanes es Ingeniero Informático por la Universidad de Sevilla (España)
Trabaja como Ingeniero de Calidad Software para diversas compañías nacionales e internacionales.
Contacta en:
www.rafablanes.com
contact@rafablanes.com
@gomezbl

Good Coding!