Bases de Datos II

Trabajo Práctico Integrador 3

Fecha de Publicación: 18/05
Fecha de Finalización: 01/06 (la entrega se hará en conjunto con la etapa siguiente)

Introducción

Este trabajo consistirá en implementar una alternativa de persistencia sobre el mismo modelo de la etapa 1 (DBlivery), pero esta vez utilizando MongoDB como almacenamiento. Nuevamente, la aplicación consistirá en una arquitectura multicapa clásica, pero requerirá de la implementación de un nuevo **servicio** y **repositorio**, ambos adaptados a la tecnología MongoDB. Además, la forma de indicar los *mappings* será diferente.

El nuevo servicio debe implementar la misma interfaz de la etapa 1, `DBliveryService`. Para esto, hay que tener en cuenta las diferencias entre Hibernate/RDBMSs y MongoDB. En primer lugar, MongoDB identifica los documentos con un **ObjectId** compuesto que desde Java se manipula como **String**. Para reflejar esto en el modelo y facilitar las consultas en el nuevo repositorio, los objetos del modelo deberán implementar la interfaz `PersistentObject`, que define un protocolo mínimo para obtener y asignar el **ObjectId**.

Para resolver este trabajo deberán trabajar sobre una rama aparte del proyecto del TP previo, con otro proyecto de base proporcionado por la cátedra. El modelo que hayan realizado debe permanecer en el paquete **model**, pero deberán realizar los mappings con annotations para el formato BSon, reemplazando las *annotations* de Hibernate.

Para persistir los objetos se utilizará un nuevo repositorio que se comunica con el driver para MongoDB mediante **PojoCodec**, un mapeador básico que convierte los objetos en documentos de MongoDB y viceversa. El repositorio `DBliveryMongoRepository` es provisto por la cátedra con algunos métodos básicos implementados. Hay en principio dos opciones para mapear las asociaciones:

- **generando documentos embebidos** que es la mecánica por defecto de PojoCodec. Si se quiere evitar esto se debe utilizar la annotation `@BsonIgnore`.
- **relacionándolos manualmente** utilizando la clase `Association` (provista), para lo cual hay que ocuparse de crear la asociación a persistir, y recuperar la relación entre los objetos al recuperarlos de la BD, **sin que esto represente un cambio en el modelo**, es decir que sólo se utilizan las instancias de Association para *preparar* los objetos antes de persistirlos, lo que generará colecciones en la BD para representar dichas relaciones (una colección para cada combinación de clases).

La infraestructura del proyecto provista se encuentra en el mismo repositorio del TP previo, **en la rama `mongo`**:

https://github.com/juliangrigera/bdlivery/tree/mongo

Presenta los siguientes cambios respecto de la etapa 1:

- Se agrega la clase `MongoDBConfiguration`, que al igual que en el caso del TP con Hibernate, deberán implementar el método `getGroupNumber()` con su número de grupo.
- Se agrega el paquete **mongo** con la clase `Association` y la interfaz `PersistentObject`.
- Se agrega el nuevo repositorio para trabajar con el driver para MongoDB, con métodos implementados.
- Hay una nueva clase de tests `DBliveryMongoTestCase`, con los mismos tests de la etapa 1 pero adaptados para utilizar los identificadores de MongoDB.

Al igual que en la etapa 1, desde la línea de comandos debe obtenerse un build exitoso en donde pasen todos los tests.

Requisitos detallados que deben satisfacerse en esta etapa

- Escribir todo el código necesario para que la aplicación compile y todos los tests de las subclases de `DBliveryMongoTestCase` pasen, tanto en su implementación de Hibernate como en la de MongoDB. Esto implicará codificar una nueva implementación concreta de la interfaz `DBliveryService`.

- El código tiene que estar subido en el mismo repositorio que la etapa previa, pero en nuevo branch llamado `practica3`.