Taller Web 1 Evaluación de Aprendizaje Nro 2

Alumno: Marcos Cabral.

ENVIO	Jueves 11/6/2020	Fecha de Envío de EV2 a alumnos
ENTREGA	Jueves 18/6/2020	Fecha de Entrega
DEVOLUCION RDOS	Jueves 25/6/2020	Fecha Máxima de Devolución de Resultados
REENTREGA	Jueves 2/7/2020	Fecha de Reentrega

Pautas de resolución

1. Entregar a través de la plataforma MIEL
2. El formato de la entrega debe ser PDF obligatoriamente
3. Agregar el nombre del alumno en el encabezado del documento
4. No incluir capturas de pantallas
5. El texto correspondiente a código fuente debe estar en tipografía `Curier` o `Consolas`

Resolver

1-Pegar código del producto que se está desarrollando en equipo donde se vea el mapeo de hibernate de una relación entre clases explicando las partes principales del mismo. Se debería mostrar un caso de una mapeo uno a muchos. Si se mapea del lado "muchos" explicar por qué es recomendable mapear de este lado de la relación; si en cambio se tomó la decisión de ir en contra de la recomendación y mapear del lado "uno" explicar el por qué de esa decisión

2- Mostrar una porción de código de un repositorio donde se haga una consulta usando Criteria (que tenga al menos 1 restricción) explicando las partes importantes del mismo

```
1.@Entity

public class Pedido {

	@Id

	@GeneratedValue(strategy = GenerationType.IDENTITY)

	private Long id;


	@ManyToOne

	private Usuario usuario;

}
```

Para empezar tenemos una relación de Muchos a uno, del lado de muchos. Esto lo sabemos gracias a la anotación @ManyToOne, que se muestra en el código.

Al tener relacionada nuestra clase Pedido con la del Usuario estamos diciendo que uno o muchos pedidos van a pertenecer a un usuario en concreto, el pedido no se realiza si no hay un usuario que sea responsable del mismo. Por lo tanto deberemos tener un campo de tipo Usuario en esta clase.

La primer razón por la que usaría esta manera es que es lo más parecido a como se trabaja en el mundo de las bases de datos relacionales. Como hay 1 usuario para N pedidos, cada pedido contiene una clave externa para el usuario al que pertenece. De esta manera se asegura recorrer un valor propio que hace referencia a uno existente.

Un problema propio de Hibernate para no utilizar un @OneToMany y quedarnos con el @ManyToOne, es que Hibernate puede llegar a crear tablas inesperadas o “fantasmas” y ejecutar más sentencias SQL de las que esperaba al momento de registrar valores.

```
2. public Boolean validarExistenciaEmail(String email) {

            final Session session = sesion.getCurrentSession();

Usuario usuarioEncontrado =
(Usuario).createCriteria(Usuario.class).add(Restrictions.eq("email",
email)).uniqueResult();

            if (usuarioEncontrado != null){

                  return true;

      }return false;

}
```

Como vemos usamos un método validarExistenciaEmail, para verificar si existe el email.

Utilizamos una sesión, asociada a la transacción iniciada en el servicio que llama a este método, y creamos el Criteria, y le especificamos en el parámetro, que será sobre la clase Usuario.

Agregar un criterio y restricciones, nos permite poder obtener registros de la base de datos, filtrados bajo una condición, y asegurándonos que nos devuelvan un valor, o una lista de valores.

Podemos agregar diferentes tipos de restricciones, y se agregan con el método add().

En este caso utilizamos el método eq(), que nos permite comparar que el email que recibo por parámetro, coincida con alguno de la tabla usuarios, donde hay un campo llamado “email”.

Y para finalizar le agregamos el uniqueResult, que sirve para indicar que esperamos un único resultado, y si en caso de traer más de uno, devuelve un error.