Taller Web 1 Evaluación de Aprendizaje Nro 1

Alumno: Marcos Cabral.

ENVIO	Jueves 28/5/2020	Fecha de Envío de EV1 a alumnos
ENTREGA	Jueves 4/6/2020	Fecha de Entrega
DEVOLUCION RDOS	Jueves 11/6/2020	Fecha Máxima de Devolución de Resultados
REENTREGA	Jueves 18/6/2020	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 están desarrollando y explicar en el mismo como se mapean un action desde una vista, explicando las partes importantes, si es por GET debe tener pasaje de parámetros (mostrar código de una vista donde se invoque un action, un controlador donde el action se procesa, explicar las anotaciones importantes, los tipos de retornos y todo lo que sea relevante para comprender las porciones de código entregadas)

2- Mostrar un Service del producto construido donde se vea inyección de dependencias de un repositorio. Mostrar el repositorio inyectado y explicar cómo se resuelve dicha inyección.
Responder: cuál es la responsabilidad de la capa de servicios y cuál es la relación de esta capa con el concepto de "unit of work"

1. A continuación podemos ver una parte de la vista jsp, donde tenemos un formulario que nos indica colocar un nombre para crear una restriccion.

Lo que escribamos dentro de action nos define hacia donde viajaran los datos del formulario, y por otro lado, el method, nos indica de que forma viajaran dichos datos. En este ejemplo, los datos van a ser enviados por el metodo GET.

```
<form action="crearrestriccion" method="GET">
            <div class="form-group">
                  <div class="row">
                        <div class="col-md-6">
                              <input path="nombre" name="nombre" type="text"/>
                        </div>
                        <div class="col-md-6">
                              <button Type="Submit"/>Crear restriccion</button>
                        </div>
                  </div>
            </div>
      </form>
```

Sin embargo, necesitamos tener una parte de nuestro programa que sepa responder ante los pedidos del formulario, en este caso lo hacemos creando un metodo en el controlador de Restriccion.

```
@RequestMapping(path="/crearrestriccion", method=RequestMethod.GET)
	Public	ModelAndView	crestriccionesUsuario(@RequestParam(value="nombre", 
required=true) String nombre) {

		ModelMap model=new ModelMap();
		Restriccion restriccion=new Restriccion();
		restriccion.setNombre(nombre);

		Long idGenerado=this.servicioRestriccion.crearRestriccion(restriccion);

		model.put("restriccion",restriccion);

		return new ModelAndView("restriccionCreada",model);
	}
```

Utilizamos la anotación de RequestMapping, para indicarle que vamos a estar juntando el path, que sería la ruta o el action que colocamos en el formulario, y nuestro método de crearRestriccionesUsuario.

Luego definimos nuestro método, el cual va a devolver un ModelAndView, le indicamos que va a esperar un parámetro en el request del envio por GET, con el valor=”nombre”, le indicamos que es requerido, y definimos su tipo de dato, en este caso un String.

Una vez con el valor del nombre pasado por get, podemos manipular dicho dato como el tipo de dato que definimos en el parámetro de nuestra función. Como el objetivo del método nuestro es crear una restricción, instanciamos un nuevo objeto Restricción, y le seteamos el nombre que el usuario relleno en el formulario.

Finalmente, nuestro método es un ModelAndView, y debemos devolver un objeto de dicho tipo de dato, este objeto nos permite asociar una vista jsp, con un modelo, que puede llegar a ser un objeto, lista variable, o simplemente una vista sin modelo.

Para poder hacerlo, creamos un objeto tipo ModelMap, mapeamos un objeto y una key, en este caso, la key “restriccion”, responderá al objeto restriccion.

Retornamos un nuevo objeto ModelAndView, donde el primer parámetro responde al nombre de la vista a la que queramos dirigir al usuario y los datos, y el segundo responde al modelo.
Para terminar, en la vista de restriccionCreada, podemos manipular y mostrar los datos de nuestro modelo.

```
<h3>Nombre: ${restriccion.nombre}</h3>
```

2.

@Service ->**Nos indica que la clase será un servicio relacionado a la capa de negocio.**

@Transactional ->**Nos permite no realizar determinada acción si existe un error, es decir, no "arrastrar" errores.**

```
public class ServicioRestriccionImpl implements ServicioRestriccion {
	@Inject ->Inyección de dependencias.
	private RestriccionDao restriccionDao; -> componente al cual se le inyectan las dependencias.
	@Override
	public Long crearRestriccion(Restriccion restriccion) {

		return restriccionDao.crearRestriccion(restriccion);
	}
}
```

@Inject nos permite inyectar comportamientos a componentes haciendo que nuestras piezas de software sean independientes y se comuniquen únicamente a través de una interfaz.

Por otro lado tenemos el repositorio.

```
public interface RestriccionDao {
	Long crearRestriccion(Restriccion restriccion);
}
```

Donde definimos la interfaz RestriccionDao, y definimos la firma del método crearRestriccion. Y lo manejamos como una interfaz de acuerdo al patrón de diseño Dao. Además de permitir tener un código más escalable, nos facilita la separación de las dependencias, haciendo así más fácil para probar cada componente.

De esta manera logramos tener de manera individual el repositorio y el servicio, donde podemos interactuar con el mediante la inyección de dependencias, y así poder manipular la capa de acceso a datos.

La responsabilidad de la capa de servicios es la de mantener la lógica de negocio de la aplicación, utilizando un servicio para cada controlador. La relación de esta capa con el concepto de unit of work, es el de tratar como unidad todos aquellos objetos nuevos, modificados o eliminados con respecto de una fuente de datos.