

# Auditoría backend

Auditoría backend sobre Spring boot con tecnologías event listener y entity listener.

## Implementación de auditoría principal backend (semiautomática/user) con eventListeners.

### Objetivos

El objetivo principal de ésta historia es el de tener una auditoria completa de los cambios hechos en base de datos, almacenar información adicional que no podía ser recolectada (cómo el usuario, navegador, dirección IP, etc) por la herramienta que ya se dispone en Oracle.

### Tecnologías

Para ésta historia se hizo uso de una tecnología llamada **EntityListener** la cual funciona a nivel del ORM (el cual es Hibernate y jpa). Se encarga de escuchar cualquier cambio persistente en base de datos de todas las entidades del proyecto, fue configurado con las anotaciones `@PreUpdate`, `@PrePersist` y `@PreRemove`, esto quiere decir que se hace el control inmediatamente antes de que los cambios sean efectuados en base de datos.

### Paso a paso

El primer paso fue configurar a través de un archivo llamado `orm.xml` la clase que va a ser instanciada como `@Component` que escuchará todos los cambios de cualquier entidad por parte de **Hibernate**.

Luego dentro de esta clase se desarrolló el método llamado `handleLifecycleEvent()` que contiene toda la lógica de negocio, para esto se hizo uso de la tecnología **MDC**, encargada de generar un id único por cada petición dentro del servidor, ésta era una variable temporal que nos ayudaba a manejar correctamente el flujo de operaciones persistidas a la base de datos, y de ésta manera insertar un único registro en la tabla de auditoria por transacción. Una vez se captura la información de la primera operación que va ser hecha por parte de **Hibernate** se inserta el registro en la tabla de auditoria junto con las demás operaciones que vengan de la transacción.

```

2  * Método principal, el cual maneja los eventos a nivel de hibernate, en este c
3  *
4  * @param entity
5  */
6  @Transactional
7  @PreUpdate
8  @PrePersist
9  @PreRemove
10 public void handleLifecycleEvent(Object entity) {
11     if (MDC.get(IConstantes.REQUEST_IDENTIFIER) == null || Objects.equals(MDC.get(
12         || entity instanceof Log)
13         return;
14     var auditor = new AuditoriaBackend();
15     var ipUser = getIpUser();
16     auditor.setIp(ipUser == null || ipUser.isEmpty() ? "No identificada" : ipUser);
17     auditor.setConexion(dbUser);
18     var userAgent = httpRequest.getHeader(IConstantes.NAVEGADOR);
19     var user = userAgent.toLowerCase(Locale.ROOT);
20     auditor.setNavegador(getNavegador(user, userAgent));
21     auditor.setSistemaOperativo(getOs(userAgent));
22     auditor.setIdTransaccion(MDC.get(IConstantes.REQUEST_IDENTIFIER));
23     var rolHeader = httpRequest.getHeader(IConstantesAdministracionPersonal.ROL);
24     var userHeader = httpRequest.getHeader(IConstantesAdministracionPersonal.USER);
25     auditor.setIdRol(Long.parseLong(rolHeader != null && !rolHeader.isEmpty() ? rolHeader : ""));
26     auditor.setIdUsuario(Long.parseLong(userHeader != null && !userHeader.isEmpty() ? userHeader : ""));
27     auditor.setProyecto(projectName.split("/")[1]);
28     MDC.clear();
29     auditoriaBackendRepository.save(auditor);
30     applicationEventPublisher.publishEvent(new Evento<>(entity));
31
32 }

```

También se hizo uso de la tecnología **EventListener**, esto con el fin de insertar un registro en la tabla de auditoria backend por transacción con operaciones de base de datos y no solo por petición.

```

1  @TransactionalEventListener(phase = TransactionPhase.AFTER_COMMIT)
2  public void handleCustom(Evento<Object> evento){
3      var uuid = UUID.randomUUID().toString().toUpperCase().replace("-", "");
4      MDC.put(IConstantes.REQUEST_IDENTIFIER, uuid);
5  }

```

## PRUEBAS

1. **Verificar que el registro de auditoria\_backend quede guardada de forma atómica junto con la transacción involucrada.**
  - ▶ Se hizo esta verificación tanto para transacciones con una operación o con varias operaciones.
2. **Solo se permite guardar un registro por transacción, no por operación de inserción, actualización, borrado (es decir si una transacción tiene varias operaciones, debe quedar solo un registro).**
  - ▶ Se hicieron varias pruebas con diferentes operaciones dentro de una misma transacción y se comprobó la inserción de un solo registro.
3. **Se debe probar que ocurre con esa inserción cuando hay excepciones.**
  - ▶ Si hay alguna excepción que involucre el rollback de las operaciones que van a ser persistidas dentro de la transacción no se hace ninguna inserción dentro de la tabla de auditoria.
4. **Se debe probar que ocurre con esa inserción cuando no hay excepciones.**
  - ▶ Cuando no hay excepciones se inserta correctamente el registro en la tabla de auditoria.
5. **Se debe probar el comportamiento con varias operaciones en un @transactional, que haya varios save, un delete, etc. (Múltiples operaciones en una transacción).**
  - ▶ Se hizo la prueba con las operaciones save, flush, save de nuevo, delete y get dentro de una misma transacción arrojando como resultado un comportamiento normal con una única inserción dentro de la tabla de auditoria.
6. **Pruebas crud básicos y complejos.**
  - ▶ Se hicieron pruebas en la tabla CLASE\_SITU\_ADMIN y RH\_TIPOS\_CONTRATO, haciendo todas las operaciones de CRUD.
7. **Se debe probar el comportamiento cuando se inserta una entidad y luego más adelante se actualiza la misma en un @transactional (Múltiple operación sobre un mismo registro).**
  - ▶ Se hizo la prueba de esto, mostrando un comportamiento correcto, una sola inserción en la tabla de auditoria a pesar de múltiples operaciones que hace Hibernate sobre un registro.
8. **Se debe probar el comportamiento cuando se usan las operaciones de un @transactional en un mismo método.**
  - ▶ Se hizo esta prueba encontrando que estaba guardando un registro por petición, se hizo la corrección con la ayuda de la tecnología EventListener y de ésta manera se logró dar solución. Ahora guarda un registro por transacción.
9. **La inserción en auditoria\_backend debe quedar cuando se haga un commit en base de datos, es decir que quede junto con las tablas analizadas, no antes ni después.**
  - ▶ Quedó probado este ítem, esto con el flashback de Oracle y la vista para mirar la auditoria de la tabla CLASE\_SITU\_ADMIN.

**10. Se debe probar con transacciones nativas a ver si es posible o no.**

- ▶ No se presentaron inconvenientes en las pruebas realizadas.

**11. Se debe comprobar que las transacciones no provoquen deadlock ni lazy initialization.**

- ▶ No se evidenció comportamiento de este tipo.

**12. Se debe probar las transacciones con el entitymanager.**

- ▶ Se hizo prueba de ésto y funciona correctamente con todo lo implementado para auditoria.

**13. Se debe probar las transacciones que se invoquen desde @repository.**

- ▶ Se probó haciendo uso de la anotación @Modifying, ésta no arrojó un comportamiento deseado, ya que el EntityHibernateListener no escucha los cambios producidos por estos métodos.

**14. Se debe probar si las inserciones deben hacerse pre o post transacción (OJO al final del commit).**

- ▶ Las inserciones se están realizando inmediatamente antes (@PreUpdate, @PrePersist, @PreDelete) de la cola de operaciones de Hibernate.