**Caso de estudio “Registrar usuario”**

Asignatura: PWeb

Proyecto: Backend con Spring Boot

Objetivo: Guiar el desarrollo en el backend, crear clases controladoras, las rutas y servicios.

Antes de comenzar con el caso de edstudio en necesario comprender que en la asignatura deben implementar **dos proyectos**:

1. Uno para el fronted, basado en Jsf + Primefaces donde están los “beans” de Jsf, las páginas web en formato “xhtml” y entre otras cosas un paquete “service” donde se define una clase servicio con su interfaz. Desde estos servicios serán llamados a los “Endpoints” del proyecto backend.
2. Un proyecto Backend, basado en Spring Boot + Java, en este caso no hay páginas web solo clase, ficheros e interfaces en Java.

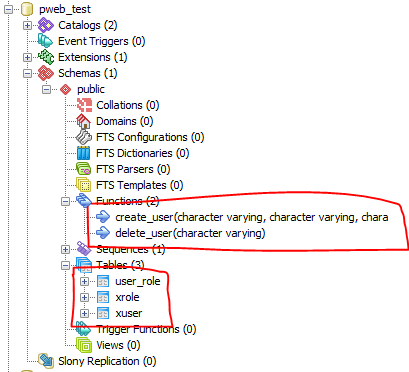
Nota: Tengan en cuenta que los Dto que está en el frontend Jsf son los mismos que estarán en el paquete Dto del backend. Estos Dto serán utilizandos para realizar el mapeo entre aplicaciones.

**El caso de estudio cuenta con:**

1. Un backup de la base de datos de prueba en Postgresql
2. El repositorio actualizado que deben descargarlo nuevamente del icloud (…/herramientas/maven/repository.rar)
3. Videos para las pruebas con swagger.
4. Manual que documenta todos los pasos del caso de estudio.

Para dar inicio es necesario seguir todos los pasos y comprender todas las explicaciones, de tal forma que comprendan el fujo.

**Paso 1.** Restaurar en el PostgreSql la base de datos “pweb\_test.backup” y al terminar este proceso revisar que tengan creadas las funciones y las tablas como se muestran a continuación:



**Paso 2.** Adicionar al proyecto las dependencias necesarias para el **acceso a los datos**. Teniendo en cuenta la precedencia de la asignatura Base de datos estaremos utilizando el mismo mecanismo con el menor cambio posible, por tanto, todos los servicios utilizados en el proyecto de BD lo utilizarán aquí mismo sin cambiar prácticamente nada, solamente la manera de obtener la conexión. Adicionar las siguientes dependencias en el pom.xml.

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-jdbc</artifactId>

</dependency>

<dependency>

<groupId>org.postgresql</groupId>

<artifactId>postgresql</artifactId>

<scope>runtime</scope>

</dependency>

La primera responde a la manera en que spring garantiza el acceso a los datos mediante JDBC y la segunda no es más que a dependencia para el driver del PostgreSql. En el caso de estudiantes que por la línea de investigación utilicen otro gestor de base de datos pues solo deben buscar la dependencia acorde al gestor.

**Paso 3.** Antes de ejecutar Maven install deben primero configurar la conexión a la base de datos para que el install no de error. La configuración a la base de datos se hace en el fichero de configuración que trae Spring Boot que es el “application.properties”. La configuración queda la siguiente manera:

spring.datasource.url=jdbc:postgresql://localhost:5433/pweb\_test

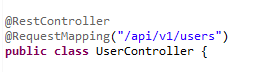
spring.datasource.username=postgres

spring.datasource.password=postgres

**Nota**: Tener en cuenta que en este ejemplo el puerto es el 5433 pero siempre dependerá del puerto configurado en su PostgreSql. Lo mismo con el username y el password.

Con esta configuración se le indica al proyecto y a Spring Boot cuál es la configuración de la base de datos, que luego será utilizada en los servicios.

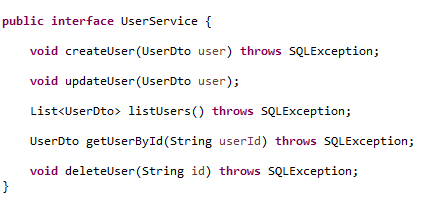
**Paso 4.** Dentro del paquete “controller” crear una nueva clase llamada UserController, indicando que es un @RestController y además definiendo la ruta en entrada usando la anotación @RequestMapping. De esta manera la clase Rest controller queda:



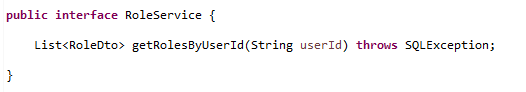
Recordar siempre que la anotación @RestController le indica a Spring Boot que esa clase debe registrarla en su contexto para exponerla como un servicio web rest, por tanto, **NO PUEDE FALTAR.**

La anotación @RequestMapping indica la ruta de entrada a los endpoints que están dentro de ese contralador.

**Paso 5.** Respetando la estructura del proyecto, crear la interfaz UserService dentro del paquete “core.service” y definir ahí los métodos que serán utilizados. Para esto solo deben fijarse en la implementación de sus servicios en el proyecto de BD y lo que deben hacer es crearle una interfaz con los métodos. En este caso queda la siguiente manera:



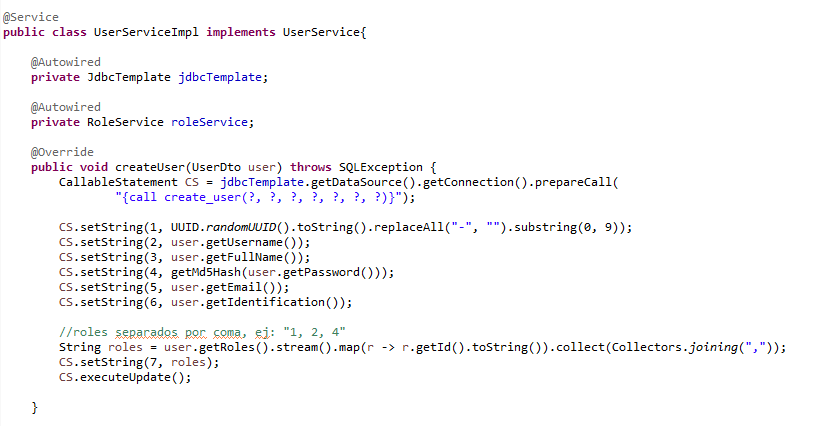
Igualmente se crea una interfaz para el servicio RoleService quedando de la siguiente manera:



**Paso 5.** Crear las clases que implementan estas interfaces dentro del paquete “service” y tengan en cuenta que no están ubicadas en el mismo paquete la interfaz y la implementación. La primera es la clase UserServiceImpl que debe tener arriba la anotación @Service que **NO PUEDE FALTAR.**

**Nota:** Estas clases servicios son los mismos servicios de BD solo cambia la manera en que se obtiene la conexión.

Un fragmento de la clase **UserServiceImpl** queda de la siguiente manera:



En este caso deben apreciar lo siguiente:

1. El método createUser(UserDto user) tiene el mismo estilo de los métodos en los servicios implementados en el proyecto de base de datos, por tanto, **todo lo que ya tienen hecho de base de datos les funcionará aquí.** En este caso se llama a la función que está en el postgres “create\_user”. La única diferencia con el código que ya tienen es:



que en sus proyectos la conexión la obtienen de una clase “ConnectionManager”, por tanto, lo que tendrían que cambiar es esa parte del código porque el resto es lo mismo.

1. La inyección de **private** JdbcTemplate jdbcTemplate; usando la anotación @Autowired. Esta es una interfaz de Spring que se encarga de manejar todo lo relacionado al acceso a datos mediante Jdbc. Esto **no es “magia”** sencillamente como se define la configuración de la bd en el fichero application.properties Spring Boot a través de una de sus dependencias, que es Spring Jdbc, es capaz de obtener la conexión de la misma forma que se hacía en Base de datos, pero ahora de una manera más sencilla.

El resto de la implementación pueden verla en el demo.

**Paso 6.** Una vez creado los servicios pasamos a los métodos dentro del UserController. Esto se crean usando las anotaciones que brinda Spring Boot acorde a los diversos métodos Http:

@GetMapping => Esta anotación es para obtener, ya se una lista, un objeto o un valor lineal.

@PostMapping => Utilizado para insertar, por tanto, el método recibe un objeto como parámetro, en este caso el Dto, pero para que spring pueda realizar el mapeo del Json que viene, al objeto se le pone delante @RequestBody que como su nombre indica no es más que el cuerpo de la solicitud.

@PutMapping => Muy similar al Post pero utilizado para modificaciones. En este caso cuando se va a modificar el objeto completo. Pero en ocasiones se utiliza @PatchMapping porque se trata de modificaciones sobre uno o algunos campos del objeto.

@DeleteMapping => Como su nombre lo indica se utiliza para borrar.

Finalmente, el UserController queda de la siguiente manera:



Como pueden apreciar en el controller se inyectan los servicios utilizando el @Autowired y deben recordar que siempre **se inyectan las interfaces**.

**Nota**: Aunque en este controller siempre se devuelve Ok() también es posible devolver errores, teniendo en cuenta validaciones que necesiten hacer y para eso utilizan el badRequest(). Un ejemplo sería:



**Consideraciones Finales**

1. Para utilizar los servicios que ya tienen de base de datos deben realizar cinco pasos:
   1. Copiar los servicios para el paquete “service”.
   2. Cambiarle el nombre agregando al final “Impl” y arriba de la clase la anotación @Service.
   3. Agregar una interfaz con los mismos métodos por cada servicio en el paquete “core.service” y luego implementarla en su respectiva clase servicio.
   4. Inyectar en cada servicio la interfaz JdbcTemplate
   5. Cambiar la manera en que obtenían la conexión por: jdbcTemplate.getDataSource().getConnection()
2. Una de las ventajas más significativas del backend y swagger es que se pueden implementar todo y probar usando swagger sin necesidad de esperar por la implementación del front.