UNIVERSIDAD DE LA FRONTERA

DCI
DEPARTAMENTO DE COMPUTACION E INFORMATICA

Desafío 5

El objetivo de este desafío es poner en práctica todo lo aprendido sobre Spring Boot que se ha desarrollado durante las clases.

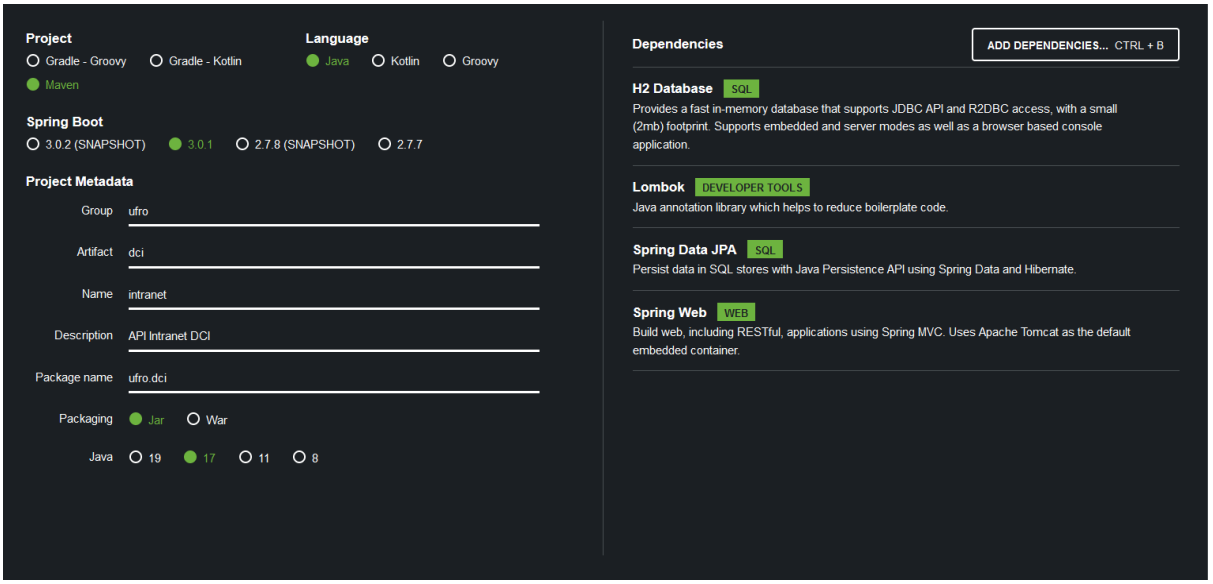
Se le ha encargado desarrollar una API para el departamento de intranet DCI. Una de las funcionalidades más importantes de esta API es la de permitir el registro y el inicio de sesión de los usuarios.

Para el registro, la API debe permitir al administrador ingresar el nombre, dirección de correo electrónico (único), contraseña y rol del usuario. Una vez se tenga toda la información requerida y ha sido verificada, la API debe crear una cuenta para el usuario y almacenar la información proporcionada en la base de datos.

Para el inicio de sesión, la API debe permitir que el usuario proporcione su dirección de correo electrónico y contraseña. La API debe verificar que esta información coincida con la almacenada en la base de datos para el usuario correspondiente y, si es así, permitir que el usuario inicie sesión en la página retornando el rol del usuario. Si la información proporcionada no es válida, la API debe rechazar el inicio de sesión y proporcionar un mensaje de error apropiado. Adicionalmente, y por motivos de seguridad, se requiere que este inicio de sesión tenga un máximo de 3 intentos, al tercer intento la cuenta debe quedar en un estado bloqueado.

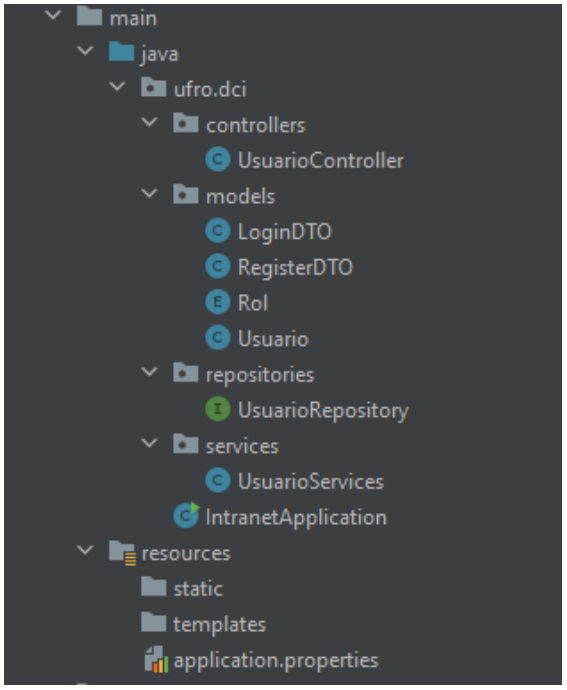
Solución Desafío

Primero, generamos el proyecto a través de la herramienta de Spring Initializr, la configuración a utilizar será la siguiente.



- H2 Database: Como nuestro sistema de base de datos.
- Lombok: Para reducir el código repetido
- Spring Data JPA: Para manejar la persistencia de los objetos en nuestra base de datos
- Spring Web: Ya que la aplicación corresponderá a una API.

Una vez descargado y extraído el archivo zip generado por la página, estructuramos nuestro proyecto de la siguiente manera.



Primero, definiremos las entidades a utilizar en nuestro proyecto.

```
@Entity
@AllArgsConstructor
@NoArgsConstructor
@Getter
@Setter
public class Usuario {

   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private long id;

   @Column(nullable = false)
   private String nombre;

   @Column(nullable = false)
   private String correo;

   @Column(nullable = false)
   private String contrasenna;

   @Enumerated(EnumType.STRING)
   @Column(nullable = false)
   private Rol rol;

   @Column(nullable = false)
   private int intentos;

   @Column(nullable = false)
   private boolean bloqueado;

}
```

En la clase usuarios, guardaremos la información referente al Usuario. Al inicio agregaremos las etiquetas de lombok para ahorrar ciertas porciones de código. También definimos que nuestros atributos no puedan ser nulos.

```
public enum Rol {
   ADMIN,PERSONAL,PROFESOR,ALUMNO
}
```

Creamos un enum que contendrá los roles para nuestros usuarios.

```java
@Data
@AllArgsConstructor
@NoArgsConstructor
public class LoginDTO {
   private String correo;
   private String contrasenna;
}
```

```java
@Data
@AllArgsConstructor
@NoArgsConstructor
public class RegisterDTO {
   private String nombre;
   private String correo;
   private String contrasenna;
   private Rol rol;
}
```

Finalmente, creamos las clases DTO, las cuales permitirán enviar la información a nuestra API.

Como necesitamos controlar los diferentes aspectos en el inicio de sesión de nuestros usuarios, verificando múltiples elementos (Si el usuario ya existe, que los datos no sean nulos, etc), la manera más sencilla es a través de Excepciones de Java manejadas, entonces al encontrar un error responderemos una solicitud incorrecta, pero si la solicitud cumple todos los requisitos, retornaremos una solicitud correcta. Para ello, realizaremos la siguiente configuración en nuestro controlador.

```java
@RestController
@RequestMapping("/api/usuario/")
public class UsuarioController {

   @Autowired
   private UsuarioServices usuarioServices;

   @PostMapping("registrar")
   public ResponseEntity<?> registrarUsuario(@RequestBody RegisterDTO
registerDTO){
       try{
           usuarioServices.registrarUsuario(registerDTO);
       }catch (Exception e){
           return ResponseEntity.badRequest().body(e.getMessage());
       }
       return ResponseEntity.status(HttpStatus.CREATED).body("Usuario
Registrado correctamente");
```

```
    }

    @GetMapping("login")
    public ResponseEntity<?> login(@RequestBody LoginDTO loginDTO){
        try {
            Rol rol = usuarioServices.login(loginDTO);
            return ResponseEntity.accepted().body(rol);
        }catch (Exception e){
            return ResponseEntity.badRequest().body(e.getMessage());
        }
    }
}
```

Así, en nuestro controlador manejaremos las entradas y salidas de nuestro programa, al detectar un error, este responderá automáticamente con un estado de "BadRequest", enviando el mensaje del error lanzado. Caso contrario, si todo ha sido ejecutado sin ningún tipo de problema, retornaremos a un estado positivo.

Siguiendo la estructura de Spring Boot, realizaremos toda la lógica de nuestra aplicación en nuestra clase de Servicio. La lógica que utilizaremos para la resolución de este desafío será la siguiente.

Registro:

1. Verificar que la solicitud de registro venga con correctamente, es decir que tenga todos los campos requeridos, y estos sean distintos de "null".
2. Verificar que no exista otra cuenta con el mismo correo electrónico a registrar.
3. Si todo lo anterior es correcto, se procederá a crear el objeto y guardarlo en nuestra base de datos.

Login:

1. Verificar si el correo electrónico se encuentra registrado
2. Verificar si el usuario está bloqueado en el sistema
3. Verificar si la contraseña es correcta, en caso contrario sumar 1 al contador de intentos de inicio de sesión
4. Si todo es correcto, devolver un mensaje con el rol del usuario.

Antes de realizar nuestra clase de Servicio, nos percatamos de que debemos hacer una consulta a la base de datos preguntando si existe algún usuario con el correo electrónico. Esta consulta será utilizada tanto para el registro como para el login, para realizar esta consulta usaremos las herramientas entregadas a través de JPA y la crearemos en nuestra clase UsuarioRepository, ya que ésta es la encargada de interactuar directamente con la base de datos.

Así, nuestra clase Repository quedará de la siguiente manera.

```
@Repository
public interface UsuarioRepository extends JpaRepository<Usuario,Long> {
   Optional<Usuario> findByCorreo(String correo);
}
```

Como podemos observar, preguntaremos si existe un usuario con determinado correo, esta consulta devolverá un objeto Optional de Java, la cuál nos permitirá realizar la lógica detrás de nuestro Registro y Login.

Teniendo nuestro repositorio listo, procederemos ahora sí, a programar nuestra clase de Servicio. Así, nuestra clase de servicio debe quedar de la siguiente manera.

```
@Service
public class UsuarioServices {
   @Autowired
   private UsuarioRepository usuarioRepository;
   public void registrarUsuario(RegisterDTO registerDTO) throws
Exception {
       // Verificar integridad de los datos
       verificarDatos(registerDTO);
       // Verificar correo electronico unico
       verificarCorreoUnico(registerDTO.getCorreo());
       // Crear objeto Usuario
       Usuario usuario = new Usuario();
       usuario.setNombre(registerDTO.getNombre());
       usuario.setCorreo(registerDTO.getCorreo());
       usuario.setContrasenna(registerDTO.getContrasenna());
       usuario.setRol(registerDTO.getRol());
       usuario.setBloqueado(false);
       usuario.setIntentos(0);
       usuarioRepository.save(usuario);
   }
   private void verificarCorreoUnico(String correo) throws Exception {
       Optional<Usuario> usuario =
usuarioRepository.findByCorreo(correo);
       if (usuario.isPresent()) throw new Exception("Correo ya se
encuentra registrado");
   }
   private void verificarDatos(RegisterDTO registerDTO) throws Exception
{
       if (registerDTO.getNombre() == null) throw new Exception("Nombre
no ingresado");
       if (registerDTO.getCorreo() == null) throw new Exception("Correo
```

```
no ingresado");
        if (registerDTO.getContrasenna() == null) throw new
Exception("Contraseña no ingresada");
        if (registerDTO.getRol() == null) throw new Exception("Rol no
ingresado");
    }
    public Rol login(LoginDTO loginDTO) throws Exception {
        // Verificar si el usuario existe
        Usuario usuario =
usuarioRepository.findByCorreo(loginDTO.getCorreo()).orElseThrow(() ->
new Exception("Usuario no existe"));
        // Verificar si usuario se encuentra bloqueado
        if (usuario.isBloqueado()) throw new Exception("Usuario
Bloqueado");
        // Verificar contraseña correcta
        if(!usuario.getContrasenna().equals(loginDTO.getContrasenna())){
            usuario.setIntentos(usuario.getIntentos()+1);
            if (usuario.getIntentos() > 3) {
                usuario.setBloqueado(true);
                throw new Exception("Exceso de intentos");
            }
            throw new Exception("Contraseña Incorrecta");
        }
        usuario.setIntentos(0);
        return usuario.getRol();
    }
}
```

Realizado todo lo anterior, nuestro desafío quedará resuelto.

Trabajo Futuro

- Agregar que los campos enviados tengan un mínimo de 3 caracteres.
- Probar el funcionamiento de la API a través de Postman o Insomnia.

Lecturas Obligatorias

- Java Lambda Expressions
- Data Transfer Objects (DTO)
- Optional Object Java
- HttpStatus responses
- Test API Postman / Test Api Insomnia

Acceso al github con la solución
https://github.com/Spawnbig/desafios-proyecto-aplicacion