Pruebas Login

Introducción

Bienvenido a nuestra plantilla de gestión de usuarios. Esta solución simple y eficaz incluye funciones para el inicio de sesión, registro de usuarios, asignación de roles, recuperación de contraseñas y una vista de bienvenida. Ideal para quienes buscan una base sólida y fácil de implementar para manejar el acceso y la administración de usuarios.

Contenido

CoreApplication

Verificación de Compilación correctamente

Esta prueba asegura que la aplicación arranca correctamente y que el contexto de Spring se carga sin errores.

Pruebas Incluidas:

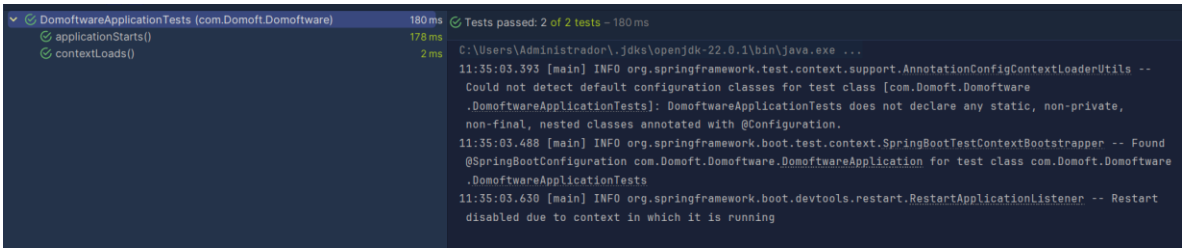
1. **Carga del Contexto (contextLoads()):** Verifica que el contexto de Spring se inicializa correctamente sin errores.
2. **Arranque de la Aplicación (applicationStarts()):** Confirma que el método main de la aplicación se ejecuta sin excepciones, garantizando un arranque exitoso.

```
9       @Test
10      void contextLoads() {
11      }
12
13      @Test
14      void applicationStarts() {
15      }
16  }
17
```

Resultados de la Prueba

1. Ambas pruebas deben completarse sin excepciones. El mensaje de éxito es: "La aplicación se ha inicializado correctamente. El contexto de Spring se ha cargado sin errores y la aplicación ha arrancado sin problemas."

Salida de la Ejecución de la Prueba



ApplicationConfigTest

Verificación de la Configuración de la Base de Datos

Esta prueba valida que los parámetros de configuración de la base de datos se carguen correctamente desde el archivo de propiedades. Verifica la URL, el nombre de usuario y la contraseña de la base de datos.

```
19      @Test
20      public void testDatabaseConfig() {
21          assertEquals( expected: "jdbc:mysql://localhost:3306/login?useSSL=false&serverTimezone=UTC&useLegacyDateTimeCode=false",
22                  env.getProperty("spring.datasource.url"));
23          assertEquals( expected: "root", env.getProperty("spring.datasource.username"));
24          assertEquals( expected: "root", env.getProperty("spring.datasource.password"));
25      }
26
```

Resultados de la Prueba

La prueba pasa si los valores de spring.datasource.url, spring.datasource.username, y spring.datasource.password coinciden con:

- URL: "jdbc:mysql://localhost:3306/login?useSSL=false&serverTimezone=UTC&useLegacyDateTimeCode=false"
- Usuario: "root"
- Contraseña: "root"

Salida de la Ejecución de la Prueba

Verificación de inicio de sesión exitoso (AuthController)

La prueba `testLogin_Success` valida el comportamiento del sistema cuando un usuario proporciona credenciales correctas. Se simula la autenticación con un token válido y se comprueba que el controlador responde con un token JWT generado exitosamente, junto con el código de estado HTTP `200 OK`. Esto asegura que el sistema maneja correctamente un inicio de sesión exitoso y devuelve la información correcta al usuario.

```
63      @Test
64      public void testLogin_Success() {
65          LoginUser loginUser = new LoginUser( username: "usuarioPrueba",  password: "contraseñaSegura");
66          Authentication authentication = mock(Authentication.class);
67          when(authenticationManager.authenticate(any(UsernamePasswordAuthenticationToken.class)))
68                  .thenReturn(authentication);
69          when(jwtProvider.generateToken(authentication)).thenReturn( value: "tokenJwtPrueba");
70          when(userService.existsByUserName(loginUser.getUsername())).thenReturn( value: true);
71
72          ResponseEntity<?> response = authController.login(loginUser, bindingResult);
73
74          assertEquals(HttpStatus.OK, response.getStatusCode());
75          assertTrue(response.getBody() instanceof JwtDto);
76          JwtDto jwtDto = (JwtDto) response.getBody();
77          assertEquals( expected: "tokenJwtPrueba", jwtDto.getToken());
78      }
```

Verificación de usuario no encontrado durante el inicio de sesión (AuthController)

La prueba `testLogin_UserNotFound` verifica cómo el sistema responde cuando un usuario intenta iniciar sesión con un nombre de usuario que no existe en la base de datos. El test asegura que el sistema responda con un código de estado HTTP `401 Unauthorized` y un mensaje de error indicando que el usuario no fue encontrado. Esto valida que el controlador maneja adecuadamente los intentos de inicio de sesión con nombres de usuario no registrados.

```
80      @Test
81      public void testLogin_UserNotFound() {
82          LoginUser loginUser = new LoginUser( username: "usuarioDesconocido",  password: "contraseñaSegura");
83          when(userService.existsByUserName(loginUser.getUsername())).thenReturn( value: false);
84
85          ResponseEntity<?> response = authController.login(loginUser, bindingResult);
86
87          assertEquals(HttpStatus.UNAUTHORIZED, response.getStatusCode());
88          assertTrue(response.getBody() instanceof Mensaje);
89          Mensaje mensaje = (Mensaje) response.getBody();
90          assertEquals( expected: "Usuario no encontrado", mensaje.getMensaje());
91      }
```

Verificación de credenciales incorrectas en inicio de sesión (AuthController)

La prueba `testLogin_BadCredentials` comprueba que el sistema maneje correctamente el caso en que un usuario proporciona una contraseña incorrecta. Se simula una excepción de credenciales incorrectas y se asegura que el sistema responda con un código de estado HTTP `401 Unauthorized` y un mensaje de error adecuado. Esto valida que el controlador maneja adecuadamente los errores de autenticación por contraseñas incorrectas.

```
93      @Test
94      public void testLogin_BadCredentials() {
95          LoginUser loginUser = new LoginUser( username: "usuarioPrueba",  password: "contraseñaSegura");
96          when(userService.existsByUserName(loginUser.getUsername())).thenReturn( value: true);
97          when(authenticationManager.authenticate(any(UsernamePasswordAuthenticationToken.class)))
98                  .thenThrow(new org.springframework.security.authentication.BadCredentialsException("Credenciales incorrectas"));
99
100         ResponseEntity<?> response = authController.login(loginUser, bindingResult);
101
102         assertEquals(HttpStatus.UNAUTHORIZED, response.getStatusCode());
103         assertTrue(response.getBody() instanceof Mensaje);
104         Mensaje mensaje = (Mensaje) response.getBody();
105         assertEquals( expected: "Contraseña incorrecta", mensaje.getMensaje());
106     }
```

Verificación de actualización de token (AuthController)

La prueba `testRefreshToken` valida que el sistema pueda generar un nuevo token JWT a partir de un token antiguo. Se simula la solicitud de actualización de token y se verifica que el controlador responda con un nuevo token y un código de estado HTTP `200 OK`. Esto asegura que la funcionalidad de actualización de token está implementada correctamente y el sistema devuelve el token nuevo esperado.

```
108     @Test
109     public void testRefreshToken() throws Exception {
110         JwtDto jwtDto = new JwtDto( token: "tokenViejo");
111         when(jwtProvider.refreshToken(jwtDto)).thenReturn( value: "tokenNuevo");
112
113         ResponseEntity<JwtDto> response = authController.refresh(jwtDto);
114
115         assertEquals(HttpStatus.OK, response.getStatusCode());
116         assertNotNull(response.getBody());
117         assertEquals( expected: "tokenNuevo", response.getBody().getToken());
118     }
119
```

Verificación de registro de usuario exitoso (AuthController)

La prueba `testRegister_Success` valida el comportamiento del sistema cuando un nuevo usuario se registra con éxito. Se simula la creación de un nuevo usuario y se comprueba que el controlador responda con un código de estado HTTP `201 Created` y un mensaje de éxito. Esto asegura que el controlador maneja correctamente el registro de nuevos usuarios y devuelve una respuesta adecuada al éxito del registro.

```
@Test
public void testRegister_Success() {
    NewUser newUser = new NewUser( name: "Juan Pérez", username: "juanperez", email: "juan.perez@ejemplo.com", password: "contraseñaSegura123", n
    User user = new User( name: "Juan Pérez", username: "juanperez", email: "juan.perez@ejemplo.com", password: "contraseñaCodificada");

    when(userService.existsByUserName(anyString())).thenReturn( value: false);
    when(userService.existsByEmail(anyString())).thenReturn( value: false);
    when(rolService.getByRolName(RolName.ROL_USER)).thenReturn(Optional.of(new Rol(RolName.ROL_USER)));

    doNothing().when(userService).save(any(User.class));

    when(bindingResult.hasErrors()).thenReturn( value: false);

    ResponseEntity<?> response = authController.nuevo(newUser, bindingResult);

    assertEquals(HttpStatus.CREATED, response.getStatusCode());
    assertTrue(response.getBody() instanceof Mensaje);
    Mensaje mensaje = (Mensaje) response.getBody();
    assertEquals( expected: "Usuario guardado satisfactoriamente", mensaje.getMensaje());
}
```

Verificación de registro con nombre de usuario existente (AuthController)

La prueba `testRegister_UsernameExists` verifica cómo el sistema maneja el intento de registro de un nuevo usuario con un nombre de usuario que ya existe. Se simula una solicitud de registro con un nombre de usuario ya registrado y se asegura que el controlador responda con un código de estado HTTP `400 Bad Request` y un mensaje indicando que el nombre de usuario ya existe. Esto valida que el controlador maneja adecuadamente los conflictos de nombre de usuario durante el registro.

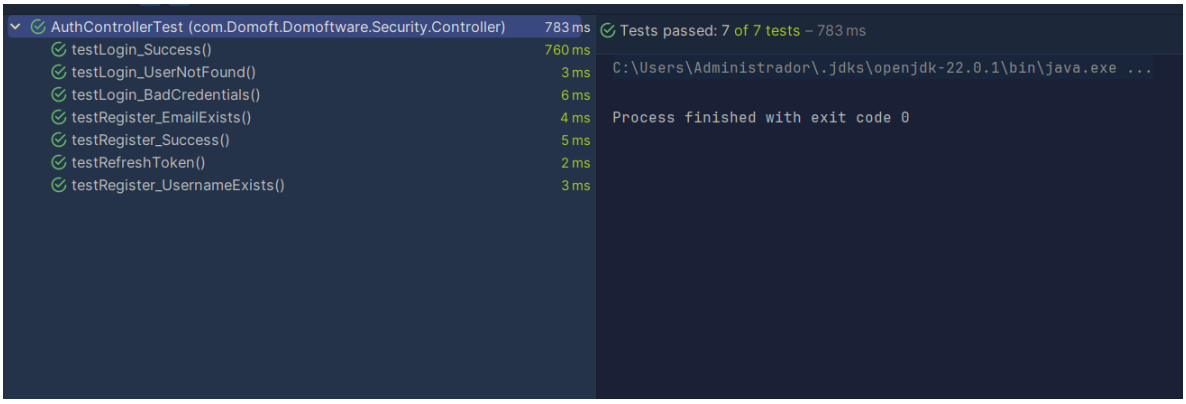
```
@Test
public void testRegister_UsernameExists() {
    NewUser newUser = new NewUser( name: "Juan Pérez", username: "juanperez", email: "juan.perez@ejemplo.com", password: "contraseñaSegura123",

    when(userService.existsByUserName(anyString())).thenReturn( value: true);
    when(bindingResult.hasErrors()).thenReturn( value: false);
    when(passwordEncoder.encode(any(CharSequence.class))).thenReturn( value: "contraseñaCodificada");

    ResponseEntity<?> response = authController.nuevo(newUser, bindingResult);

    assertEquals(HttpStatus.BAD_REQUEST, response.getStatusCode());
    assertTrue(response.getBody() instanceof Mensaje);
    Mensaje mensaje = (Mensaje) response.getBody();
    assertEquals( expected: "Ese nombre ya existe", mensaje.getMensaje());
}
```

Verificación de registro con email existente (AuthController)

La prueba `testRegister_EmailExists` valida cómo el sistema maneja el intento de registro de un nuevo usuario con un email que ya está en uso. Se simula una solicitud de registro con un email ya registrado y se verifica que el controlador responda con un código de estado HTTP `400 Bad Request` y un mensaje indicando que el email ya existe. Esto asegura que el controlador maneja adecuadamente los conflictos de email durante el registro.

```
157     @Test
158     public void testRegister_EmailExists() {
159         NewUser newUser = new NewUser( name: "Juan Pérez",  username: "juanperez",  email: "juan.perez@ejemplo.com",  password: "contras
160
161         when(userService.existsByUserName(anyString())).thenReturn( value: false);
162         when(userService.existsByEmail(anyString())).thenReturn( value: true);
163         when(bindingResult.hasErrors()).thenReturn( value: false);
164
165         ResponseEntity<?> response = authController.nuevo(newUser, bindingResult);
166
167         assertEquals(HttpStatus.BAD_REQUEST, response.getStatusCode());
168         assertTrue(response.getBody() instanceof Mensaje);
169         Mensaje mensaje = (Mensaje) response.getBody();
170         assertEquals( expected: "Ese email ya existe", mensaje.getMensaje());
171     }
172 }
173
```

Resultado de pruebas:



Verificación de obtención de usuario por nombre o nombre de usuario (UserService)

La prueba `testGetBynameOrUsername` valida que el método `getBynameOrUsername` del `UserService` funcione correctamente al buscar un usuario por nombre o nombre de usuario. Se simula la respuesta del repositorio con un usuario que coincide con el nombre proporcionado y se verifica que el resultado devuelto contenga el usuario esperado. Además, se confirma que el método del repositorio fue llamado una vez.

```
30      @Test
31      public void testGetBynameOrUsername() {
32          String nameOrUsername = "testUser";
33          User user = new User();
34          user.setUsername(nameOrUsername);
35          Optional<User> optionalUser = Optional.of(user);
36          when(userRepository.findBynameOrUsername(nameOrUsername, nameOrUsername)).thenReturn(optionalUser);
37
38          Optional<User> result = userService.getBynameOrUsername(nameOrUsername);
39
40          assertTrue(result.isPresent());
41          assertEquals(nameOrUsername, result.get().getUsername());
42          verify(userRepository, times( wantedNumberOfInvocations: 1)).findBynameOrUsername(nameOrUsername, nameOrUsername);
43      }
44
```

Verificación de obtención de usuario por token de contraseña (UserService)

La prueba `testGetByTokenPassword` verifica que el método `getByTokenPassword` del `UserService` retorna un usuario basado en el token de contraseña proporcionado. Se simula el comportamiento del repositorio para devolver un usuario con el token especificado y se asegura que el método del servicio devuelva el usuario correcto. También se comprueba que el repositorio fue invocado una vez.

```
45      @Test
46      public void testGetByTokenPassword() {
47          String tokenPassword = "testToken";
48          User user = new User();
49          user.setTokenPassword(tokenPassword);
50          Optional<User> optionalUser = Optional.of(user);
51          when(userRepository.findByTokenPassword(tokenPassword)).thenReturn(optionalUser);
52
53          Optional<User> result = userService.getByTokenPassword(tokenPassword);
54
55          assertTrue(result.isPresent());
56          assertEquals(tokenPassword, result.get().getTokenPassword());
57          verify(userRepository, times( wantedNumberOfInvocations: 1)).findByTokenPassword(tokenPassword);
58      }
59
```

Verificación de obtención de usuario por nombre de usuario (UserService)

La prueba `testGetByUserName` valida que el método `getByUserName` del `UserService` devuelve el usuario correcto basado en el nombre de usuario proporcionado. Se simula una respuesta del repositorio con un usuario que tiene el nombre de usuario especificado y se verifica que el resultado es el esperado. Se confirma que el repositorio fue llamado una vez.

```
@Test
public void testGetByUserName() {
    String username = "testUsername";
    User user = new User();
    user.setUsername(username);
    Optional<User> optionalUser = Optional.of(user);
    when(userRepository.findByUsername(username)).thenReturn(optionalUser);

    Optional<User> result = userService.getByUserName(username);

    assertTrue(result.isPresent());
    assertEquals(username, result.get().getUsername());
    verify(userRepository, times( wantedNumberOfInvocations: 1)).findByUsername(username);
}
```

Verificación de existencia de nombre de usuario (UserService)

La prueba `testExistsByUserName` asegura que el método `existsByUserName` del `UserService` correctamente determina si un nombre de usuario existe en la base de datos. Se simula que el repositorio indica que el nombre de usuario existe y se verifica que el método del servicio retorne el valor esperado. También se confirma que el repositorio fue llamado una vez.

```
@Test
public void testExistsByUserName() {
    String username = "testUsername";
    when(userRepository.existsByUsername(username)).thenReturn( value: true);

    boolean result = userService.existsByUserName(username);

    assertTrue(result);
    verify(userRepository, times( wantedNumberOfInvocations: 1)).existsByUsername(username);
}
```

Verificación de existencia de email (UserService)

La prueba `testExistsByEmail` valida que el método `existsByEmail` del `UserService` determina correctamente si un email ya está registrado. Se simula una respuesta positiva del repositorio y se asegura que el método del servicio devuelva el resultado esperado. También se verifica que el repositorio fue llamado una vez.

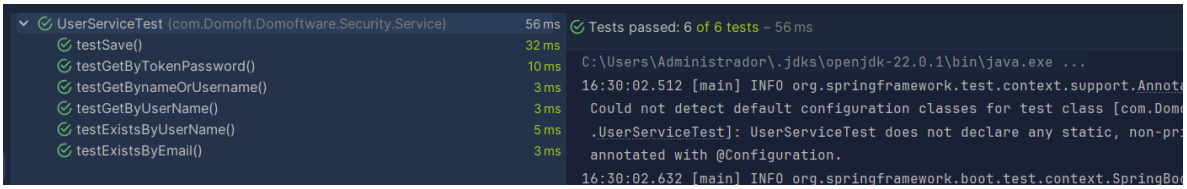
```
86      @Test
87      public void testExistsByEmail() {
88          String email = "test@example.com";
89          when(userRepository.existsByEmail(email)).thenReturn( value: true);
90
91          boolean result = userService.existsByEmail(email);
92
93          assertTrue(result);
94          verify(userRepository, times( wantedNumberOfInvocations: 1)).existsByEmail(email);
95      }
```

Verificación de guardado de usuario (UserService)

La prueba `testSave` comprueba que el método `save` del `UserService` llama correctamente al método `save` del repositorio para guardar un usuario. Se simula la llamada al repositorio y se verifica que el método `save` del repositorio sea invocado una vez con el usuario proporcionado.

```
97      @Test
98      public void testSave() {
99          User user = new User();
100
101         userService.save(user);
102
103         verify(userRepository, times( wantedNumberOfInvocations: 1)).save(user);
104     }
105 }
106
```

Resultado de pruebas:

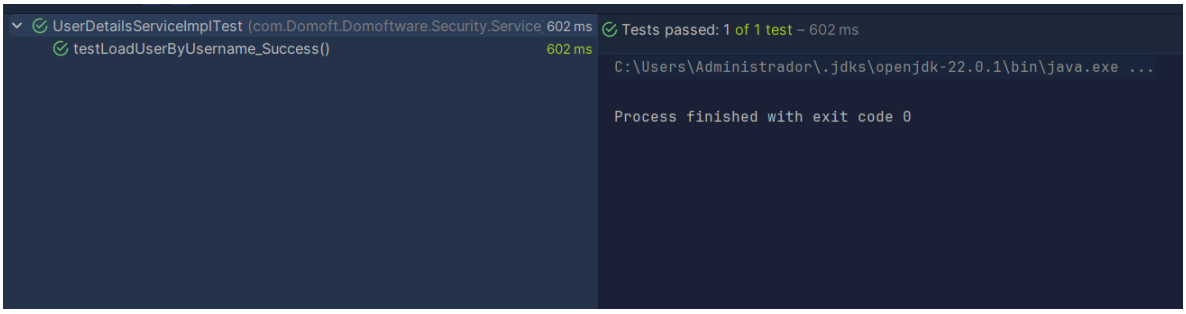


Verificación de carga de usuario por nombre de usuario (UserDetailsServiceImpl)

La prueba `testLoadUserByUsername_Success` valida que el método `loadUserByUsername` del `UserDetailsServiceImpl` funcione correctamente al cargar un usuario basado en su nombre de usuario. Se simula que el `UserService` devuelve un usuario que coincide con el nombre de usuario proporcionado. Se verifica que el objeto `UserDetails` no sea nulo y que el nombre de usuario coincida con el esperado. Además, se confirma que el método `getByUserName` del `UserService` fue llamado una vez.

```
31
32      @Test
33      public void testLoadUserByUsername_Success() {
34          String username = "testUser";
35          User user = new User();
36          user.setUsername(username);
37          when(userService.getByUserName(username)).thenReturn(Optional.of(user));
38
39          UserDetails userDetails = userDetailsService.loadUserByUsername(username);
40
41          assertNotNull(userDetails);
42          assertEquals(username, userDetails.getUsername());
43          verify(userService, times( wantedNumberOfInvocations: 1)).getByUserName(username);
44      }
45
46  }
```

Resultado de pruebas:



Verificación de recuperación de rol por nombre (RolService)

La prueba `testGetByRolName_Success` valida que el método `getByRolName` del `RolService` funcione correctamente al recuperar un rol basado en el nombre de rol proporcionado. Se simula que el `RolRepository` devuelve un rol con el nombre especificado. Se verifica que el resultado no esté vacío y que el nombre del rol coincida con el esperado. Además, se confirma que el método `findByRolName` del `RolRepository` haya sido llamado una vez.

```
30      @Test
31      public void testGetByRolName_Success() {
32          RolName rolName = RolName.ROL_USER;
33          Rol rol = new Rol();
34          rol.setRolName(rolName);
35          Optional<Rol> optionalRol = Optional.of(rol);
36          when(rolRepository.findByRolName(rolName)).thenReturn(optionalRol);
37
38          Optional<Rol> result = rolService.getByRolName(rolName);
39
40          assertTrue(result.isPresent());
41          assertEquals(rolName, result.get().getRolName());
42          verify(rolRepository, times( wantedNumberOfInvocations: 1)).findByRolName(rolName);
43      }
44
```

Verificación de recuperación de rol por nombre no encontrado (RolService)

La prueba `testGetByRolName_NotFound` valida que el método `getByRolName` del `RolService` maneje correctamente el caso en que no se encuentra un rol con el nombre de rol proporcionado. Se simula que el `RolRepository` devuelve un `Optional.empty()`. Se verifica que el resultado esté vacío y que el método `findByRolName` del `RolRepository` haya sido llamado una vez.

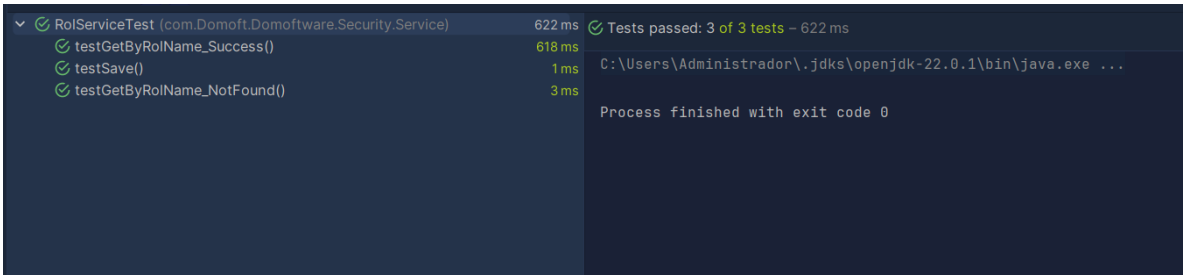
```
45      @Test
46      public void testGetByRolName_NotFound() {
47          RolName rolName = RolName.ROL_ADMIN;
48          when(rolRepository.findByRolName(rolName)).thenReturn( value: Optional.empty());
49
50          Optional<Rol> result = rolService.getByRolName(rolName);
51
52          assertFalse(result.isPresent());
53          verify(rolRepository, times( wantedNumberOfInvocations: 1)).findByRolName(rolName);
54      }
```

Verificación de guardado de rol (RolService)

La prueba `testSave` valida que el método `save` del `RolService` funcione correctamente al guardar un rol en la base de datos. Se simula que el `RolRepository` guarda el rol sin errores. Se confirma que el método `save` del `RolRepository` haya sido llamado una vez con el rol proporcionado.

```
55
56      @Test
57      public void testSave() {
58          Rol rol = new Rol();
59
60          rolService.save(rol);
61
62          verify(rolRepository, times( wantedNumberOfInvocations: 1)).save(rol);
63      }
64  }
65
```

Resultado de pruebas:



Verificación de envío de correo cuando el usuario no se encuentra (EmailPasswordController)

La prueba `testSendEmailTemplate_UserNotFound` valida que el método `sendEmailTemplate` del `EmailPasswordController` maneje correctamente el caso en que no se encuentra un usuario con el correo electrónico proporcionado. Se simula que el `UserService` no encuentra ningún usuario con el correo electrónico dado. Se verifica que la respuesta tenga el estado HTTP `NOT_FOUND` y que el mensaje de respuesta indique que no se encontró ningún usuario con las credenciales proporcionadas.

```
46
47      @Test
48      public void testSendEmailTemplate_UserNotFound() {
49          EmailValuesDTO dto = new EmailValuesDTO();
50          dto.setMailTo("test@example.com");
51
52          when(userService.getBynameOrUsername(dto.getMailTo())).thenReturn( value: Optional.empty());
53
54          ResponseEntity<?> response = emailPasswordController.sendEmailTemplate(dto);
55          assertEquals(HttpStatus.NOT_FOUND, response.getStatusCode());
56          assertEquals( expected: "No existe ningún usuario con esas credenciales", ((Mensaje) response.getBody()).getMensaje(
57      }
58
```

Verificación de envío de correo exitoso (EmailPasswordController)

La prueba `testSendEmailTemplate_Success` valida que el método `sendEmailTemplate` del `EmailPasswordController` funcione correctamente cuando se encuentra un usuario con el correo electrónico proporcionado. Se simula que el `UserService` encuentra un usuario con el correo electrónico dado y que el `EmailService` envía el correo electrónico. Se verifica que la respuesta tenga el estado HTTP `OK` y que el mensaje de respuesta indique que el correo electrónico ha sido enviado. Además, se confirma que el `EmailService` ha sido llamado para enviar el correo electrónico.

```
59      @Test
60      public void testSendEmailTemplate_Success() {
61          EmailValuesDTO dto = new EmailValuesDTO();
62          dto.setMailTo("test@example.com");
63
64          User user = new User();
65          user.setEmail("test@example.com");
66          user.setUsername("testUser");
67
68          when(userService.getBynameOrUsername(dto.getMailTo())).thenReturn(Optional.of(user));
69
70          ResponseEntity<?> response = emailPasswordController.sendEmailTemplate(dto);
71
72          assertEquals(HttpStatus.OK, response.getStatusCode());
73          assertEquals( expected: "Te hemos enviado un correo", ((Mensaje) response.getBody()).getMensaje());
74
75          verify(emailService).sendEmail(dto);
76      }
77
```

Verificación de cambios de contraseña con errores de validación (EmailPasswordController)

La prueba `testChangePassword_ValidationErrors` valida que el método `changePassword` del `EmailPasswordController` maneje correctamente los casos en que hay errores de validación en los campos del `ChangePasswordDTO`. Se simula que el `BindingResult` reporta errores de validación. Se verifica que la respuesta tenga el estado HTTP `BAD_REQUEST` y que el mensaje de respuesta indique que los campos están mal puestos.

```
79      @Test
80      public void testChangePassword_ValidationErrors() {
81          ChangePasswordDTO dto = new ChangePasswordDTO();
82          when(bindingResult.hasErrors()).thenReturn( value: true);
83
84          ResponseEntity<?> response = emailPasswordController.changePassword(dto, bindingResult);
85          assertEquals(HttpStatus.BAD_REQUEST, response.getStatusCode());
86          assertEquals( expected: "Campos mal puestos", ((Mensaje) response.getBody()).getMensaje());
87      }
88
```

Verificación de cambio de contraseña cuando las contraseñas no coinciden (EmailPasswordController)

La prueba `testChangePassword_PasswordsDoNotMatch` valida que el método `changePassword` del `EmailPasswordController` maneje correctamente el caso en que las contraseñas proporcionadas no coinciden. Se configura el `ChangePasswordDTO` con contraseñas que no coinciden y se verifica que la respuesta tenga el estado HTTP `BAD_REQUEST` y que el mensaje de respuesta indique que las contraseñas no coinciden.

```
89      @Test
90      public void testChangePassword_PasswordsDoNotMatch() {
91          ChangePasswordDTO dto = new ChangePasswordDTO();
92          dto.setPassword("newPassword");
            dto.setConfirmarPassword("differentPassword");
94
95          ResponseEntity<?> response = emailPasswordController.changePassword(dto, bindingResult);
96          assertEquals(HttpStatus.BAD_REQUEST, response.getStatusCode());
97          assertEquals( expected: "Las contraseñas no coinciden", ((Mensaje) response.getBody()).getMensaje());
98      }
99
```

Verificación de cambio de contraseña cuando el usuario no se encuentra (EmailPasswordController)

La prueba `testChangePassword_UserNotFound` valida que el método `changePassword` del `EmailPasswordController` maneje correctamente el caso en que no se encuentra un usuario con el token de contraseña proporcionado. Se simula que el `UserService` no encuentra ningún usuario con el token dado. Se verifica que la respuesta tenga el estado HTTP `NOT_FOUND` y que el mensaje de respuesta indique que no se encontró ningún usuario con las credenciales proporcionadas.

```
99
100         @Test
101         public void testChangePassword_UserNotFound() {
102             ChangePasswordDTO dto = new ChangePasswordDTO();
103             dto.setPassword("newPassword");
104             dto.setConfirmarPassword("newPassword");
105             dto.setTokenPassword(UUID.randomUUID().toString());
106
107             when(userService.getByTokenPassword(dto.getTokenPassword())).thenReturn( value: Optional.empty());
108
109             ResponseEntity<?> response = emailPasswordController.changePassword(dto, bindingResult);
110
111             assertEquals(HttpStatus.NOT_FOUND, response.getStatusCode());
112             assertEquals( expected: "No existe ningún usuario con esas credenciales", ((Mensaje) response.getBody()).getMensaje())
113         }
114
115
```

Verificación de cambio de contraseña exitoso (EmailPasswordController)

La prueba `testChangePassword_Success` valida que el método `changePassword` del `EmailPasswordController` funcione correctamente cuando el token de contraseña es válido y las contraseñas coinciden. Se simula que el `UserService` encuentra un usuario con el token dado y que el `PasswordEncoder` codifica la nueva contraseña correctamente. Se verifica que la respuesta tenga el estado HTTP `OK` y que el mensaje de respuesta indique que la contraseña ha sido actualizada. Además, se confirma que el `UserService` ha sido llamado para guardar el usuario con la nueva contraseña.

```
116         @Test
117         public void testChangePassword_Success() {
118             ChangePasswordDTO dto = new ChangePasswordDTO();
119             dto.setPassword("newPassword");
120             dto.setConfirmarPassword("newPassword");
121             dto.setTokenPassword(UUID.randomUUID().toString());
122
123             User user = new User();
124             when(userService.getByTokenPassword(dto.getTokenPassword())).thenReturn(Optional.of(user));
125             when(passwordEncoder.encode(dto.getPassword())).thenReturn( value: "encodedPassword");
126
127             ResponseEntity<?> response = emailPasswordController.changePassword(dto, bindingResult);
128             assertEquals(HttpStatus.OK, response.getStatusCode());
129             assertEquals( expected: "Contraseña actualizada", ((Mensaje) response.getBody()).getMensaje());
130
131             verify(userService).save(user);
132         }
133     }
```

Resultado de pruebas: