INFORME DE TEST UNITARIOS BACKEND – SPRINT III

Los tests unitarios en el backend se están realizando con base en el framework Junit. Para este sprint se inició con su implementación en los Service que se han creado hasta el momento, por lo que a continuación detallaré lo realizado.

- **UserServiceImplTest**
  En esta clase se realizó la validación de los métodos crear usuario y cargar usuario. Es importante aclarar que se validaron casos positivos como negativos con el fin de dar una mayor cobertura al testing.
  Se implementó la anotación @Order para que los test se ejecutaran llevando una secuencia predeterminada y de esta forma el resultado de uno pudiera ser la entrada del siguiente.

Nombre y descripción	Código del Test	Resultado

1. **mustCreateUser()**: Valida la posibilidad de crear un usuario ingresando el request correcto. Se utiliza la funcionalidad assertEquals para validar que el mail del request es igual al mail del usuario que se guardó utilizando el service.	`@Test`
	`@Order(1)`
	`@Sql(statements = {"delete from users where email = 'estefy@mail.com'"})`
	`void mustCreateUser() {`
	`  SignUpRequest request = new SignUpRequest("estefy@mail.com","12345678","Estefania","Escarria", "1234","12345678901234567890","3125478965",true);`
	`  UserDetails user = userService.createUser(request);`
	`  assertEquals("estefy@mail.com",user.getUsername());`
	`}`
		✔ Test passed

2. **mustLoadUserByUsername()**: Valida la posibilidad de cargar un usuario luego de que este se ha registrado en el testa nterior. Se utiliza la funcionalidad assertEquals para validar que el mail del request del punto 1 es igual al mail del usuario que se cargó utilizando el service.	`@Test`
	`@Order(2)`
	`void mustLoadUserByUsername() {`
	`  UserDetails user = userService.loadUserByUsername("estefy@mail.com");`
	`  assertEquals("estefy@mail.com",user.getUsername());`
	`}`
		✔ Test passed

3. mustNotCreateExistentUser(): Valida la ejecución de la Excepción EntityExistsException en caso de querer guardar un usuario que ya existe. En este caso se intenta volver a registrar el usuario que se guardó en el paso 1. Se utiliza la funcionalidad assertThrows para validar que efectivamente se active la excepción que está parametrizada para este caso.	`@Test`
	`@Order(3)`
	`void mustNotCreateExistentUser(){`
	`SignUpRequest request = new SignUpRequest("estefy@mail.com","12345678","Estefania","Escarria","1234","12345678901234567890123","3125478965",true);`
	
	`Assertions.assertThrows(EntityExistsException.class,() -> userService.createUser(request));`
	`}`
	
	`}`
		✔ Test passed

4. mustNotCreateUser WithNullParameters(): Valida la ejecución de la Excepción MissingParameterValueException en caso de querer guardar un usuario que tiene algún dato null. En este caso se intenta guardar un usuario con cuill null. Se utiliza la funcionalidad assertThrows para validar que efectivamente se active la excepción que está parametrizada para este caso.	`@Test`
	`@Order(4)`
	`void mustNotCreateUserWithNullParameters(){`
	`SignUpRequest request = new SignUpRequest("ana@mail.com","12345678","Ana","Perez","1234",null,"3125478965",true);`
	
	`Assertions.assertThrows(MissingParameterValueException.class,() -> userService.createUser(request));`
		✔ Test passed

5. mustNotLoadUserByUsername(): Valida la ejecución de la Excepción UsernameNotFoundException en caso de que no se encuentre el username que se intenta cargar. Se utiliza la funcionalidad assertThrows para validar que efectivamente se active la excepción que está parametrizada para este caso.	`@Test`
	`@Order(5)`
	`void mustNotLoadUserByUsername() {`
	
	`Assertions.assertThrows(UsernameNotFoundException.class,() -> userService.loadUserByUsername("ana@mail.com"));`
	
	`}`
		✔ Test passed

- **BarrioServiceImplTest**
  En esta clase se realizó la validación de los métodos crear guardar, buscar y borrar barrios.
  Se implementó la anotación @Order para que los test se ejecutaran llevando una secuencia predeterminada y de esta forma el resultado de uno pudiera ser la entrada del siguiente.

Nombre y descripción	Código del Test	Resultado

1. **guardarTest()**: Valida la posibilidad de guardar un barrio ingresando el request correcto. Se utiliza la funcionalidad assertEquals para validar que el nombre del barrio del request es igual al nombre del barrio que se guardó utilizando el service.	`@Test`
	`@Order(1)`
	`void guardarTest() {`
	`  Barrio barrio = new Barrio();`
	`  barrio.setNombre("Las flores");`
	
	`  Barrio barrioGuardado = barrioService.guardar(barrio);`
	
	`  assertEquals("Las flores",barrioGuardado.getNombre());`
	`}`
		✔ Test passed

2. **buscarTodosTest():** Valida que al ejecutar el método se genere una lista con una longitud mayor a 0, ya que de entrada sabemos que hay datos cargados en la BD (el del punto anterior)
Se utiliza la funcionaildad assertNotEquals para validar que la longitud no sea igual a 0.
	`@Test`
	`@Order(2)`
	`void buscarTodosTest() throws ResourceNotFoundException {`
	`  Integer length = barrioService.buscarTodos().size();`
	
	`  assertNotEquals(0,length);`
	`}`
		✔ Test passed

3. **borrarXIdTest**: Valida que después de haber guardado un barrio (el del punto 1), se pueda buscar con el nombre (método findByNombre), capture su id y luego ejecute el método borrar del service.
Se utiliza la funcionalidad assertThrows para validar que después de borrado si se intenta buscar se ejecute la excepción ResourceNotFoundException.
	`@Test`
	`@Order(3)`
	`void borrarXIdTest() throws ResourceNotFoundException {`
	`  Barrio barrioAEliminar =`
	`barrioRepository.findByNombre("Las flores");`
	`  Long id = barrioAEliminar.getId();`
	
	`  barrioService.borrarXId(id);`
	
	`Assertions.assertThrows(ResourceNotFoundException.class,()->barrioService.borrarXId(id)); }`
		✔ Test passed

- **CategoriaServiceImplTest**

  En esta clase se realizó la validación de los métodos crear guardar, buscar y borrar categorías.

  Se implementó la anotación @Order para que los test se ejecutaran llevando una secuencia predeterminada y de esta forma el resultado de uno pudiera ser la entrada del siguiente.

Nombre y descripción	Código del Test	Resultado

1. **agregarCategoriaTest()**: Valida la posibilidad de agregar una categoría ingresando el request correcto. Se utiliza la funcionalidad assertEquals para validar que el nombre de la categoría del request es igual al nombre de la categoría que se guardó utilizando el service.	`@Test`
	`@Order(1)`
	`void agregarCategoriaTest() {`
	`Categoria categoria = new Categoria();`
	`categoria.setNombre("Natación");`
	`categoria.setDescripcion("Piscinas");`
	`categoria.setUrl("natacion.jpg");`
	
	`Categoria categoriaGuardada = categoriaService.agregarCategoria(categoria);`
	`assertEquals("Natación",categoriaGuardada.getNombre());}`
		✔ Test passed

2. **listarCagetoriasTest():** Valida que al ejecutar el método se genere una lista con una longitud mayor a 0, ya que de entrada sabemos que hay datos cargados en la BD (el del punto anterior)
Se utiliza la funcionaildad assertNotEquals para validar que la longitud no sea igual a 0.
	`@Test`
	`@Order(2)`
	`void listarCategoriasTest() throws ResourceNotFoundException {`
	`Integer listaLength = categoriaService.listarCategorias().size();`
	
	`Assertions.assertNotEquals(0,listaLength);`
	`}`
		✔ Test passed

3. **eliminarCategoriaTest()**: Valida que después de haber guardado una categoría (la del punto 1), se pueda buscar con el nombre (método findByNombre), capture su id y luego ejecute el método borrar del service.
Se utiliza la funcionalidad assertThrows para validar que después de borrado si se intenta buscar se ejecute la excepción ResourceNotFoundException.
	`@Test`
	`@Order(3)`
	`void eliminarCategoriaTest() throws ResourceNotFoundException {`
	`Optional<Categoria> categoria = Optional.ofNullable(categoriaRepository.findByNombre("Natación"));`
	`Long id = categoria.get().getId();`
	`categoriaService.eliminarCategoria(id);`
	
	`Assertions.assertThrows(ResourceNotFoundException.class,()->categoriaService.eliminarCategoria(id));`
	`}`
		✔ Test passed

- **ServicioServiceImplTest**

  En esta clase se realizó la validación de los métodos crear guardar, buscar y borrar servicios.

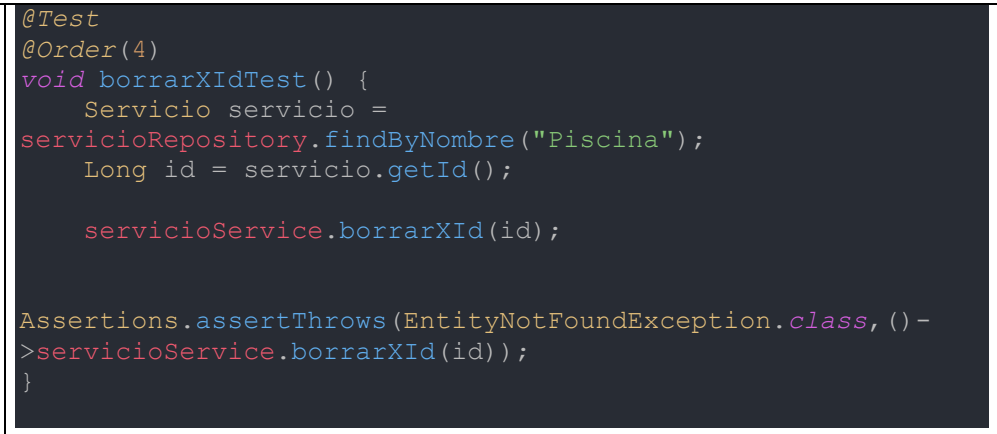
  Se implementó la anotación @Order para que los test se ejecutaran llevando una secuencia predeterminada y de esta forma el resultado de uno pudiera ser la entrada del siguiente.

Nombre y descripción	Código del Test	Resultado

1. **guardarServicioTest()**: Valida la posibilidad de agregar un servicio ingresando el request correcto. Se utiliza la funcionalidad assertEquals para validar que el nombre del servicio del request es igual al nombre del servicio que se guardó utilizando el service.	`@Test`
	`@Order(1)`
	`void guardarServicioTest() {`
	`  Servicio servicio = new Servicio();`
	`  servicio.setNombre("Piscina");`
	`  servicio.setDescripcion("Piscina");`
	`  servicio.setUrl("piscina.jpg");`
	
	`  Servicio servicioGuardado = servicioService.guardarServicio(servicio);`
	
	`  assertEquals("Piscina",servicioGuardado.getNombre());`
	`}`
		✔ Test passed

2. **buscarXIdTest():** Primero ejecuta el método findByNombre el cual busca por el nombre del servicio y devuelve un servicio. Luego toma el id del servicio encontrado y lo busca utilizando el método del service.
Se utiliza la funcionalidad assertTrue para validar que el Optional de Servicio esté presente y no sea nulo.
	`@Test`
	`@Order(2)`
	`void buscarXIdTest() {`
	`  Servicio servicio = servicioRepository.findByNombre("Piscina");`
	`  Long id = servicio.getId();`
	
	`  Optional<Servicio> servicioBuscado = servicioService.buscarXId(id);`
	`  assertTrue(servicioBuscado.isPresent());`
	`}`
		✔ Test passed

3. **buscarTodosTest()**: Valida que al ejecutar el método se genere una lista con una longitud mayor a 0, ya que de entrada sabemos que hay datos cargados en la BD (el del punto 1)
Se utiliza la funcionaildad assertNotEquals para validar que la longitud no sea igual a 0.
	`@Test`
	`@Order(3)`
	`void buscarTodosTest() {`
	`  Integer length = servicioService.buscarTodos().size();`
	
	`  assertNotEquals(0,length);`
	`}`
		✔ Test passed

4. **borrarXIdTest():** Valida que después de haber guardado un servicio (el del punto 1), se pueda buscar con el nombre (método findByNombre), capture su id y luego ejecute el método borrar del service. Se utiliza la funcionalidad assertThrows para validar que después de borrado si se intenta buscar se ejecute la excepción EntityNotFoundException.		✔ Test passed

- **CanchaServiceImplTest**

  En esta clase se realizó la validación del método crear cancha.

  Se implementó la anotación @Order para que los test se ejecutaran llevando una secuencia predeterminada y de esta forma el resultado de uno pudiera ser la entrada del siguiente.

Nombre y descripción	Código del Test	Resultado

1. **mustSaveField()**: Valida la posibilidad de crear una cancha o producto ingresando el request correcto. En caso, los parámetros solicitados son:
• CanchaDTO, la cual se crea utilizando esta clase.
• Token, el cual se genera con la clase JwtService y creando un User.
• File, que hace referencia a las imágenes de la cancha. Esto se simula utilizando el método MockMultipartFile.
Se utiliza la funcionalidad assertEquals para validar que el nombre de la cancha o producto que se ingresó en el request es el mismo que el de la cancha que se guardó utilizando el método del service.
	`@Test`
	`@Order(1)`
	`void mustSaveField() throws Exception {`
	
	`Domicilio domicilio = new Domicilio();`
	`domicilioService.guardar(domicilio);`
	
	`CanchaDTO canchaDTO = new CanchaDTO();`
	`canchaDTO.setNombre("Las pilas");`
	`canchaDTO.setDomicilio(domicilio);`
	`canchaDTO.setPrecio(80000.0);`
	`canchaDTO.setTelefono("1234567891");`
	
	`User user = new User();`
	`user.setName("Pepe");`
	`user.setApellido("Perez");`
	`user.setTelefono("1234567890");`
	`user.setDomicilio(domicilio);`
	`user.setEmail("pepe@gmail.com");`
	`user.setRole(Role.ADMIN);`
	
	`String token = jwtService.generateToken(user);`
	
	`MultipartFile file = new MockMultipartFile("natacion.jpg","imagen".getBytes());`
	
	`Cancha cancha = canchaService.guardar(canchaDTO,token,file);`
	
	`assertEquals("Las pilas",cancha.getNombre());`
	
	`}`
		✔ Test passed