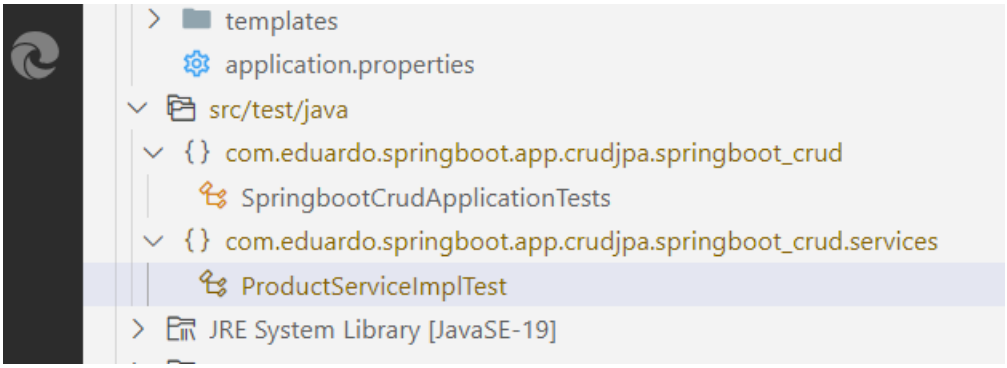
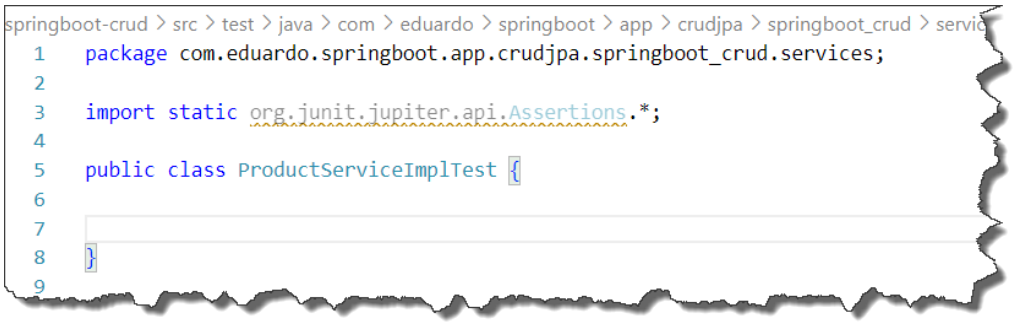
PROBANDO EL “SERVICES”, USANDO MOCKITO.

Usamos nuestro proyecto CRUD para esta prueba.

1. Primero crearemos el package “xxxxxxx.services” (igual como lo tenemos en “src/main/java”) dentro de la ruta **“src/test/java” .** Dentro de este package creamos la clase “ProductServiceImplTest”.





2. Creamos dos atributos en esta clase,
    a. private ProductServiceImpl service;
    b. private ProductoRepository repository;
3. Luego duplicamos la interface “ProductoRepository”, a través de la notación “@Mock”, adicionalmente debemos importar el Mock desde Mockito.

4. Además, debemos agregar la notación @InjectMocks al atributo service. Con esta anotación le estoy diciendo a la clase ProductServiceImpl que escanee todas las clases que tiene la anotación @Mock.

```
5  import org.mockito.InjectMocks;
6  import org.mockito.Mock;
7
8  import com.eduardo.springboot.app.crudjpa.springboot_crud.repository.ProductoRepository;
9
10 public class ProductServiceImplTest {
11
12     @InjectMocks
13     private ProductServiceImpl service;
14
15     @Mock
16     private ProductoRepository repository;
17 }
18
```

5. Creamos un método para que se muestre antes que se realice cada prueba unitaria, para lo cual utilizamos la notación Junit @BeforeEach. En este método invocamos una clase de Mockito para indicarle que abra todos los Mocks para iniciar la configuración necesaria.

```
13 public class ProductServiceImplTest {
14
15     @InjectMocks
16     private ProductServiceImpl service;
17
18     @Mock
19     private ProductoRepository repository;
20
21     @BeforeEach
22     public void init() {
23         MockitoAnnotations.openMocks(this);
24     }
25 }
26
```

6. Ahora crearemos el método para probar “el buscar” de nuestro CRUD. Utilizamos comando claves de Mockito (when,thenReturn), lo que nos indica que cuando se llame al repository.findAll() entonces devolverá una lista, para esto debemos crear un objeto lista(List<>).

```
27      @BeforeEach
28      public void init() {
29          MockitoAnnotations.openMocks(this);
30      }
31
32      @Test
33      public void findByAllTest() {
34
35          when(repository.findAll()).thenReturn(list);
36      }
37  }
38
```

```
23      @Mock
24      private ProductoRepository repository;
25
26      List<Producto> list = new ArrayList<Producto>();
27
28      @BeforeEach
```

7. Crearemos un método para llenar la lista.

```
39      public void chargeProducto(){
40          Producto prod1 = new Producto(Long.valueOf(1:1), nombre:"Galletitas", descripcion:"De chocolate", precio:1500);
41          Producto prod2 = new Producto(Long.valueOf(1:2), nombre:"Donas", descripcion:"rellenas con manjar", precio:850);
42          Producto prod3 = new Producto(Long.valueOf(1:3), nombre:"Queque", descripcion:"de vainilla", precio:12000);
43
44          list.add(prod1);
45          list.add(prod2);
46          list.add(prod3);
47
48      }
```

8. Debemos cargar el método anterior en el método init() para que lo ejecute y cargue la lista.

```
27
28      @BeforeEach
29      public void init() {
30          MockitoAnnotations.openMocks(this);
31
32          this.chargeProducto();
33      }
34
```

9. El siguiente paso será configurar la llamada al método original para que busque los elementos que construimos en nuestra lista ficticia, el objetivo de esto es probar que el método de búsqueda funcione, en este caso sin ir a la base de datos, solo que traiga la lista como corresponde usando esta simulación. Esto lo hacemos en el método "findByAllTest()". Luego evaluamos usando Junit para saber si trajo lo que esperamos, en este caso preguntaremos por la cantidad de elementos de la lista.

```
37    @Test
38    public void findByAllTest() {
39
40        when(repository.findAll()).thenReturn(list);
41
42        List<Producto> response = service.findByAll();
43
44        //Esperamos que traiga los 3 productos que tenemos en la lista que simula la base de datos.
45        assertEquals(expected:3, response.size());
46
47        //Verificamos que se llame al metodo findAll una vez.
48        verify(repository, times(wantedNumberOfInvocations:1)).findAll();
49    }
50
```

10. Por último, ejecutamos la prueba y comprobamos su resultado.