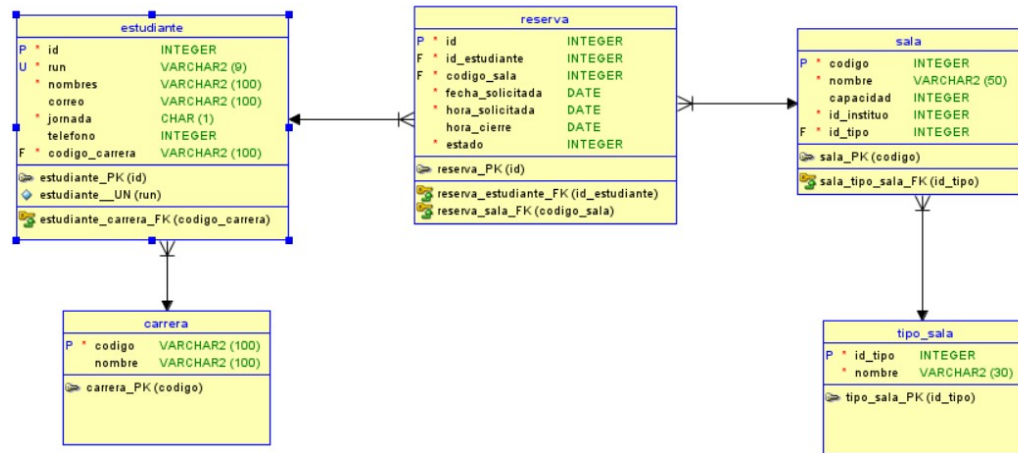


## Guía práctica

### Proyecto reserva de salas biblioteca

### Análisis a los test

En esta guía, detallaremos paso a paso el desarrollo de un proyecto en Spring llamado "Salas Bibliotecas".



## Plan de Prueba de Testing con JUnit 5 para el Proyecto de Reserva de Salas

### Introducción

El plan de prueba pretende verificar la funcionalidad de los servicios del sistema de reserva de salas. Se utilizará **JUnit 5** para las pruebas unitarias y **Mockito** para la simulación de dependencias.

### Estructura de la Base de Datos

La estructura de la base de datos incluye las siguientes tablas:

- Carrera
- Estudiante
- Reserva
- Sala
- Tipo de Sala

## Configuración de Pruebas

Dependencias Maven: agregar las dependencias necesarias en el archivo pom.xml:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>org.mockito</groupId>
  <artifactId>mockito-core</artifactId>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter-engine</artifactId>
  <scope>test</scope>
</dependency>
```

## Problemas al ejecutar un test ✕

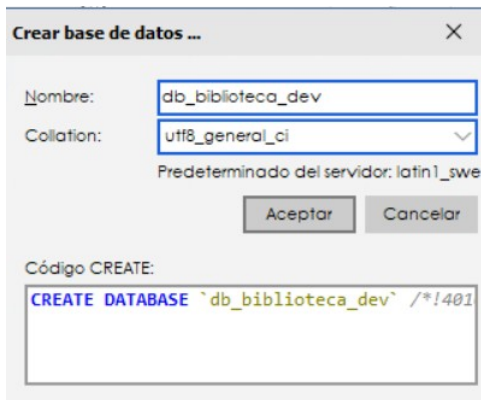
Cuando realizamos una prueba de testing, estaremos ejecutando en la misma base de datos. Para esto se crea una base de datos de test y otra de desarrollo.

Para separar la base de datos local de la de pruebas en un proyecto Spring Boot, se pueden usar diferentes configuraciones en el archivo application.properties o application.yml para cada perfil de entorno (e.g., dev para desarrollo y test para pruebas). Esto asegura que se utilicen bases de datos diferentes para el desarrollo y para ejecutar las pruebas unitarias.

## Configuración de la Base de Datos

Archivo de Configuración para Desarrollo (application-dev.properties):

- Crear dos archivos nuevos application-dev.properties y application-test.properties
- Crear dos bases de datos



Crear base de datos ...

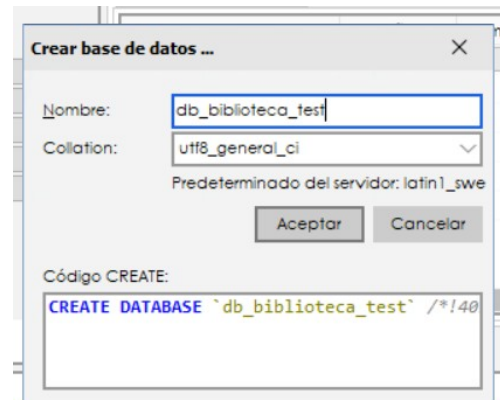
Nombre:

Collation:

Predeterminado del servidor: latin1\_swe

Código CREATE:

```
CREATE DATABASE `db_biblioteca_dev` /*!401
```



Crear base de datos ...

Nombre:

Collation:

Predeterminado del servidor: latin1\_swe

Código CREATE:

```
CREATE DATABASE `db_biblioteca_test` /*!40
```

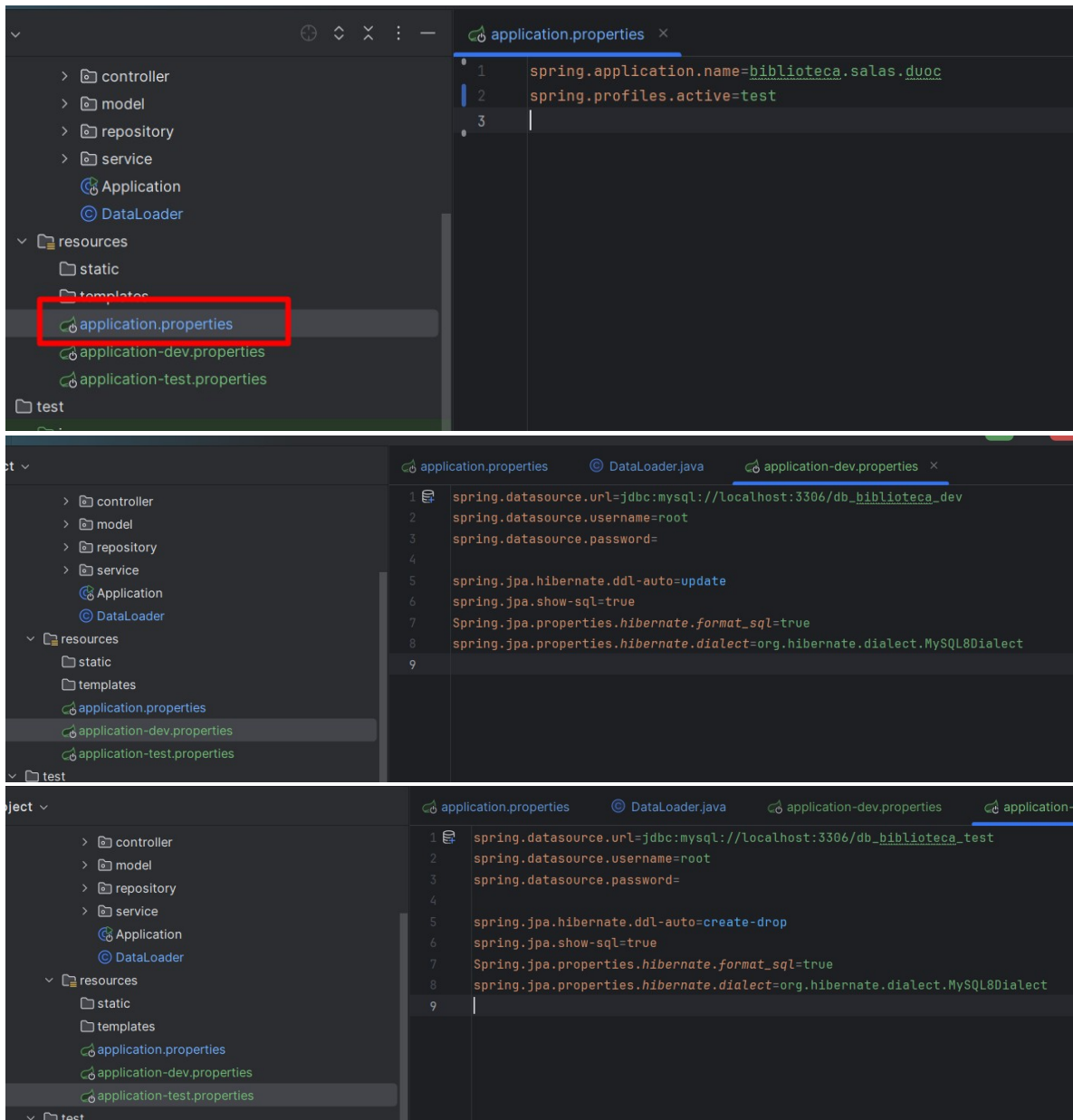
# src/main/resources/application-dev.properties

```
spring.datasource.url=jdbc:mysql://localhost:3306/mi_base_datos_dev
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
```

# src/main/resources/application-test.properties

```
spring.datasource.url=jdbc:mysql://localhost:3306/mi_base_datos_test
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
```

Entonces nos queda así.



The first screenshot shows the file explorer with `application.properties` selected in the `resources` folder. The editor displays the following content:

```
1 spring.application.name=biblioteca.salas.duoc
2 spring.profiles.active=test
3
```

The second screenshot shows the file explorer with `application-dev.properties` selected. The editor displays the following content:

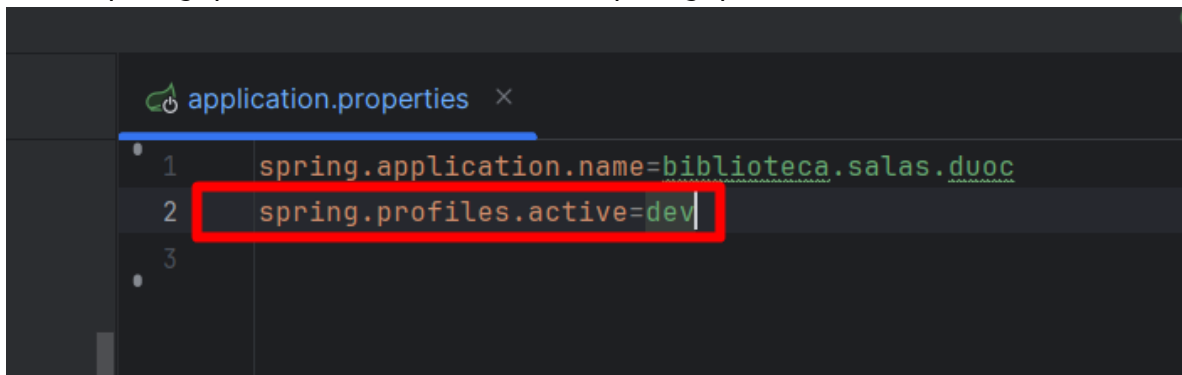
```
1 spring.datasource.url=jdbc:mysql://localhost:3306/db_biblioteca_dev
2 spring.datasource.username=root
3 spring.datasource.password=
4
5 spring.jpa.hibernate.ddl-auto=update
6 spring.jpa.show-sql=true
7 Spring.jpa.properties.hibernate.format_sql=true
8 spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
9
```

The third screenshot shows the file explorer with `application-test.properties` selected. The editor displays the following content:

```
1 spring.datasource.url=jdbc:mysql://localhost:3306/db_biblioteca_test
2 spring.datasource.username=root
3 spring.datasource.password=
4
5 spring.jpa.hibernate.ddl-auto=create-drop
6 spring.jpa.show-sql=true
7 Spring.jpa.properties.hibernate.format_sql=true
8 spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
9
```

Para cambiar de test a desarrollo es solo ir cambiando, si queremos ejecutar test cambiar a test.

spring.profiles.active=test o spring.profiles.active=dev



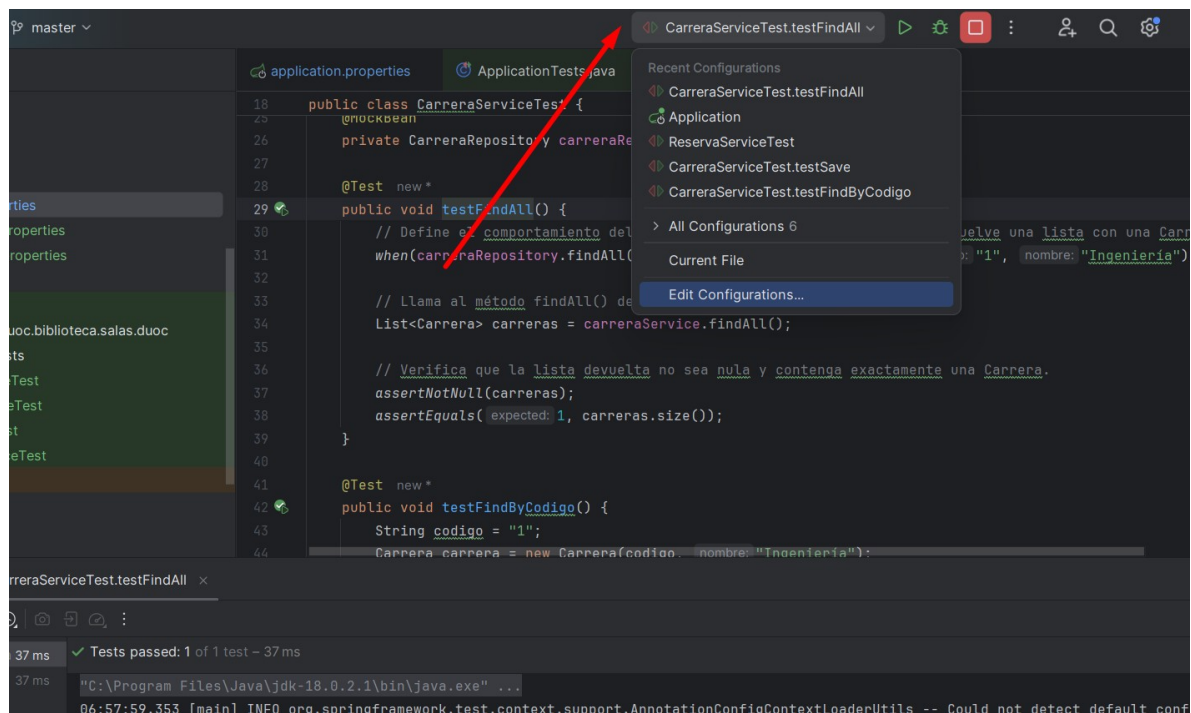
```

1  spring.application.name=biblioteca.salas.duoc
2  spring.profiles.active=dev
3

```

## Ejecutar los test

Se puede ejecutar las pruebas de manera individual



```

18  public class CarreraServiceTest {
19      @MockBean
20      private CarreraRepository carreraRepository;
21
22      @Test new *
23      public void testFindAll() {
24          // Define el comportamiento del mock
25          when(carreraRepository.findAll())
26              .thenReturn(new ArrayList<>());
27
28          // Llama al método findAll() de
29          List<Carrera> carreras = carreraService.findAll();
30
31          // Verifica que la lista devuelta no sea nula y contenga exactamente una Carrera.
32          assertNotNull(carreras);
33          assertEquals(expected: 1, carreras.size());
34      }
35
36      @Test new *
37      public void testFindByCodigo() {
38          String codigo = "1";
39          Carrera carrera = new Carrera(codigo, nombre: "Ingeniería");
40      }
41  }

```

Recent Configurations

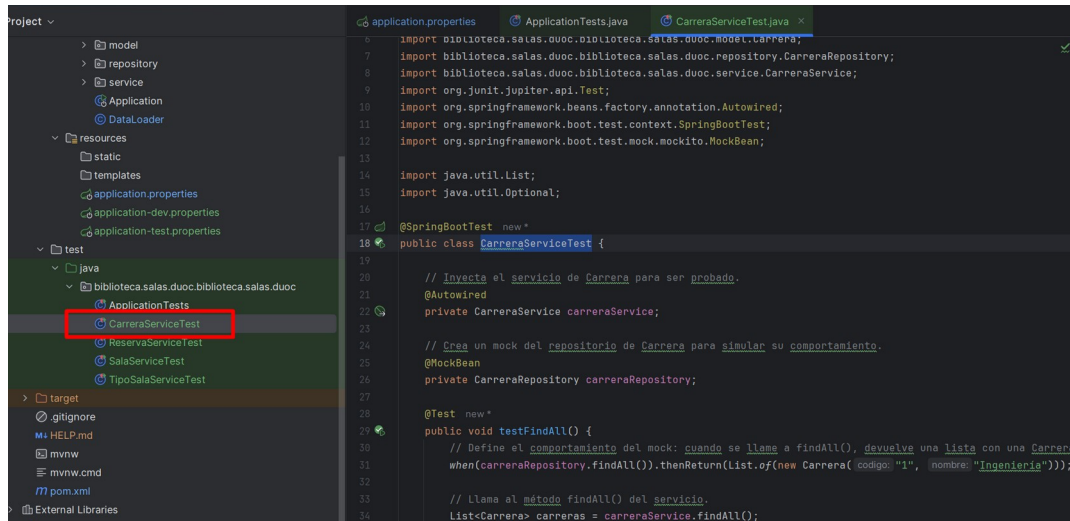
- CarreraServiceTest.testFindAll
- Application
- ReservaServiceTest
- CarreraServiceTest.testSave
- CarreraServiceTest.testFindByCodigo
- All Configurations 6
- Current File
- Edit Configurations...

Tests passed: 1 of 1 test - 37 ms

## Análisis de testing nivel servicio:

Descomprime el zip y analiza junto a tu equipo, los diferentes pruebas realizados y completa las preguntas, de la guía.

### TESTING - Analizar Services Test - biblioteca.salas.duoc.zip



The screenshot shows an IDE with the project structure on the left and the code of `CarreraServiceTest.java` on the right. The project structure includes:

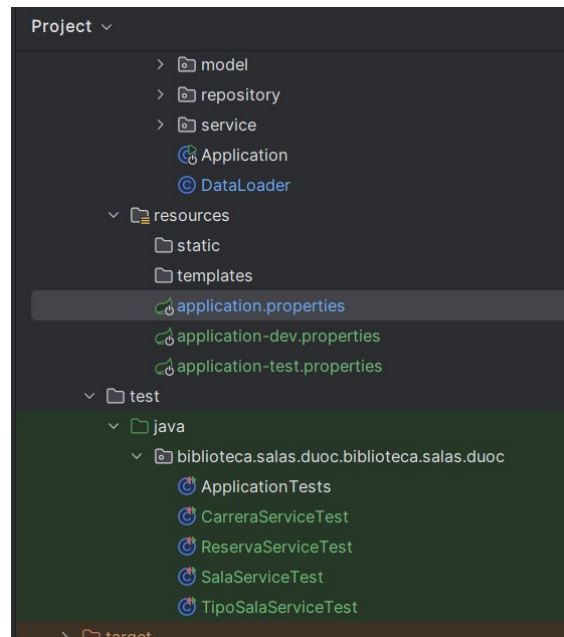
- model
- repository
- service
  - Application
  - DataLoader
- resources
  - static
  - templates
  - application.properties
  - application-dev.properties
  - application-test.properties
- test
  - java
    - biblioteca.salas.duoc.biblioteca.salas.duoc
      - ApplicationTests
      - CarreraServiceTest
      - ReservaServiceTest
      - SalaServiceTest
      - TipoSalaServiceTest

The code in `CarreraServiceTest.java` is as follows:

```

1 import biblioteca.salas.duoc.modelo.biblioteca.salas.duoc.modelo.Carrera;
2 import biblioteca.salas.duoc.repository.CarreraRepository;
3 import biblioteca.salas.duoc.service.CarreraService;
4 import org.junit.jupiter.api.Test;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.boot.test.context.SpringBootTest;
7 import org.springframework.boot.test.mock.mockito.MockBean;
8
9 import java.util.List;
10 import java.util.Optional;
11
12 @SpringBootTest
13 public class CarreraServiceTest {
14
15     // Inyecta el servicio de Carrera para ser probado.
16     @Autowired
17     private CarreraService carreraService;
18
19     // Crea un mock del repositorio de Carrera para simular su comportamiento.
20     @MockBean
21     private CarreraRepository carreraRepository;
22
23     @Test
24     public void testFindAll() {
25         // Define el comportamiento del mock: cuando se llama a findAll(), devuelve una lista con una Carrera
26         when(carreraRepository.findAll()).thenReturn(List.of(new Carrera(codigo: "1", nombre: "Ingeniería")));
27
28         // Llama al método findAll() del servicio.
29         List<Carrera> carreras = carreraService.findAll();
30     }
31 }

```



```

@Autowired
private CarreraService carreraService;

// Crea un mock del repositorio de Carrera para simular su comportamiento.
@MockBean
private CarreraRepository carreraRepository;

@Test
public void testFindAll() {
    // Define el comportamiento del mock: cuando se llame a findAll(), devuelve una lista con una Carrera.
    when(carreraRepository.findAll()).thenReturn(List.of(new Carrera(codigo: "1", nombre: "Ingeniería")));

    // Llama al método findAll() del servicio.
    List<Carrera> carreras = carreraService.findAll();

    // Verifica que la lista devuelta no sea nula y contenga exactamente una Carrera.
    assertNotNull(carreras);
    assertEquals(1, carreras.size());
}

```

¿Qué hace `testFindAll()`?

El propósito de este método es verificar que el método `findAll` del servicio `CarreraService` funcione correctamente. Específicamente, se asegura de que el método `findAll` devuelva una lista de todas las carreras almacenadas en el repositorio.

#### Pasos Detallados

##### Configuración del Mock (when...thenReturn):

- `when(carreraRepository.findAll()).thenReturn(List.of(new Carrera("1", "Ingeniería")));`
- En este paso, se configura el comportamiento simulado (mock) del `carreraRepository`.
- Específicamente, se indica que cuando el método `findAll()` del `carreraRepository` sea llamado, debe devolver una lista que contiene un solo objeto `Carrera` con código "1" y nombre "Ingeniería".

##### Llamada al Método del Servicio:

- `List<Carrera> carreras = carreraService.findAll();`
- Aquí se llama al método `findAll()` del `carreraService`.
- El servicio, a su vez, llama al `findAll()` del repositorio.
- Debido a la configuración del mock en el paso anterior, el repositorio devolverá la lista simulada de carreras.

##### Verificación de la Lista Devuelta:

- `assertNotNull(carreras);`
- Este paso verifica que la lista de carreras devuelta no sea nula. Esto asegura que el método `findAll` del servicio realmente devuelve una lista.
- `assertEquals(1, carreras.size());`

Este paso verifica que la lista contenga exactamente un elemento. Esto asegura que la lista de carreras devuelta por el servicio contiene el número esperado de elementos.

¿Analiza los demás testing y comenta con tu compañero su implementación a los servicios?

¿Porque se implementan testing a nivel de servicio, se pueden implementar a nivel de controller?



**Análisis de testing nivel de controller:**



TESTING - CONTROLLER - [bibliteca.salas.duoc.zip](http://bibliteca.salas.duoc.zip)

¿Qué se espera realizar en este testing?

Analiza la prueba estudiante Controller ¿Qué se espera realizar en el **testCreateEstudiante**?

Ejecuta todos los test. ¿Existe alguna falla? Corrige la falla y comenta ¿qué era?

🎉🎉 **BUEN TRABAJO** 🎉🎉