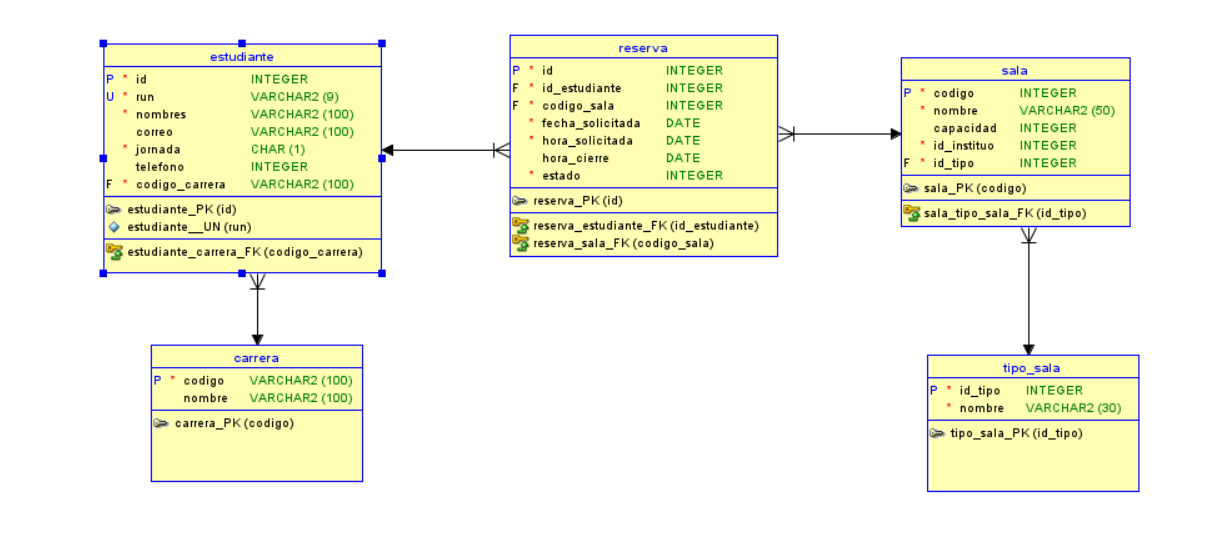
## **Guía práctica**

## **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:



#### **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
* Crea dos base de datos

|  |  |
| --- | --- |
|  |  |

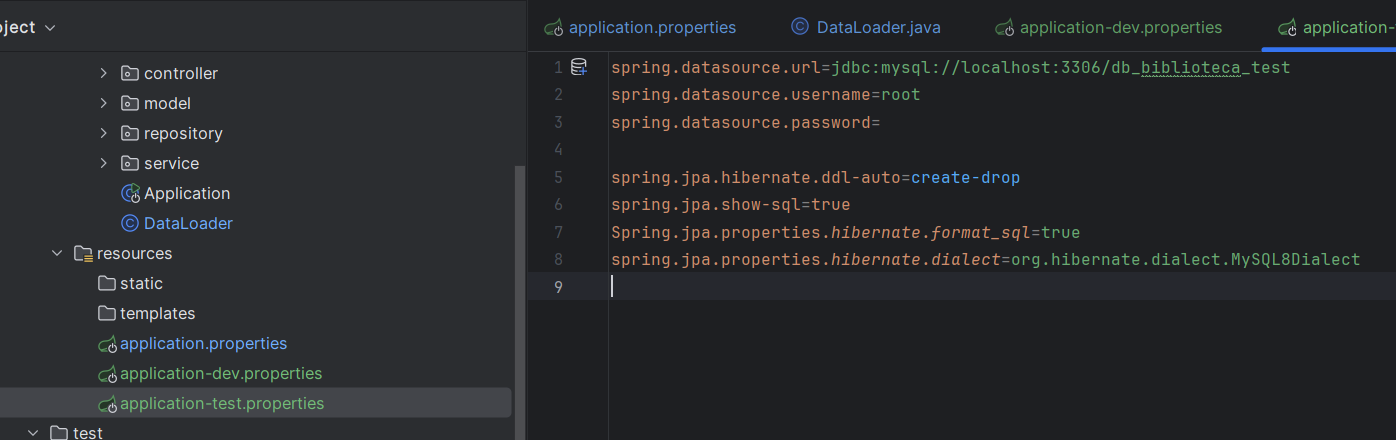
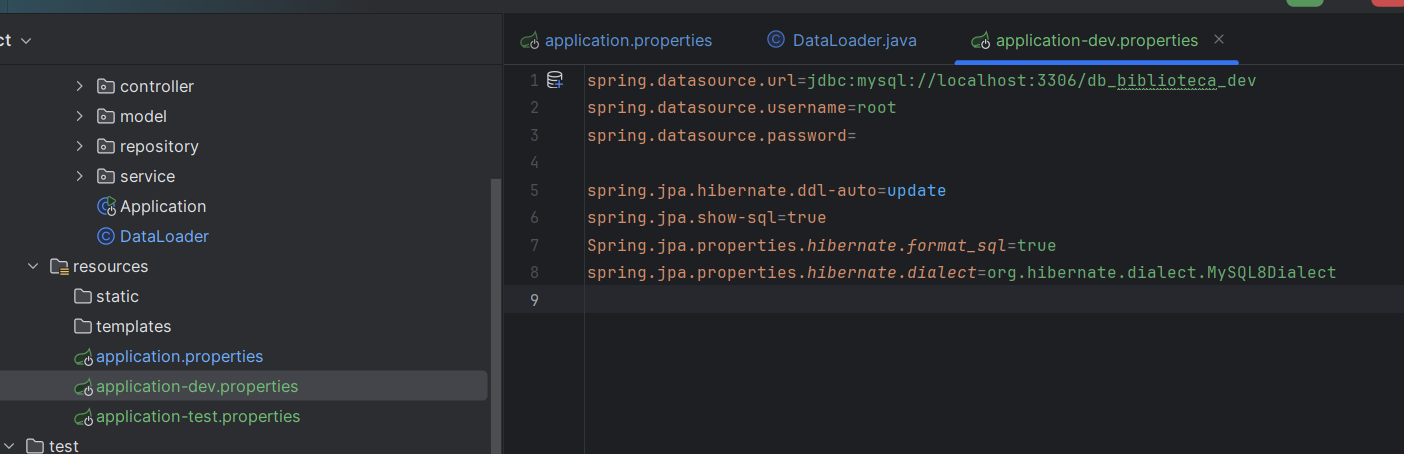
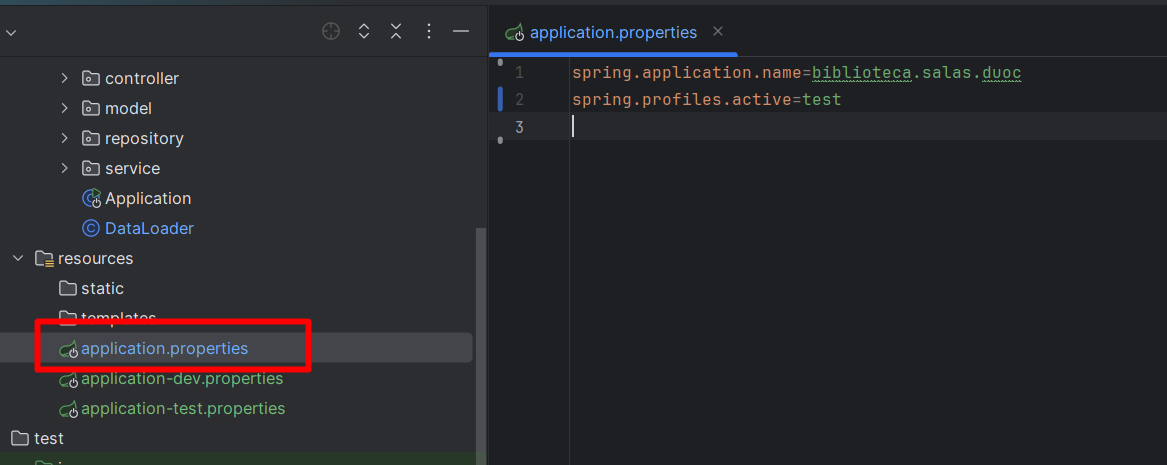
# src/main/resources/application-dev.properties

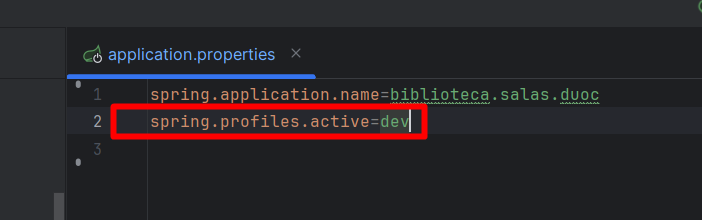
|  |
| --- |
| spring.datasource.url=jdbc:mysql://localhost:3306/mi\_base\_datos\_dev spring.datasource.username=root spring.datasource.password=root spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver spring.jpa.hibernate.ddl-auto=update spring.jpa.show-sql=true spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect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

# 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 |

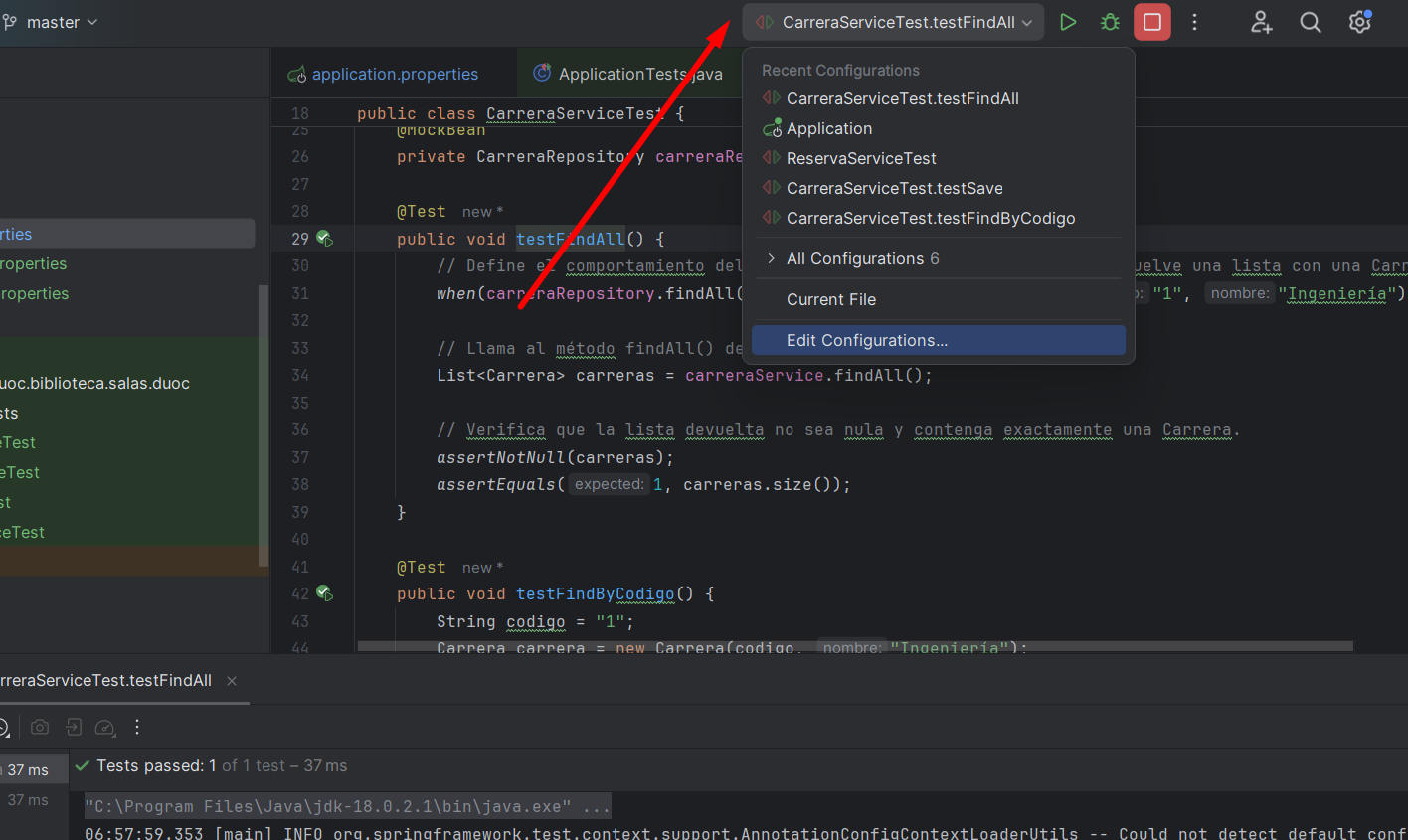
Entonces nos queda así.

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 

**Ejecutar los test**

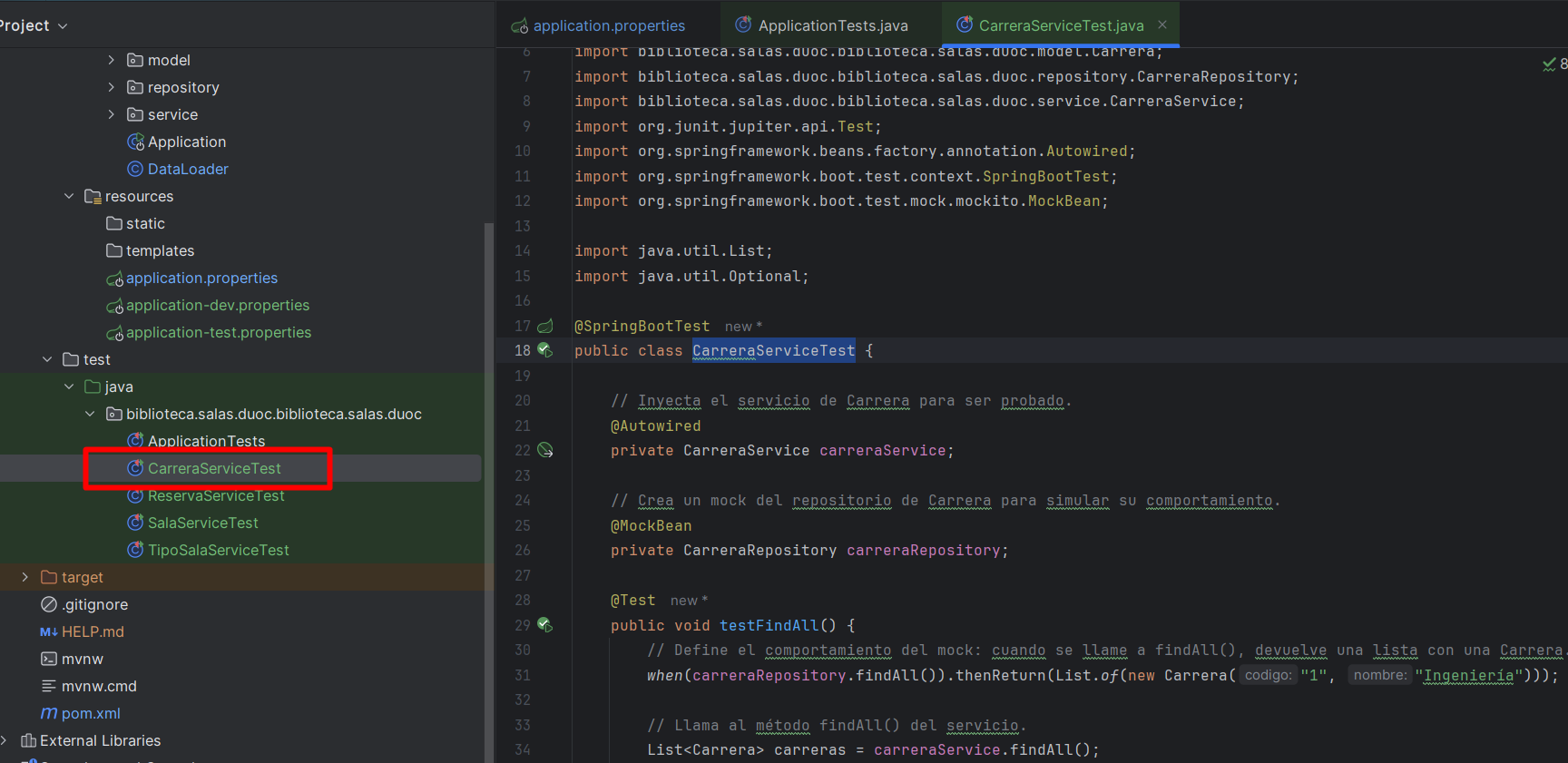
Se puede ejecutar las pruebas de manera individual

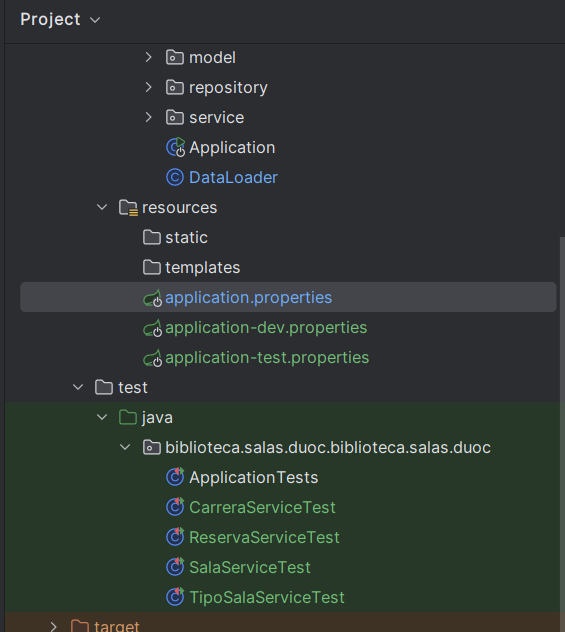


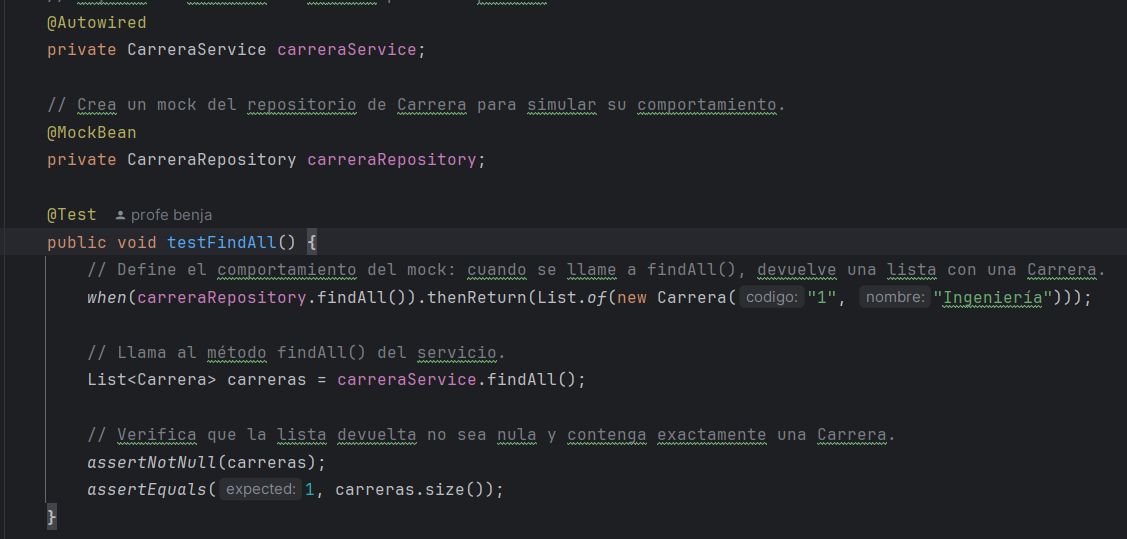
**Análisis de testing nivel servicio:**

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









¿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:**



¿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 🎉🎉***