

INSTITUTO TECNOLOGICO DE FRONTERA COMALAPA

CATEDRÁTICO: Mingo Velázquez Francisco Javier.

MATERIA: Tópicos Selectos de Base de datos.

NOMBRE DEL ALUMNO: Moguel Recinos Miriam.

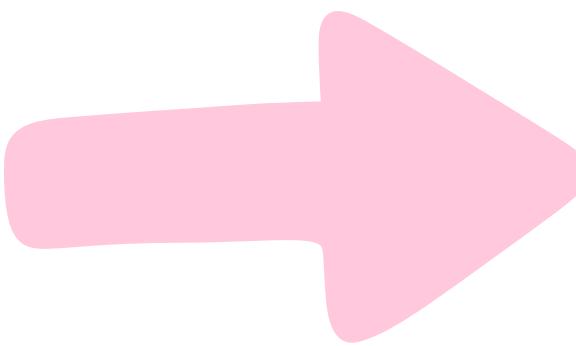
NUMERO DE CONTROL: 211260044

CARRERA: Ingeniería en Sistemas Computacionales.

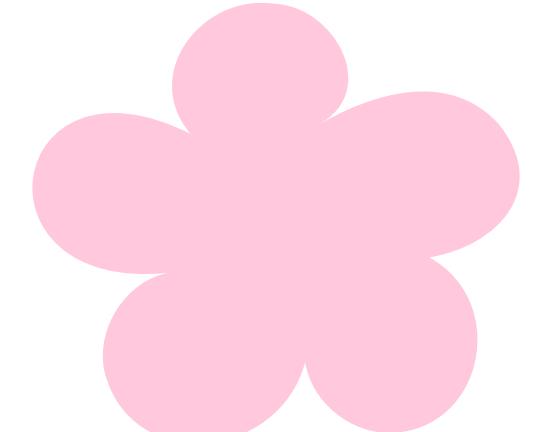
NOMBRE DEL TRABAJO: Reporte de Práctica.

FRONTERA COMALAPA, CHIAPAS A 14 DE NOVIEMBRE DE 2025

REPORTE DE PRACTICA



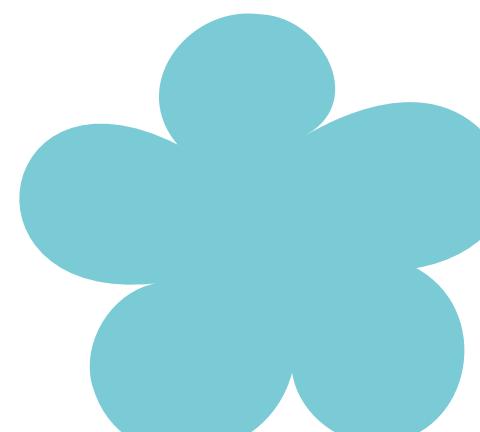
Introducción



El buen funcionamiento de un sistema de software se asegura mediante la ejecución de pruebas que verifican su correcta operación, seguridad, rendimiento y experiencia de usuario. En este proyecto se aplicaron pruebas a microservicios desarrollados en Java (Spring Boot), JavaScript, Astro, React, NestJS y Angular, utilizando herramientas como JUnit 5, Mockito, MockMvc, Jest, Supertest y Testing Library.



Las pruebas incluyeron desde evaluaciones unitarias hasta revisiones de sistema, seguridad, desempeño y usabilidad, analizando controladores, servicios, componentes y flujos críticos bajo escenarios similares a los de uso real.



Justificación

La implementación de distintos tipos de pruebas resulta fundamental para asegurar que el sistema cumpla con los requisitos funcionales y operativos establecidos. Mediante evaluaciones como pruebas unitarias, de integración, aceptación, seguridad, sistema, rendimiento y usabilidad, es posible identificar errores tempranos, verificar la correcta comunicación entre módulos, validar el cumplimiento de las necesidades del usuario final, garantizar la protección de recursos y roles, analizar el comportamiento global del software, medir su eficiencia y confirmar que la interfaz sea accesible y funcional. Esta diversidad de pruebas justifica su aplicación, ya que permite entregar una solución confiable, robusta y segura, capaz de responder adecuadamente a las condiciones reales de uso.

MOKITO

A continuación se muestran las imágenes que permiten identificar la versión de Mockito empleada para llevar a cabo las pruebas unitarias, de integración, seguridad, sistema, aceptación, desempeño y usabilidad. Esta información es indispensable para agregar la dependencia correspondiente en el archivo pom.xml y asegurar el correcto desarrollo del proceso de verificación del sistema.

The screenshot shows the Maven Repository website at <https://mvnrepository.com/artifact/org.mockito/mockito-all/2.0.2-beta>. The page displays the Mockito 2.0.2-beta artifact details, including its license (MIT), categories (Mocking), and tags (quality, mock, mocking, testing, Mockito). It also shows the homepage URL (<http://www.mockito.org>) and a download count of 15,764 artifacts. The central part of the page features a dark blue background with white text advertising "Un CRM + que trabaja tan rápido como tú". Below the main content, there is a snippet of the pom.xml dependency code.

```
<!-- https://mvnrepository.com/artifact/org.mockito/mockito-all -->
<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-all</artifactId>
    <version>2.0.2-beta</version>
    <scope>test</scope>
</dependency>
```

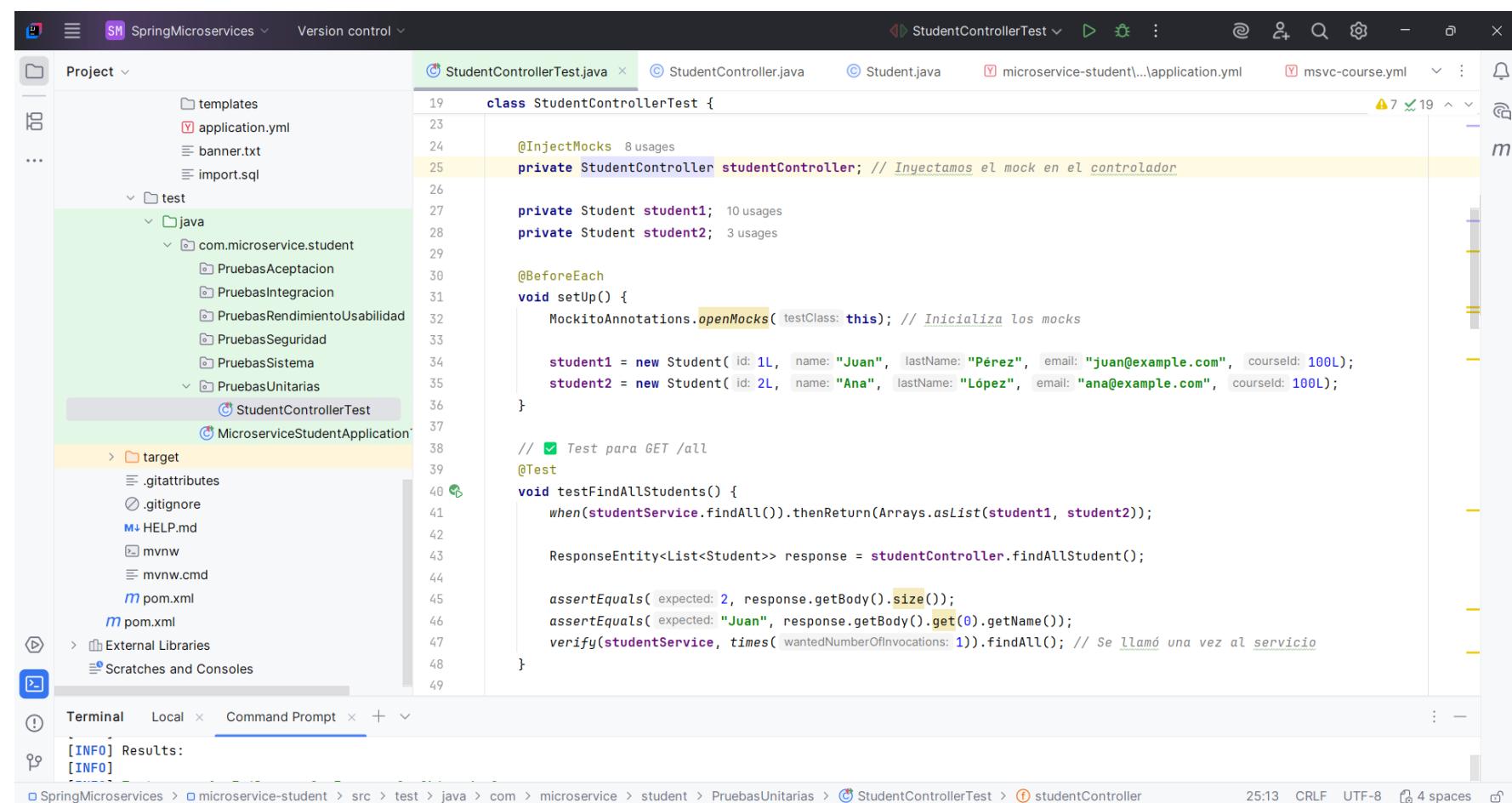
The screenshot shows an IDE interface with the project structure and the pom.xml file for the "MicroserviceGatewayApplication" module. The pom.xml file is open, showing the dependency section for Mockito. The dependency is defined as follows:

```
<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-all</artifactId>
    <version>2.0.2-beta</version>
    <scope>test</scope>
</dependency>
```

The pom.xml file also includes other dependencies and build configurations for Spring Boot and Maven.

Pruebas unitarias microservicios

El código corresponde a una clase de pruebas unitarias en Java que evalúa el comportamiento del `StudentController` en un microservicio, utilizando JUnit 5 y Mockito para simular la capa de servicio sin ejecutar toda la aplicación. La clase verifica las operaciones CRUD y la búsqueda por curso mediante objetos simulados y datos de prueba. Cada método valida un endpoint específico, contemplando tanto respuestas correctas como casos de error, como cuando un estudiante no se encuentra registrado.



The screenshot shows an IDE interface with the following details:

- Project View:** Shows the project structure under "Project". It includes a "templates" folder, "application.yml", "banner.txt", "import.sql", and a "test" directory containing "java" subfolders for "PruebasAceptacion", "PruebasIntegracion", "PruebasRendimientoUsabilidad", "PruebasSeguridad", "PruebasSistema", and "PruebasUnitarias". The "StudentControllerTest" class is selected in the "PruebasUnitarias" folder.
- Code Editor:** The "StudentControllerTest.java" file is open. The code uses Mockito annotations like `@InjectMocks` and `@Mock` to set up a mock `StudentService`. It includes methods for testing the `findAllStudents` endpoint, asserting the size of the response, and verifying the number of invocations on the service.
- Terminal:** At the bottom, there is a terminal window showing "[INFO] Results:" and "[INFO]" messages.

```
class StudentControllerTest {
    @InjectMocks
    private StudentController studentController; // Inyectamos el mock en el controlador

    private Student student1; 10 usages
    private Student student2; 3 usages

    @BeforeEach
    void setUp() {
        MockitoAnnotations.openMocks(this); // Inicializa los mocks

        student1 = new Student(id: 1L, name: "Juan", lastName: "Pérez", email: "juan@example.com", courseId: 100L);
        student2 = new Student(id: 2L, name: "Ana", lastName: "López", email: "ana@example.com", courseId: 100L);
    }

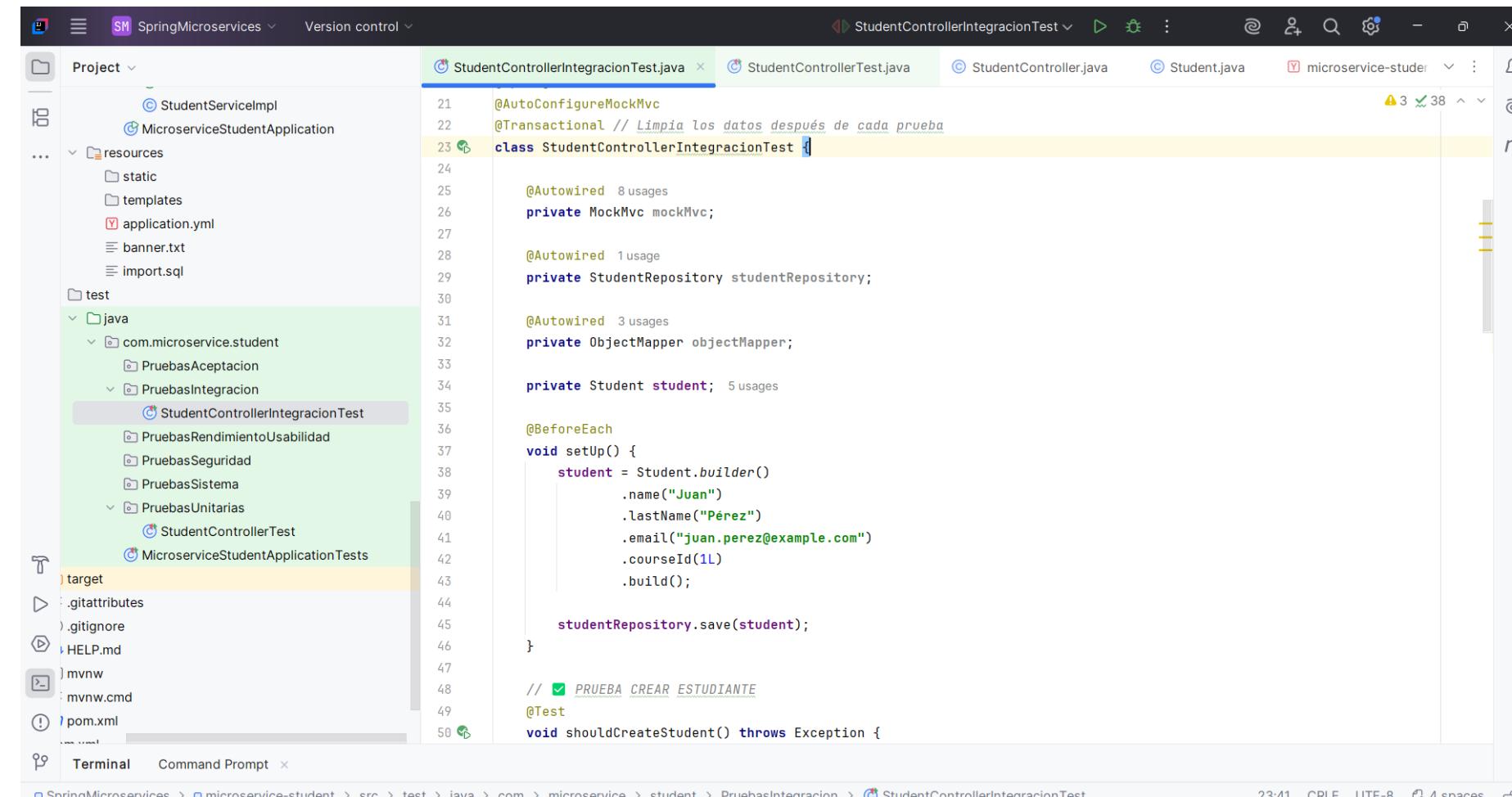
    // Test para GET /all
    @Test
    void testfindAllStudents() {
        when(studentService.findAll()).thenReturn(Arrays.asList(student1, student2));

        ResponseEntity<List<Student>> response = studentController.findAllStudent();

        assertEquals(expected: 2, response.getBody().size());
        assertEquals(expected: "Juan", response.getBody().get(0).getName());
        verify(studentService, times(wantedNumberOfInvocations: 1)).findAll(); // Se llamó una vez al servicio
    }
}
```

Pruebas integración student

El código corresponde a una clase de pruebas de integración en Java, `StudentControllerIntegrationTest`, desarrollada con Spring Boot Test, JUnit 5 y MockMvc. Su propósito es verificar que los endpoints del `StudentController` interactúen correctamente con la base de datos y el microservicio. Utiliza anotaciones como `@SpringBootTest`, `@AutoConfigureMockMvc` y `@Transactional` para ejecutar pruebas completas sin afectar los datos reales. Las pruebas abarcan operaciones de creación, consulta, actualización, eliminación y búsqueda por curso, incluyendo escenarios con estudiantes inexistentes, asegurando el correcto funcionamiento del controlador y la consistencia entre las capas del sistema.



The screenshot shows a Java IDE interface with the following details:

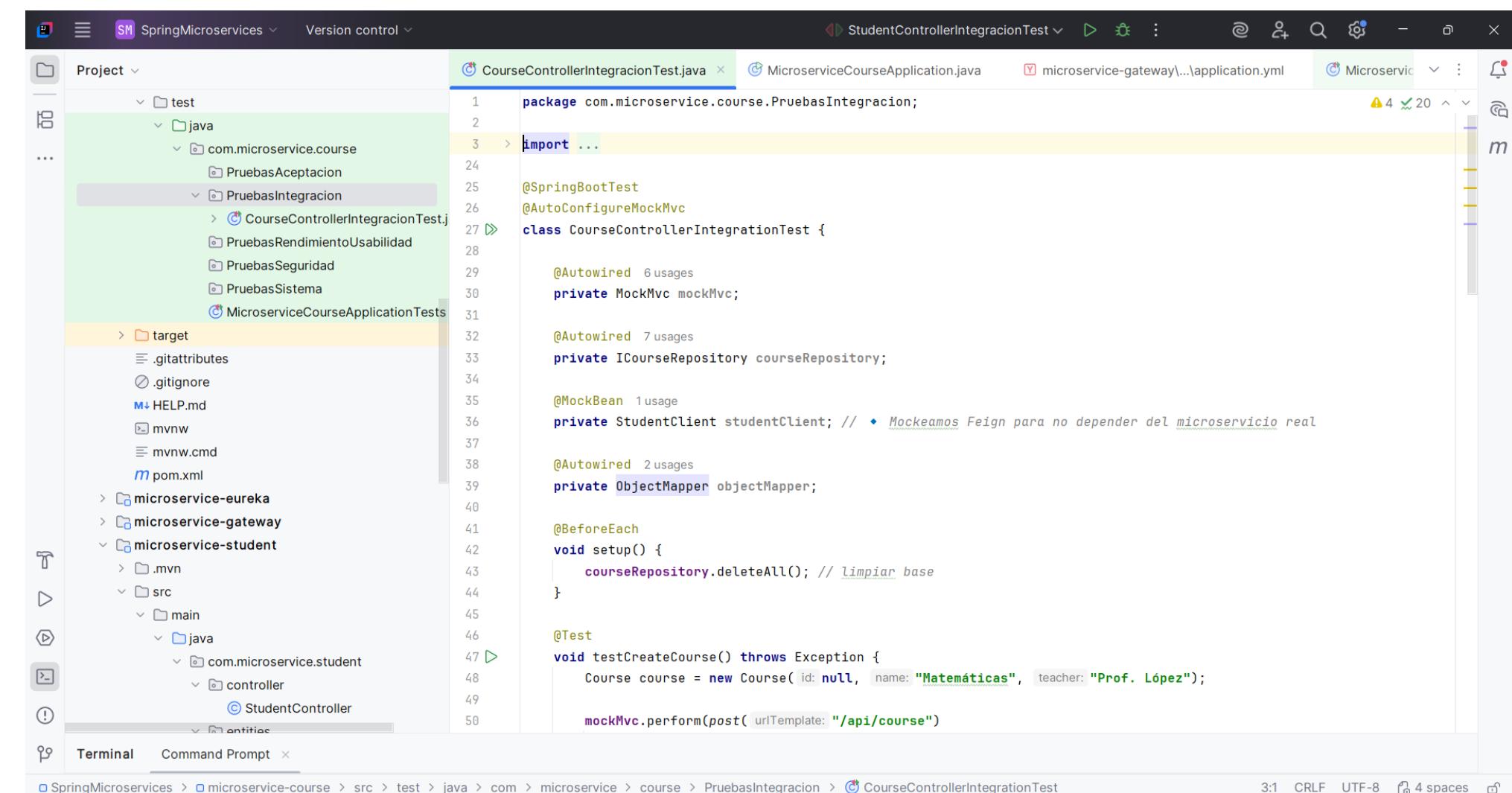
- Project Structure:** The project is named "SpringMicroservices". It contains a "resources" folder with static files like banner.txt and import.sql, and a "test" folder containing "java" subfolders for "PruebasAceptacion", "PruebasIntegracion", and "PruebasUnitarias". The "PruebasUnitarias" folder is currently selected, and it contains the `StudentControllerIntegrationTest` class.
- Code Editor:** The code editor displays the `StudentControllerIntegrationTest.java` file. The code is as follows:

```
21 @AutoConfigureMockMvc
22 @Transactional // Limpia los datos después de cada prueba
23 class StudentControllerIntegrationTest {
24
25     @Autowired 8 usages
26     private MockMvc mockMvc;
27
28     @Autowired 1 usage
29     private StudentRepository studentRepository;
30
31     @Autowired 3 usages
32     private ObjectMapper objectMapper;
33
34     private Student student; 5 usages
35
36     @BeforeEach
37     void setUp() {
38         student = Student.builder()
39             .name("Juan")
40             .lastName("Pérez")
41             .email("juan.perez@example.com")
42             .courseId(1L)
43             .build();
44
45         studentRepository.save(student);
46     }
47
48     // PRUEBA CREAR ESTUDIANTE
49     @Test
50     void shouldCreateStudent() throws Exception {
```

The code uses annotations like `@AutoConfigureMockMvc` and `@Transactional`. It autowires `MockMvc`, `StudentRepository`, and `ObjectMapper`. It also defines a `student` object using `Student.builder()` and saves it to the repository. A test method `shouldCreateStudent()` is defined to verify the creation of a student.

Pruebas integración course

El código corresponde a una clase de pruebas de integración para CourseController, desarrollada con Spring Boot, JUnit 5, MockMvc y un *mock* de Feign Client (StudentClient). Su objetivo es verificar que los endpoints funcionen correctamente, interactuando tanto con la base de datos como con servicios externos simulados. Las pruebas incluyen operaciones de creación, consulta, búsqueda, actualización y eliminación de cursos, así como la obtención de los estudiantes asociados a cada curso.



The screenshot shows a Java IDE interface with the following details:

- Project Structure:** The project is named "SpringMicroservices". It contains several modules: "test", "MicroserviceCourseApplication.java", "microservice-gateway...application.yml", "MicroserviceCourseApplicationTests", "target", ".gitattributes", ".gitignore", "HELP.md", "mvnw", "mvnw.cmd", "pom.xml", "microservice-eureka", "microservice-gateway", and "microservice-student".
- Code Editor:** The file "CourseControllerIntegrationTest.java" is open. The code is as follows:

```
package com.microservice.course.PruebasIntegracion;
import ...;
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;
import org.springframework.boot.test.mock.mockito.MockBean;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.ResultActions;
import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;
import org.springframework.test.web.servlet.result.MockMvcResultMatchers;
import com.microservice.course.CourseController;
import com.microservice.course.entity.Course;
import com.microservice.course.repository.ICourseRepository;
import com.microservice.course.service.StudentClient;
import com.fasterxml.jackson.databind.ObjectMapper;
import static org.mockito.Mockito.*;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.*;

@WebMvcTest
@AutoConfigureMockMvc
public class CourseControllerIntegrationTest {

    @Autowired
    private MockMvc mockMvc;

    @Autowired
    private ICourseRepository courseRepository;

    @MockBean
    private StudentClient studentClient; // ✨ Mockeamos Feign para no depender del microservicio real

    @Autowired
    private ObjectMapper objectMapper;

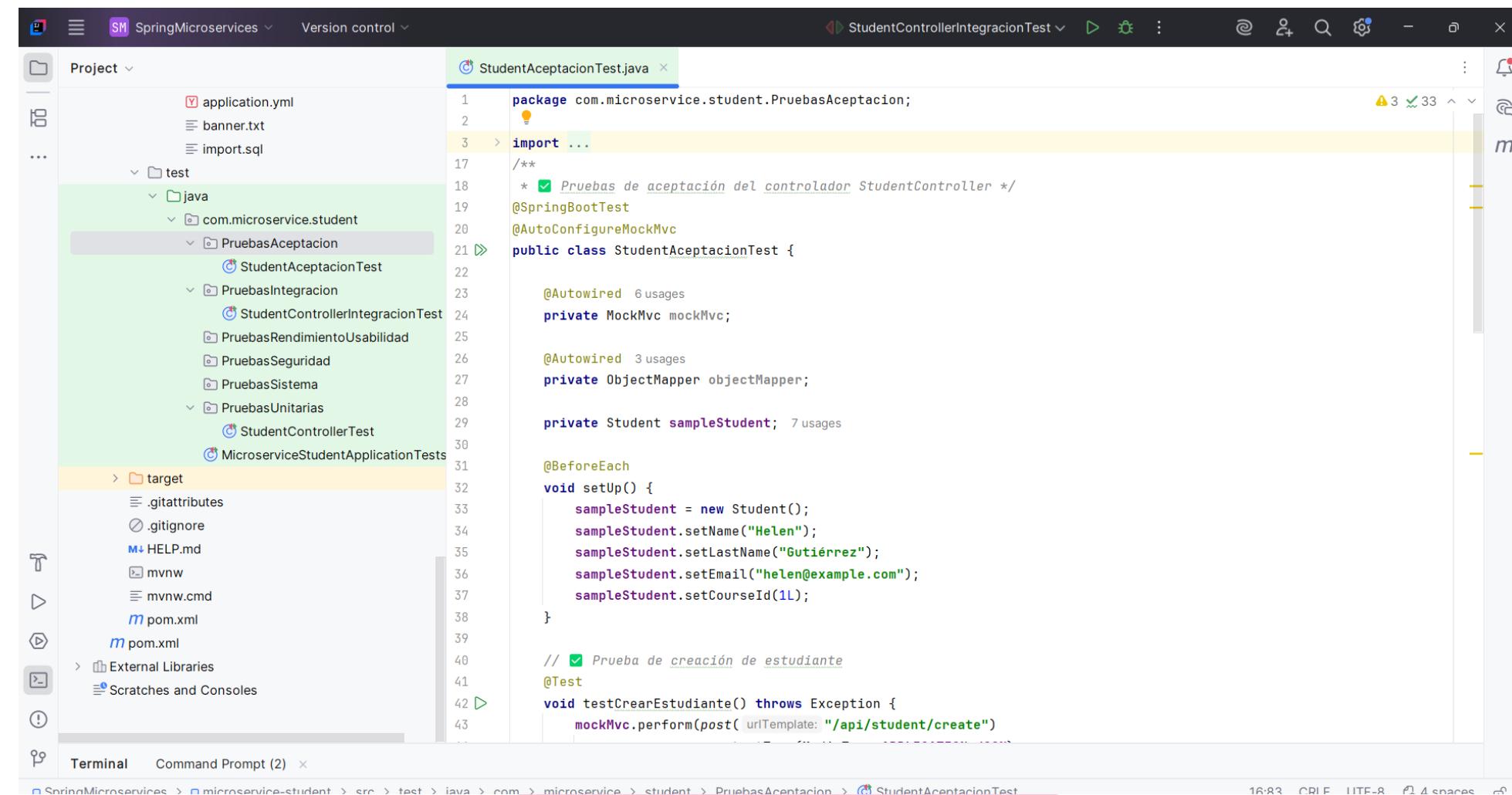
    @BeforeEach
    void setup() {
        courseRepository.deleteAll(); // limpiar base
    }

    @Test
    void testCreateCourse() throws Exception {
        Course course = new Course( id: null, name: "Matemáticas", teacher: "Prof. López");
        mockMvc.perform(post( urlTemplate: "/api/course"))
            .andExpect(status().isOk())
            .andExpect(content().string("Course created successfully"));
    }
}
```

The code uses JUnit 5 assertions and Spring Boot's MockMvc and AutoConfigureMockMvc annotations to test the CourseController. It also uses Mockito to mock the StudentClient and Jackson's ObjectMapper. The test specifically checks if a new course is created successfully.

Pruebas aceptación student

El código corresponde a una clase de pruebas de aceptación en Java, `StudentAceptacionTest`, desarrollada con Spring Boot Test, JUnit 5 y MockMvc. Su propósito es verificar que los endpoints del microservicio Student operen correctamente desde la perspectiva del usuario. Empleando `@SpringBootTest`, `@AutoConfigureMockMvc` y `@Transactional`, las pruebas se ejecutan de manera completa sin afectar la base de datos. Se evalúan las operaciones CRUD —crear, listar, actualizar, consultar y eliminar estudiantes— asegurando que cada endpoint cumpla su función correctamente en un entorno real.

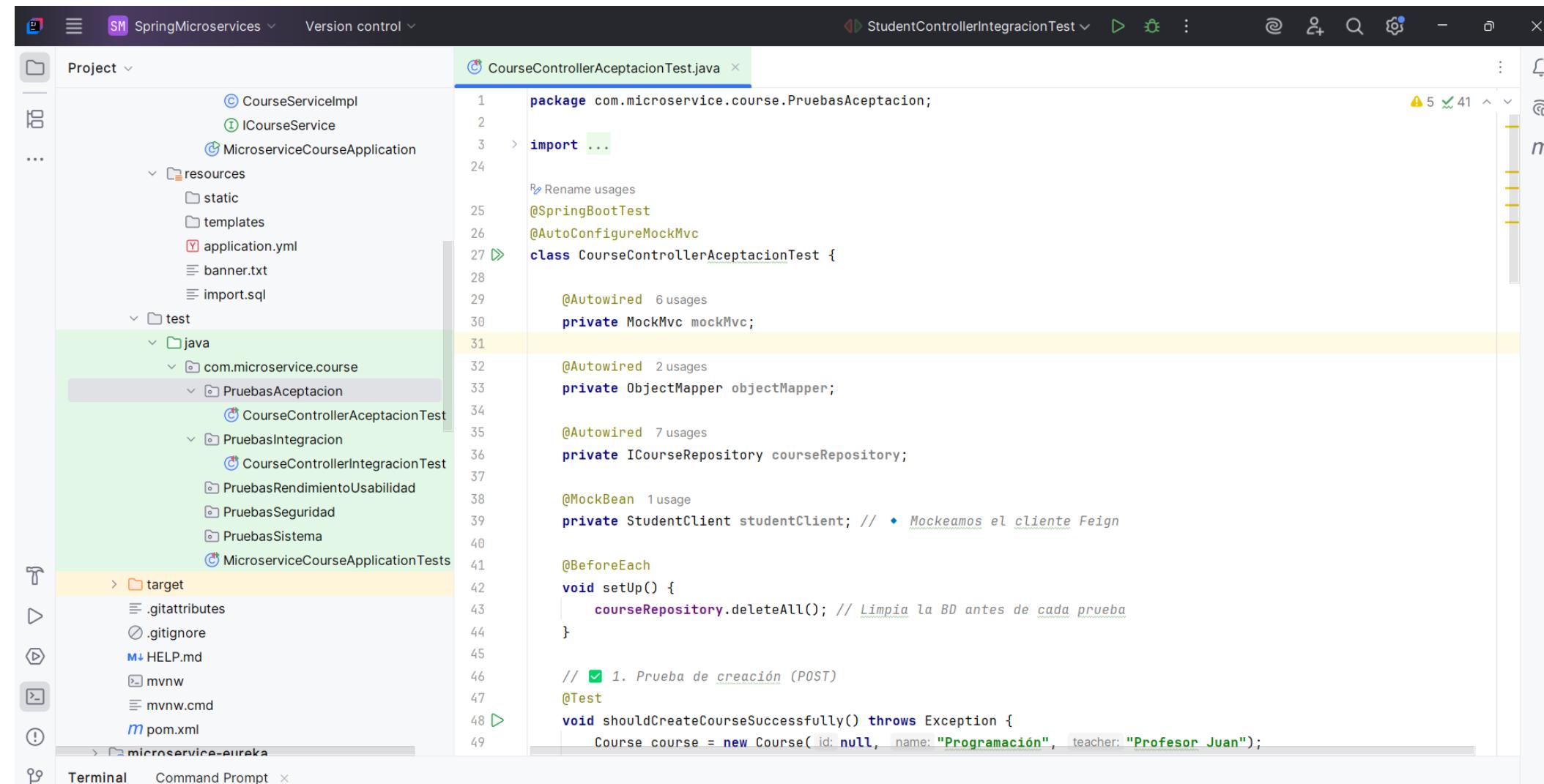


The screenshot shows a Java IDE interface with the following details:

- Project Structure:** The project is named "SpringMicroservices". It contains an "application.yml" file, a "banner.txt" file, an "import.sql" file, and a "test" directory. Inside "test", there is a "java" directory which contains several test classes under the package "com.microservice.student.PruertasAceptacion". These include "StudentAceptacionTest", "StudentControllerIntegracionTest", "PruebasRendimientoUsabilidad", "PruebasSeguridad", "PruebasSistema", "PruebasUnitarias", and "MicroserviceStudentApplicationTests".
- Code Editor:** The code editor displays the `StudentAceptacionTest.java` file. The code is annotated with `@SpringBootTest`, `@AutoConfigureMockMvc`, and `@Transactional`. It includes a `@BeforeEach` method to set up a sample student object with name "Helen", last name "Gutiérrez", email "helen@example.com", and course ID 1L. It also includes a `@Test` method to test the creation of a student using MockMvc.
- Terminal:** A terminal window at the bottom left shows the command "mvnw" being run.
- Status Bar:** The status bar at the bottom right shows the current time as 16:53, the file encoding as UTF-8, and the number of spaces as 4.

Pruebas de aceptación course

El código corresponde a una clase de pruebas de aceptación para CourseController, desarrollada con Spring Boot, JUnit 5, MockMvc y un *mock* de Feign Client para simular estudiantes. Su objetivo es verificar que los endpoints cumplan los requerimientos funcionales. Las pruebas incluyen operaciones de creación, consulta, búsqueda por ID, actualización y eliminación de cursos, así como la obtención de los estudiantes asociados a cada curso.



The screenshot shows a Java IDE interface with the following details:

- Project Structure:** The project is named "SpringMicroservices". It contains a "MicroserviceCourseApplication" module with "CourseServiceImpl", "ICourseService", and "application.yml" files. A "test" directory contains "java" sub-directories for "com.microservice.course" (containing "PruebasAceptacion", "PruebasIntegracion", "PruebasRendimientoUsabilidad", "PruebasSeguridad", "PruebasSistema", and "MicroserviceCourseApplicationTests") and "target" (containing ".gitattributes", ".gitignore", "HELP.md", "mvnw", "mvnw.cmd", and "pom.xml").
- Code Editor:** The file "CourseControllerAceptacionTest.java" is open. The code is as follows:

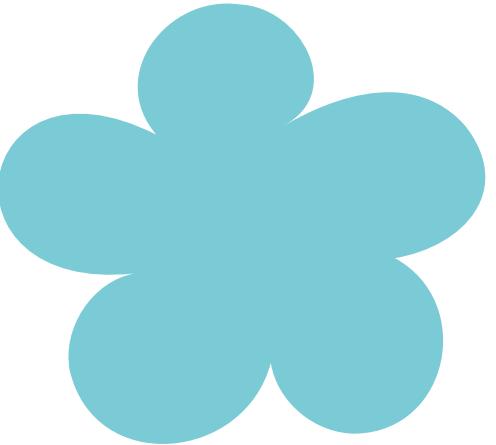
```
package com.microservice.course.PruebasAceptacion;
import ...;
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;
import org.springframework.boot.test.mock.mockito.MockBean;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.microservice.course.Course;
import com.microservice.course.ICourseRepository;
import com.microservice.course.StudentClient;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.Mockito;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.setup.MockMvcBuilders;
import org.springframework.web.context.WebApplicationContext;
import java.util.List;
import java.util.Optional;
import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.*;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.*;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.*;

@ExtendWith(MockitoExtension.class)
@WebMvcTest
@AutoConfigureMockMvc
public class CourseControllerAceptacionTest {
    @Autowired
    private MockMvc mockMvc;
    @Autowired
    private ObjectMapper objectMapper;
    @MockBean
    private ICourseRepository courseRepository;
    @MockBean
    private StudentClient studentClient; // ✨ Mockeamos el cliente Feign

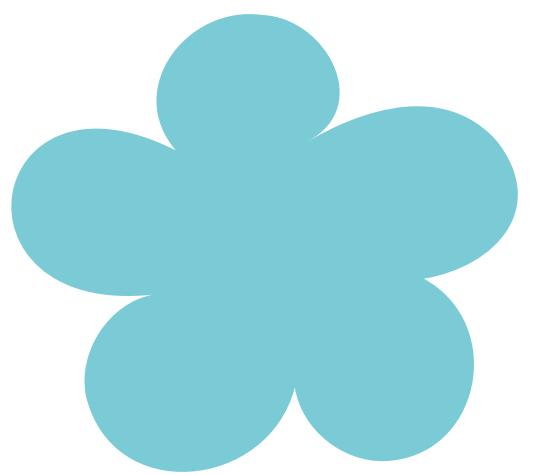
    @BeforeEach
    void setUp() {
        courseRepository.deleteAll(); // Limpia la BD antes de cada prueba
    }

    // ✅ 1. Prueba de creación (POST)
    @Test
    void shouldCreateCourseSuccessfully() throws Exception {
        Course course = new Course(id: null, name: "Programación", teacher: "Profesor Juan");
        courseRepository.save(course);
        mockMvc.perform(post("/api/courses"))
            .andExpect(status().isCreated())
            .andExpect(header().contains("Location"));
    }
}
```

Pruebas seguridad student



El código corresponde a una clase de pruebas de seguridad para StudentController, desarrollada con Spring Boot, JUnit 5 y Mockito. Su objetivo es garantizar que los endpoints solo sean accesibles por usuarios autenticados, respetando los roles asignados, y que operaciones críticas, como la eliminación de estudiantes, estén restringidas únicamente a administradores.

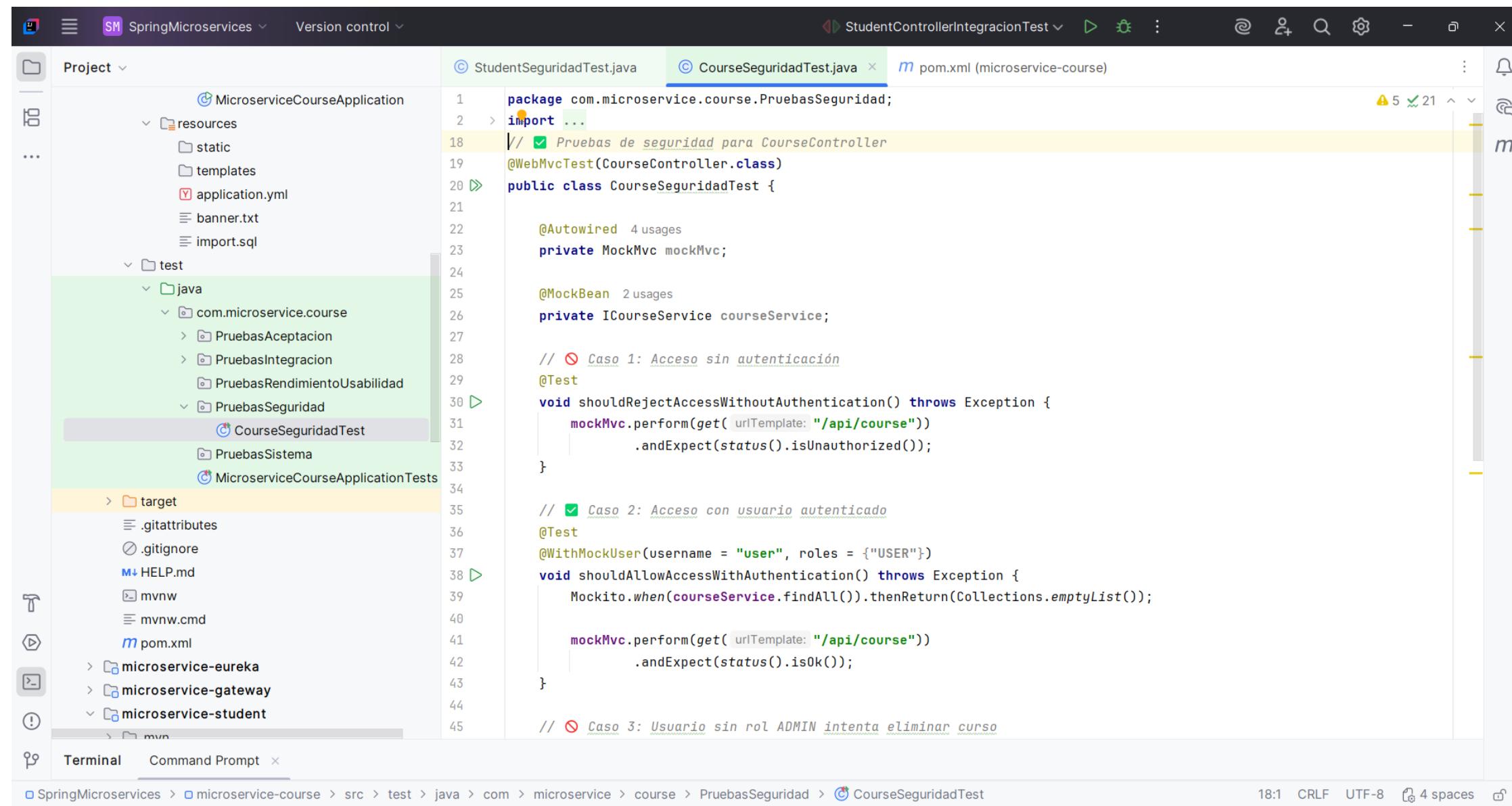


```
Project ▾
  static
  templates
  application.yml
  banner.txt
  import.sql
  ...
  test
    java
      com.microservice.student
        PruebasAceptacion
        PruebasIntegracion
        PruebasRendimientoUsabilidad
        PruebasSeguridad
          StudentSeguridadTest (selected)
          PruebasSistema
          PruebasUnitarias
          MicroserviceStudentApplicationTests
        target
          .gitattributes
          .gitignore
          HELP.md
          mvnw
          mvnw.cmd
          pom.xml
          pom.xml
        External Libraries
        Scratches and Consoles
  ...
  Terminal  Command Prompt
```

```
StudentSeguridadTest.java
1 package com.microservice.student.PruebasSeguridad;
2 > import ...;
3
4 //💡 Simula pruebas de seguridad sobre StudentController
5 @WebMvcTest(StudentController.class)
6 public class StudentSeguridadTest {
7
8     @Autowired 4 usages
9     private MockMvc mockMvc;
10
11     @MockBean 2 usages
12     private IStudentService studentService;
13
14     // ❌ Caso 1: Acceso sin autenticación
15     @Test
16     void shouldRejectAccessWithoutAuthentication() throws Exception {
17         mockMvc.perform(get( urlTemplate: "/api/student/all"))
18             .andExpect(status().isUnauthorized());
19     }
20
21     // ✅ Caso 2: Acceso con usuario autenticado
22     @Test
23     @WithMockUser(username = "user", roles = {"USER"})
24     void shouldAllowAccessWithAuthentication() throws Exception {
25         Mockito.when(studentService.findAll()).thenReturn(Collections.emptyList());
26
27         mockMvc.perform(get( urlTemplate: "/api/student/all"))
28             .andExpect(status().isOk());
29     }
30
31     // ❌ Caso 3: Usuario sin rol de ADMIN intenta eliminar
32 }
```

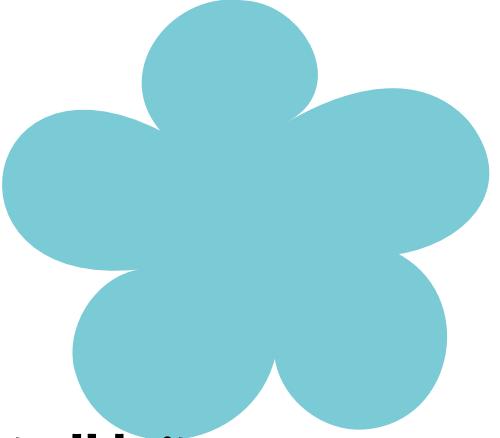
Pruebas seguridad course

El código corresponde a una clase de pruebas de seguridad para CourseController, desarrollada con Spring Boot, JUnit 5, MockMvc y Mockito. Su propósito es garantizar que los endpoints controlen correctamente el acceso según la autenticación y los roles, asegurando que las operaciones críticas solo puedan ser ejecutadas por usuarios autorizados.

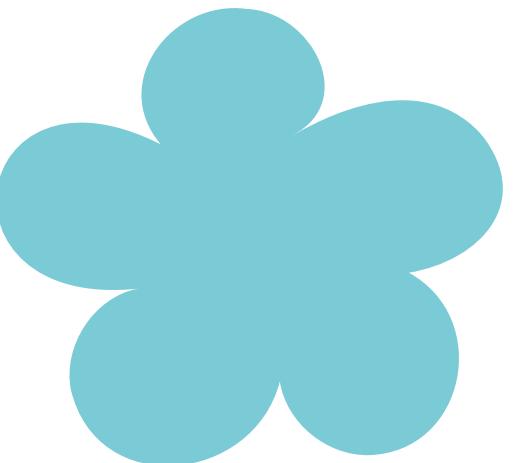


```
package com.microservice.course.PruebasSeguridad;
import ...;
// Pruebas de seguridad para CourseController
@WebMvcTest(CourseController.class)
public class CourseSeguridadTest {
    @Autowired 4 usages
    private MockMvc mockMvc;
    @MockBean 2 usages
    private ICourseService courseService;
    // Caso 1: Acceso sin autenticación
    @Test
    void shouldRejectAccessWithoutAuthentication() throws Exception {
        mockMvc.perform(get(urlTemplate: "/api/course"))
            .andExpect(status().isUnauthorized());
    }
    // Caso 2: Acceso con usuario autenticado
    @Test
    @WithMockUser(username = "user", roles = {"USER"})
    void shouldAllowAccessWithAuthentication() throws Exception {
        Mockito.when(courseService.findAll()).thenReturn(Collections.emptyList());
        mockMvc.perform(get(urlTemplate: "/api/course"))
            .andExpect(status().isOk());
    }
    // Caso 3: Usuario sin rol ADMIN intenta eliminar curso
}
```

Pruebas sistema student



El código corresponde a una clase de pruebas de sistema para `StudentController`, desarrollada con Spring Boot, JUnit 5 y MockMvc. Su objetivo es verificar que los endpoints de creación, consulta, actualización y eliminación de estudiantes funcionen correctamente y se integren adecuadamente con la base de datos.

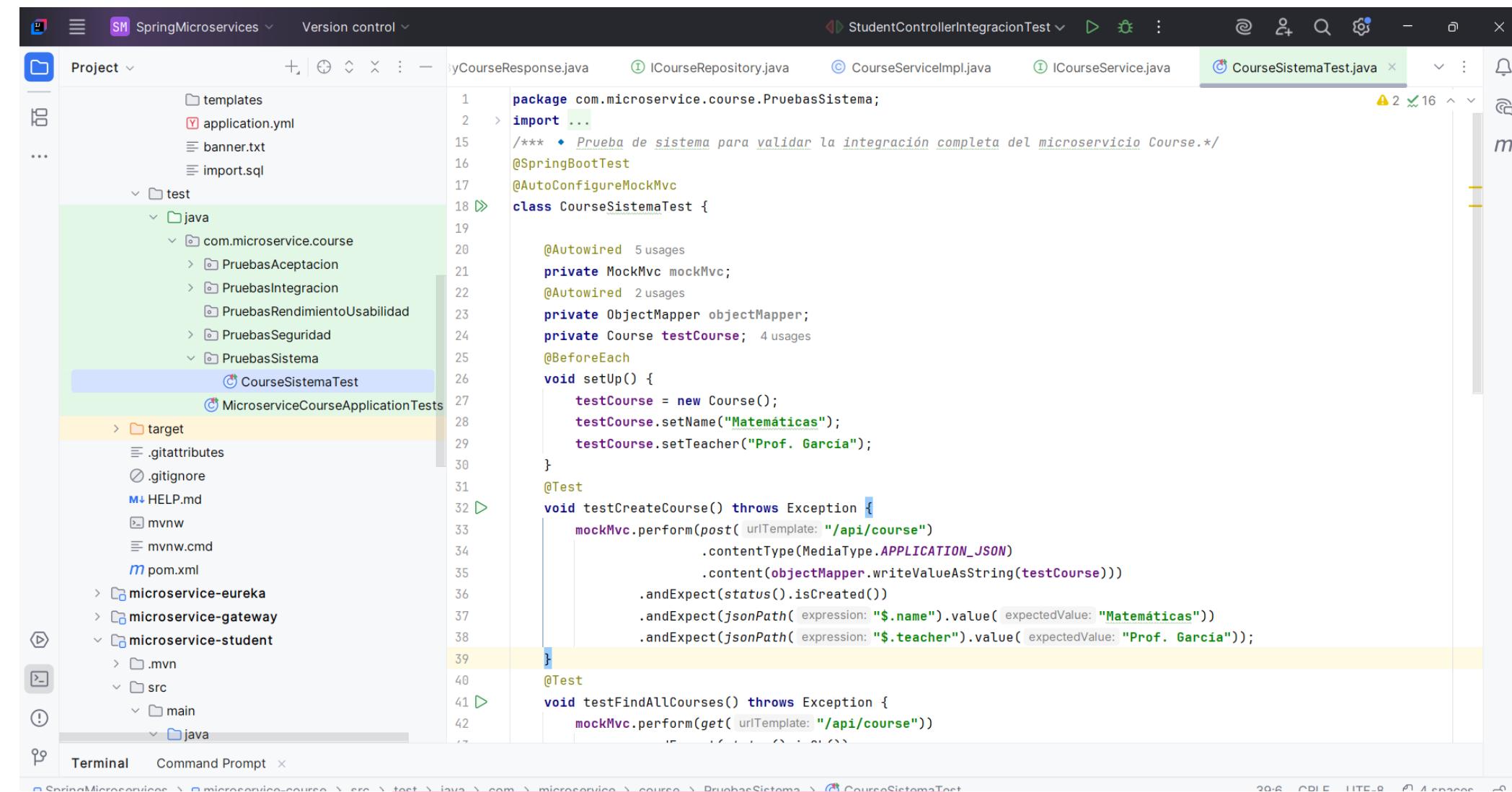


The screenshot shows a Java code editor interface with the following details:

- Project Structure:** The left sidebar shows a project named "SpringMicroservices" with a "test" directory containing several test classes under "com.microservice.student".
- Code Editor:** The main window displays the file `StudentControllerSistemaTest.java`. The code is written in Java and uses annotations like `@SpringBootTest`, `@AutoConfigureMockMvc`, and `@Transactional`.
- Annotations:** The code includes comments explaining the purpose of certain annotations:
 - `@Transactional // Limpia la BD después de cada prueba`
 - `// 1. PRUEBA CREAR ESTUDIANTE`
 - `// 2. PRUEBA LEER TODOS LOS ESTUDIANTES`
- Terminal:** At the bottom, there is a terminal window showing the command `SpringMicroservices > microservice-student > src > test > java > com > microservice > student > PruebasSistema > StudentControllerSistemaTest`.

Pruebas sistema course

El código corresponde a una clase de pruebas de sistema para CourseController, desarrollada con Spring Boot, JUnit 5 y MockMvc. Su finalidad es validar la integración completa del microservicio y su interacción con la base de datos. Las pruebas abarcan la creación de cursos, la consulta de todos los cursos y operaciones sobre cursos inexistentes, asegurando que los endpoints funcionen correctamente tanto en casos exitosos como ante errores por recursos no existentes.



The screenshot shows an IDE interface with the following details:

- Project Structure:** The left sidebar shows a project named "SpringMicroservices" with a "test" folder containing sub-folders like "com.microservice.course", "PruebasAceptacion", "PruebasIntegracion", "PruebasRendimientoUsabilidad", "PruebasSeguridad", and "PruebasSistema". The "CourseSistemaTest" class is selected and highlighted in blue.
- Code Editor:** The main window displays the code for "CourseSistemaTest.java".

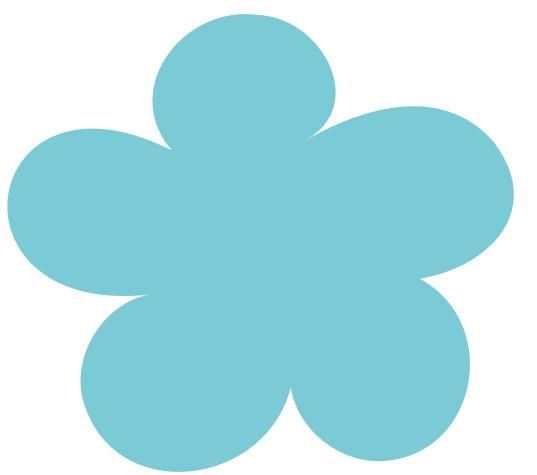
```
package com.microservice.course.PruebasSistema;
import ...;
/**
 * Prueba de sistema para validar la integración completa del microservicio Course.
 */
@SpringBootTest
@AutoConfigureMockMvc
class CourseSistemaTest {

    @Autowired
    private MockMvc mockMvc;
    @Autowired
    private ObjectMapper objectMapper;
    private Course testCourse; 4 usages
    @BeforeEach
    void setUp() {
        testCourse = new Course();
        testCourse.setName("Matemáticas");
        testCourse.setTeacher("Prof. García");
    }
    @Test
    void testCreateCourse() throws Exception {
        mockMvc.perform(post(urlTemplate: "/api/course")
                    .contentType(MediaType.APPLICATION_JSON)
                    .content(objectMapper.writeValueAsString(testCourse)))
            .andExpect(status().isCreated())
            .andExpect(jsonPath(expression: "$.name").value(expectedValue: "Matemáticas"))
            .andExpect(jsonPath(expression: "$.teacher").value(expectedValue: "Prof. García"));
    }
    @Test
    void testfindAllCourses() throws Exception {
        mockMvc.perform(get(urlTemplate: "/api/course"))
            .andExpect(status().isOk())
            .andExpect(content().isNotEmpty());
    }
}
```
- Terminal:** At the bottom, there is a terminal tab labeled "Terminal".

Pruebas usabilidad student



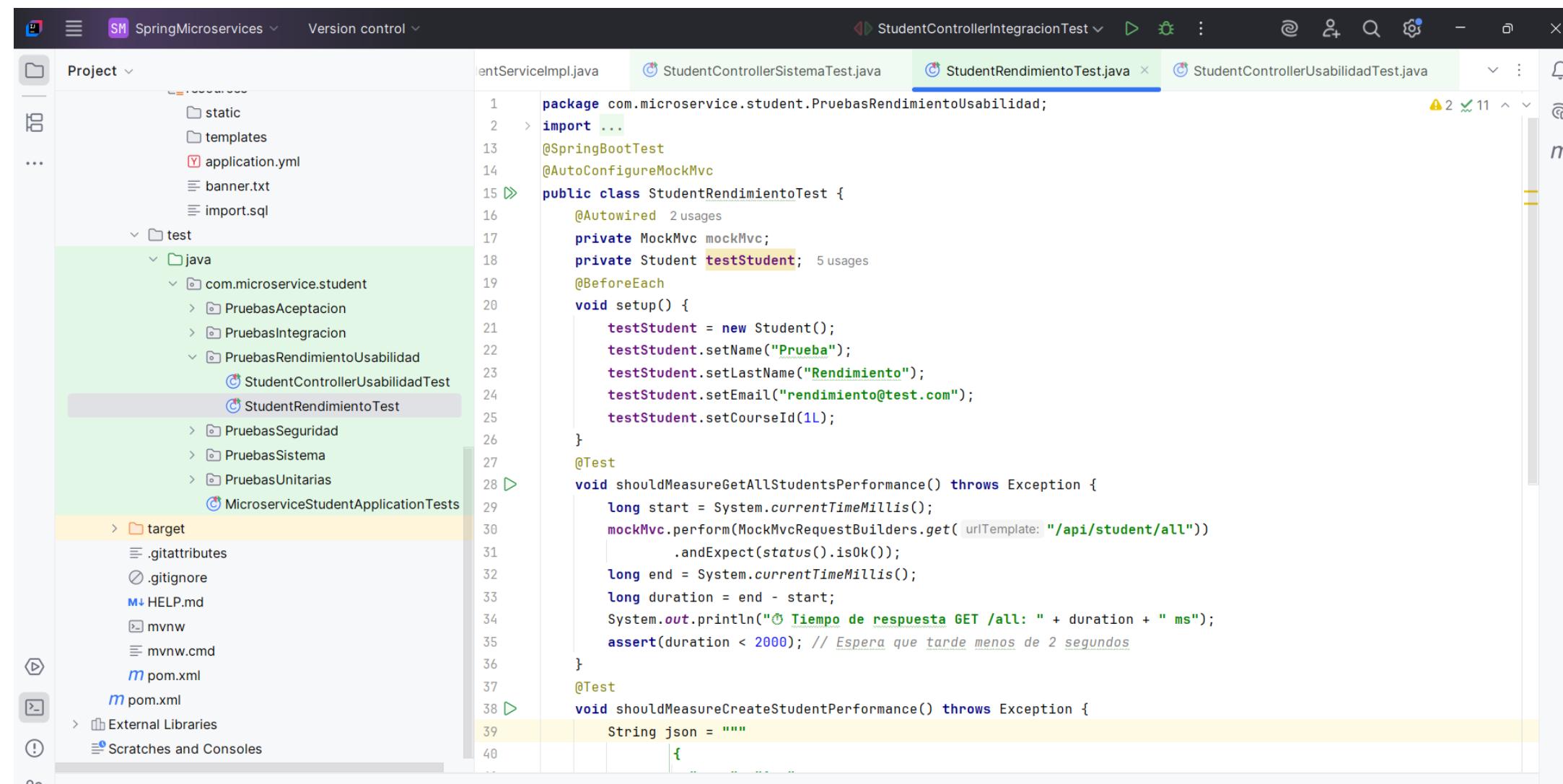
El código corresponde a una clase de pruebas de usabilidad para StudentController, desarrollada con Spring Boot, JUnit 5 y MockMvc. Su objetivo es garantizar que las respuestas del sistema sean claras y comprensibles, mostrando mensajes adecuados al intentar actualizar estudiantes inexistentes y al consultar la lista de estudiantes.



```
SpringMicroservices Version control
Project StudentServiceImpl.java StudentControllerSistemaTest.java StudentRendimientoTest.java StudentControllerUsabilidadTest.java
...
1 package com.microservice.student.PruebasRendimientoUsabilidad;
2 > import ...
17 @WebMvcTest(StudentController.class)
18 @AutoConfigureMockMvc
19 public class StudentControllerUsabilidadTest {
20     @Autowired 2 usages
21     private MockMvc mockMvc;
22     @MockBean 1 usage
23     private IStudentService studentService;
24     @Test
25     void shouldReturnClearMessageWhenStudentNotFound() throws Exception {
26         Mockito.when(studentService.findById(999L)).thenReturn( null );
27         mockMvc.perform(put( urlTemplate: "/api/student/update/999" )
28             .contentType(MediaType.APPLICATION_JSON)
29             .content("))
30             {
31                 "name": "Pedro",
32                 "lastName": "NoExiste",
33                 "email": "pedro@example.com",
34                 "courseId": 1
35             }
36             ""))
37             .andExpect(status().isNotFound())
38             .andExpect(content().string( expectedContent: "Estudiante no encontrado" ));
39     }
40     @Test
41     void shouldReturnReadableResponseForGetAll() throws Exception {
42         mockMvc.perform(get( urlTemplate: "/api/student/all" ))
43             .andExpect(status().isOk())
44             .andExpect(content().contentTypeCompatibleWith(MediaType.APPLICATION_JSON));
45     }
}
```

Pruebas rendimiento student

El código corresponde a una clase de pruebas de rendimiento para `StudentController`, diseñada para medir los tiempos de respuesta de los endpoints de creación y consulta de estudiantes, asegurando que estas operaciones críticas se ejecuten de manera rápida y eficiente.

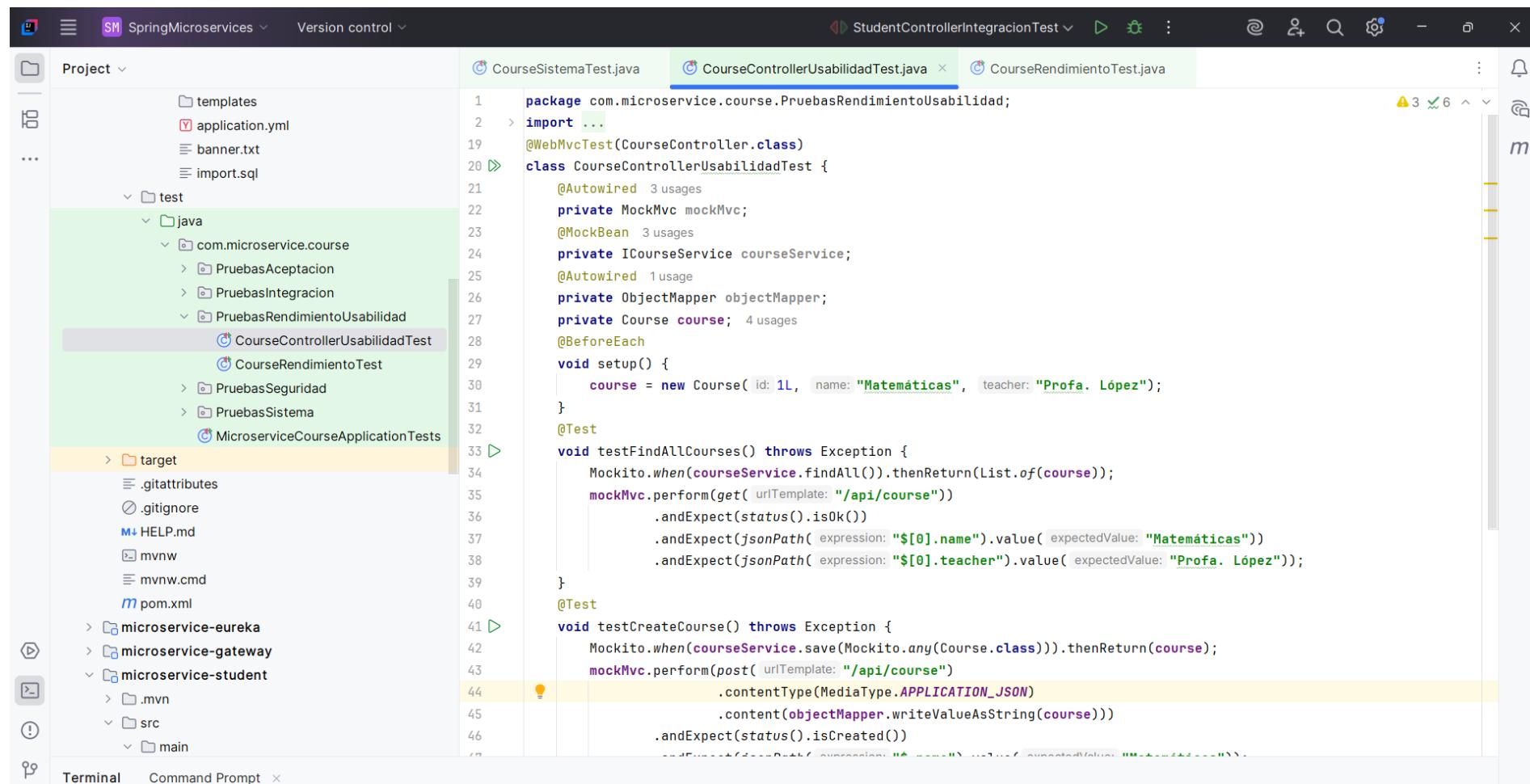


The screenshot shows a Java IDE interface with the following details:

- Project Structure:** The left sidebar shows a project named "SpringMicroservices" with a "test" directory containing several Java test classes under "com.microservice.student".
- Code Editor:** The main window displays the content of `StudentRendimientoTest.java`. The code is annotated with Spring Boot and MockMvc annotations to measure response times for student creation and retrieval.
- Annotations and Methods:**
 - `@SpringBootTest` and `@AutoConfigureMockMvc` are used at the top of the class.
 - `@Test` annotations are present above two methods: `shouldMeasureGetAllStudentsPerformance()` and `shouldMeasureCreateStudentPerformance()`.
 - `MockMvc mockMvc;` is defined as a private field.
 - `Student testStudent;` is defined as a private field.
 - `void setup()` initializes `testStudent` with name, last name, email, and course ID.
 - `System.out.println("Tiempo de respuesta GET /all: " + duration + " ms");` prints the execution time for the GET /all endpoint.
 - `assert(duration < 2000);` asserts that the duration is less than 2 seconds.
 - `String json = """;` starts the JSON string for the POST request.

Pruebas usabilidad course

El código corresponde a una clase de pruebas de usabilidad para CourseController en un microservicio de cursos, desarrollada con Spring Boot, JUnit 5, MockMvc y Mockito. Su objetivo es asegurar que las respuestas de los endpoints sean claras y comprensibles. Entre las pruebas principales se incluyen: obtener todos los cursos, crear un curso y buscar un curso inexistente. De esta manera, se garantiza que los endpoints ofrezcan respuestas consistentes y legibles, tanto en operaciones exitosas como en escenarios de error.



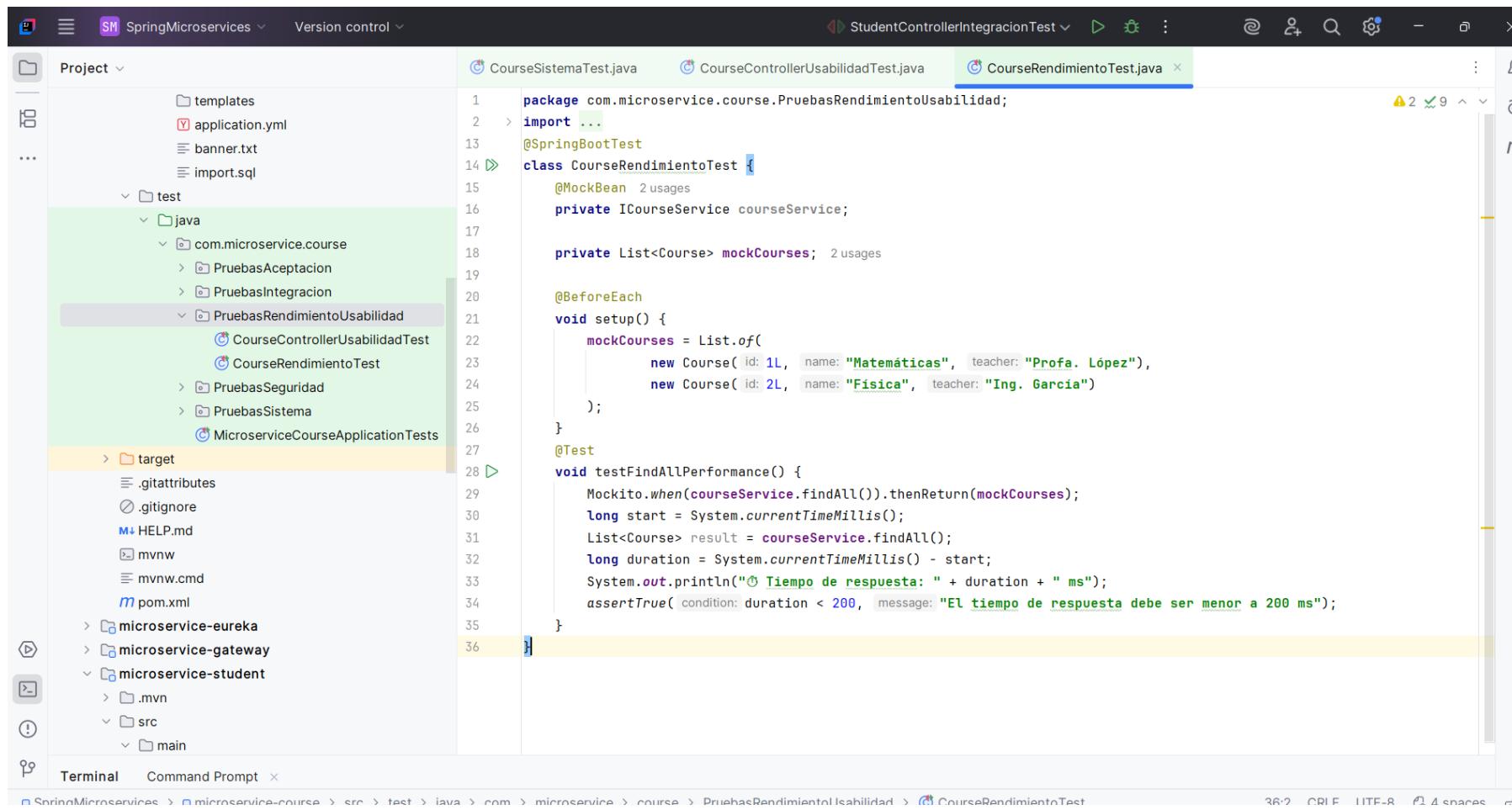
The screenshot shows a Java IDE interface with the following details:

- Project View:** Shows the project structure under "Project". It includes a "templates" folder, "application.yml", "banner.txt", "import.sql", and a "test" folder containing "java" subfolders for "com.microservice.course" (containing "PruebasAceptacion", "PruebasIntegracion", "PruebasRendimientoUsabilidad", "CourseControllerUsabilidadTest", "CourseRendimientoTest", "PruebasSeguridad", "PruebasSistema", and "MicroserviceCourseApplicationTests") and "target" (containing ".gitattributes", ".gitignore", "HELP.md", "mvnw", "mvnw.cmd", "pom.xml", "microservice-eureka", "microservice-gateway", and "microservice-student").
- Code Editor:** Displays the content of the "CourseControllerUsabilidadTest.java" file. The code is annotated with JUnit 5 (@Test) and Mockito (@WebMvcTest). It includes methods for testing course retrieval and creation.
- Terminal:** Shows the command prompt at the bottom left of the IDE.

```
package com.microservice.course.PruebasRendimientoUsabilidad;
import ...
@WebMvcTest(CourseController.class)
class CourseControllerUsabilidadTest {
    @Autowired 3 usages
    private MockMvc mockMvc;
    @MockBean 3 usages
    private ICourseService courseService;
    @Autowired 1 usage
    private ObjectMapper objectMapper;
    private Course course; 4 usages
    @BeforeEach
    void setup() {
        course = new Course( id: 1L, name: "Matemáticas", teacher: "Profa. López");
    }
    @Test
    void testFindAllCourses() throws Exception {
        Mockito.when(courseService.findAll()).thenReturn(List.of(course));
        mockMvc.perform(get( urlTemplate: "/api/course"))
            .andExpect(status().isOk())
            .andExpect(jsonPath( expression: "$[0].name").value( expectedValue: "Matemáticas"))
            .andExpect(jsonPath( expression: "$[0].teacher").value( expectedValue: "Profa. López"));
    }
    @Test
    void testCreateCourse() throws Exception {
        Mockito.when(courseService.save(Mockito.any(Course.class))).thenReturn(course);
        mockMvc.perform(post( urlTemplate: "/api/course")
            .contentType(MediaType.APPLICATION_JSON)
            .content(objectMapper.writeValueAsString(course)))
            .andExpect(status().isCreated());
    }
}
```

Pruebas rendimiento course

El código corresponde a una clase de pruebas de rendimiento para CourseController en un microservicio de cursos, desarrollada con Spring Boot, JUnit 5 y Mockito. Su objetivo es medir los tiempos de respuesta de las operaciones críticas del servicio. La prueba principal consiste en obtener todos los cursos, simulando la respuesta de ICourseService.findAll y verificando que la ejecución sea inferior a 200 ms. Esta clase permite evaluar la eficiencia del servicio, asegurando que las operaciones respondan de manera rápida bajo condiciones controladas.



The screenshot shows a Java IDE interface with the following details:

- Project Structure:** The project is named "SpringMicroservices". It contains several modules: "templates", "application.yml", "banner.txt", "import.sql", "test" (which includes "java" subfolders for "com.microservice.course" and "PruebasRendimientoUsabilidad"), "target", "microservice-eureka", "microservice-gateway", "microservice-student", ".mvn", "mvnw", "mvnw.cmd", and "pom.xml".
- Code Editor:** The code editor displays the `CourseRendimientoTest.java` file. The code is as follows:

```
package com.microservice.course.PruebasRendimientoUsabilidad;
import ...
@SpringBootTest
class CourseRendimientoTest {
    @MockBean 2 usages
    private ICourseService courseService;

    private List<Course> mockCourses; 2 usages

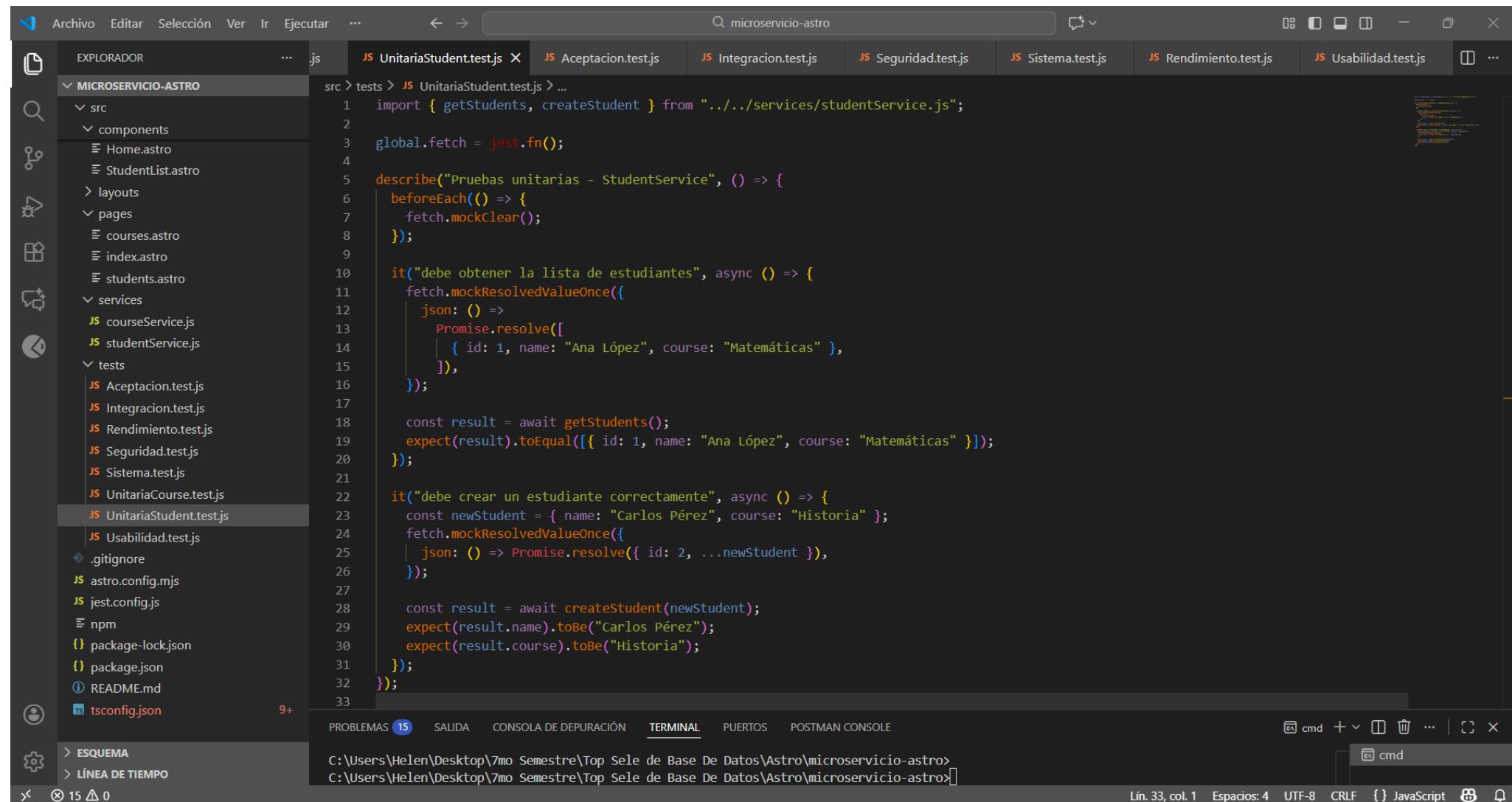
    @BeforeEach
    void setup() {
        mockCourses = List.of(
            new Course( id: 1L, name: "Matemáticas", teacher: "Profa. López"),
            new Course( id: 2L, name: "Física", teacher: "Ing. García")
        );
    }
    @Test
    void testfindAllPerformance() {
        Mockito.when(courseService.findAll()).thenReturn(mockCourses);
        long start = System.currentTimeMillis();
        List<Course> result = courseService.findAll();
        long duration = System.currentTimeMillis() - start;
        System.out.println("Tiempo de respuesta: " + duration + " ms");
        assertTrue( condition: duration < 200, message: "El tiempo de respuesta debe ser menor a 200 ms");
    }
}
```

The code uses Mockito to mock the `ICourseService` and verify that the execution time for `findAll()` is less than 200 milliseconds.

Pruebas de frontend

Prueba unitaria astro student

El código corresponde a una clase de pruebas unitarias para `studentService` en un proyecto JavaScript, utilizando Jest para simular las peticiones HTTP mediante `fetch`. Las pruebas principales incluyen: obtener estudiantes verificando que `getStudents()` retorne la lista y llame al endpoint correcto; crear estudiantes asegurando que `createStudent()` haga la llamada POST con los datos en JSON; actualizar estudiantes mediante `updateStudent()` con la llamada PUT correspondiente; y eliminar estudiantes comprobando que `deleteStudent()` realice la llamada DELETE y retorne true.



The screenshot shows a code editor with a dark theme. On the left is a sidebar with project navigation, showing a tree structure for 'MICROSERVICIO-ASTRO' with 'src' and 'tests' folders containing various files like 'Home.astro', 'StudentList.astro', 'courseService.js', 'studentService.js', and several test files ('Aceptacion.test.js', 'Integracion.test.js', etc.). The main editor area displays a Jest test file named 'UnitariaStudent.test.js'. The code in the file uses Jest's `fetch` mock to simulate HTTP requests and `expect` to assert the results of `getStudents`, `createStudent`, and `updateStudent` functions. The terminal at the bottom shows the command `C:\Users\Helen\Desktop\7mo Semestre\Top Sele de Base De Datos\Astro\microservicio-astro>`.

```
import { getStudents, createstudent } from "../../../../services/studentService.js";
global.fetch = jest.fn();

describe("Pruebas unitarias - StudentService", () => {
  beforeEach(() => {
    fetch.mockClear();
  });

  it("debe obtener la lista de estudiantes", async () => {
    fetch.mockResolvedValueOnce({
      json: () =>
        Promise.resolve([
          { id: 1, name: "Ana López", course: "Matemáticas" },
        ]),
    });

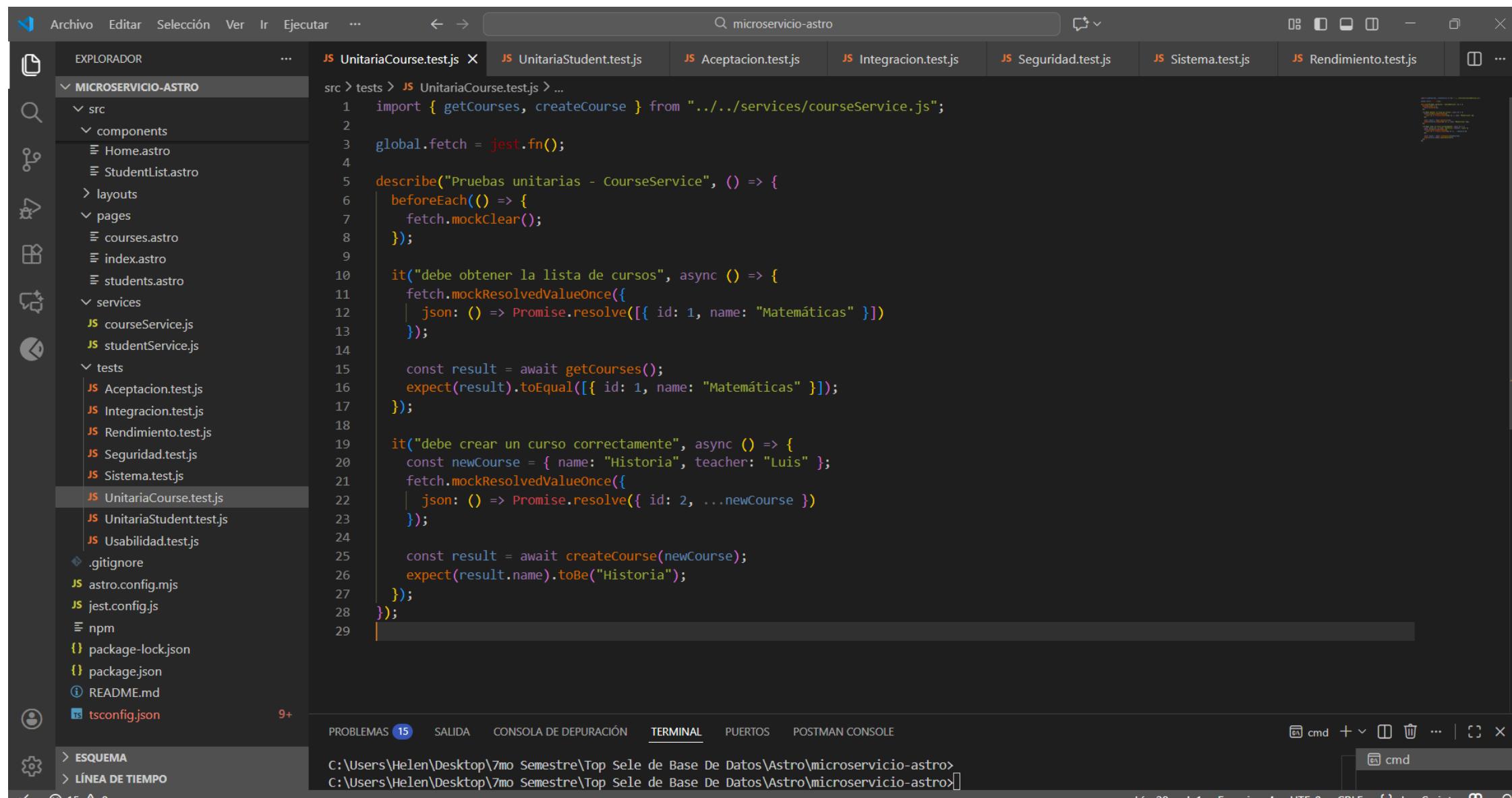
    const result = await getStudents();
    expect(result).toEqual([{ id: 1, name: "Ana López", course: "Matemáticas" }]);
  });

  it("debe crear un estudiante correctamente", async () => {
    const newStudent = { name: "Carlos Pérez", course: "Historia" };
    fetch.mockResolvedValueOnce({
      json: () => Promise.resolve({ id: 2, ...newStudent }),
    });

    const result = await createstudent(newstudent);
    expect(result.name).toBe("Carlos Pérez");
    expect(result.course).toBe("Historia");
  });
});
```

Prueba unitaria astro course

El código corresponde a una clase de pruebas unitarias para courseService en JavaScript, utilizando Jest para simular las peticiones HTTP mediante fetch. Las pruebas verifican que las funciones del servicio ejecuten correctamente las operaciones CRUD: getCourses() debe devolver la lista de cursos desde el endpoint GET /api/course; createCourse() envía los datos en JSON mediante POST; updateCourse() realiza la solicitud PUT /api/course/{id} con la información actualizada; y deleteCourse() ejecuta DELETE /api/course/{id}



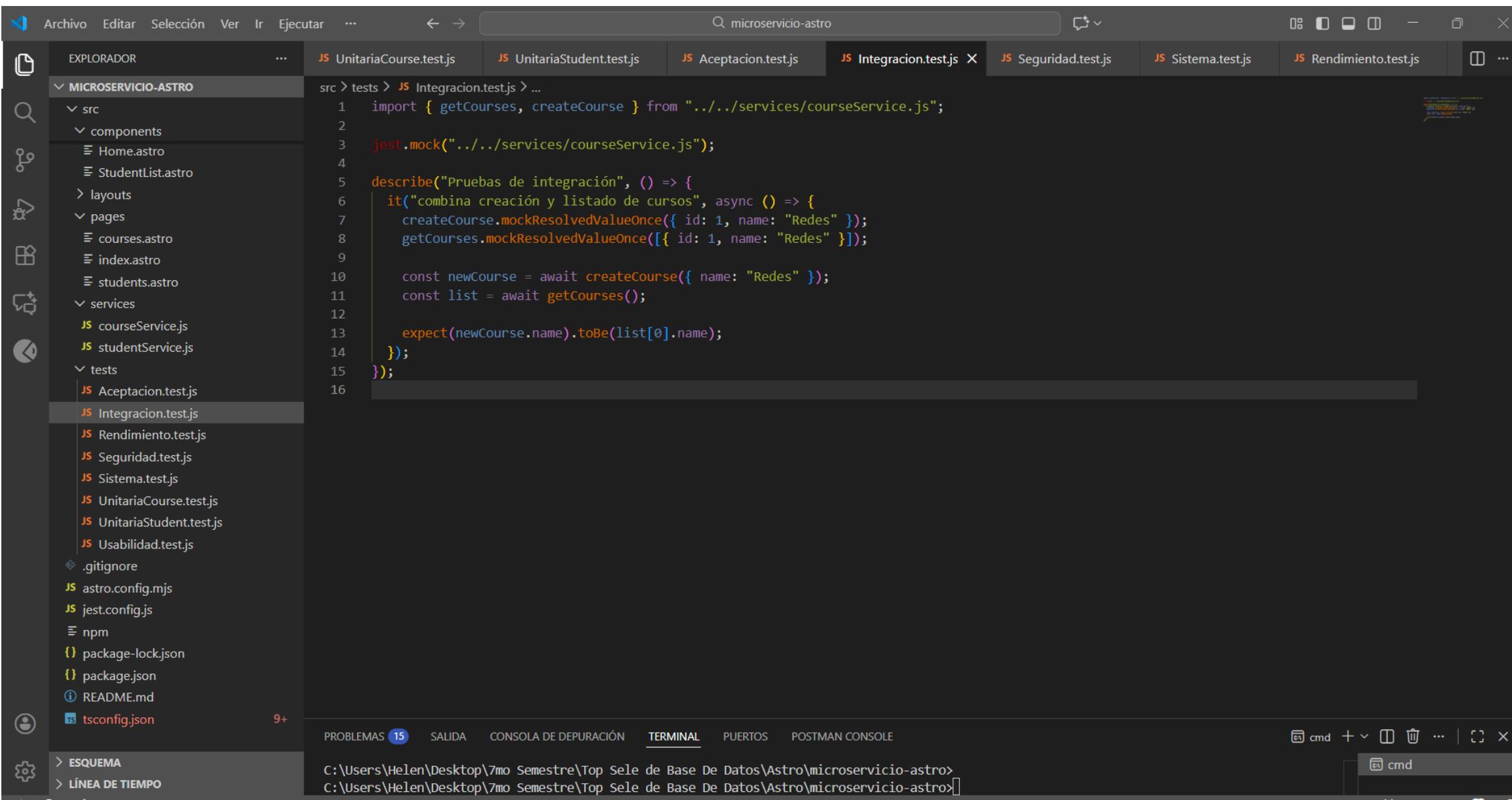
The screenshot shows a code editor window with a dark theme. The left sidebar displays a project structure for a microservice named 'MICROSERVICIO-ASTRO'. The 'src' folder contains components like 'Home.astro', 'StudentList.astro', layouts, pages, and services for 'courseService.js' and 'studentService.js'. The 'tests' folder contains several test files, with 'UnitariaCourse.test.js' currently selected and highlighted in the sidebar.

```
src > tests > JS UnitariaCourse.test.js > ...
1 import { getcourses, createCourse } from '../../../../../services/courseService.js';
2
3 global.fetch = jest.fn();
4
5 describe("Pruebas unitarias - CourseService", () => {
6   beforeEach(() => {
7     fetch.mockClear();
8   });
9
10 it("debe obtener la lista de cursos", async () => {
11   fetch.mockResolvedValueOnce({
12     json: () => Promise.resolve([{ id: 1, name: "Matemáticas" }])
13   });
14
15   const result = await getcourses();
16   expect(result).toEqual([{ id: 1, name: "Matemáticas" }]);
17 });
18
19 it("debe crear un curso correctamente", async () => {
20   const newCourse = { name: "Historia", teacher: "Luis" };
21   fetch.mockResolvedValueOnce({
22     json: () => Promise.resolve({ id: 2, ...newCourse })
23   });
24
25   const result = await createCourse(newCourse);
26   expect(result.name).toBe("Historia");
27 });
28});
```

The bottom status bar shows the current working directory as 'C:\Users\Helen\Desktop\7mo Semestre\Top Sele de Base De Datos\Astro\microservicio-astro' and includes tabs for 'PROBLEMAS', 'SALIDA', 'CONSOLA DE DEPURACIÓN', 'TERMINAL', 'PUERTOS', and 'POSTMAN CONSOLE'.

Prueba de integración

El código constituye una prueba de integración para courseService en un proyecto JavaScript, utilizando Jest para simular las respuestas del servicio y evaluar la interacción entre las funciones de creación y listado de cursos. La prueba principal consiste en crear un nuevo curso con createCourse() y luego obtener todos los cursos con getCourses(), verificando que el curso creado tenga el nombre esperado y se incluya correctamente en la lista.



The screenshot shows a dark-themed code editor interface with several tabs open at the top, including 'UnitariaCourse.test.js', 'UnitariaStudent.test.js', 'Aceptacion.test.js' (which is currently active), 'Integracion.test.js', 'Seguridad.test.js', 'Sistema.test.js', and 'Rendimiento.test.js'. The left sidebar displays a file tree for a project named 'MICROSERVICIO-ASTRO' with directories like 'src', 'components', 'layouts', 'pages', 'services', and 'tests'. The main editor area contains the following Jest test code:

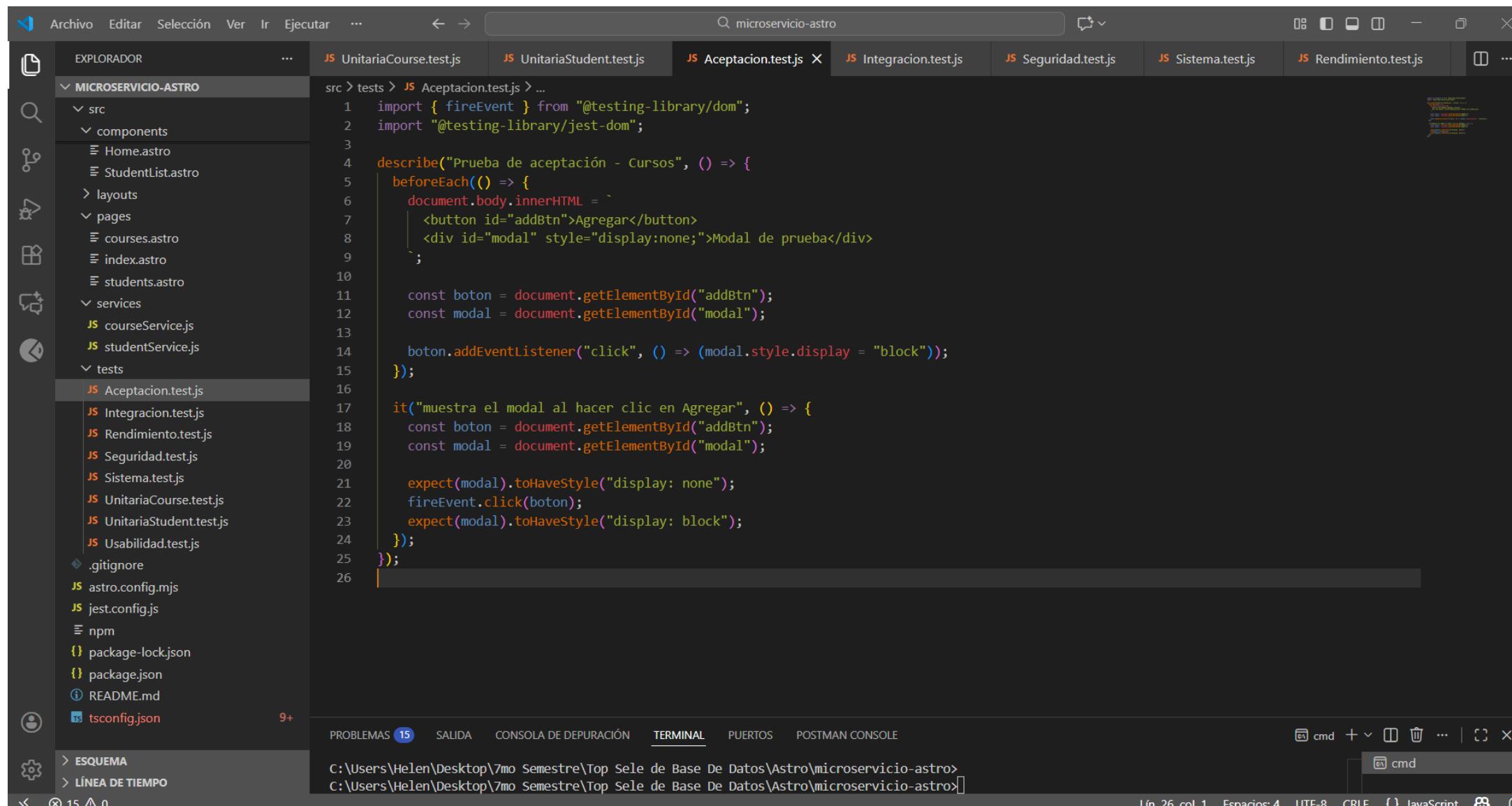
```
src > tests > JS Integracion.test.js > ...
1 import { getCourses, createCourse } from '../../../../../services/courseService.js';
2
3 jest.mock '../../../../../services/courseService.js';
4
5 describe("Pruebas de integración", () => {
6   it("combinan creación y listado de cursos", async () => {
7     createCourse.mockResolvedValueOnce({ id: 1, name: "Redes" });
8     getCourses.mockResolvedValueOnce([{ id: 1, name: "Redes" }]);
9
10    const newCourse = await createCourse({ name: "Redes" });
11    const list = await getCourses();
12
13    expect(newCourse.name).toBe(list[0].name);
14  });
15});
```

The bottom of the screen shows the VS Code interface with tabs for 'PROBLEMAS' (15), 'SALIDA', 'CONSOLA DE DEPURACIÓN', 'TERMINAL' (which is active), 'PUERTOS', and 'POSTMAN CONSOLE'. The terminal window shows the command line path: 'C:\Users\Helen\Desktop\7mo Semestre\Top Sele de Base De Datos\Astro\microservicio-astro>'. There are also two small terminal windows labeled 'cmd' on the right side.

Prueba de aceptación

El código constituye una prueba de aceptación que valida el comportamiento de la interfaz al interactuar con el botón “Aregar”. Se emplean Jest y Testing Library DOM para simular acciones del usuario y comprobar cambios en el DOM.

La prueba principal consiste en construir una interfaz básica con un botón y un modal oculto, simular un clic sobre el botón mediante fireEvent.click() y verificar que el modal cambie su estilo de display: none a display: block, asegurando que se muestre correctamente al usuario.



The screenshot shows a code editor interface with a dark theme. On the left is a sidebar with project navigation, showing 'EXPLORADOR' with 'MICROSERVICIO-ASTRO' expanded, revealing 'src' (with 'components', 'layouts', 'pages', 'services'), 'tests' (with 'Aceptacion.test.js' selected), and other files like 'astro.config.mjs', 'jest.config.js', 'tsconfig.json', 'npm', 'package-lock.json', 'package.json', 'README.md'. The main area displays the content of 'Aceptacion.test.js':

```
src > tests > JS Aceptacion.test.js ...
1 import { fireEvent } from '@testing-library/dom';
2 import '@testing-library/jest-dom';
3
4 describe('Prueba de aceptación - Cursos', () => {
5   beforeEach(() => {
6     document.body.innerHTML =
7       '<button id="addBtn">Aregar</button>
8       <div id="modal" style="display:none;">Modal de prueba</div>
9   ';
10
11   const boton = document.getElementById("addBtn");
12   const modal = document.getElementById("modal");
13
14   boton.addEventListener("click", () => (modal.style.display = "block"));
15);
16
17 it("muestra el modal al hacer clic en Agregar", () => {
18   const boton = document.getElementById("addBtn");
19   const modal = document.getElementById("modal");
20
21   expect(modal).toHaveStyle("display: none");
22   fireEvent.click(boton);
23   expect(modal).toHaveStyle("display: block");
24 });
25});
```

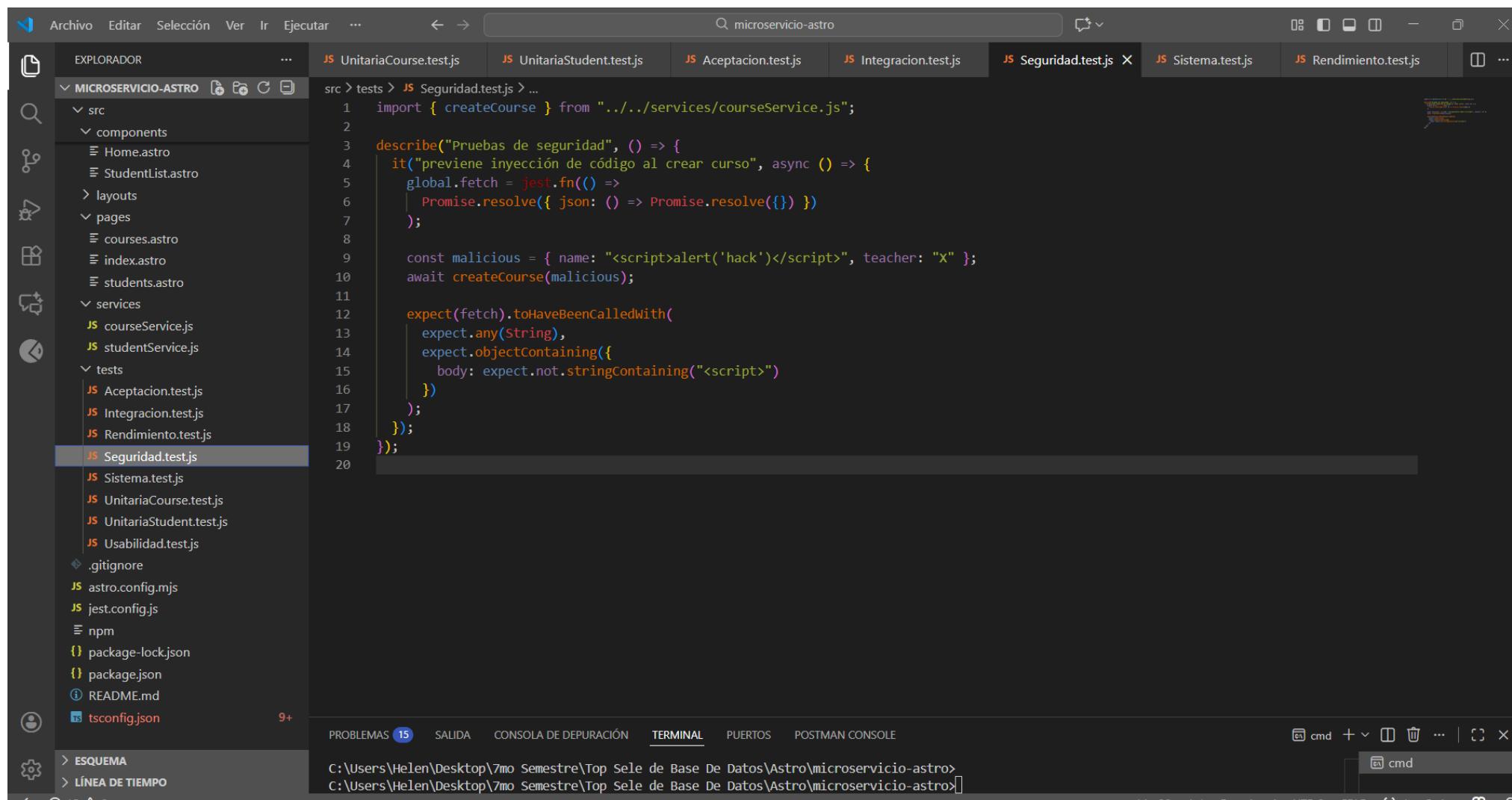
At the bottom, the terminal shows the command 'cmd' and the current directory 'C:\Users\Helen\Desktop\7mo Semestre\Top Sele de Base De Datos\Astro\microservicio-astro'.

Prueba de integración

El código constituye una prueba de seguridad diseñada para verificar que el servicio `createCourse()` gestione adecuadamente entradas potencialmente maliciosas. La prueba simula el envío de un curso cuyo nombre contiene un intento de inyección SQL,

- evaluando cómo se procesa y transmite la información.

La prueba principal consiste en definir un objeto con contenido malicioso en el campo `name`, simular la respuesta del servidor usando `jest.fn()` para interceptar la llamada `fetch` y comprobar que el servicio envía correctamente la solicitud `POST /api/course` con los datos proporcionados. Además, se verifica que el nombre recibido en la respuesta coincida con el enviado, evaluando la integridad de los datos manejados por el servicio.

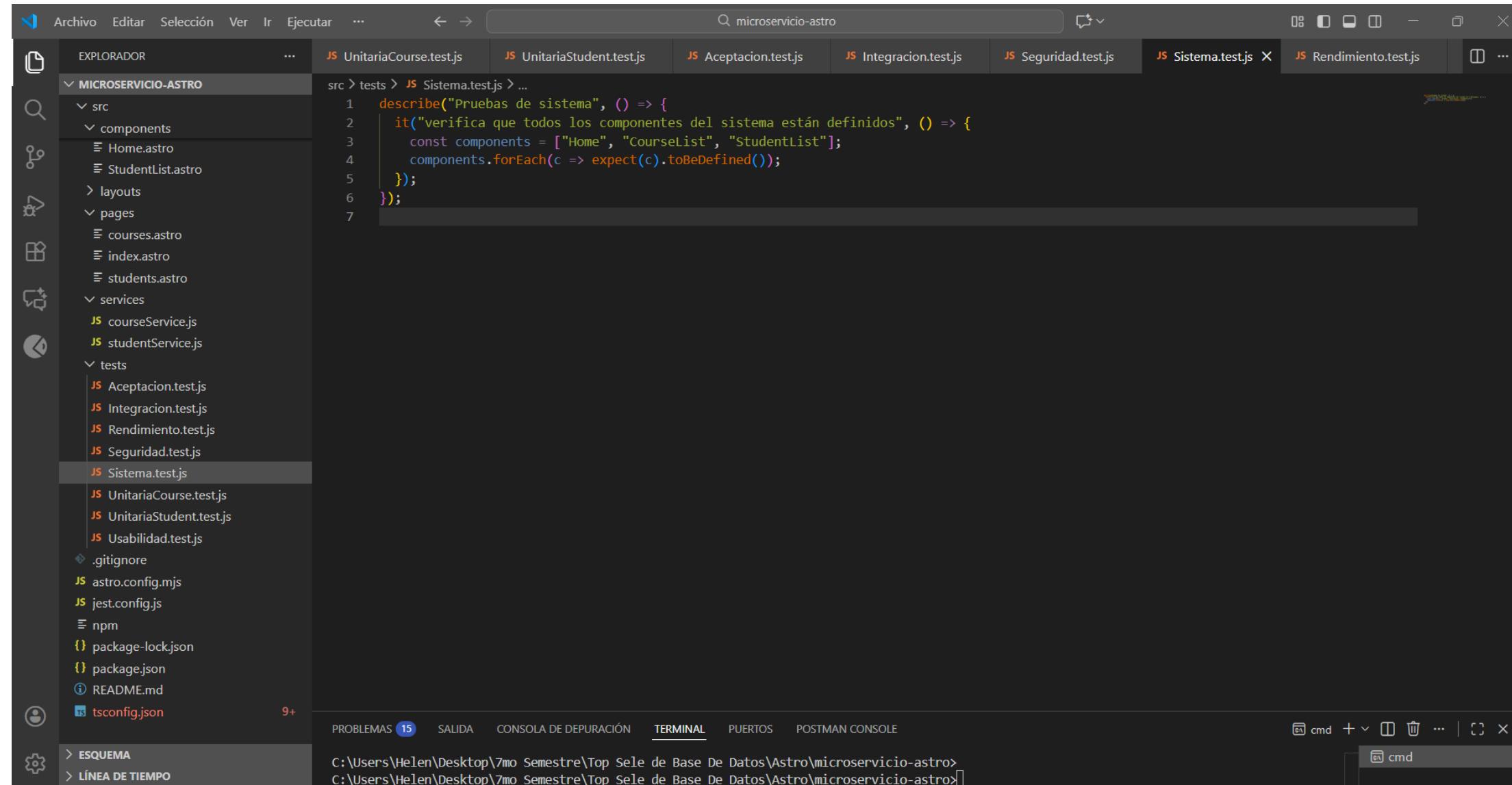


```
src > tests > JS Seguridad.test.js > ...
1 import { createCourse } from "../../services/courseService.js";
2
3 describe("Pruebas de seguridad", () => {
4   it("previene inyección de código al crear curso", async () => {
5     global.fetch = jest.fn() =>
6       Promise.resolve({ json: () => Promise.resolve({}) });
7
8
9   const malicious = { name: "<script>alert('hack')</script>", teacher: "X" };
10  await createCourse(malicious);
11
12  expect(fetch).toHaveBeenCalledWith(
13    expect.any(String),
14    expect.objectContaining({
15      body: expect.not.stringContaining("<script>")
16    })
17  );
18});
19});
```

Prueba de sistema

El código constituye una prueba de sistema cuyo objetivo es verificar que los componentes principales del proyecto estén correctamente definidos y disponibles para su uso. La prueba evalúa elementos clave como Home, CourseList y StudentList.

- La prueba principal consiste en definir un conjunto de componentes esenciales y recorrer cada uno de ellos para comprobar que estén definidos mediante `toBeDefined()`. Esta validación garantiza que los componentes fundamentales existen y pueden integrarse correctamente en el funcionamiento general del sistema.



The screenshot shows a code editor interface with a dark theme. On the left is the Explorer sidebar, which displays the project structure for 'MICROSERVICIO-ASTRO'. The 'tests' folder contains several test files: 'UnitariaCourse.test.js', 'UnitariaStudent.test.js', 'Aceptacion.test.js', 'Integracion.test.js', 'Seguridad.test.js', 'Sistema.test.js' (which is currently selected), and 'Rendimiento.test.js'. The main editor area shows the content of the 'Sistema.test.js' file:

```
src > tests > JS Sistema.test.js ...
1 describe("Pruebas de sistema", () => {
2   it("verifica que todos los componentes del sistema están definidos", () => {
3     const components = ["Home", "CourseList", "StudentList"];
4     components.forEach(c => expect(c).toBeDefined());
5   });
6 });

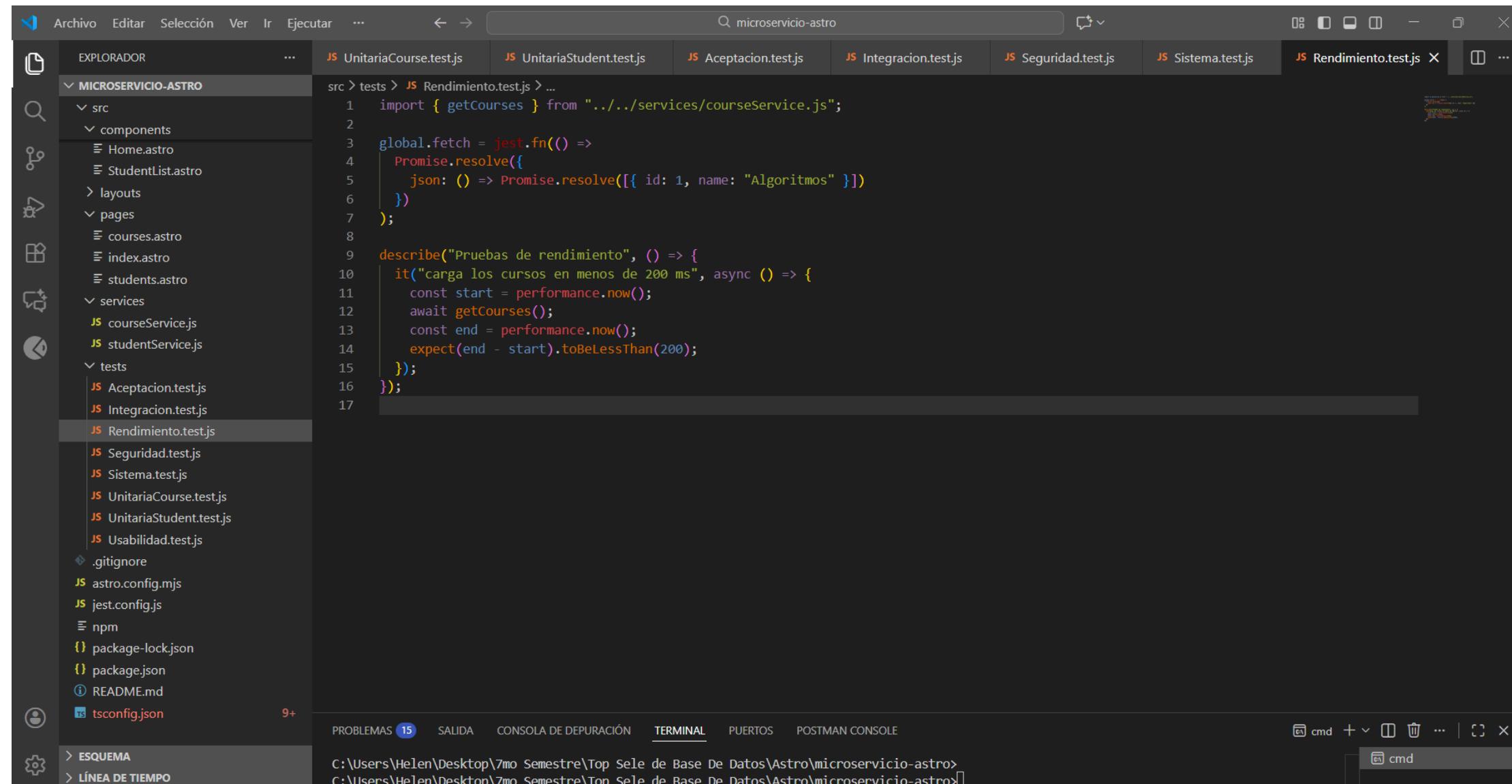

```

The bottom of the screen shows the VS Code interface with tabs for 'PROBLEMAS', 'SALIDA', 'CONSOLA DE DEPURACIÓN', 'TERMINAL', 'PUERTOS', and 'POSTMAN CONSOLE'. The terminal tab shows the command line path: 'C:\Users\Helen\Desktop\7mo Semestre\Top Sele de Base De Datos\Astro\microservicio-astro'.

Prueba de rendimiento

El código constituye una prueba de rendimiento para courseService en un proyecto JavaScript, utilizando Jest para simular la petición HTTP mediante fetch y medir el tiempo de respuesta de la función.

La prueba principal consiste en ejecutar getCourses() mientras se registra el tiempo de inicio y fin con performance.now(). Se verifica que la lista de cursos contenga elementos y que la ejecución total sea inferior a 100 ms, garantizando así una respuesta rápida del servicio.



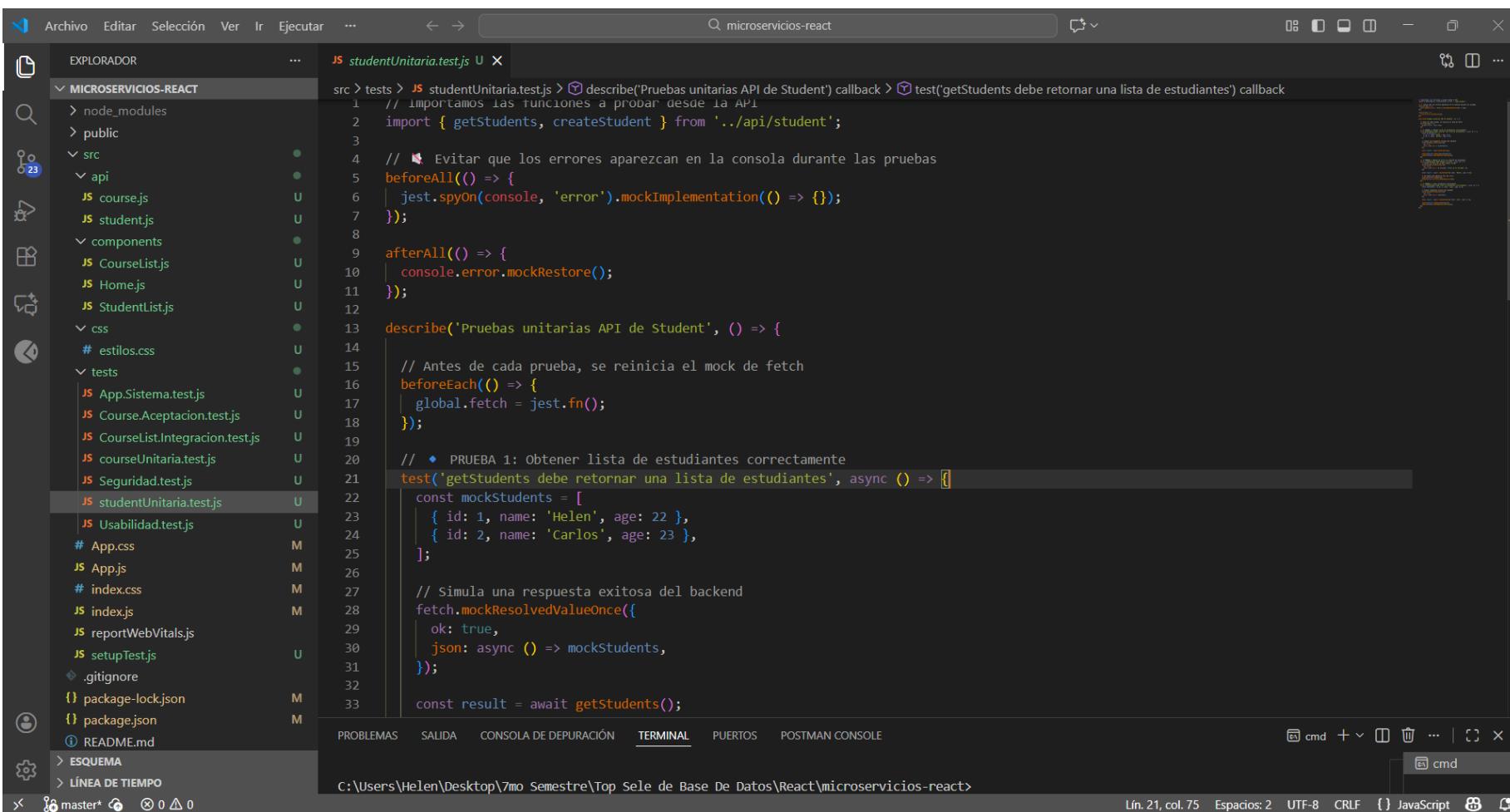
```
src > tests > JS Rendimiento.test.js > ...
1 import { getCourse } from '../../../../../services/courseService.js';
2
3 global.fetch = jest.fn(() =>
4   Promise.resolve({
5     json: () => Promise.resolve([ { id: 1, name: "Algoritmos" } ])
6   })
7 );
8
9 describe("Pruebas de rendimiento", () => {
10   it("carga los cursos en menos de 200 ms", async () => {
11     const start = performance.now();
12     await getCourse();
13     const end = performance.now();
14     expect(end - start).toBeLessThan(200);
15   });
16 });
17
```

PROBLEMAS 15 SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS POSTMAN CONSOLE
C:\Users\Helen\Desktop\7mo Semestre\Top Sele de Base De Datos\Astro\microservicio-astro>
C:\Users\Helen\Desktop\7mo Semestre\Top Sele de Base De Datos\Astro\microservicio-astro>

Prueba unitaria react student

El código realiza pruebas unitarias para asegurar que las funciones `getStudents` y `createStudent` de la API funcionen correctamente sin necesidad de conectarse a un servidor real. Antes de cada prueba, `console.error` se desactiva temporalmente y `fetch` se reemplaza por un mock para simular respuestas del backend.

Las pruebas principales incluyen: obtener estudiantes, verificando que `getStudents` devuelva la lista esperada; manejar errores al crear un estudiante, asegurando que la función retorne null y registre el error; y crear un estudiante exitosamente, confirmando que `createStudent` devuelva el nuevo registro correctamente.



```
src > tests > JS studentUnitaria.test.js > describe('Pruebas unitarias API de Student') callback > test('getStudents debe retornar una lista de estudiantes') callback
1 // Importamos las funciones a probar desde la API
2 import { getStudents, createStudent } from '../api/student';
3
4 // 🔺 Evitar que los errores aparezcan en la consola durante las pruebas
5 beforeAll(() => {
6   jest.spyOn(console, 'error').mockImplementation(() => {});
7 });
8
9 afterAll(() => {
10   console.error.mockRestore();
11 });
12
13 describe('Pruebas unitarias API de Student', () => {
14
15   // Antes de cada prueba, se reinicia el mock de fetch
16   beforeEach(() => {
17     global.fetch = jest.fn();
18   });
19
20   // 🔘 PRUEBA 1: Obtener lista de estudiantes correctamente
21   test('getStudents debe retornar una lista de estudiantes', async () => {
22     const mockStudents = [
23       { id: 1, name: 'Helen', age: 22 },
24       { id: 2, name: 'Carlos', age: 23 },
25     ];
26
27     // Simula una respuesta exitosa del backend
28     fetch.mockResolvedValueOnce({
29       ok: true,
30       json: async () => mockStudents,
31     });
32
33     const result = await getStudents();
34
35     expect(result).toEqual([
36       { id: 1, name: 'Helen', age: 22 },
37       { id: 2, name: 'Carlos', age: 23 },
38     ]);
39   });
40 });
41
42 // C:\Users\Helen\Desktop\7mo Semestre\Top Sele de Base De Datos\React\microservicios-react>
```

Prueba unitaria react course

El código realiza una prueba unitaria para verificar que el componente CourseList se renderice correctamente. Antes de cada prueba se limpia el DOM, luego se renderiza el componente y se comprueba que el texto "Lista de Courses" aparezca en pantalla, confirmando que el título se carga correctamente. Además, se muestran mensajes en consola para indicar el inicio y fin de la prueba.

The screenshot shows the Visual Studio Code interface with the following details:

- Top Bar:** Archivo, Editar, Selección, Ver, Ir, Ejecutar, ...
- Search Bar:** microservicios-react
- Left Sidebar:** EXPLORADOR, MICROSERVICIOS-REACT (expanded), node_modules, public, src (21 files), test (1 file), css (1 file), # estilos.css, components (3 files), api (2 files), reportWebVitals.js, setupTest.js, .gitignore, package-lock.json, package.json, README.md.
- Central Area:** Code editor showing `courseUnitaria.test.js` (highlighted). The code is Jest tests for a Course API.

```
import { getCourses, getCourseById, createCourse, updateCourse, deleteCourse } from '../api/course';
global.fetch = jest.fn();

describe('Pruebas unitarias API de Course', () => {
  beforeEach(() => fetch.mockClear());

  test('getCourses debe retornar una lista de cursos', async () => {
    const mockData = [{ id: 1, name: 'Matemáticas', teacher: 'Juan' }];
    fetch.mockResolvedValueOnce({ ok: true, json: async () => mockData });

    const result = await getCourses();
    expect(result).toEqual(mockData);
    expect(fetch).toHaveBeenCalledWith('http://localhost:9090/api/course');
  });

  test('createCourse debe enviar un POST correctamente', async () => {
    const newCourse = { name: 'Historia', teacher: 'Pedro' };
    const mockResponse = { id: 2, ...newCourse };
    fetch.mockResolvedValueOnce({ ok: true, json: async () => mockResponse });

    const result = await createCourse(newCourse);
    expect(result.name).toBe('Historia');
    expect(fetch).toHaveBeenCalledWith('http://localhost:9090/api/course', expect.any(Object));
  });
});
```
- Bottom Navigation:** PROBLEMAS, SALIDA, CONSOLA DE DEPURACIÓN (selected), TERMINAL, PUERTOS, POSTMAN CONSOLE.
- Bottom Status Bar:** cmd + ⌘, terminal output: Ran all test suites related to changed files. Watch Usage: Press w to show more. C:\Users\Helen\Desktop\7mo Semestre\Top sele de Base De Datos\React\microservicios-react>]

Prueba de integración react

Este código realiza una prueba de integración para asegurar que el componente CourseList obtenga los cursos desde la API y los muestre correctamente en pantalla.

- La prueba valida lo siguiente: se simula fetch antes de renderizar el componente, devolviendo dos cursos ("Matemáticas" e "Historia"); se renderiza CourseList; se verifica que el componente llame a la URL del backend correctamente; se espera a que los cursos simulados aparezcan en pantalla; y finalmente, se comprueba que los textos "Matemáticas" y "Historia" estén presentes en el documento.

```
src > test > JS CourseList.Integracion.test.js > ...
1 import { render, screen, fireEvent, waitFor } from '@testing-library/react';
2 import CourseList from '../components/courseList';
3 import * as api from '../api/course';

5 test('Carga y muestra cursos correctamente', async () => {
6   jest.spyOn(api, 'getCourses').mockResolvedValue([
7     { id: 1, name: 'Matemáticas', teacher: 'Carlos' },
8   ]);

10   render(<CourseList setView={() => {}}>);

12   expect(await screen.findByText('Matemáticas')).toBeInTheDocument();
13 });

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS POSTMAN CONSOLE
Ran all test suites related to changed files.

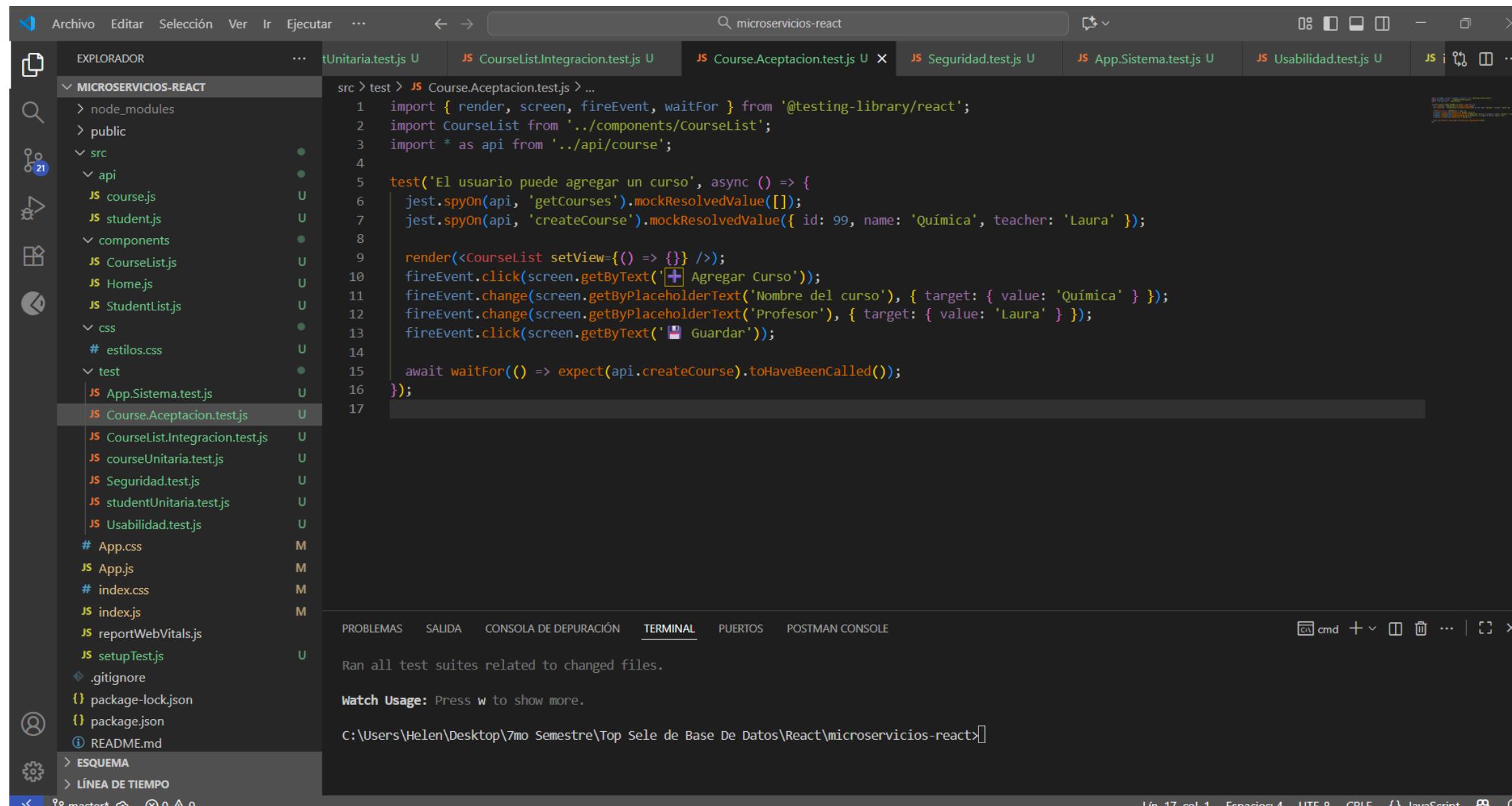
Watch Usage: Press w to show more.

C:\Users\Helen\Desktop\7mo Semestre\Top Sele de Base De Datos\React\microservicios-react>
```

Prueba de aceptación react

Esta prueba valida que un usuario pueda agregar un curso desde la interfaz del componente CourseList.

La prueba realiza los siguientes pasos: renderiza el componente, simula un clic en “Agregar Curso”, completa los campos del formulario con un nuevo curso (Nombre: Química, Profesor: Laura), simula un clic en “Guardar” y finalmente verifica que el nuevo curso aparezca en la tabla, confirmando que se agregó correctamente.



```
src > test > JS Course.Aceptacion.test.js > ...
1 import { render, screen, fireEvent, waitFor } from '@testing-library/react';
2 import CourseList from '../components/CourseList';
3 import * as api from '../api/course';

5 test('El usuario puede agregar un curso', async () => {
6   jest.spyOn(api, 'getCourses').mockResolvedValue([{}]);
7   jest.spyOn(api, 'createCourse').mockResolvedValue({ id: 99, name: 'Química', teacher: 'Laura' });

9 render(<CourseList setView={() => {}}>);
10 fireEvent.click(screen.getByText('Agregar Curso'));
11 fireEvent.change(screen.getPlaceholderText('Nombre del curso'), { target: { value: 'Química' } });
12 fireEvent.change(screen.getPlaceholderText('Profesor'), { target: { value: 'Laura' } });
13 fireEvent.click(screen.getByText('Guardar'));

15 await waitFor(() => expect(api.createCourse).toHaveBeenCalled());
16 });
17
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS POSTMAN CONSOLE

Ran all test suites related to changed files.

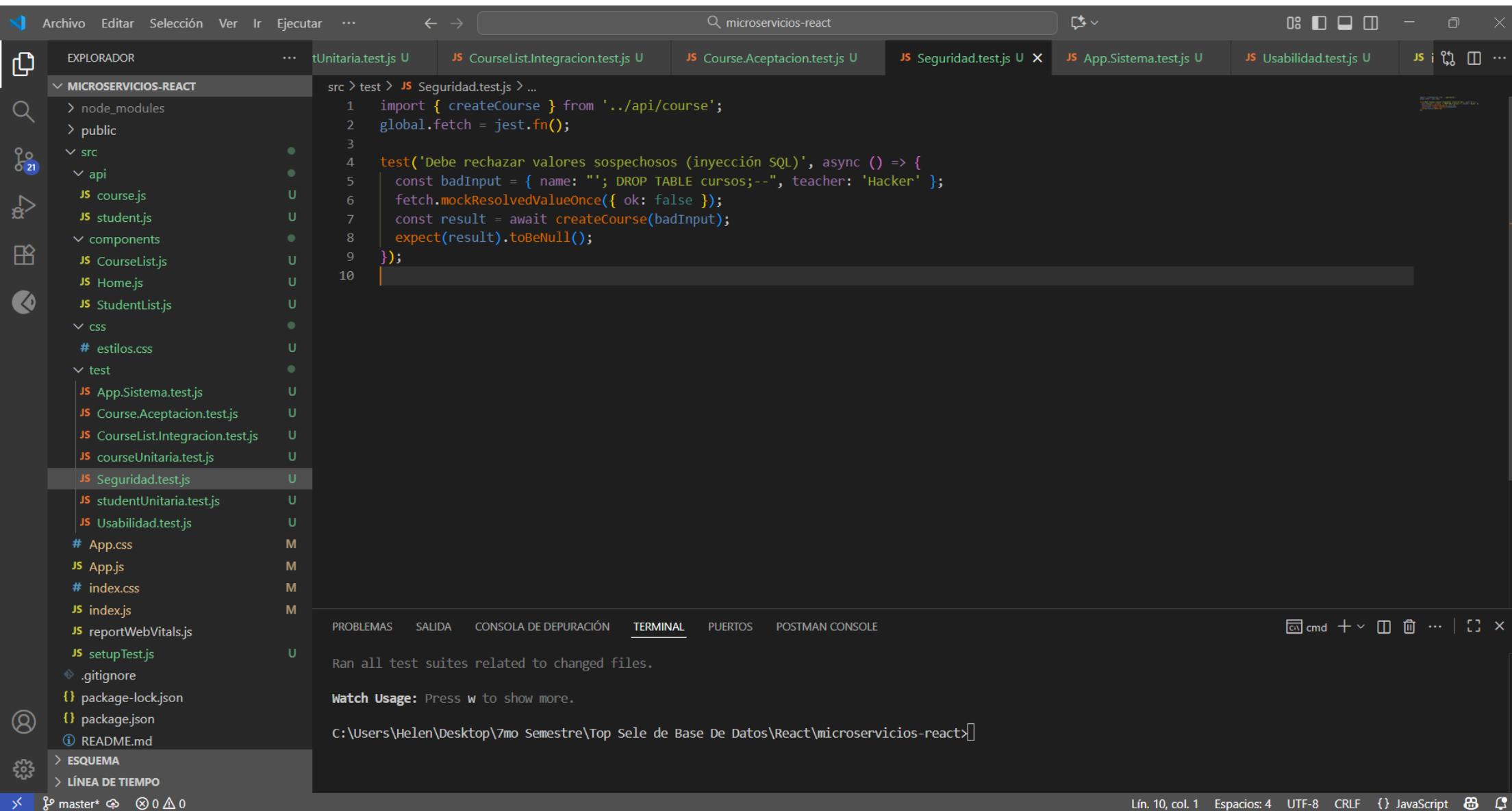
Watch Usage: Press w to show more.

C:\Users\Helen\Desktop\7mo Semestre\Top Sele de Base De Datos\React\microservicios-react\

Prueba de seguridad react

Esta prueba valida que la función createCourse detecte y rechace datos potencialmente peligrosos, como intentos de inyección SQL.

- Para ello, se simula el envío de un curso con contenido malicioso en el nombre, se *mockea* fetch para que responda como un error del servidor (ok: false), se llama a createCourse con el dato inseguro y se verifica que la función devuelva null, indicando que el sistema no acepta información riesgosa.



The screenshot shows a dark-themed code editor interface with multiple tabs open at the top, including "tUnitaria.test.js", "CourseList.Integracion.test.js", "Course.Aceptacion.test.js", "Seguridad.test.js" (which is the active tab), "App.Sistema.test.js", "Usabilidad.test.js", and others. The left sidebar shows a project structure under "EXPLORADOR" named "MICROSERVICIOS-REACT", with folders like node_modules, public, src, api, components, and test, along with files such as course.js, student.js, CourseList.js, Home.js, StudentList.js, estilos.css, and various test files. The main editor area displays the content of "Seguridad.test.js":

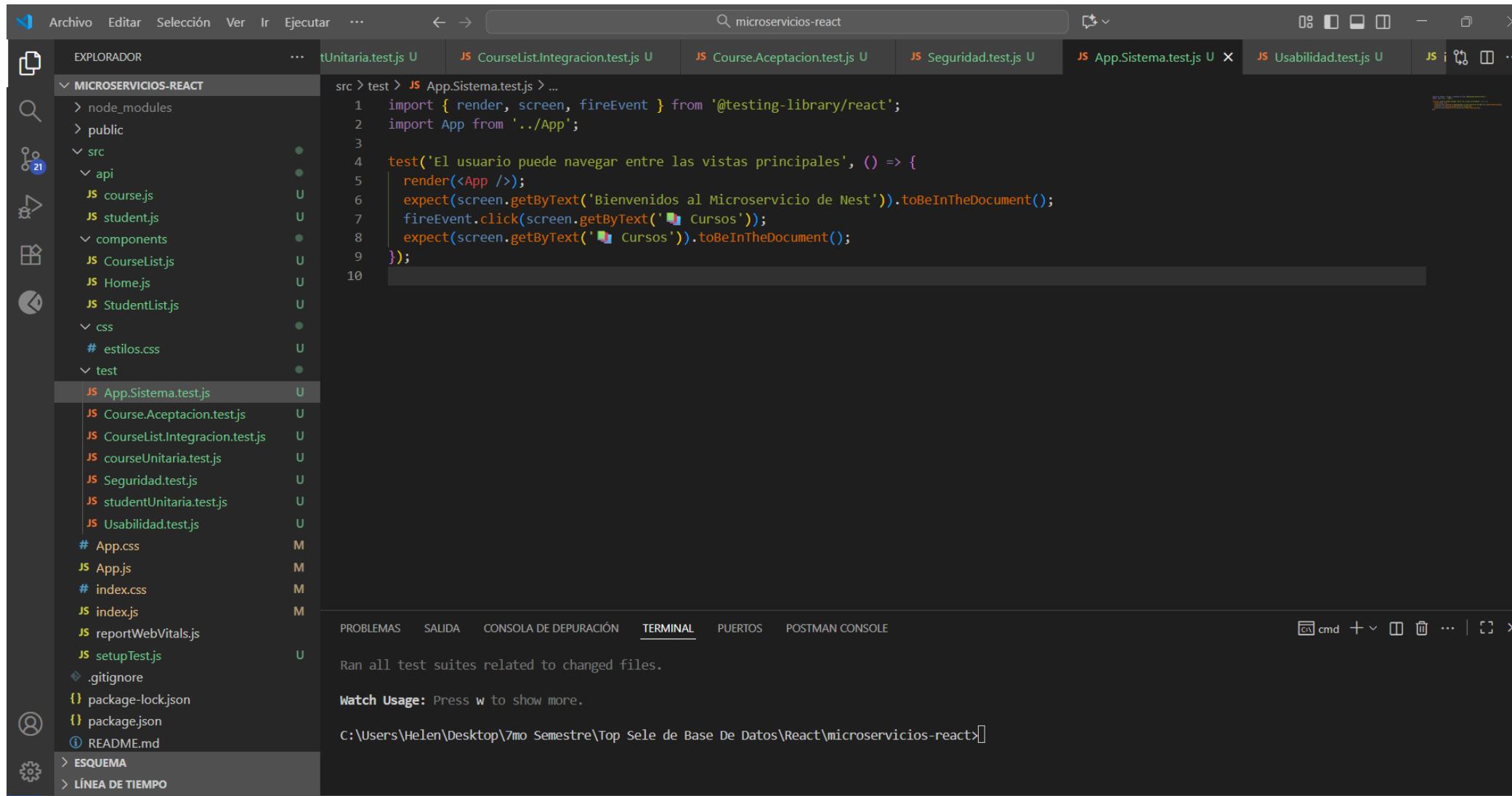
```
src > test > JS Seguridad.test.js > ...
1 import { createCourse } from '../api/course';
2 global.fetch = jest.fn();
3
4 test('Debe rechazar valores sospechosos (inyección SQL)', async () => {
5   const badInput = { name: ''; DROP TABLE cursos;--, teacher: 'Hacker' };
6   fetch.mockResolvedValueOnce({ ok: false });
7   const result = await createCourse(badInput);
8   expect(result).toBeNull();
9 });
10 
```

At the bottom, the terminal window shows the command "Ran all test suites related to changed files." and the path "C:\Users\Helen\Desktop\7mo Semestre\Top Sele de Base De Datos\React\microservicios-react>". The status bar at the bottom indicates "Lín. 10, col. 1 Espacios: 4 UTF-8 CRLF {} JavaScript".

Prueba sistema react

Esta prueba valida que el componente principal App muestre correctamente la información de la lista de cursos.

La prueba renderiza el componente App, verifica que el título "Lista de Cursos" aparezca en pantalla y comprueba que los cursos Base de Datos, Programación Web y Sistemas Distribuidos se muestren correctamente en la interfaz.



```
src > test > JS App.Sistema.test.js > ...
1 import { render, screen, fireEvent } from '@testing-library/react';
2 import App from '../App';
3
4 test('El usuario puede navegar entre las vistas principales', () => {
5   render(<App />);
6   expect(screen.getByText('Bienvenidos al Microservicio de Nest')).toBeInTheDocument();
7   fireEvent.click(screen.getByText('Base de Datos'));
8   expect(screen.getByText('Base de Datos')).toBeInTheDocument();
9 });
10
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS POSTMAN CONSOLE

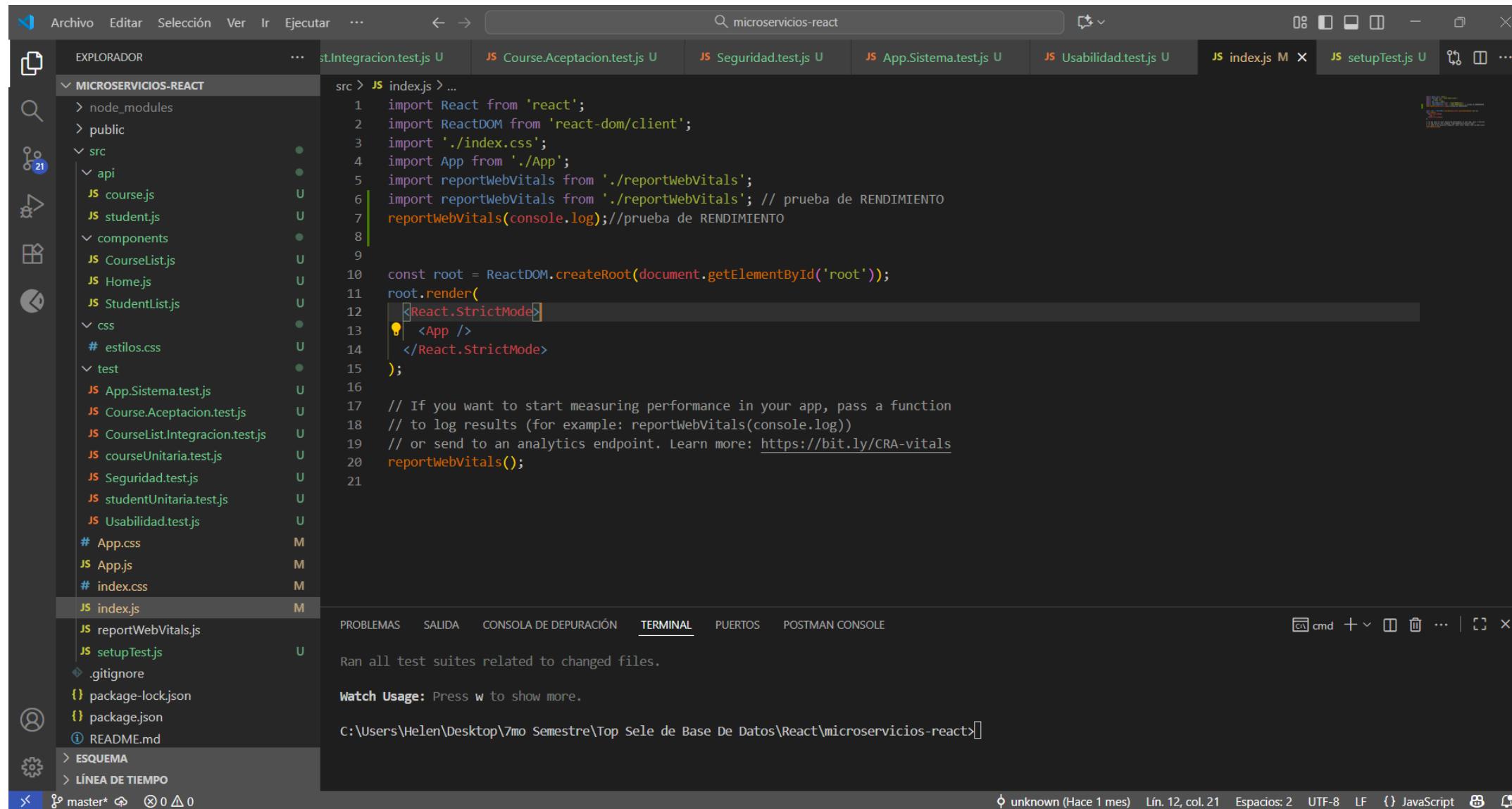
Ran all test suites related to changed files.

Watch Usage: Press w to show more.

C:\Users\Helen\Desktop\7mo Semestre\Top Sele de Base De Datos\React\microservicios-react>

Prueba rendimiento react

Este archivo es el punto de entrada principal de la aplicación React. Importa las dependencias, estilos y el componente App, obtiene el elemento HTML con id "root" y renderiza <App /> dentro de <React.StrictMode> para detectar errores y fomentar buenas prácticas. Opcionalmente, llama a reportWebVitals(console.log) para medir el rendimiento de la aplicación.



The screenshot shows a dark-themed code editor interface with several tabs at the top: Archivo, Editar, Selección, Ver, Ir, Ejecutar, and others. The main area displays a file named index.js. The code imports React and ReactDOM, and defines a root element with id 'root'. It uses StrictMode and the App component. There are comments about measuring performance using reportWebVitals. The left sidebar shows a project structure with folders like node_modules, public, src, api, components, css, test, and files like course.js, student.js, CourseList.js, Home.js, StudentList.js, estilos.css, App.Sistema.test.js, Course.Aceptacion.test.js, CourseList.Integracion.test.js, courseUnitaria.test.js, Seguridad.test.js, studentUnitaria.test.js, Usabilidad.test.js, App.css, App.js, and index.css. The bottom status bar shows the terminal path C:\Users\Helen\Desktop\7mo Semestre\Top Sele de Base De Datos\React\microservicios-react, and the bottom right corner shows the file is a JavaScript file.

```
src > JS index.js > ...
1 import React from 'react';
2 import ReactDOM from 'react-dom/client';
3 import './index.css';
4 import App from './App';
5 import reportWebVitals from './reportWebVitals';
6 import reportWebVitals from './reportWebVitals'; // prueba de RENDIMIENTO
7 reportWebVitals(console.log); //prueba de RENDIMIENTO
8
9
10 const root = ReactDOM.createRoot(document.getElementById('root'));
11 root.render(
12   <React.StrictMode>
13     <App />
14   </React.StrictMode>
15 );
16
17 // If you want to start measuring performance in your app, pass a function
18 // to log results (for example: reportWebVitals(console.log))
19 // or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
20 reportWebVitals();
21
```

Prueba usabilidad react

Esta prueba verifica que la función de búsqueda en CourseList funcione correctamente. Se simula la respuesta del servidor con los cursos Matemáticas, Historia y Física, se renderiza el componente y se ingresa "historia" en el campo de búsqueda. La prueba confirma que solo Historia permanece visible, asegurando que la búsqueda filtra correctamente la lista de cursos según la entrada del usuario.

```
src > test > JS Usabilidad.test.js ...
1 import { render, screen } from '@testing-library/react';
2 import userEvent from '@testing-library/user-event';
3 import CourseList from '../components/CourseList';
4
5 test('El campo de búsqueda filtra correctamente', async () => {
6   const mockCourses = [
7     { id: 1, name: 'Matemáticas', teacher: 'Luis' },
8     { id: 2, name: 'Historia', teacher: 'Ana' }
9   ];
10 jest.spyOn(require('../api/course'), 'getCourses').mockResolvedValue(mockCourses);
11
12 render(<CourseList setView={() => {}}>);
13 await screen.findByText('Matemáticas');
14
15 await userEvent.type(screen.getByPlaceholderText('Buscar curso o profesor...'), 'historia');
16 expect(screen.queryByText('Matemáticas')).not.toBeInTheDocument();
17 expect(screen.getText('Historia')).toBeInTheDocument();
18 });
19
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS POSTMAN CONSOLE

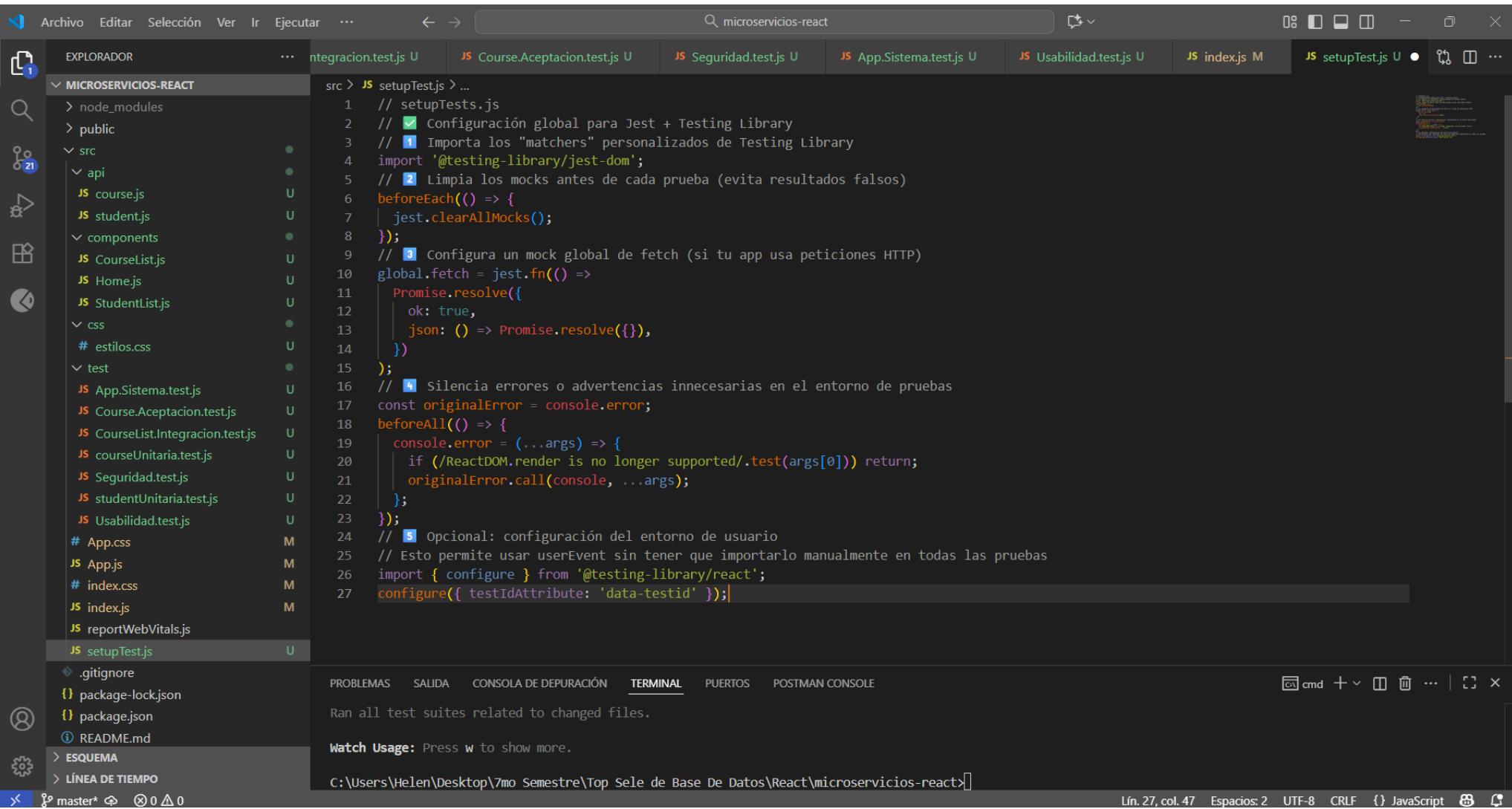
Ran all test suites related to changed files.

Watch Usage: Press w to show more.

C:\Users\Helen\Desktop\7mo Semestre\Top Sele de Base De Datos\React\microservicios-react\]

Configuración de pruebas en react

El archivo `setupTests.js` configura el entorno global de Jest, permitiendo ejecutar pruebas de manera consistente y sin repetir configuraciones. Activa los *matchers* de Testing Library, limpia los *mocks* antes de cada prueba, simula fetch si no existe, oculta errores innecesarios y configura `data-testid` para facilitar la identificación de elementos en los tests.



```
// Configuración global para Jest + Testing Library
// Importa los "matchers" personalizados de Testing Library
import '@testing-library/jest-dom';
// Limpia los mocks antes de cada prueba (evita resultados falsos)
beforeEach(() => {
  jest.clearAllMocks();
});
// Configura un mock global de fetch (si tu app usa peticiones HTTP)
global.fetch = jest.fn(() =>
  Promise.resolve({
    ok: true,
    json: () => Promise.resolve({})
  })
);
// Silencia errores o advertencias innecesarias en el entorno de pruebas
const originalError = console.error;
beforeAll(() => {
  console.error = (...args) => {
    if (/ReactDOM.render is no longer supported/.test(args[0])) return;
    originalError(...args);
  };
});
// Opcional: configuración del entorno de usuario
// Esto permite usar userEvent sin tener que importarlo manualmente en todas las pruebas
import { configure } from '@testing-library/react';
configure({ testIdAttribute: 'data-testid' });
```

Prueba unitaria student nest

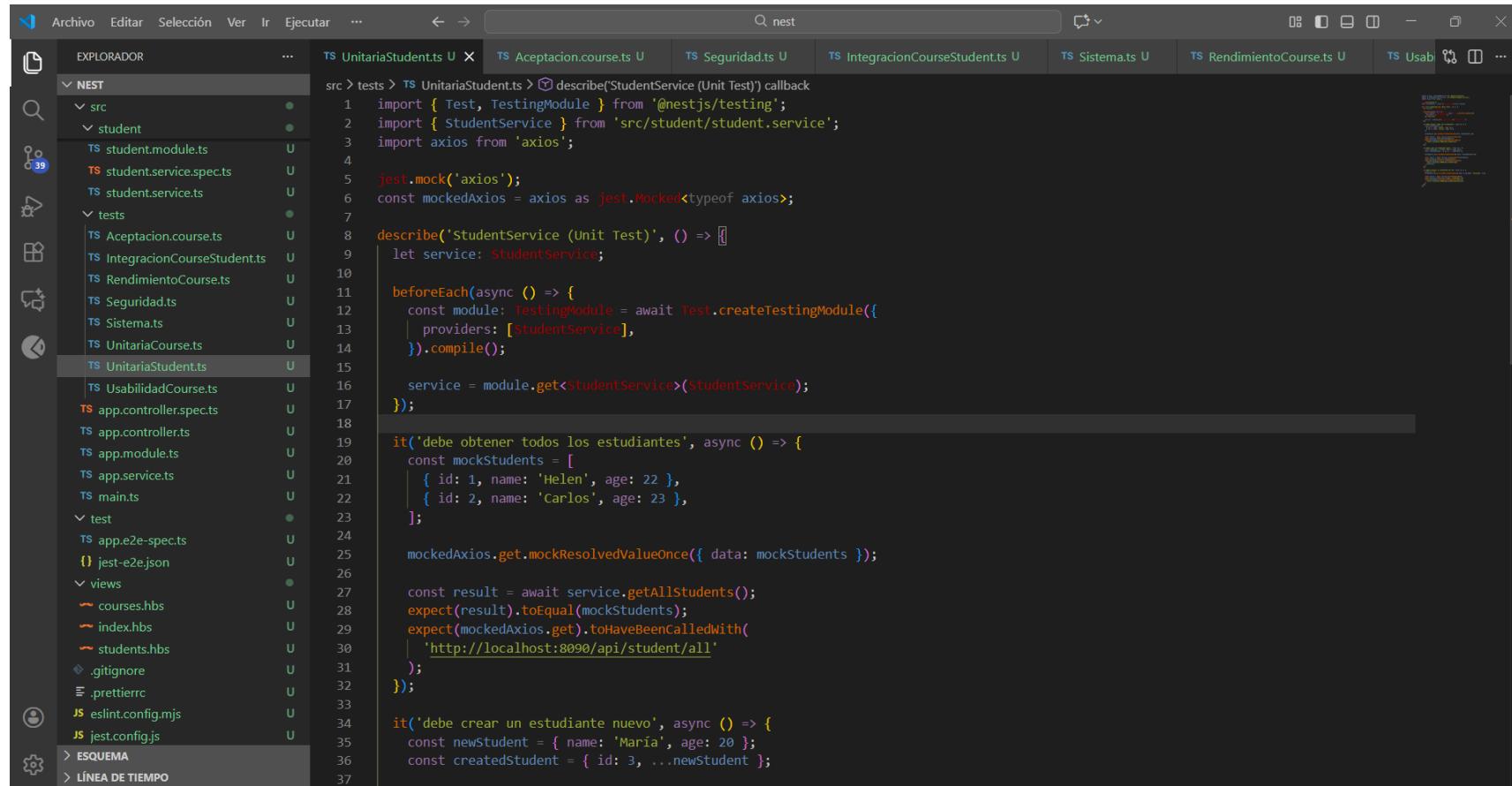


Las pruebas unitarias verifican que los métodos de `StudentService` funcionen correctamente simulando Axios, sin realizar llamadas reales al backend. Se utiliza un *mock* de Axios y un módulo de prueba que carga únicamente el servicio.

- Se comprueba que:**

 - Obtener estudiantes: devuelve la lista simulada y usa la URL correcta.
 - Crear estudiante: envía los datos adecuados y recibe la respuesta simulada.
 - Eliminar estudiante: llama a la ruta con el ID y obtiene el mensaje esperado.

En conjunto, el archivo asegura que `StudentService` realice correctamente sus operaciones principales (listar, crear y eliminar estudiantes) usando *mocks* de `Axios`.

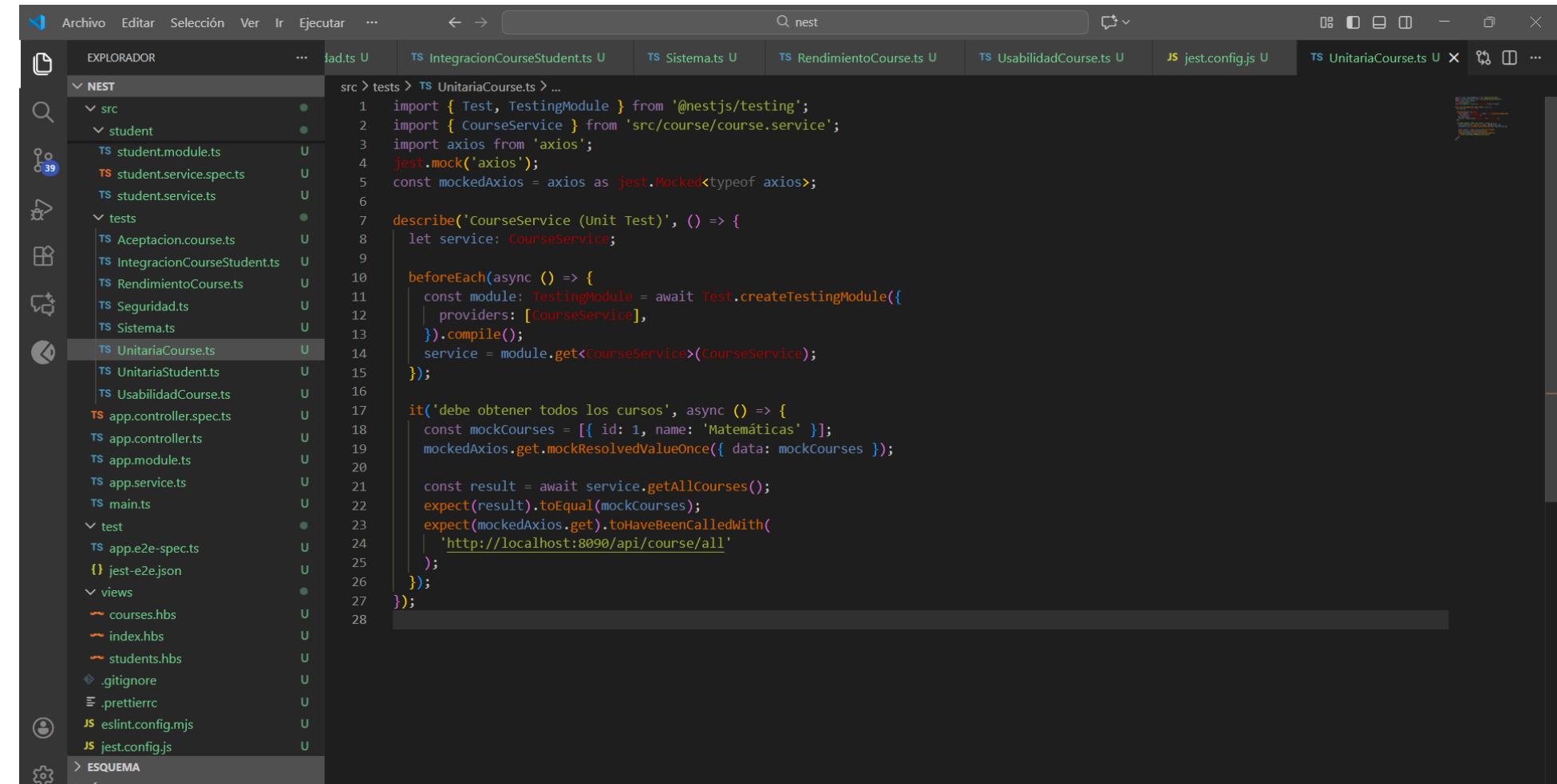


prueba unitaria student nest

Este archivo realiza pruebas unitarias para garantizar que CourseService funcione correctamente sin conectarse a un backend real, utilizando *mocks* de Axios. Se crea un módulo de prueba de NestJS que solo incluye el servicio evaluado.

- Las pruebas principales son:

- Obtener todos los cursos: se simula axios.get, se verifica que getAllCourses() devuelva los datos simulados y que se haya llamado a la URL /all.
- Obtener un curso por ID: se simula axios.get con un curso específico, se comprueba que getCourseById(1) retorne ese curso y que se haya llamado a /search/1.



```
src > tests > TS UnitariaCourse.ts ...
1 import { Test, TestingModule } from '@nestjs/testing';
2 import { CourseService } from 'src/course/course.service';
3 import axios from 'axios';
4 jest.mock('axios');
5 const mockedAxios = axios as jest.Mocked;
6
7 describe('CourseService (Unit Test)', () => {
8     let service: CourseService;
9
10    beforeEach(async () => {
11        const module: TestingModule = await test.createTestingModule({
12            providers: [courseService],
13        }).compile();
14        service = module.get<CourseService>(CourseService);
15    });
16
17    it('debe obtener todos los cursos', async () => {
18        const mockCourses = [{ id: 1, name: 'Matemáticas' }];
19        mockedAxios.get.mockResolvedValueOnce({ data: mockCourses });
20
21        const result = await service.getAllCourses();
22        expect(result).toEqual(mockCourses);
23        expect(mockedAxios.get).toHaveBeenCalledWith(
24            'http://localhost:8090/api/course/all'
25        );
26    });
27});
```

Prueba integración nest

Este archivo realiza una prueba de integración para verificar que la API de NestJS obtenga correctamente los estudiantes asociados a un curso. Se utiliza Supertest para llamar a los endpoints reales y Axios se simula con Jest para evitar llamadas a otros microservicios, devolviendo dos estudiantes falsos al consultar el curso 1.

La prueba se ejecuta en un entorno de pruebas basado en AppModule, permitiendo evaluar la ruta /api/course/1/students sin depender de servicios externos. Se verifica que la respuesta tenga código 200, devuelva un arreglo con estudiantes y que cada uno incluya información como nombre y courseId.

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer (Left):** Shows the project structure under the "EXPLORADOR" tab. The "src" folder contains "student" and "tests" subfolders. "student" contains "student.module.ts", "student.service.spec.ts", and "student.service.ts". "tests" contains "Aceptacion.course.ts" and "IntegracionCourseStudent.ts". Other files include "RendimientoCourse.ts", "Seguridad.ts", "Sistema.ts", "UnitariaCourse.ts", "UnitariaStudent.ts", "UsabilidadCourse.ts", "app.controller.spec.ts", "app.controller.ts", "app.module.ts", "app.service.ts", "main.ts", "test/app.e2e-spec.ts", "jest-e2e.json", "views/courses.hbs", "views/index.hbs", "views/students.hbs", ".gitignore", ".prettierrc", "eslint.config.mjs", and "jest.config.js".
- Search Bar (Top):** Contains the text "nest".
- Code Editor (Center):** Displays the content of "IntegracionCourseStudent.ts". The code is a Jest test for the "student" module, specifically for the "Integración - Curso y Estudiantes" scenario. It uses the "supertest" library to interact with the application's API.
- Activity Bar (Bottom):** Shows icons for "ESQUEMA" and "LÍNEA DE TIEMPO".

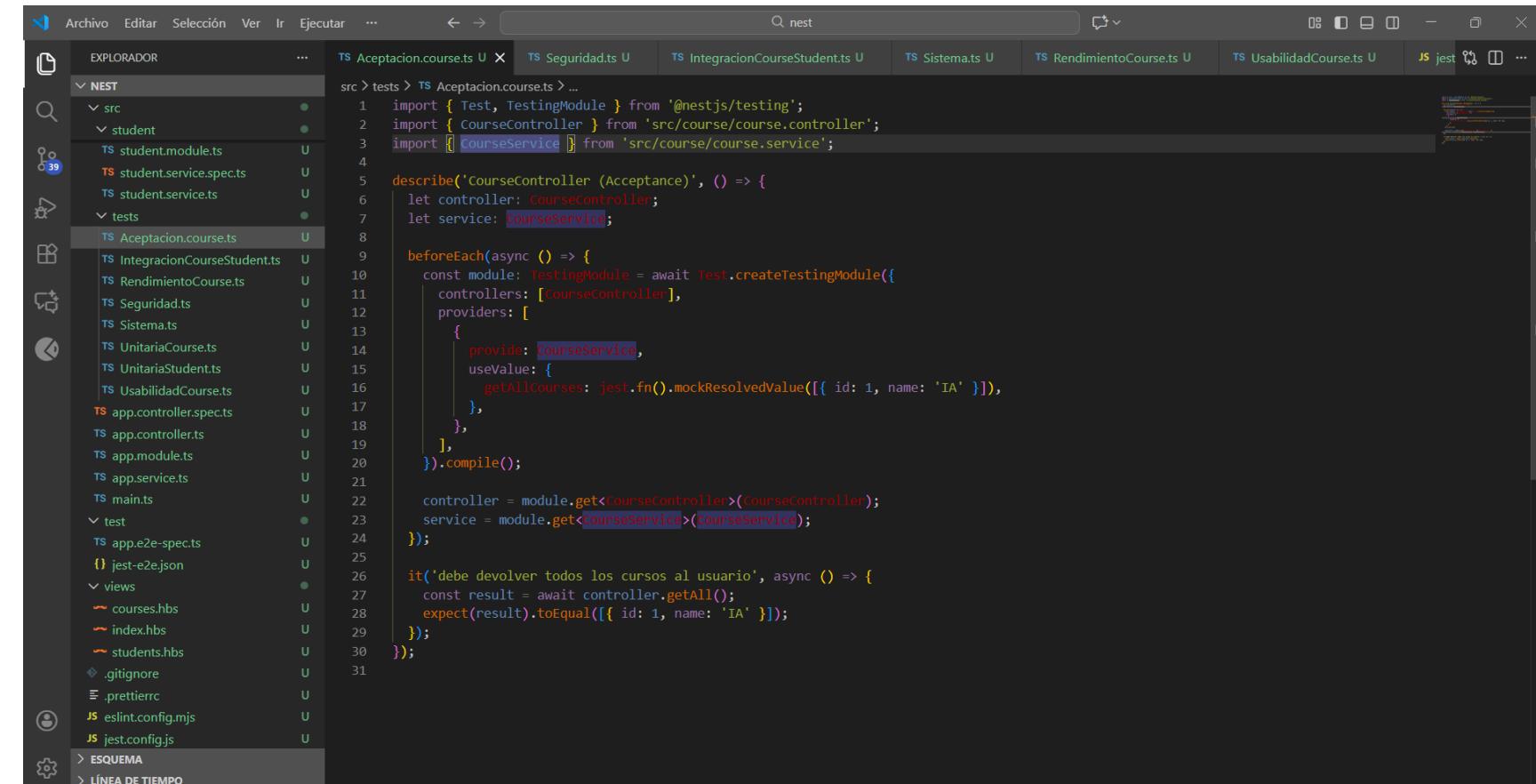
Prueba aceptación nest

Este archivo realiza una prueba de aceptación para CourseController en NestJS, con el objetivo de verificar que el controlador devuelva correctamente la información al usuario final. Se utiliza un CourseService completamente simulado con Jest.

Se crea un módulo de prueba que incluye el controlador real, pero reemplaza el servicio por un *mock* que devuelve cursos simulados, enfocando la prueba únicamente en el comportamiento del controlador.

Las pruebas principales son:

- Verificar que el método que obtiene todos los cursos devuelva la lista simulada y que el servicio haya sido llamado.
- Confirmar que el método que obtiene un curso por ID retorne el curso correcto y que el controlador envíe el ID adecuado al servicio.



```
src > tests > TS Aceptacion.course.ts ... TS Seguridad.ts U TS IntegracionCourseStudent.ts U TS Sistema.ts U TS RendimientoCourse.ts U TS UsabilidadCourse.ts U JS jest U ...
1 import { Test, TestingModule } from '@nestjs/testing';
2 import { CourseController } from 'src/course/course.controller';
3 import { CourseService } from 'src/course/course.service';
4
5 describe('CourseController (Acceptance)', () => {
6   let controller: CourseController;
7   let service: CourseService;
8
9   beforeEach(async () => {
10     const module: TestingModule = await Test.createTestingModule({
11       controllers: [courseController],
12       providers: [
13         {
14           provide: CourseService,
15           useValue: {
16             getAllCourses: jest.fn().mockResolvedValue([
17               { id: 1, name: 'IA' }
18             ]),
19           },
20         ],
21       ],
22     }).compile();
23
24     controller = module.get<CourseController>(courseController);
25     service = module.get<CourseService>(CourseService);
26   });
27
28   it('debe devolver todos los cursos al usuario', async () => {
29     const result = await controller.getAll();
30     expect(result).toEqual([{ id: 1, name: 'IA' }]);
31   });
32 })
```

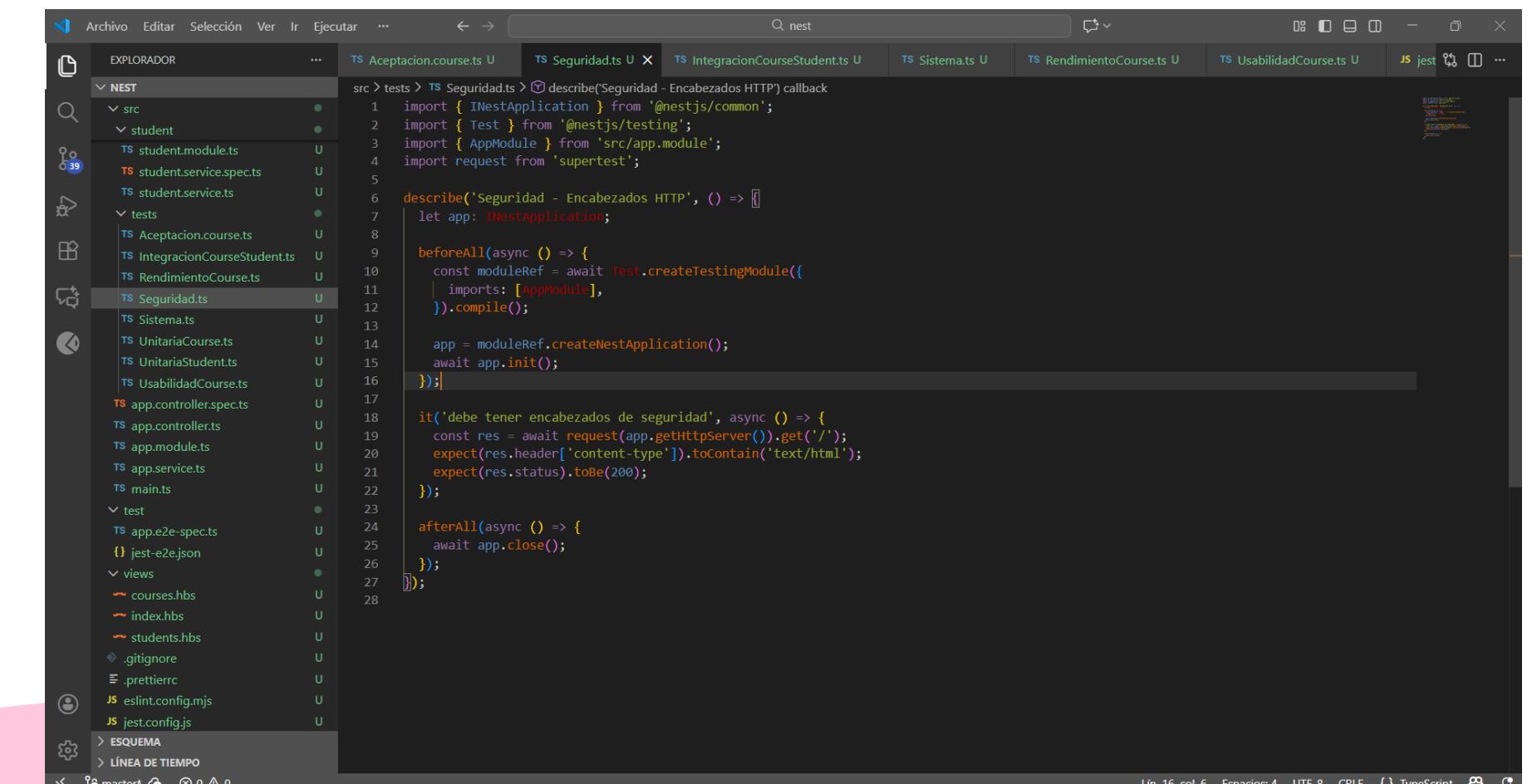
Prueba seguridad nest

Este archivo realiza una prueba end-to-end para verificar que la aplicación NestJS responda con los encabezados HTTP adecuados, como parte de las validaciones básicas de seguridad. Se levanta una instancia real de la aplicación mediante AppModule y se ejecuta una petición HTTP con Supertest.

La prueba realiza un GET al endpoint raíz / y valida que:

- La respuesta tenga código 200, indicando que la aplicación funciona correctamente.
- El encabezado content-type contenga text/html, asegurando que la respuesta se envía correctamente.

En conjunto, la prueba garantiza que la aplicación se inicia correctamente y que los encabezados HTTP esenciales están presentes, cumpliendo buenas prácticas de seguridad.



The screenshot shows a code editor with a dark theme. On the left is a sidebar labeled 'EXPLORADOR' containing a tree view of files and folders related to a NestJS project. The main area displays a Jest test file named 'Seguridad.ts'. The code in the file is as follows:

```
src > tests > TS Seguridad.ts U
1 import { INestApplication } from '@nestjs/common';
2 import { Test } from '@nestjs/testing';
3 import { AppModule } from 'src/app.module';
4 import request from 'supertest';
5
6 describe('Seguridad - Encabezados HTTP', () => [
7   let app: INestApplication;
8
9   beforeAll(async () => {
10     const moduleRef = await Test.createTestingModule({
11       imports: [AppModule],
12     }).compile();
13
14     app = moduleRef.createNestApplication();
15     await app.init();
16   });
17
18   it('debe tener encabezados de seguridad', async () => {
19     const res = await request(app.getHttpServer()).get('/');
20     expect(res.header['content-type']).toContain('text/html');
21     expect(res.status).toBe(200);
22   });
23
24   afterAll(async () => {
25     await app.close();
26   });
27 ]);
28
```

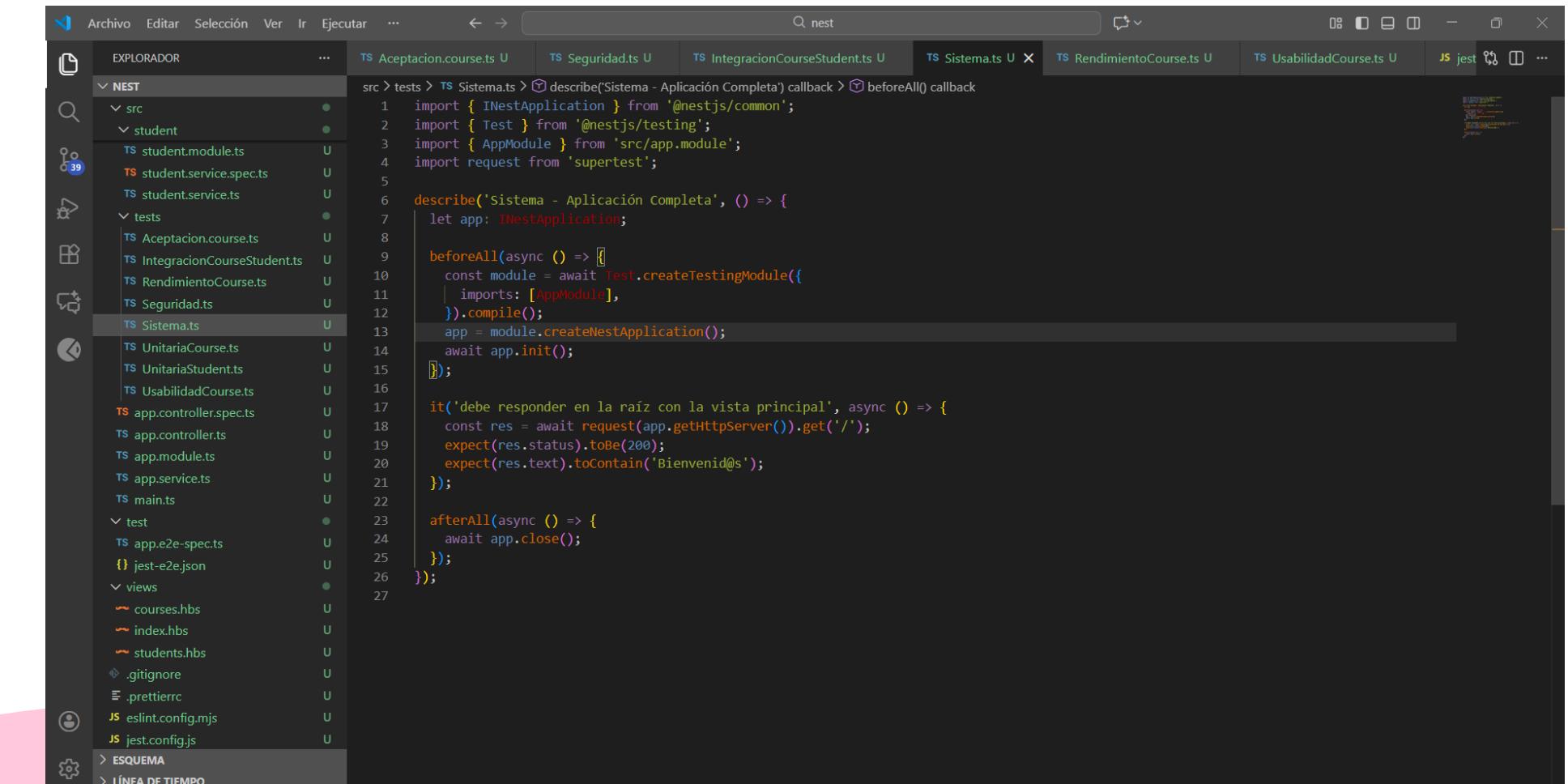
Prueba de sistema nest

Este archivo realiza una prueba de sistema para verificar el funcionamiento global de la aplicación NestJS utilizando su módulo principal (AppModule). La prueba evalúa la aplicación completa, no solo partes aisladas del código.

Se levanta toda la aplicación y se envía una petición HTTP a la ruta raíz con Supertest, verificando que:

- La respuesta tenga código 200 (OK).
- El contenido incluya la palabra “Bienvenid@s”, confirmando que la vista principal se carga correctamente.

En conjunto, la prueba asegura que la aplicación funciona correctamente y que el usuario recibe la página esperada al acceder al sitio.



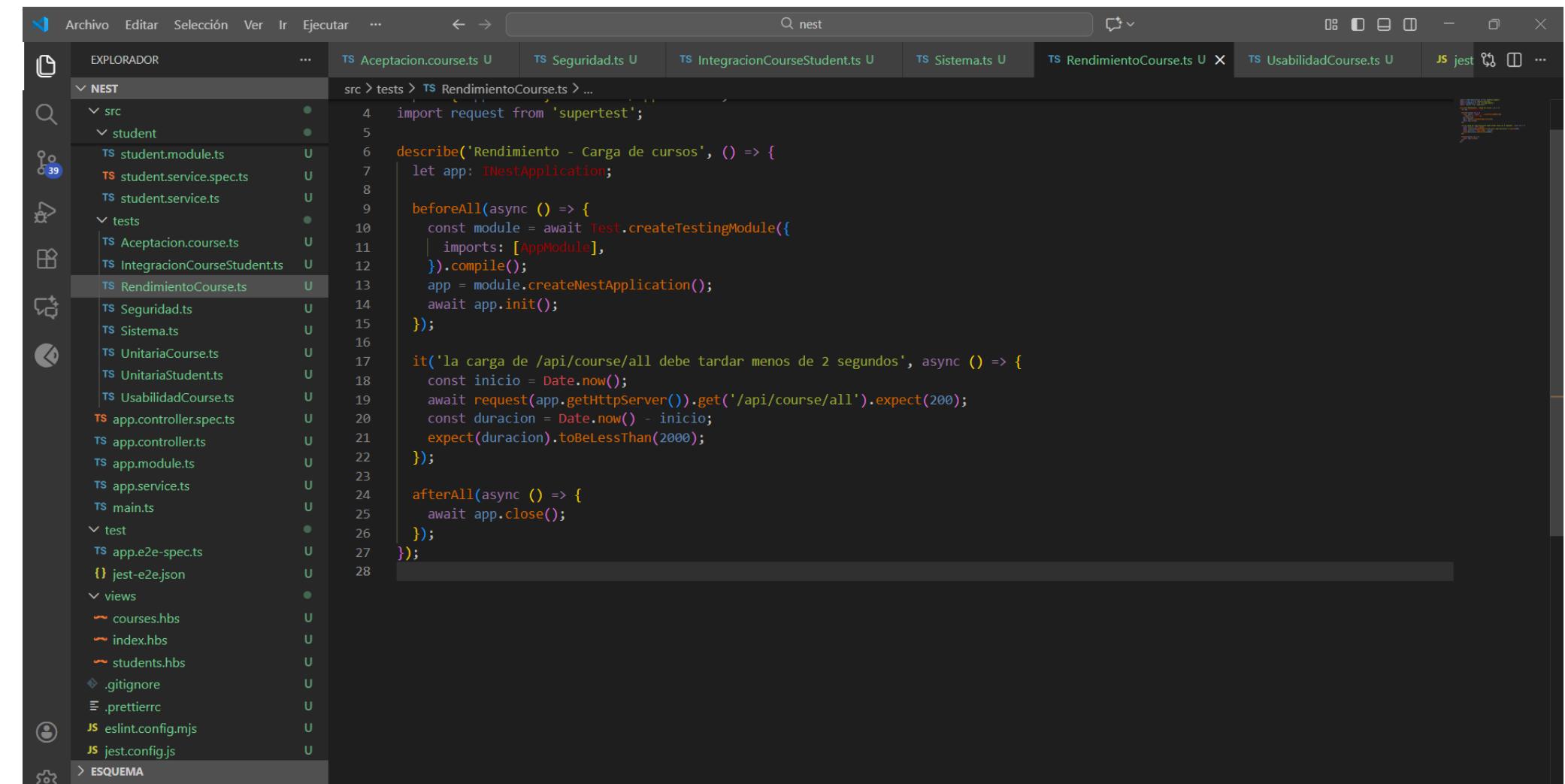
```
src > tests > TS Sistema.ts > describe('Sistema - Aplicación Completa') callback > beforeAll() callback
1 import { INestApplication } from '@nestjs/common';
2 import { Test } from '@nestjs/testing';
3 import { AppModule } from 'src/app.module';
4 import request from 'supertest';

6 describe('Sistema - Aplicación Completa', () => {
7   let app: INestApplication;
8
9   beforeAll(async () => {
10     const module = await Test.createTestingModule({
11       imports: [AppModule],
12     }).compile();
13     app = module.createNestApplication();
14     await app.init();
15   });
16
17   it('debe responder en la raíz con la vista principal', async () => {
18     const res = await request(app.getHttpServer()).get('/');
19     expect(res.status).toBe(200);
20     expect(res.text).toContain('Bienvenid@s');
21   });
22
23   afterAll(async () => {
24     await app.close();
25   });
26 });
27
```

Prueba rendimiento nest

Esta prueba evalúa el rendimiento del sistema midiendo el tiempo de respuesta de la ruta /api/course/all. Se levanta la aplicación completa mediante AppModule y se realiza una petición HTTP con Supertest.

La prueba registra el tiempo antes y después de la solicitud, verificando que la duración total sea inferior a 2 segundos, lo que garantiza que la carga de todos los cursos es rápida y cumple con los estándares de rendimiento del sistema.

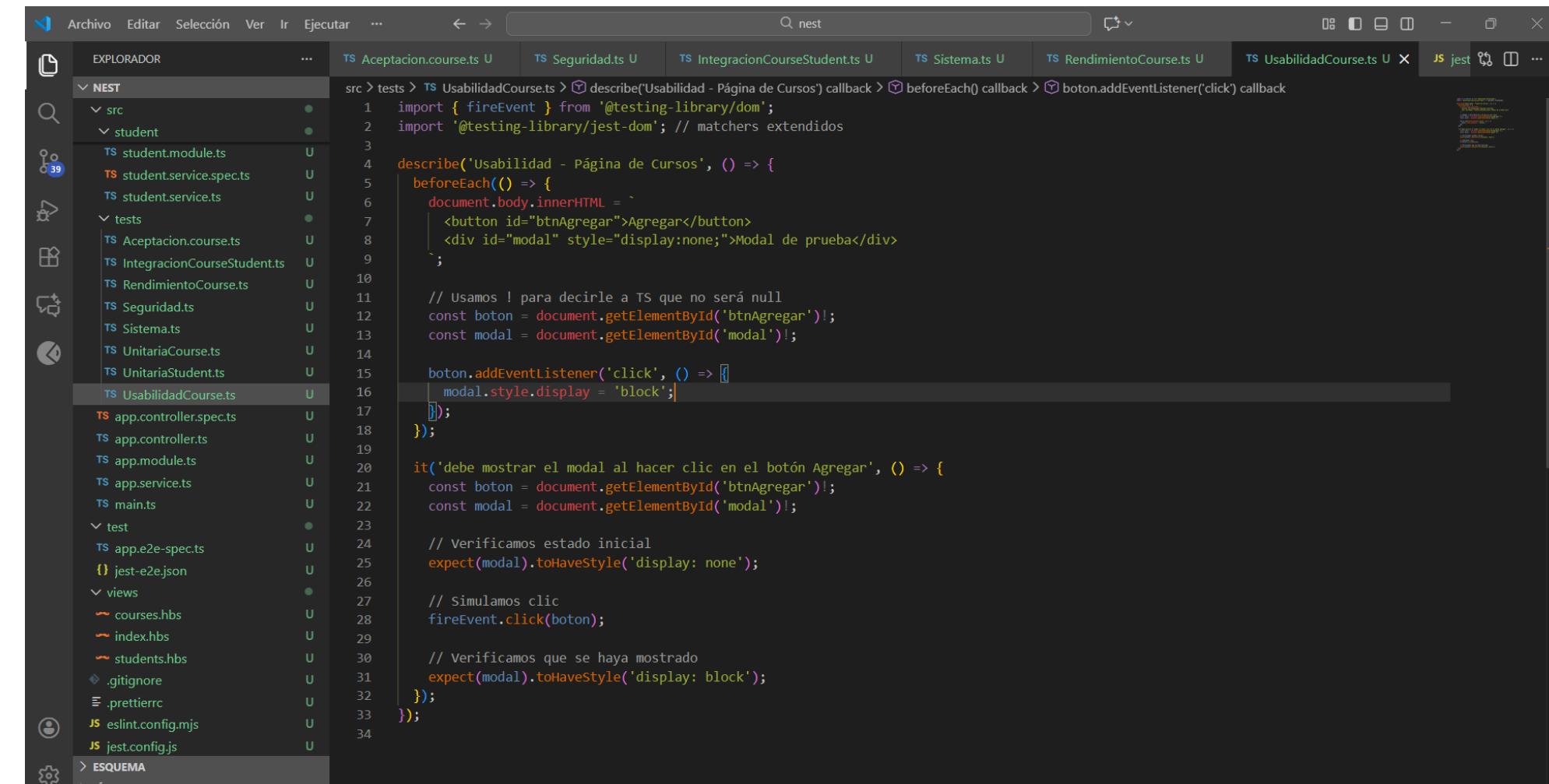


```
src > tests > TS RendimientoCourse.ts ...
4 import request from 'supertest';
5
6 describe('Rendimiento - Carga de cursos', () => {
7   let app: INestApplication;
8
9   beforeAll(async () => {
10     const module = await Test.createTestingModule({
11       imports: [AppModule],
12     }).compile();
13     app = module.createNestApplication();
14     await app.init();
15   });
16
17   it('la carga de /api/course/all debe tardar menos de 2 segundos', async () => {
18     const inicio = Date.now();
19     await request(app.getHttpServer()).get('/api/course/all').expect(200);
20     const duracion = Date.now() - inicio;
21     expect(duracion).toBeLessThan(2000);
22   });
23
24   afterAll(async () => {
25     await app.close();
26   });
27 });

> ESQUEMA
```

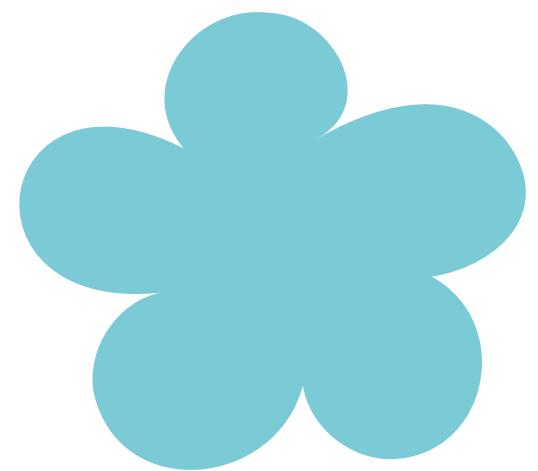
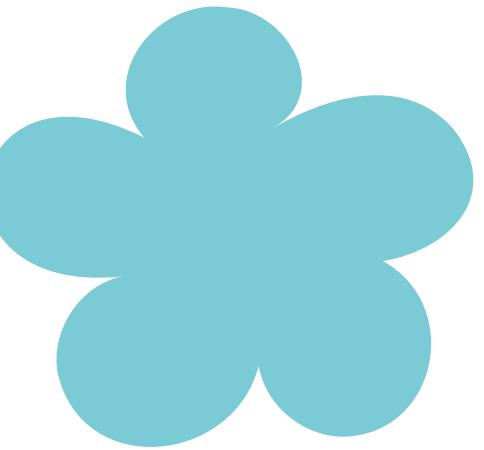
Prueba usabilidad nest

Esta prueba verifica la usabilidad de la página de cursos simulando la interacción del usuario en un entorno DOM con Jest. Antes de cada prueba se construye un HTML simple con un botón y un modal oculto. Se añade un evento al botón que muestra el modal al hacer clic. La prueba ejecuta un clic sobre el botón “Agregar” y confirma que el modal cambia su estilo a display: block, comprobando que la interfaz responde correctamente a la interacción del usuario.



```
src > tests > TS UsabilidadCourse.ts U | TS Aceptacion.course.ts U | TS Seguridad.ts U | TS IntegracionCourseStudent.ts U | TS Sistema.ts U | TS RendimientoCourse.ts U | TS UsabilidadCourse.ts U X | JS jest
1 import { fireEvent } from '@testing-library/dom';
2 import '@testing-library/jest-dom'; // matchers extendidos
3
4 describe('Usabilidad - Página de Cursos', () => {
5   beforeEach(() => {
6     document.body.innerHTML =
7       '<button id="btnAgregar">Agregar</button>
8       <div id="modal" style="display:none;">Modal de prueba</div>
9   ';
10
11   // Usamos ! para decirle a TS que no será null
12   const boton = document.getElementById('btnAgregar')!;
13   const modal = document.getElementById('modal')!;
14
15   boton.addEventListener('click', () => [
16     modal.style.display = 'block'];
17   });
18
19 it('debe mostrar el modal al hacer clic en el botón Agregar', () => {
20   const boton = document.getElementById('btnAgregar')!;
21   const modal = document.getElementById('modal')!;
22
23   // Verificamos estado inicial
24   expect(modal).toHaveStyle('display: none');
25
26   // Simulamos clic
27   fireEvent.click(boton);
28
29   // Verificamos que se haya mostrado
30   expect(modal).toHaveStyle('display: block');
31 });
32 });
33 });
34 }
```

Link



Conclusión

Las pruebas realizadas aseguran que la aplicación funcione de manera confiable, segura y eficiente. Las pruebas unitarias verifican que los servicios y componentes realicen correctamente sus operaciones. Las de integración confirman la correcta interacción entre módulos y servicios, mientras que las de aceptación garantizan que los controladores y la interfaz respondan correctamente a las acciones del usuario.

Las pruebas de seguridad protegen contra accesos no autorizados y datos maliciosos, y las de sistema y end-to-end verifican que la aplicación completa se ejecute correctamente y que las rutas principales devuelvan la información esperada.

Finalmente, las pruebas de rendimiento y usabilidad aseguran tiempos de respuesta óptimos y una interfaz clara.

En conjunto, estas pruebas garantizan un sistema confiable, seguro y eficiente, cumpliendo con los requisitos funcionales y de calidad.