VALIDACIÓN Y VERIFICACIÓN DE SOFTWARE (INF 732)
GUÍA DE LABORATORIO 4
Pruebas unitarias para el Controlador de Notas

En esta guía, nos enfocaremos en probar el controlador de notas de nuestra API construida con NestJS y MySQL, utilizando Jest como framework de testing. Al finalizar, tendrás un conjunto de pruebas robustas que facilitarán el mantenimiento y la escalabilidad del proyecto.

Objetivos

Implementar pruebas unitarias para un controlador en NestJS utilizando **Jest**, cubriendo los siguientes aspectos:

- Configuración del entorno de pruebas.
- Mocking de dependencias (servicios).
- Pruebas de endpoints (GET, POST, PUT, DELETE).
- Manejo de errores (NotFoundException).

Prerrequisitos

Haber desarrollado las Guías 1, 2 y 3 de la asignatura.

1. Configuración inicial

notas.controller.spec.ts

```
import { Test, TestingModule } from '@nestjs/testing';
import { NotasController } from './notas.controller';
import { NotasService } from './notas.service';
import { NotFoundException } from '@nestjs/common';
import { Nota } from './nota.entity';
import { CreateNotaDto } from './dto/create-nota.dto';
import { UpdateNotaDto } from './dto/update-nota.dto';

describe('NotasController', () => {
  let controller: NotasController;
  let service: NotasService;

  beforeEach(async () => {
    const mockService = {
      create: jest.fn(),
```

```
      findOne: jest.fn(),
      findAll: jest.fn(),
      update: jest.fn(),
      remove: jest.fn(),
    };

    const module: TestingModule = await Test.createTestingModule({
      controllers: [NotasController],
      providers: [
        {
          provide: NotasService,
          useValue: mockService,
        },
      ],
    }).compile();

    controller = module.get<NotasController>(NotasController);
    service = module.get<NotasService>(NotasService);

    // Limpiar los mocks antes de cada prueba
    jest.clearAllMocks();
  });

  it('should be defined', () => {
    expect(controller).toBeDefined();
  });

...
});
```

2. Casos de prueba

Crear una nota

notas.controller.spec.ts

```
  describe('create', () => {
    it('debería crear una nota', async () => {
      const mockNota: Nota = {
        id: 1,
        title: 'Nota 1',
        content: 'Contenido',
      };
      const createNotaDto: CreateNotaDto = {
        title: 'Nota 1',
        content: 'Contenido',
      };
```

```
    // Simulamos que el servicio devuelve la nota creada
    jest.spyOn(service, 'create').mockResolvedValue(mockNota);

    const result = await controller.create(createNotaDto);

    expect(result).toEqual(mockNota);
    expect(service.create).toHaveBeenCalledWith(createNotaDto);
  });
});
```

Mostrar todas las notas

notas.controller.spec.ts

```
describe('findAll', () => {
  it('debería retornar un array de notas', async () => {
    const mockNotas: Nota[] = [
      { id: 1, title: 'Nota 1', content: 'Contenido' },
      { id: 2, title: 'Nota 2', content: 'Contenido 2' },
    ];
    jest.spyOn(service, 'findAll').mockResolvedValue(mockNotas);

    const result = await controller.findAll();

    expect(result).toEqual(mockNotas);
    expect(service.findAll).toHaveBeenCalled();
  });
});
```

Buscar una nota

notas.controller.spec.ts

```
describe('findOne', () => {
  it('debería retornar una nota si existe', async () => {
    const mockNota: Nota = {
      id: 1,
      title: 'Nota 1',
      content: 'Contenido',
    };
    jest.spyOn(service, 'findOne').mockResolvedValue(mockNota);

    const result = await controller.findOne('1');

    expect(result).toEqual(mockNota);
    expect(service.findOne).toHaveBeenCalledWith(1);
  });
```

```
    it('debería lanzar NotFoundException si la nota no existe', async () =>
{
      jest.spyOn(service, 'findOne').mockRejectedValue(new
NotFoundException());

      await expect(controller.findOne('999')).rejects.toThrow(
        NotFoundException,
      );
      expect(service.findOne).toHaveBeenCalledWith(999);
    });
  });
```

Modificar una nota

notas.controller.spec.ts

```
  describe('update', () => {
    it('debería actualizar una nota existente', async () => {
      const mockNota: Nota = {
        id: 1,
        title: 'Nota Actualizada',
        content: 'Nuevo contenido',
      };
      const updateNotaDto: UpdateNotaDto = { title: 'Nota Actualizada' };

      jest.spyOn(service, 'update').mockResolvedValue(mockNota);

      const result = await controller.update('1', updateNotaDto);

      expect(result).toEqual(mockNota);
      expect(service.update).toHaveBeenCalledWith(1, updateNotaDto);
    });

    it('debería lanzar NotFoundException si la nota no existe', async () =>
{
      jest.spyOn(service, 'update').mockRejectedValue(new
NotFoundException());

      await expect(
        controller.update('999', { title: 'No existe' }),
      ).rejects.toThrow(NotFoundException);
      expect(service.update).toHaveBeenCalledWith(999, {
        title: 'No existe',
      });
    });
  });
```

Eliminar una nota

notas.controller.spec.ts

```
  describe('remove', () => {
    it('debería eliminar una nota existente', async () => {
      jest.spyOn(service, 'remove').mockResolvedValue(undefined);

      await controller.remove('1');

      expect(service.remove).toHaveBeenCalledWith(1);
    });

    it('debería lanzar NotFoundException si la nota no existe', async () =>
{
      jest.spyOn(service, 'remove').mockRejectedValue(new
NotFoundException());

      await
expect(controller.remove('999')).rejects.toThrow(NotFoundException);
      expect(service.remove).toHaveBeenCalledWith(999);
    });
  });
```

Finalmente, para ejecutar los test, utilice el siguiente comando:

```
npm run test
```