VALIDACIÓN Y VERIFICACIÓN DE SOFTWARE (INF 732)
GUÍA DE LABORATORIO 3
Pruebas unitarias para el Servicio de Notas

En el desarrollo de software, las pruebas unitarias son esenciales para garantizar la fiabilidad y el correcto funcionamiento de cada componente de manera aislada. En esta guía, nos enfocaremos en probar el servicio de notas de nuestra API construida con NestJS y MySQL, utilizando Jest como framework de testing. Aprenderemos a simular dependencias (como TypeORM), diseñar casos de prueba para los métodos CRUD y validar el comportamiento esperado del servicio, asegurando que nuestra lógica de negocio responda correctamente incluso en escenarios de error. Al finalizar, tendrás un conjunto de pruebas robustas que facilitarán el mantenimiento y la escalabilidad del proyecto.

Objetivos

- Aprender a escribir **pruebas unitarias** para un servicio (NotasService) en NestJS.
- Utilizar **Jest** (framework de testing incluido en NestJS) para verificar la lógica de negocio.
- Aislar dependencias con **mocks** (simulaciones) de TypeORM y otros módulos.
- Garantizar que los métodos del servicio (create(), findAll(), update(), delete()) funcionen correctamente.

Prerrequisitos

Haber desarrollado las Guías 1 y 2 de la asignatura.

1. Configuración inicial

Para aislar el servicio de la base de datos, creamos un mock del repositorio y configuramos el módulo testing.

notas.service.spec.ts

```
import { Test, TestingModule } from '@nestjs/testing';
import { NotasService } from './notas.service';
import { ObjectLiteral, Repository, UpdateResult } from 'typeorm';
import { Nota } from './nota.entity';
import { getRepositoryToken } from '@nestjs/typeorm';
import { NotFoundException } from '@nestjs/common';

const mockNotaRepository = () => ({
  create: jest.fn(),
  save: jest.fn(),
  find: jest.fn(),
  findOneBy: jest.fn(),
  update: jest.fn(),
  delete: jest.fn(),
});

const mockNota = {
  id: 1,
  title: 'Test Nota',
  content: 'Test Content',
};

type MockRepository<T extends ObjectLiteral = any> = Partial<
  Record<keyof Repository<T>, jest.Mock>
>;

describe('NotasService', () => {
  let service: NotasService;
  let repository: MockRepository<Nota>;

  beforeEach(async () => {
    const module: TestingModule = await Test.createTestingModule({
      providers: [
        NotasService,
        {
          provide: getRepositoryToken(Nota),
          useValue: mockNotaRepository(),
        },
      ],
    }).compile();

    service = module.get<NotasService>(NotasService);
    repository = module.get<MockRepository<Nota>>(getRepositoryToken(Nota));
  });
```

2. Casos de prueba

Crear una nota

notas.service.spec.ts

```
it('deberia crear una nota', async () => {
    jest.spyOn(repository, 'save').mockResolvedValue(mockNota as Nota);

    const result = await service.create({
      title: 'Test Nota',
      content: 'Test Content',
    });
    expect(result).toEqual(mockNota);

    expect(repository.save).toHaveBeenCalled();
    expect(repository.create).toHaveBeenCalled();
  });
```

Mostrar todas las notas

notas.service.spec.ts

```
  it('deberia encontrar todas las notas', async () => {
    jest.spyOn(repository, 'find').mockResolvedValue([mockNota] as Nota[]);

    const result = await service.findAll();
    expect(result).toEqual([mockNota]);

    expect(repository.find).toHaveBeenCalled();
  });
```

Buscar una nota

notas.service.spec.ts

```
describe('select (buscar una nota)', () => {
    describe('cuando la nota existe', () => {
      it('deberia encontrar una nota por id', async () => {
        jest.spyOn(repository, 'findOneBy').mockResolvedValue(mockNota as
Nota);

        const id: number = 1;
        const result = await service.findOne(id);
        expect(result).toEqual(mockNota);
```

```
        expect(repository.findOneBy).toHaveBeenCalledWith({ id: id });
      });
    });

    describe('cuando la nota no existe', () => {
      it('deberia lanzar un error si no encuentra una nota por id', async ()
=> {
        jest.spyOn(repository, 'findOneBy').mockResolvedValue(null);

        const id: number = 999;
        await
expect(service.findOne(id)).rejects.toThrow(NotFoundException);

        expect(repository.findOneBy).toHaveBeenCalledWith({ id: id });
      });
    });
  });
```

Modificar una nota

notas.service.spec.ts

```
  describe('update (modificar una nota)', () => {
    describe('cuando la nota existe', () => {
      it('deberia modificar una nota', async () => {
        const id = 1;
        const updateNotaDto = {
          title: 'Updated Nota',
        };
        const notaActualizada = {
          ...mockNota,
          ...updateNotaDto,
        } as Nota;

        const updateResult = {
          affected: 1,
          raw: {},
          generatedMaps: [],
        } as UpdateResult;
        jest.spyOn(repository, 'update').mockResolvedValue(updateResult);
        jest.spyOn(service, 'findOne').mockResolvedValue(notaActualizada);

        const result = await service.update(id, updateNotaDto);

        expect(repository.update).toHaveBeenCalledWith(id, updateNotaDto);
        expect(service.findOne).toHaveBeenCalledWith(id);
        expect(result).toEqual(notaActualizada);
      });
    });
```

```
    describe('cuando la nota no existe', () => {
      it('deberia lanzar NotFoundException si la nota no exite', async () =>
{
        const id = 999;
        const updateNotaDto = {
          title: 'Updated Nota',
        };
        const updateResult = {
          affected: 0,
          raw: {},
          generatedMaps: [],
        } as UpdateResult;

        jest.spyOn(repository, 'update').mockResolvedValue(updateResult);
        jest.spyOn(service, 'findOne').mockImplementation();

        await expect(service.update(id, updateNotaDto)).rejects.toThrow(
          NotFoundException,
        );

        expect(repository.update).toHaveBeenCalledWith(id, updateNotaDto);
        expect(service.findOne).not.toHaveBeenCalled();
      });
    });
  });
```

Eliminar una nota

notas.service.spec.ts

```
describe('eliminar nota)', () => {
    describe('cuando la nota existe', () => {
      it('deberia eliminar la nota', async () => {
        const id = 1;
        jest.spyOn(repository, 'delete').mockResolvedValue({ affected: 1 });

        await service.remove(id);

        expect(repository.delete).toHaveBeenCalledWith(id);
      });
    });

    describe('cuando la nota no existe', () => {
      it('deberia lanzar NotFoundException si la nota no existe', async ()
=> {
        const id = 999;
        jest.spyOn(repository, 'delete').mockResolvedValue({ affected: 0 });

        await expect(service.remove(id)).rejects.toThrow(NotFoundException);
```

```
        expect(repository.delete).toHaveBeenCalledWith(id);
      });
    });
  });
```

No olvide cerrar "});" correspondiente a:

```
describe('NotasService', () => {
```

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

```
npm run test
```

3. Glosario de términos

Glosario de Términos - Pruebas Unitarias en NestJS

1. `Test` / `TestingModule`
   - Clases de `@nestjs/testing` que permiten configurar un entorno de pruebas aislado para módulos NestJS.

2. `Mock` (Simulación)
   - Objeto falso que imita el comportamiento de dependencias reales (ej: Base de datos) para aislar las pruebas.

3. `Repository` (TypeORM)
   - Patrón que abstrae el acceso a la base de datos. En pruebas, se reemplaza por un mock (`MockRepository`).

4. `jest.fn()`
   - Función de Jest para crear mocks de métodos, permitiendo rastrear llamadas y definir respuestas simuladas.

5. `getRepositoryToken`
   - Utilidad de `@nestjs/typeorm` para inyectar repositorios en pruebas usando tokens.

6. `Partial<T>`

- Tipo de TypeScript que hace opcionales todas las propiedades de `T` (usado para mocks parciales).

7. `describe` / `it`
   - Bloques de Jest para organizar pruebas:
     - `describe`: Agrupa pruebas relacionadas (ej: por método).
     - `it`: Define un caso de prueba individual.

8. `beforeEach`
   - Hook de Jest que ejecuta código antes de cada prueba (aquí se inicializa el módulo de testing).

9. `spyOn`
   - Método de Jest para "espiar" funciones y controlar su comportamiento (ej: `mockResolvedValue`).

10. `NotFoundException`
    - Excepción de NestJS lanzada cuando un recurso no existe (manejo de errores HTTP).

11. `UpdateResult`
    - Tipo de TypeORM que representa el resultado de operaciones `update` o `delete` (ej: `affected: number`).

12. `rejects.toThrow()`
    - Assertion de Jest para verificar que una promesa rechaza con un error específico.

13. `toHaveBeenCalled` / `toEqual`
    - Assertions de Jest:
      - `toHaveBeenCalled`: Verifica si un mock fue llamado.
      - `toEqual`: Compara objetos/valores profundamente.

14. `ObjectLiteral`
    - Tipo de TypeORM que representa objetos genéricos (usado en el tipo `MockRepository`).

15. `TestingModule`

- Simula un módulo NestJS para pruebas, permitiendo sobrescribir proveedores (ej: repositorios).

Glosario de Métodos de Jest

Tabla 1: Creación de Mocks y Espías

Método	Descripción	Ejemplo
jest.fn()	Crea una función mock	const mock = jest.fn();
jest.spyOn()	Espía un método de un objeto	jest.spyOn(repo, 'save')
jest.mock()	Reemplaza un módulo completo	jest.mock('@nestjs/typeorm')

Tabla 2: Configuración de Comportamiento

Método	Descripción	Ejemplo
mockReturnValue(val)	Valor de retorno fijo	mockFn.mockReturnValue(42)
mockResolvedValue(val)	Promesa resuelta	mockFn.mockResolvedValue({id: 1})
mockRejectedValue(error)	Promesa rechazada	mockFn.mockRejectedValue(new Error())
mockImplementation(fn)	Implementación personalizada	mockFn.mockImplementation((x) => x*2)

Tabla 3: Verificaciones (Assertions)

Método	Descripción	Ejemplo
expect().toBe()	Comparación estricta	expect(result).toBe(42)
expect().toEqual()	Comparación profunda	expect(result).toEqual({id: 1})
expect().toHaveBeenCalled()	Verifica llamada	expect(mockFn).toHaveBeenCalled()
expect().toHaveBeenCalledWith()	Verifica argumentos	expect(mockFn).toHaveBeenCalledWith(1)
expect().toThrow()	Verifica error	expect(fn).toThrow(Error)

Tabla 4: Configuración del Entorno

Método	Descripción	Ejemplo
beforeEach(fn)	Antes de cada prueba	beforeEach(() => init())

afterEach(fn)	Después de cada prueba	afterEach(() => clean())
beforeAll(fn)	Antes de todas las pruebas	beforeAll(async () => await setup())
afterAll(fn)	Después de todas las pruebas	afterAll(() => teardown())

Tabla 5: Agrupación y Organización

Método	Descripción	Ejemplo
describe(name, fn)	Agrupa pruebas	describe('Servicio', () => {...})
it(name, fn)	Caso de prueba	it('should work', () => {...})
test(name, fn)	Alias de it	test('...', () => {...})

Tabla 6: Métodos Avanzados

Método	Descripción	Ejemplo
jest.clearAllMocks()	Limpia mocks	afterEach(jest.clearAllMocks)
jest.resetAllMocks()	Restablece mocks	afterEach(jest.resetAllMocks)
jest.useFakeTimers()	Simula temporizadores	jest.useFakeTimers()