VALIDACIÓN Y VERIFICACIÓN DE SOFTWARE (INF 732)
GUÍA DE LABORATORIO 5
Pruebas de integración para el módulo Notas

Las pruebas de integración son esenciales para garantizar que los diferentes componentes de una aplicación backend funcionen correctamente en conjunto, especialmente cuando interactúan con bases de datos. En este laboratorio, exploraremos cómo implementar pruebas de integración para un servicio NestJS utilizando TypeORM, validando operaciones CRUD contra una base de datos MySQL real. Aprenderemos a configurar un entorno de pruebas aislado, escribir casos de prueba significativos y manejar escenarios de error, asegurando que nuestra aplicación sea robusta y confiable antes de llegar a producción. Mediante ejercicios prácticos, descubriremos cómo estas pruebas ayudan a detectar problemas temprano y mantienen la calidad del código en sistemas complejos.

Objetivos

- Comprender la importancia de las pruebas de integración en aplicaciones backend.
- Implementar pruebas de integración para un servicio NestJS con TypeORM.
- Validar el comportamiento de operaciones CRUD contra una base de datos real.

Prerrequisitos

Haber desarrollado las Guías 1, 2, 3 y 4 de la asignatura.

1. Configuración inicial

notas.integration.spec.ts

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

```
describe('Notas Integration Tests', () => {
  let service: NotasService;
  let repository: Repository<Nota>;
  beforeAll(async () => {
    const module: TestingModule = await Test.createTestingModule({
      imports: [
        TypeOrmModule.forRoot({
          type: 'mysql',
          host: 'localhost',
          port: 3306,
          username: 'root',
          password: '',
          database: 'notas_test',
          entities: [Nota],
          synchronize: true,
        }),
        TypeOrmModule.forFeature([Nota]),
      ],
      providers: [NotasService],
    }).compile();
    service = module.get<NotasService>(NotasService);
    repository = module.get<Repository<Nota>>(getRepositoryToken(Nota));
  });

  afterAll(async () => {
    const connection = repository.manager.connection;
    if (connection.isInitialized) {
      await connection.destroy();
    }
  });

  afterEach(async () => {
    await repository.query('DELETE FROM nota;');
  });

...
});
```

2. Casos de prueba

Crear una nota

notas.integration.spec.ts

```
  it('Deberia crear una nueva nota en la base de datos', async () => {
    const nuevaNota = {
      title: 'Nota de prueba',
```

```
      content: 'Contenido de prueba',
    };

    const notaCreada = await service.create(nuevaNota);

    // Verificar la respuesta del servicio
    expect(notaCreada).toHaveProperty('id');
    expect(notaCreada.title).toEqual(nuevaNota.title);
    expect(notaCreada.content).toEqual(nuevaNota.content);

    // Verificar que realmente se guardó en la base de datos
    const notasEnDB = await repository.findOneBy({ id: notaCreada.id });
    expect(notasEnDB).not.toBeNull();
    if (notasEnDB) {
      expect(notasEnDB.title).toEqual(nuevaNota.title);
      expect(notasEnDB.content).toEqual(nuevaNota.content);
    }
  });
```

Mostrar todas las notas

notas.integration.spec.ts

```
  it('Deberia obtener todas las notas de la base de datos', async () => {
    await repository.save([
      { title: 'Nota 1', content: 'Contenido 1' },
      { title: 'Nota 2', content: 'Contenido 2' },
    ]);

    const notas = await service.findAll();
    expect(notas.length).toBe(2);
    expect(notas[0].title).toBe('Nota 1');
    expect(notas[1].title).toBe('Nota 2');
  });
```

Buscar una nota

notas.integration.spec.ts

```
  describe('findOne()', () => {
    it('Debería obtener una nota por ID', async () => {
      const nuevaNota = await repository.save({
        title: 'Nota específica',
        content: 'Contenido específico',
      });

      const notaEncontrada = await service.findOne(nuevaNota.id);
```

```
      expect(notaEncontrada).toBeDefined();
      expect(notaEncontrada.title).toEqual('Nota específica');
      expect(notaEncontrada.content).toEqual('Contenido específico');
    });

    it('Debería lanzar NotFoundException al no encontrar una nota por ID',
async () => {
      const notaInexistente = 999;
      try {
        await service.findOne(notaInexistente);
      } catch (error) {
        expect(error).toBeInstanceOf(NotFoundException);
      }
    });
  });
```

Modificar una nota

notas.integration.spec.ts

```
  describe('update()', () => {
    it('Debería actualizar una nota existente', async () => {
      const nuevaNota = await repository.save({
        title: 'Nota antes de actualizar',
        content: 'Contenido antes de actualizar',
      });

      const notaActualizada = await service.update(nuevaNota.id, {
        title: 'Nota actualizada',
        content: 'Contenido actualizado',
      });

      expect(notaActualizada).toBeDefined();
      expect(notaActualizada.title).toEqual('Nota actualizada');
      expect(notaActualizada.content).toEqual('Contenido actualizado');

      const notasEnDB = await repository.findOneBy({ id: nuevaNota.id });
      expect(notasEnDB).not.toBeNull();
      if (notasEnDB) {
        expect(notasEnDB.title).toEqual(notaActualizada.title);
        expect(notasEnDB.content).toEqual(notaActualizada.content);
      }
    });

    it('Debería lanzar NotFoundException al no encontrar una nota para
modificar', async () => {
      const notaInexistente = 999;
      try {
        await service.update(notaInexistente, {
          title: 'Nota actualizada',
          content: 'Contenido actualizado',
```

```
        });
      } catch (error) {
        expect(error).toBeInstanceOf(NotFoundException);
      }
    });
  });
```

Eliminar una nota

notas.integration.spec.ts

```
  describe('remove()', () => {
    it('Debería eliminar una nota existente', async () => {
      const notaExistente = await repository.save({
        title: 'Nota para eliminar',
        content: 'Contenido para eliminar',
      });

      await service.remove(notaExistente.id);
      const notaEliminada = await repository.findOneBy({
        id: notaExistente.id,
      });
      expect(notaEliminada).toBeNull();
    });

    it('debería lanzar NotFoundException al intentar eliminar una nota
inexistente', async () => {
      const idInexistente = 9999;

      await expect(service.remove(idInexistente)).rejects.toThrow(
        NotFoundException,
      );
    });
  });
```

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

```
npm run test
```