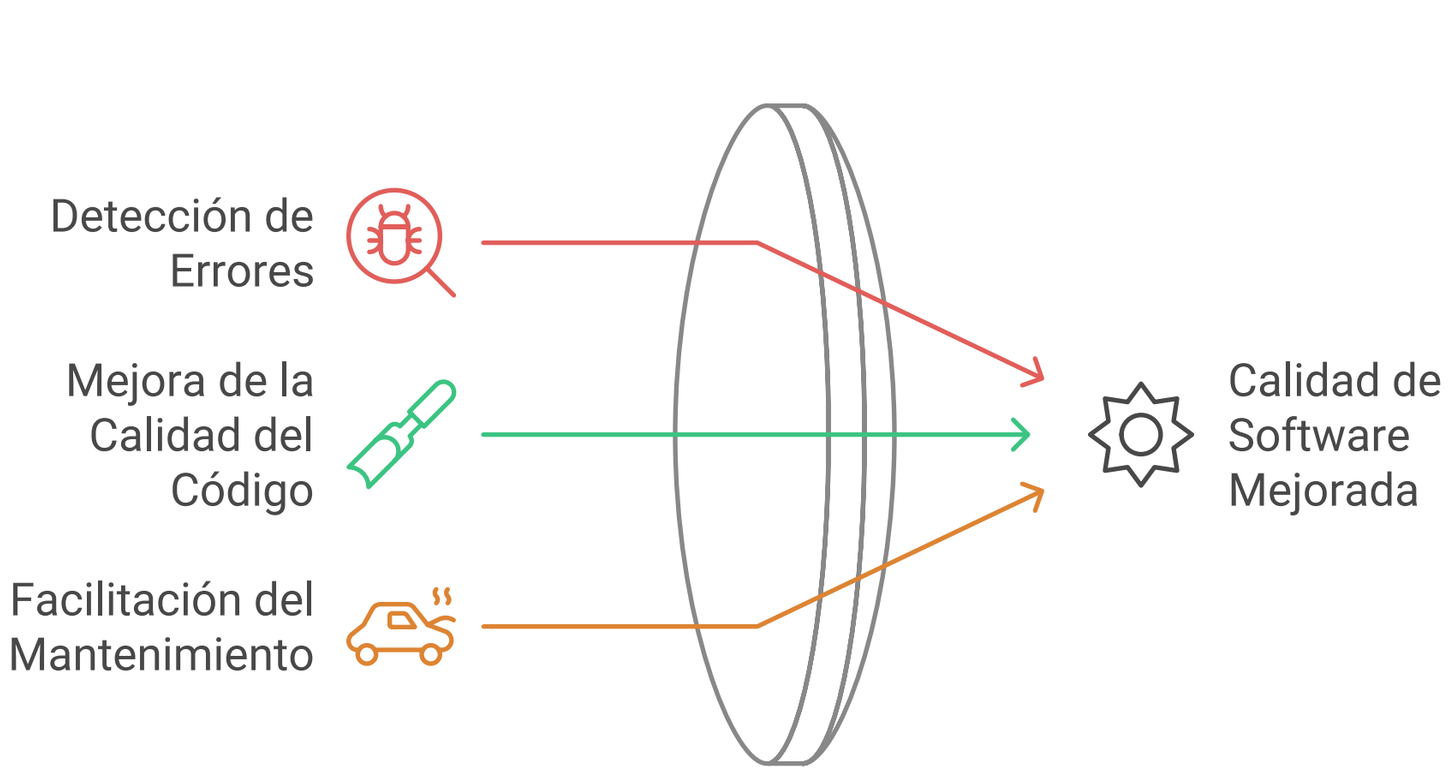
Introducción a las pruebas unitarias en NestJS

¿Qué son las pruebas unitarias?

Las pruebas unitarias son un tipo de prueba de software que se centra en verificar el funcionamiento de unidades individuales de código, como funciones, métodos o clases, de manera aislada. Estas pruebas ayudan a:

- Detectar errores rápidamente.
- Mejorar la calidad del código.
- Facilitar el mantenimiento y refactorización.



En NestJS, las pruebas unitarias se realizan comúnmente con **Jest**, un framework de pruebas incluido por defecto.

Configuración del entorno de pruebas

Para asegurarte de que tu proyecto NestJS está listo para pruebas unitarias:

1. **Crear un nuevo proyecto NestJS (si no tienes uno):**

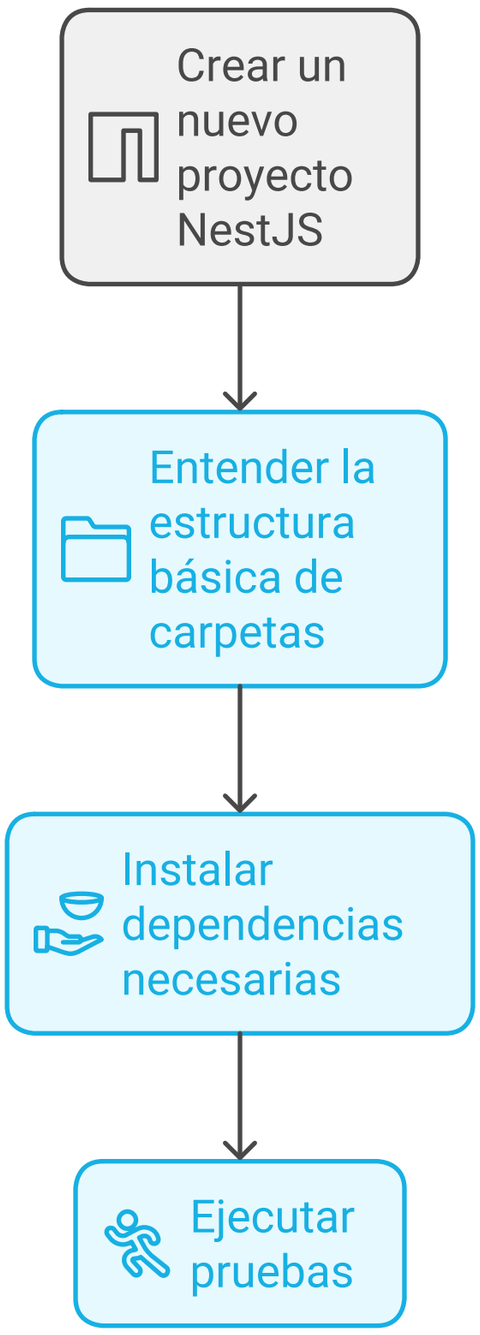
```
nest new my-nestjs-project
```

2. **Estructura básica de carpetas:**
    - **src/**: Contiene la lógica principal de tu aplicación.
    - **test/**: Contiene pruebas de integración y unitarias.
3. **Dependencias necesarias:**
    - Jest ya está configurado en proyectos generados con NestJS. Si necesitas instalarlo manualmente:

```
npm install --save-dev jest @nestjs/testing
```

4. **Ejecutar pruebas:**

```
npm run test
```



Creación de un ejemplo básico: Servicio y pruebas unitarias

1. Crear el servicio calculator.service.ts

En el directorio **src/**, crea un archivo llamado **calculator.service.ts**:

```
import { Injectable } from '@nestjs/common';

@Injectable()
export class CalculatorService {
  add(a: number, b: number): number {
    return a + b;
  }

  subtract(a: number, b: number): number {
    return a - b;
  }
}
```

2. Escribir pruebas unitarias para el servicio

Crea el archivo **calculator.service.spec.ts** en el mismo directorio:

```
import { Test, TestingModule } from '@nestjs/testing';
import { CalculatorService } from './calculator.service';

describe('CalculatorService', () => {
  let service: CalculatorService;

  beforeEach(async () => {
    const module: TestingModule = await Test.createTestingModule({
      providers: [CalculatorService],
    }).compile();

    service = module.get<CalculatorService>(CalculatorService);
  });

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

  it('should add two numbers', () => {
    expect(service.add(2, 3)).toBe(5);
  });

  it('should subtract two numbers', () => {
    expect(service.subtract(5, 3)).toBe(2);
  });
});
```

Explicación:

- **beforeEach**: Configura un módulo de pruebas antes de cada caso de prueba.
- **it**: Define un caso de prueba individual.

Ejecuta las pruebas:

```
npm run test
```

Pruebas con dependencias: Controlador y servicio

1. Crear el controlador calculator.controller.ts

En el directorio **src/**, crea un archivo llamado **calculator.controller.ts**:

```
import { Controller, Get, Query } from '@nestjs/common';
import { CalculatorService } from './calculator.service';

@Controller('calculator')
export class CalculatorController {
  constructor(private readonly calculatorService: CalculatorService) {}

  @Get('add')
  add(@Query('a') a: string, @Query('b') b: string): number {
    return this.calculatorService.add(+a, +b);
  }
}
```

2. Escribir pruebas para el controlador con mocks

Crea el archivo **calculator.controller.spec.ts**:

```
import { Test, TestingModule } from '@nestjs/testing';
import { CalculatorController } from './calculator.controller';
import { CalculatorService } from './calculator.service';

describe('CalculatorController', () => {
  let controller: CalculatorController;
  let service: CalculatorService;

  beforeEach(async () => {
    const mockService = {
      add: jest.fn((a, b) => a + b),
    };

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

    controller = module.get<CalculatorController>(CalculatorController);
    service = module.get<CalculatorService>(CalculatorService);
  });

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

  it('should call add method from service', () => {
    const result = controller.add('2', '3');
    expect(service.add).toHaveBeenCalledWith(2, 3);
    expect(result).toBe(5);
  });
});
```

Explicación:

- **Mocks:** Utilizamos **jest.fn()** para crear funciones simuladas que controlan el comportamiento de las dependencias externas.
- **useValue:** Proporciona una implementación simulada del servicio en el módulo de pruebas.
- **Verificación de llamadas:** Aseguramos que el método **add** del servicio fue llamado con los argumentos correctos.

Buenas prácticas y consejos

1. Usa nombres descriptivos para las pruebas, como **should add two numbers**.
2. Escribe pruebas pequeñas y específicas para una responsabilidad.
3. Usa mocks para dependencias externas y evita pruebas acopladas.
4. Cubre casos positivos y negativos (como valores nulos o extremos).

5. Ejecuta las pruebas frecuentemente durante el desarrollo.
6. Verifica la cobertura de pruebas:

```
npm run test:cov
```

Conclusión y recursos adicionales

Resumen:

- Configuraste el entorno de pruebas en un proyecto NestJS.
- Escribiste pruebas unitarias para un servicio y un controlador.
- Usaste mocks para manejar dependencias.

Recursos:

- Documentación oficial de NestJS Testing. [https://docs.nestjs.com/fundamentals/testing]
- Guía de Jest. [https://docs.nestjs.com/fundamentals/testing]

¡Felicidades por aprender cómo escribir pruebas unitarias en NestJS!