



Diego Felipe Solorzano  
Juan David Loaiza  
Daniel Aguilar  
Juan David Chacón

UNIVERSIDAD NACIONAL DE  
COLOMBIA SEDE BOGOTÁ  
FACULTAD DE  
INGENIERÍA  
DEPARTAMENTO DE SISTEMAS E INDUSTRIAL

INGENIERÍA DE SOFTWARE 1  
BOGOTÁ D.C  
2024-2

# Taller testing

## Introducción

El proyecto consiste en una aplicación móvil, diseñada con una interfaz intuitiva para facilitar viajes compartidos. Los usuarios pueden registrar o buscar viajes, indicando días, horarios y ubicaciones, mientras que los conductores pueden publicar sus rutas y horarios para conectar con pasajeros. La app incluye funciones como notificaciones en tiempo real, calificaciones, pasarela de pago y medidas de seguridad como verificación de identidad y registro de viajes, promoviendo una movilidad económica, eficiente y segura dentro de la comunidad universitaria, sin responsabilizarse por los acuerdos entre usuarios.

Por lo tanto, el presente trabajo tiene como objetivo poder implementar test unitarios y de integración para poder comprender su uso y utilidad dentro del entorno del desarrollo. Las herramientas que vamos a utilizar para testear están enfocadas en el testing del backend sobre el framework nestjs:

- **Jest:** Es un framework de JS destinado al testeo y corrección de cualquier código base de JS. Funciona de manera sencilla sobre JS y sus derivados (en este caso TS).
- **Supertest:** Es una librería de Node.js diseñada para facilitar las pruebas de APIs. Es útil para pruebas que incluyen Jest verificando el estado de las rutas.
- **@Nestjs/testing:** Es un módulo proporcionado por el framework NestJS para facilitar las pruebas sobre aplicaciones construidas en este. Simplifica las pruebas unitarias integrándose de forma predeterminada con Jest y Supertest.
- **@Types/supertest:** Este paquete contiene las definiciones de supertest de tipo TS, esta librería supertest está creada para probar APIs en NodeJS.
- **@Types/jest:** Es un paquete con las definiciones de Jest para Typescript, agregando funcionalidades globales de Jest que simplifican el trabajo de testing.

Con estas herramientas vamos a hacer los test unitarios sobre los módulos de login y pagos, ya que estos son críticos en la aplicación.

Para llevar a cabo los tests, dado que algunos teníamos dudas e inexperiencia en el tema, decidimos reunirnos en grupo para, entre todos, comprender el proceso, identificar los componentes a testear y determinar la mejor forma de hacerlo.

|   |  |
|---|--|
| <b>Nombre</b>                                   | Juan David Loaiza Reyes  |
| <b>Tipo de prueba</b>                           | Unitaria   |
| <b>Descripción breve del componente probado</b> | El componente probado tiene la función de generar JWT con la información de este mismo (email, id y rol). Estos tokens deben funcionar para mantener constancia en la sesión del usuario, además de que genera un token de refresco de la sesión, para cuando el usuario ha expirado el tiempo de su JWT original.   |
| <b>Screenshot del código</b>                    | <pre> import { Test, TestingModule } from '@nestjs/testing'; import { AuthService } from '../auth.service'; import { PrismaService } from '../../prisma/prisma.service'; import { UsersService } from '../../users/users.service'; import { JwtService } from '@nestjs/jwt'; import { ConfigService } from '@nestjs/config'; import { UtilsService } from '@app/utils'; import { Payload } from '../types/payload.types'; import { Role } from '../enums/role.enum';  describe('AuthService', () =&gt; {   let service: AuthService;   let usersService: jest.Mocked&lt;UsersService&gt;;   let jwtService: jest.Mocked&lt;JwtService&gt;;   let configService: jest.Mocked&lt;ConfigService&gt;;   let utilsService: jest.Mocked&lt;UtilsService&gt;;    beforeEach(async () =&gt; {     const mockUsersService = {       findOneByEmail: jest.fn(),       updateUserById: jest.fn(),     };      const mockJwtService = {       signAsync: jest.fn(),       verifyAsync: jest.fn(),     };      const mockPayload: Payload = {       email: 'test@test.com',       sub: '123',       role: Role.ADMIN,     };      const mockConfigService = {       get: jest.fn(),     }; </pre> |

```

const mockUtilsService = {
  comparePassword: jest.fn(),
  hashPassword: jest.fn(),
};

const module: TestingModule = await
Test.createTestingModule({
  providers: [
    AuthService,
    { provide: UsersService, useValue: mockUsersService },
    { provide: JwtService, useValue: mockJwtService },
    { provide: ConfigService, useValue:
mockConfigService },
    { provide: UtilsService, useValue: mockUtilsService },
  ],
}).compile();

service = module.get<AuthService>(AuthService);
userService = module.get(UsersService);
jwtService = module.get(JwtService);
configService = module.get(ConfigService);
utilsService = module.get(UtilsService);
});

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

describe('generateToken', () => {
  it('should return access_token and refresh_token', async
() => {
    const mockPayload: Payload = {
      email: 'test@test.com',
      sub: '123',
      role: Role.ADMIN,
    };

    configService.get.mockImplementation((key: string) =>
{
      if (key === 'JWT_SECRET') return 'test-secret';
      if (key === 'JWT_SECRET_REFRESH') return
'test-refresh-secret';
      return null;
    });
  });
});

```

```
});

jwtService.signAsync.mockResolvedValueOnce('mock-access-token');

jwtService.signAsync.mockResolvedValueOnce('mock-refresh-token');

const result = await (service as any).generateToken(mockPayload);

expect(result).toEqual({
  access_token: 'mock-access-token',
  refresh_token: 'mock-refresh-token',
});

expect(jwtService.signAsync).toHaveBeenCalledTimes(2);

expect(jwtService.signAsync).toHaveBeenNthCalledWith(1,
mockPayload, {
  secret: 'test-secret',
  expiresIn: '15m',
});

expect(jwtService.signAsync).toHaveBeenNthCalledWith(2,
mockPayload, {
  secret: 'test-refresh-secret',
  expiresIn: '7d',
});
});
});
});
```

```

1 < core > auth > @auth.service.spec.ts > @describe(AuthService) callback > @describe(generateToken) callback > @it('should return access_token and refresh_token') callback
2 import { Test, TestingModule } from '@nestjs/testing';
3 import { AuthService } from '../auth.service';
4 import { PrismaService } from '../prisma/prisma.service';
5 import { UserService } from '../users/users.service';
6 import { JwtService } from '@nestjs/jwt';
7 import { ConfigService } from '@nestjs/config';
8 import { UtilsService } from '@app/utils';
9 import { Payload } from '../types/payload.types';
10 import { Role } from '../enums/role.enum';
11
12 * @Run Debug
13 describe('AuthService', () => {
14   let service: AuthService;
15   let userService: jest.Mocked<UserService>;
16   let jwtService: jest.Mocked<JwtService>;
17   let configService: jest.Mocked<ConfigService>;
18   let utilsService: jest.Mocked<UtilsService>;
19
20   beforeEach(async () => {
21     const mockUserService = {
22       findOneByEmail: jest.fn(),
23       updateUserById: jest.fn(),
24     };
25
26     const mockJwtService = {
27       signAsync: jest.fn(),
28       verifyAsync: jest.fn(),
29     };
30
31     const mockPayload: Payload = {
32       email: 'test@test.com',
33       sub: '123',
34       role: Role.ADMIN,
35     };
36
37     const mockConfigService = {
38       get: jest.fn(),
39     };
40
41     const mockUtilsService = {
42       comparePassword: jest.fn(),
43       hashPassword: jest.fn(),
44     };
45
46     const module: TestingModule = await Test.createTestingModule({
47       providers: [
48         AuthService,
49         { provide: UserService, useValue: mockUserService },
50       ],
51     }).compile();
52
53     service = module.get<AuthService>(AuthService);
54     userService = module.get<UserService>(UserService);
55     jwtService = module.get<JwtService>(JwtService);
56     configService = module.get<ConfigService>(ConfigService);
57     utilsService = module.get<UtilsService>(UtilsService);
58   });
59
60   * @Run Debug
61   it('should be defined', () => {
62     expect(service).toBeDefined();
63   });
64
65   * @Run Debug
66   describe('generateToken', () => {
67     * @Run Debug
68     it('should return access token and refresh token', async () => {
69       const mockPayload: Payload = {
70         email: 'test@test.com',
71         sub: '123',
72         role: Role.ADMIN,
73       };
74
75       configService.get.mockImplementation((key: string) => {
76         if (key === 'JWT_SECRET') return 'test-secret';
77         if (key === 'JWT_SECRET_REFRESH') return 'test-refresh-secret';
78         return null;
79       });
80
81       jwtService.signAsync.mockResolvedValueOnce('mock-access-token');
82       jwtService.signAsync.mockResolvedValueOnce('mock-refresh-token');
83
84       const result = await (service as any).generateToken(mockPayload);
85
86       expect(result).toEqual({
87         access_token: 'mock-access-token',
88         refresh_token: 'mock-refresh-token',
89       });
90     });
91   });
92
93   * @Run Debug
94   describe('refreshToken', () => {
95     * @Run Debug
96     it('should return access token and refresh token', async () => {
97       const mockRefreshToken: string = 'mock-refresh-token';
98
99       jwtService.verifyAsync.mockResolvedValueOnce(mockPayload);
100       jwtService.signAsync.mockResolvedValueOnce('mock-access-token');
101       jwtService.signAsync.mockResolvedValueOnce('mock-refresh-token');
102
103       const result = await (service as any).refreshToken(mockRefreshToken);
104
105       expect(result).toEqual({
106         access_token: 'mock-access-token',
107         refresh_token: 'mock-refresh-token',
108       });
109
110       expect(jwtService.signAsync).toHaveBeenCalledTimes(2);
111       expect(jwtService.signAsync).toHaveBeenCalledWith(1, mockPayload, {
112         secret: 'test-secret',
113         expiresIn: '15m',
114       });
115       expect(jwtService.signAsync).toHaveBeenCalledTimes(2);
116       expect(jwtService.signAsync).toHaveBeenCalledWith(2, mockPayload, {
117         secret: 'test-refresh-secret',
118         expiresIn: '7d',
119       });
120     });
121   });
122 });

```

```

1 < core > auth > @auth.service.spec.ts > @describe(AuthService) callback > @describe(generateToken) callback > @it('should return access_token and refresh_token') callback
2 import { Test, TestingModule } from '@nestjs/testing';
3 import { AuthService } from '../auth.service';
4 import { PrismaService } from '../prisma/prisma.service';
5 import { UserService } from '../users/users.service';
6 import { JwtService } from '@nestjs/jwt';
7 import { ConfigService } from '@nestjs/config';
8 import { UtilsService } from '@app/utils';
9 import { Payload } from '../types/payload.types';
10 import { Role } from '../enums/role.enum';
11
12 * @Run Debug
13 describe('AuthService', () => {
14   let service: AuthService;
15   let userService: jest.Mocked<UserService>;
16   let jwtService: jest.Mocked<JwtService>;
17   let configService: jest.Mocked<ConfigService>;
18   let utilsService: jest.Mocked<UtilsService>;
19
20   beforeEach(async () => {
21     const mockUserService = {
22       findOneByEmail: jest.fn(),
23       updateUserById: jest.fn(),
24     };
25
26     const mockJwtService = {
27       signAsync: jest.fn(),
28       verifyAsync: jest.fn(),
29     };
30
31     const mockPayload: Payload = {
32       email: 'test@test.com',
33       sub: '123',
34       role: Role.ADMIN,
35     };
36
37     const mockConfigService = {
38       get: jest.fn(),
39     };
40
41     const mockUtilsService = {
42       comparePassword: jest.fn(),
43       hashPassword: jest.fn(),
44     };
45
46     const module: TestingModule = await Test.createTestingModule({
47       providers: [
48         AuthService,
49         { provide: UserService, useValue: mockUserService },
50         { provide: JwtService, useValue: mockJwtService },
51         { provide: ConfigService, useValue: mockConfigService },
52         { provide: UtilsService, useValue: mockUtilsService },
53       ],
54     }).compile();
55
56     service = module.get<AuthService>(AuthService);
57     userService = module.get<UserService>(UserService);
58     jwtService = module.get<JwtService>(JwtService);
59     configService = module.get<ConfigService>(ConfigService);
60     utilsService = module.get<UtilsService>(UtilsService);
61   });
62
63   * @Run Debug
64   it('should be defined', () => {
65     expect(service).toBeDefined();
66   });
67
68   * @Run Debug
69   describe('generateToken', () => {
70     * @Run Debug
71     it('should return access token and refresh token', async () => {
72       const mockPayload: Payload = {
73         email: 'test@test.com',
74         sub: '123',
75         role: Role.ADMIN,
76       };
77
78       configService.get.mockImplementation((key: string) => {
79         if (key === 'JWT_SECRET') return 'test-secret';
80         if (key === 'JWT_SECRET_REFRESH') return 'test-refresh-secret';
81         return null;
82       });
83
84       jwtService.signAsync.mockResolvedValueOnce('mock-access-token');
85       jwtService.signAsync.mockResolvedValueOnce('mock-refresh-token');
86
87       const result = await (service as any).generateToken(mockPayload);
88
89       expect(result).toEqual({
90         access_token: 'mock-access-token',
91         refresh_token: 'mock-refresh-token',
92       });
93     });
94   });
95
96   * @Run Debug
97   describe('refreshToken', () => {
98     * @Run Debug
99     it('should return access token and refresh token', async () => {
100       const mockRefreshToken: string = 'mock-refresh-token';
101
102       jwtService.verifyAsync.mockResolvedValueOnce(mockPayload);
103       jwtService.signAsync.mockResolvedValueOnce('mock-access-token');
104       jwtService.signAsync.mockResolvedValueOnce('mock-refresh-token');
105
106       const result = await (service as any).refreshToken(mockRefreshToken);
107
108       expect(result).toEqual({
109         access_token: 'mock-access-token',
110         refresh_token: 'mock-refresh-token',
111       });
112
113       expect(jwtService.signAsync).toHaveBeenCalledTimes(2);
114       expect(jwtService.signAsync).toHaveBeenCalledWith(1, mockPayload, {
115         secret: 'test-secret',
116         expiresIn: '15m',
117       });
118       expect(jwtService.signAsync).toHaveBeenCalledTimes(2);
119       expect(jwtService.signAsync).toHaveBeenCalledWith(2, mockPayload, {
120         secret: 'test-refresh-secret',
121         expiresIn: '7d',
122       });
123     });
124   });
125 });

```

```

1 < core > auth > @auth.service.spec.ts > @describe(AuthService) callback > @describe(generateToken) callback > @it('should return access_token and refresh_token') callback
2 import { Test, TestingModule } from '@nestjs/testing';
3 import { AuthService } from '../auth.service';
4 import { PrismaService } from '../prisma/prisma.service';
5 import { UserService } from '../users/users.service';
6 import { JwtService } from '@nestjs/jwt';
7 import { ConfigService } from '@nestjs/config';
8 import { UtilsService } from '@app/utils';
9 import { Payload } from '../types/payload.types';
10 import { Role } from '../enums/role.enum';
11
12 * @Run Debug
13 describe('AuthService', () => {
14   let service: AuthService;
15   let userService: jest.Mocked<UserService>;
16   let jwtService: jest.Mocked<JwtService>;
17   let configService: jest.Mocked<ConfigService>;
18   let utilsService: jest.Mocked<UtilsService>;
19
20   beforeEach(async () => {
21     const mockUserService = {
22       findOneByEmail: jest.fn(),
23       updateUserById: jest.fn(),
24     };
25
26     const mockJwtService = {
27       signAsync: jest.fn(),
28       verifyAsync: jest.fn(),
29     };
30
31     const mockPayload: Payload = {
32       email: 'test@test.com',
33       sub: '123',
34       role: Role.ADMIN,
35     };
36
37     const mockConfigService = {
38       get: jest.fn(),
39     };
40
41     const mockUtilsService = {
42       comparePassword: jest.fn(),
43       hashPassword: jest.fn(),
44     };
45
46     const module: TestingModule = await Test.createTestingModule({
47       providers: [
48         AuthService,
49         { provide: UserService, useValue: mockUserService },
50         { provide: JwtService, useValue: mockJwtService },
51         { provide: ConfigService, useValue: mockConfigService },
52         { provide: UtilsService, useValue: mockUtilsService },
53       ],
54     }).compile();
55
56     service = module.get<AuthService>(AuthService);
57     userService = module.get<UserService>(UserService);
58     jwtService = module.get<JwtService>(JwtService);
59     configService = module.get<ConfigService>(ConfigService);
60     utilsService = module.get<UtilsService>(UtilsService);
61   });
62
63   * @Run Debug
64   it('should be defined', () => {
65     expect(service).toBeDefined();
66   });
67
68   * @Run Debug
69   describe('generateToken', () => {
70     * @Run Debug
71     it('should return access token and refresh token', async () => {
72       const mockPayload: Payload = {
73         email: 'test@test.com',
74         sub: '123',
75         role: Role.ADMIN,
76       };
77
78       configService.get.mockImplementation((key: string) => {
79         if (key === 'JWT_SECRET') return 'test-secret';
80         if (key === 'JWT_SECRET_REFRESH') return 'test-refresh-secret';
81         return null;
82       });
83
84       jwtService.signAsync.mockResolvedValueOnce('mock-access-token');
85       jwtService.signAsync.mockResolvedValueOnce('mock-refresh-token');
86
87       const result = await (service as any).generateToken(mockPayload);
88
89       expect(result).toEqual({
90         access_token: 'mock-access-token',
91         refresh_token: 'mock-refresh-token',
92       });
93
94       expect(jwtService.signAsync).toHaveBeenCalledTimes(2);
95       expect(jwtService.signAsync).toHaveBeenCalledWith(1, mockPayload, {
96         secret: 'test-secret',
97         expiresIn: '15m',
98       });
99       expect(jwtService.signAsync).toHaveBeenCalledTimes(2);
100       expect(jwtService.signAsync).toHaveBeenCalledWith(2, mockPayload, {
101         secret: 'test-refresh-secret',
102         expiresIn: '7d',
103       });
104     });
105   });
106
107   * @Run Debug
108   describe('refreshToken', () => {
109     * @Run Debug
110     it('should return access token and refresh token', async () => {
111       const mockRefreshToken: string = 'mock-refresh-token';
112
113       jwtService.verifyAsync.mockResolvedValueOnce(mockPayload);
114       jwtService.signAsync.mockResolvedValueOnce('mock-access-token');
115       jwtService.signAsync.mockResolvedValueOnce('mock-refresh-token');
116
117       const result = await (service as any).refreshToken(mockRefreshToken);
118
119       expect(result).toEqual({
120         access_token: 'mock-access-token',
121         refresh_token: 'mock-refresh-token',
122       });
123
124       expect(jwtService.signAsync).toHaveBeenCalledTimes(2);
125       expect(jwtService.signAsync).toHaveBeenCalledWith(1, mockPayload, {
126         secret: 'test-secret',
127         expiresIn: '15m',
128       });
129       expect(jwtService.signAsync).toHaveBeenCalledTimes(2);
130       expect(jwtService.signAsync).toHaveBeenCalledWith(2, mockPayload, {
131         secret: 'test-refresh-secret',
132         expiresIn: '7d',
133       });
134     });
135   });
136 });

```

|  |   |
|--|---|
| <b>Resultado de la ejecución</b>           | <pre> npm run test:watch PASS src/core/auth/auth.service.spec.ts AuthService   ✓ should be defined (8 ms)   generateToken     ✓ should return access_token and refresh_token (2 ms)  Test Suites: 1 passed, 1 total Tests:      2 passed, 2 total Snapshots:  0 total Time:       3.011 s Ran all test suites related to changed files.  npm run test:watch PASS src/core/auth/auth.service.spec.ts AuthService   ✓ should be defined (8 ms)   generateToken     ✓ should return access_token and refresh_token (2 ms)  Test Suites: 1 passed, 1 total Tests:      2 passed, 2 total Snapshots:  0 total Time:       3.011 s Ran all test suites related to changed files.  Watch Usage: Press w to show more. </pre> |
| <b>Lecciones aprendidas y dificultades</b> | <p>La dificultad de este test radicó en la implementación del módulo, jest solicita que las rutas que se usan sean rutas relativas a la posición del archivo y no absolutas al sistema operativo o la carpeta madre del código, esto para poder tener una mejor cohesión del código y en caso de cambiar la arquitectura facilitar la actualización de las importaciones</p>  |

Nótese que, bastante del código arriba mostrado se va a repetir, ya que vamos a reutilizar bastante código para facilitarnos los testeos.

|   |  |
|---|--|
| <b>Nombre</b>                                   | Daniel Aguilar Castro (DAGUILASTRO)  |
| <b>Tipo de prueba</b>                           | Unitaria   |
| <b>Descripción breve del componente probado</b> | Obtener la información de los usuarios es un proceso constante dentro del servicio y que se utiliza en gran medida tanto para poder referenciarse en la DB como obtener sus datos para la facturación por ejemplo.   |
| <b>Screenshot del código</b>                    | <pre> import { Test, TestingModule } from '@nestjs/testing'; import { UsersService } from './users.service'; import { PrismaService } from '../core/prisma/prisma.service'; import { UtilsService } from '@app/utils'; import { CreateUserDto } from './dto/create-user.dto'; import { UserRole } from './entities/user.entity';  describe('UsersService', () =&gt; {   let service: UsersService;   let prismaService: Partial&lt;PrismaService&gt;;   let utilsService: Partial&lt;UtilsService&gt;;    beforeEach(async () =&gt; {     prismaService = {       user: {         create: jest.fn(),         findFirst: jest.fn(),         findMany: jest.fn(),         delete: jest.fn(),         update: jest.fn(),       },     } as unknown as PrismaService;      utilsService = {       hashPassword: jest.fn(),     };      const module: TestingModule = await Test.createTestingModule({   providers: [     UsersService,     { provide: PrismaService, useValue: prismaService },     { provide: UtilsService, useValue: utilsService },   ], }).compile(); </pre> |



```

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

  it('should be defined', () => {
    expect(service).toBeDefined();
  });
  describe('get', () => {
    it('should find a user by email successfully',
  async () => {
    const mockEmail = 'test@example.com';
    const mockUser = {
      id: '1',
      email: mockEmail,
      phoneNumber: '1234567890',
      name: 'Test User',
      role: 'USER',
      password: 'hashed_password',
      birthDate: new Date('2000-01-01'),
    };

    (prismaService.user.findFirst as
  jest.Mock).mockResolvedValue(mockUser);

    const result = await
  service.findOneByEmail(mockEmail);

    expect(prismaService.user.findFirst).toHaveBeenCalledW
  ith({
      where: {
        email: mockEmail,
      },
    });
    expect(result).toEqual(mockUser);
  });

  it('should return null if user with email does not
  exist', async () => {
    const mockEmail = 'nonexistent@example.com';

    (prismaService.user.findFirst as
  jest.Mock).mockResolvedValue(null);

    const result = await
  service.findOneByEmail(mockEmail);

```

```
expect(prismaService.user.findFirst).toHaveBeenCalledW
ith({
  where: {
    email: mockEmail,
  },
});
expect(result).toBeNull();
});

it('should throw an exception if there is an error
finding user by email', async () => {
  const mockEmail = 'test@example.com';

  (prismaService.user.findFirst as
jest.Mock).mockRejectedValue(
    new Error('Database error'),
  );

  await
expect(service.findOneByEmail(mockEmail)).rejects.toTh
row(
  'Error finding user',
);

expect(prismaService.user.findFirst).toHaveBeenCalledW
ith({
  where: {
    email: mockEmail,
  },
});
});
});
});
```

```

src\users > src\users.service.spec.ts > describe('userService') callback > beforeEach() callback
1 import { Test, TestingModule } from '@nestjs/testing';
2 import { UsersService } from './users.service';
3 import { PrismaService } from '../core/prisma/prisma.service';
4 import { UtilsService } from '../utils';
5 import { CreateUserDto } from '../dto/create-user.dto';
6 import { UserRole } from '../entities/user.entity';
7
8 // eslint-disable-next-line
9 describe('UsersService', () => {
10   let service: UsersService;
11   let prismaService: Partial<PrismaService>;
12   let utilsService: Partial<UtilsService>;
13
14   beforeEach(async () => {
15     prismaService = {
16       user: {
17         create: jest.fn(),
18         findFirst: jest.fn(),
19         findMany: jest.fn(),
20         delete: jest.fn(),
21         update: jest.fn(),
22       },
23     } as unknown as PrismaService;
24
25     utilsService = {
26       hashPassword: jest.fn(),
27     };
28
29     const module: TestingModule = await Test.createTestingModule({
30       providers: [
31         UsersService,
32         { provide: PrismaService, useValue: prismaService },
33         { provide: UtilsService, useValue: utilsService },
34       ],
35     }).compile();
36
37     service = module.get<UsersService>(UsersService);
38   });
39
40   // eslint-disable-next-line
41   it('should be defined', () => {
42     expect(service).toBeDefined();
43   });
44
45   // eslint-disable-next-line
46   describe('create', () => {
47     // ...
48   });
49
50   // ...
51 });

```

```

src\users > src\users.service.spec.ts > describe('userService') callback > beforeEach() callback
8 describe('UsersService', () => {
9   // ...
10
11   // eslint-disable-next-line
12   it('should be defined', () => {
13     expect(service).toBeDefined();
14   });
15
16   // eslint-disable-next-line
17   describe('create', () => {
18     // ...
19   });
20
21   // ...
22 });

```

```

src\users > src\users.service.spec.ts > describe('userService') callback > beforeEach() callback
8 describe('UsersService', () => {
9   // ...
10
11   // eslint-disable-next-line
12   it('should be defined', () => {
13     expect(service).toBeDefined();
14   });
15
16   // eslint-disable-next-line
17   describe('create', () => {
18     // ...
19   });
20
21   // ...
22 });

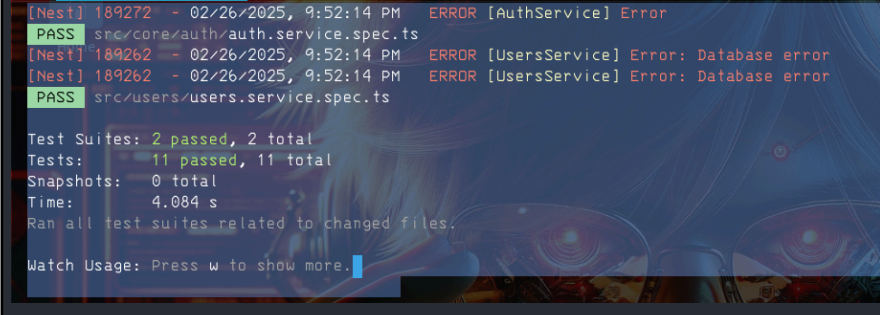
```

## Resultado de la ejecución

```

[Nest] 189272 - 02/26/2025, 9:52:14 PM ERROR
[AuthService] Error
PASS src/core/auth/auth.service.spec.ts
[Nest] 189262 - 02/26/2025, 9:52:14 PM ERROR
[UserService] Error: Database error
[Nest] 189262 - 02/26/2025, 9:52:14 PM ERROR

```

|   |   |
|---|---|
|   | <pre> [UserService] Error: Database error PASS  src/users/users.service.spec.ts  Test Suites: 2 passed, 2 total Tests:      11 passed, 11 total Snapshots:  0 total Time:       4.084 s Ran all test suites related to changed files.  Watch Usage: Press w to show more. </pre>                          |
| <p><b>Lecciones aprendidas y dificultades</b></p> | <p>A diferencia de la creación de usuarios, al obtener datos de la base de datos el moqueado simplifica mucho lo que podemos esperar y los resultados son más previsibles, el manejo de errores generales en estos casos ayuda a poder aislar la información del usuario a la que tenemos desde la consola de desarrollo por lo que se retorna, es decir hay un buen manejo de errores.</p> |

|  |   |
|--|---|
| Nombre                                   | Diego Felipe Solorzano Aponte   |
| Tipo de prueba                           | Unit test   |
| Descripción breve del componente probado | Refresh token, es similar a la función anterior, pero debe generar dos tokens distintos a partir del refresh token, esta funcionalidad es vital en su funcionamiento ya que nos asegura la persistencia en la sesión del usuario y que no existan cierres abruptos  |
| Screenshot del código                    | <pre>import { Test, TestingModule } from '@nestjs/testing'; import { AuthService } from '../auth.service'; import { PrismaService } from '../prisma/prisma.service'; import { UsersService } from '../../users/users.service'; import { JwtService } from '@nestjs/jwt'; import { ConfigService } from '@nestjs/config'; import { UtilsService } from '@app/utils'; import { Payload } from '../types/payload.types'; import { Role } from '../enums/role.enum'; import { UnauthorizedException } from '@nestjs/common';  describe('AuthService', () =&gt; {   let service: AuthService;   let usersService: jest.Mocked&lt;UsersService&gt;;   let jwtService: jest.Mocked&lt;JwtService&gt;;   let configService: jest.Mocked&lt;ConfigService&gt;;   let utilsService: jest.Mocked&lt;UtilsService&gt;;    beforeEach(async () =&gt; {     const mockUsersService = {       findOneByEmail: jest.fn(),       updateUserById: jest.fn(),     };      const mockJwtService = {       signAsync: jest.fn(),       verifyAsync: jest.fn(),     };      const mockPayload: Payload = {       email: 'test@test.com',       sub: '123',       role: Role.ADMIN,     };   });</pre> |

```

const mockConfigService = {
  get: jest.fn(),
};

const mockUtilsService = {
  comparePassword: jest.fn(),
  hashPassword: jest.fn(),
};

const module: TestingModule = await
Test.createTestingModule({
  providers: [
    AuthService,
    { provide: UsersService, useValue:
mockUsersService },
    { provide: JwtService, useValue: mockJwtService
},
    { provide: ConfigService, useValue:
mockConfigService },
    { provide: UtilsService, useValue:
mockUtilsService },
  ],
}).compile();

service = module.get<AuthService>(AuthService);
userService = module.get(UsersService);
jwtService = module.get(JwtService);
configService = module.get(ConfigService);
utilsService = module.get(UtilsService);
});

it('should be defined', () => {
  expect(service).toBeDefined();
});
describe('update token', () => {
  it('should update refresh token hash', async () =>
{
    const mockPayload: Payload = {
      email: 'test@test.com',
      sub: '123',
      role: Role.ADMIN,
    };
    const mockRefreshToken = 'mock-refresh-token';

utilsService.hashPassword.mockResolvedValue('hashed-re
fresh-token');

```

```
usersService.updateUserById.mockResolvedValue(undefined);

    await (service as any).updateRtHash(mockRefreshToken, mockPayload);

expect(utilsService.hashPassword).toHaveBeenCalledWith(mockRefreshToken);

expect(usersService.updateUserById).toHaveBeenCalledWith('123', {
    refreshToken: 'hashed-refresh-token',
});

describe('refreshTokens', () => {
    it('should refresh tokens successfully', async () => {
        const mockRefreshToken = 'valid-refresh-token';
        const mockPayload: Payload = {
            email: 'test@test.com',
            sub: '123',
            role: Role.ADMIN,
        };
        const mockUser = {
            email: 'test@test.com',
            id: '123',
            role: Role.ADMIN,
            refreshToken: 'hashed-refresh-token',
            name: 'Test User',
            phoneNumber: '1234567890',
            password: 'hashedpassword',
            birthDate: new Date(),
            createdAt: new Date(),
            updatedAt: new Date(),
        };
        const mockTokens = {
            access_token: 'new-access-token',
            refresh_token: 'new-refresh-token',
        };

        jwtService.verifyAsync.mockResolvedValueOnce(mockPayload);
```

```
userService.findOneByEmail.mockResolvedValueOnce(mockUser);

utilsService.comparePassword.mockResolvedValueOnce(true);

configService.get.mockReturnValue('test-secret');

jwtService.signAsync.mockResolvedValueOnce('new-access-token');

jwtService.signAsync.mockResolvedValueOnce('new-refresh-token');

utilsService.hashPassword.mockResolvedValue('new-hashe
d-token');

userService.updateUserById.mockResolvedValue(undefined);

    const result = await
service.refreshTokens(mockRefreshToken);

    expect(result).toEqual({
      access_token: 'new-access-token',
      refresh_token: 'new-refresh-token',
    });

expect(jwtService.verifyAsync).toHaveBeenCalled();

expect(userService.findOneByEmail).toHaveBeenCalledWith(
  mockPayload.email,
);

expect(utilsService.comparePassword).toHaveBeenCalledWith(
  mockRefreshToken,
  mockUser.refreshToken,
);

    it('should throw UnauthorizedException when token
verification fails', async () => {
```



```

39 expect(result).toEqual({
40   access token: 'new-access-token',
41   refresh token: 'new-refresh-token',
42 })
43 expect(jwtService.verifyAsync).toHaveBeenCalled()
44 expect(userService.findOneByEmail).toHaveBeenCalled()
45
46 // For examples refer to the Github
47 (method).jest.mockInstance(Promise.resolve, {
48   [password: string]: hash: string, unknown: 6 (password:
49   string, hash: string) => Promise.resolve(), tokaveBeenCalledWith(string, string)(params: 0: string, params: 1: string):
50   void
51 })
52
53 Ensure that a mock function is called with specified arguments
54
55 Optionally, you can provide a type for the expected arguments via a generic. Note that the type must be either an array or a tuple.

```

## AuthService

## Generate token

(2 ms)

update token

---

|   |   |
|---|---|
|   | <pre> ✓ should update refresh token hash (1 ms) refreshTokens   ✓ should refresh tokens successfully (1 ms)   ✓ should throw UnauthorizedException when token verification fails (7 ms)  Test Suites: 1 passed, 1 total Tests:       5 passed, 5 total Snapshots:  0 total Time:        3.097 s Ran all test suites related to changed files.  Watch Usage: Press w to show more. PASS src/core/auth/auth.service.spec.ts AuthService   ✓ should be defined (7 ms)   Generate token     ✓ should return access_token and refresh_token (2 ms)   update_token     ✓ should update refresh token hash (1 ms)   refreshTokens     ✓ should refresh tokens successfully (1 ms)     ✓ should throw UnauthorizedException when token verification fails (7 ms)  Test Suites: 1 passed, 1 total Tests:       5 passed, 5 total Snapshots:  0 total Time:        3.097 s Ran all test suites related to changed files.  Watch Usage: Press w to show more. </pre> |
| <p><b>Lecciones aprendidas y dificultades</b></p> | <p>Las funcionalidades de los tokens y usuarios al ser moldeados se vuelven manipulables a las necesidades de quien testea, es decir, es muy fácil lograr que el output sea el deseado ya que a través de las librerías tenemos un control absoluto, en este caso lo que hicimos fue verificar si funcionaba la validación de usuario con la destrucción del jwt refresh token que teníamos almacenado en la DB, y en caso de que no va a darnos un error.</p> <p>También es apreciable que en TS la test suite es el módulo que estamos probando, y los test son los individuales sin importar a qué función pertenece.</p>  |

|  |  |
|--|--|
| Nombre                                   | Juan David Chacon Munoz  |
| Tipo de prueba                           | Unitaria   |
| Descripción breve del componente probado | Se evalúa la creación de usuarios de manera correcta, que los valores de entrada sean los adecuados y que se cree un usuario correctamente,  |
| <div>Screenshot del código</div>         | <pre>import { Test, TestingModule } from '@nestjs/testing'; import { UsersService } from './users.service'; import { PrismaService } from '../core/prisma/prisma.service'; import { UtilsService } from '@app/utils'; import { CreateUserDto } from './dto/create-user.dto'; import { UserRole } from './entities/user.entity';  describe('UsersService', () =&gt; {   let service: UsersService;   let prismaService: Partial&lt;PrismaService&gt;;   let utilsService: Partial&lt;UtilsService&gt;;    beforeEach(async () =&gt; {     prismaService = {       user: {         create: jest.fn(),         findFirst: jest.fn(),         findMany: jest.fn(),         delete: jest.fn(),         update: jest.fn(),       },     } as unknown as PrismaService;      utilsService = {       hashPassword: jest.fn(),     };      const module: TestingModule = await Test.createTestingModule({   providers: [     UsersService,     { provide: PrismaService, useValue: prismaService },     { provide: UtilsService, useValue: utilsService },   ], }).compile();</pre> |

```

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

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

  describe('create', () => {
    const mockCreateUserDto: CreateUserDto = {
      id: 1,
      email: 'test@example.com',
      phoneNumber: '1234567890',
      name: 'Test User',
      role: UserRole.USER,
      password: 'password',
      birthDate: '2000-01-01',
    };

    it('should create a user successfully', async () => {
      const hashedPassword = 'hashed_password';
      const expectedDate = new Date('2000-01-01');

      const mockCreatedUser = {
        id: '1',
        email: 'test@example.com',
        phoneNumber: '1234567890',
        name: 'Test User',
        role: 'USER',
        password: hashedPassword,
        birthDate: expectedDate,
      };

      (utilsService.hashPassword as
jest.Mock).mockResolvedValue(
        hashedPassword,
      );
      (prismaService.user.create as
jest.Mock).mockResolvedValue(
        mockCreatedUser,
      );

      const result = await
service.createUser(mockCreateUserDto);

      expect(prismaService.user.create).toHaveBeenCalledWith

```

```
({
  data: {
    id: mockCreateUserDto.id.toString(),
    email: mockCreateUserDto.email,
    phoneNumber: mockCreateUserDto.phoneNumber,
    name: mockCreateUserDto.name,
    role: mockCreateUserDto.role,
    password: hashedPassword,
    birthDate: expectedDate,
  },
});
expect(result).toEqual(mockCreatedUser);
});

it('should throw an exception if there is an error
during user creation', async () => {
  const hashedPassword = 'hashed_password';

  (utilsService.hashPassword as
jest.Mock).mockResolvedValue(
    hashedPassword,
  );
  (prismaService.user.create as
jest.Mock).mockRejectedValue(
    new Error('Database error'),
  );

  await
expect(service.createUser(mockCreateUserDto)).rejects.
toThrow(
  'Error creating user',
);

expect(utilsService.hashPassword).toHaveBeenCalledWith(
(
  mockCreateUserDto.password,
));
});
});
});
```

```

1 import { Test, TestingModule } from '@nestjs/testing';
2 import { UserService } from './users.service';
3 import { PrismaService } from '../core/prisma/prisma.service';
4 import { UtilsService } from '@app/utils';
5 import { CreateUserDto } from '../dto/create-user.dto';
6 import { UserRole } from '../entities/user.entity';
7
8 describe('UserService', () => {
9   let service: UserService;
10   let prismaService: Partial<PrismaService>;
11   let utilsService: Partial<UtilsService>;
12
13   beforeEach(async () => {
14     prismaService = {
15       user: {
16         create: jest.fn(),
17         findFirst: jest.fn(),
18         findMany: jest.fn(),
19         delete: jest.fn(),
20         update: jest.fn(),
21       },
22     } as unknown as PrismaService;
23
24     utilsService = {
25       hashPassword: jest.fn(),
26     };
27
28     const module: TestingModule = await Test.createTestingModule({
29       providers: [
30         UserService,
31         { provide: PrismaService, useValue: prismaService },
32         { provide: UtilsService, useValue: utilsService },
33       ],
34     }).compile();
35
36     service = module.get<UserService>(UserService);
37   });
38
39   it('should be defined', () => {
40     expect(service).toBeDefined();
41   });
42
43   describe('create', () => {
44     const mockCreateUserDto: CreateUserDto = {
45       id: 1,
46       email: 'test@example.com',
47       phoneNumber: '1234567890',
48       name: 'Test User',
49       role: UserRole.USER,
50       password: 'password',
51       birthDate: '2000-01-01',
52     };
53
54     it('should create a user successfully', async () => {
55       const hashedPassword = 'hashed_password';
56       const expectedDate = new Date('2000-01-01');
57
58       const mockCreatedUser = {
59         id: 1,
60         email: 'test@example.com',
61         phoneNumber: '1234567890',
62         name: 'Test User',
63         role: 'USER',
64         password: hashedPassword,
65         birthDate: expectedDate,
66       };
67
68       (utilsService.hashPassword as jest.Mock).mockResolvedValue(
69         hashedPassword,
70       );
71       (prismaService.user.create as jest.Mock).mockResolvedValue(
72         mockCreatedUser,
73       );
74
75       const result = await service.createUser(mockCreateUserDto);
76
77       expect(prismaService.user.create).toHaveBeenCalledWith({
78         data: {
79           id: mockCreateUserDto.id.toString(),
80           email: mockCreateUserDto.email,
81           phoneNumber: mockCreateUserDto.phoneNumber,
82           name: mockCreateUserDto.name,
83           role: mockCreateUserDto.role,
84           password: hashedPassword,
85           birthDate: expectedDate,
86         },
87       });
88
89       expect(result).toEqual(mockCreatedUser);
90     });
91
92     it('should throw an exception if there is an error during user creation', async () => {
93       const hashedPassword = 'hashed_password';
94
95       (utilsService.hashPassword as jest.Mock).mockResolvedValue(
96         hashedPassword,
97       );
98       (prismaService.user.create as jest.Mock).mockRejectedValue(
99         new Error('Database error'),
100       );
101
102       await expect(service.createUser(mockCreateUserDto)).rejects.toThrow(
103         'Error creating user',
104       );
105
106       expect(utilsService.hashPassword).toHaveBeenCalled();
107     });
108   });
109 });

```

Resultado de la  
ejecución

[Nest] 184742 - 02/26/2025, 9:40:51 PM ERROR  
[UserService] Error: Database error  
PASS src/users/users.service.spec.ts

|   |   |
|---|---|
|   | <pre> [Nest] 184743 - 02/26/2025, 9:40:51 PM ERROR [AuthService] Error PASS src/core/auth/auth.service.spec.ts  Test Suites: 2 passed, 2 total Tests:      8 passed, 8 total Snapshots:  0 total Time:       4.235 s Ran all test suites related to changed files.  Watch Usage &gt; Press a to run all tests. &gt; Press f to run only failed tests. &gt; Press p to filter by a filename regex pattern. &gt; Press t to filter by a test name regex pattern. &gt; Press q to quit watch mode. &gt; Press Enter to trigger a test run.  [Nest] 184742 - 02/26/2025, 9:40:51 PM ERROR [UserService] Error: Database error PASS src/users/users.service.spec.ts [Nest] 184743 - 02/26/2025, 9:40:51 PM ERROR [AuthService] Error PASS src/core/auth/auth.service.spec.ts  Test Suites: 2 passed, 2 total Tests:      8 passed, 8 total Snapshots:  0 total Time:       4.235 s Ran all test suites related to changed files.  Watch Usage &gt; Press a to run all tests. &gt; Press f to run only failed tests. &gt; Press p to filter by a filename regex pattern. &gt; Press t to filter by a test name regex pattern. &gt; Press q to quit watch mode. &gt; Press Enter to trigger a test run. </pre> |
| <p><b>Lecciones aprendidas y dificultades</b></p> | <p>El uso de ORM cuando son usados directamente en funciones vuelve exageradamente complejos los test, ya que en caso de haber relaciones es necesario ponerlas en el esquema del ORM, lo que puede volverse complejo si es una primera vez, y la poca documentación de estos casos lo vuelve aún más doloroso, esto también por el tipado de typescript, que vuelve el lenguaje en uno fuertemente tipado en estos casos. Finalmente, la técnica de hash de contraseña puede resultar bastante pesada para probarlo en un test, solamente se puede probar la comparación.</p>  |

## Reflexión grupal

Solo uno de nosotros tenía experiencia con la implementación de tests, así que en un primer acercamiento para intentar resolver el taller se nos dificultó aprender a usar las herramientas de testing y asegurarnos de que las pruebas realmente cubrían los casos más relevantes tomó más tiempo del que esperábamos, sin embargo fue una excelente oportunidad para afianzar ciertas soft skills que son relevantes en un entorno de desarrollo real, como por ejemplo la comunicación asertiva, el trabajo en equipo y el seguimiento de instrucciones. Finalmente gracias a la constante retroalimentación grupal y la

guía de nuestro compañero más experimentado Juan Loaiza, logramos cubrir de una manera más general estos vacíos conceptuales y enriquecer nuestro proceso de aprendizaje sobre el testing.