Informe Testing

Presentado por Grupo 6:

Ana María González Hernández - anagonzalezhe@unal.edu.co
Daniel Felipe Soracipa - dsoracipa@unal.edu.co
Juan José Medina Guerrero - jmedinagu@unal.edu.co
Samuel Josué Vargas Castro - samvargasca@unal.edu.co

Profesor:
Oscar Eduardo Alvarez Rodriguez
oalvarezr@unal.edu.co

Febrero 27 de 2025

Universidad Nacional de Colombia
Facultad de Ingeniería
Departamento de Ingeniería de Sistemas e Industrial
2025

Descripción general de la aplicación

La aplicación consiste en desarrollar una plataforma de e-commerce diseñada específicamente para conectar a los usuarios (principalmente estudiantes y miembros de la comunidad universitaria) con las chazas de la universidad que suelen ofrecer productos dentro del campus, como comida, artículos de segunda mano, material académico, entre otros. Permitiendo que los estudiantes puedan tomar mejores decisiones sobre donde comprar y de la misma manera que todas las chazas de la universidad tengan un nivel similar de exposición en cuanto a los que pueden ofrecer.

Resumen de los tests

<table>
<tr><th colspan="4">Testing Backend</th></tr>
<tr><td>Herramienta o framework usado</td><td colspan="3">Para las pruebas se utilizó Jest, una biblioteca diseñada para la ejecución de tests en aplicaciones JavaScript, que se integra fácilmente con NestJS. Jest permite la escritura de pruebas unitarias en Node.js y su flexibilidad le permite adaptarse a distintos entornos y frameworks. Adicionalmente, se empleó @nestjs/testing para configurar el entorno de pruebas y gestionar la inyección de dependencias mediante el uso de mocks.</td></tr>
<tr><th colspan="4">Tests</th></tr>
<tr><td rowspan="3">Test #1</td><th>Integrante</th><th>Tipo de Prueba</th><th>Descripción</th></tr>
<tr><td>Daniel Felipe Soracipa</td><td>Test Unitario</td><td>Se realiza del test del use-case que permite crear una chaza, este test verifica más específicamente que se cree una chaza correctamente.</td></tr>
<tr><td colspan="3">Estructura general para escribir un test</td></tr>
<tr><td colspan="3">

```
import { Test, TestingModule } from '@nestjs/testing';
import { CreateChazaUseCase } from '../use-cases/create-chaza.use-case';

const mockChazaRepository = {
  createChaza: jest.fn(),
};

const mockUserRepository = {
  findUserById: jest.fn(),
};

describe('CreateChazaUseCase', () => {
  let useCase: CreateChazaUseCase;

  beforeEach(async () => {
    const module: TestingModule = await Test.createTestingModule({
      providers: [
        CreateChazaUseCase,
        { provide: 'IChazaRepository', useValue: mockChazaRepository },
        { provide: 'IUserRepository', useValue: mockUserRepository },
      ],
    }).compile();

    useCase = module.get<CreateChazaUseCase>(CreateChazaUseCase);
  });

  afterEach(() => {
    jest.clearAllMocks();
  });
```

</td></tr>
</table>

<table>
<tr><td rowspan="3"></td><td>Prueba unitaria específica</td></tr>
<tr><td>

```
it('debería crear una chaza exitosamente', async () => {
  const mockUser = { id: '123' };
  mockUserRepository.findUserById.mockResolvedValue(mockUser);
  mockChazaRepository.createChaza.mockResolvedValue(undefined);

  const result = await useCase.execute('Chaza 1', 'Desc', 'Ubicacion', 1,
  '123');

  expect(mockUserRepository.findUserById).toHaveBeenCalledWith('123');
  expect(mockChazaRepository.createChaza).toHaveBeenCalled();
  expect(result).toEqual({
    chaza: expect.objectContaining({
      nombre: 'Chaza 1',
      descripcion: 'Desc',
      ubicacion: 'Ubicacion',
      foto_id: 1,
      id_usuario: '123',
    }),
  });
});
```

</td></tr>
<tr><td>Resultado de la ejecución</td></tr>
<tr><td></td><td>

```
C:\Users\DANIEL\Documents\UNI\sem 6\ingesoft\proyecto\Proyecto_IngeSoft\Proyecto\Backend>npx jest --config
st.config.ts --verbose

 RUNS  src/chaza/application/use-cases/create-chaza.use-case.spec.ts
 RUNS  src/user/application/use-cases/register-user.use-cas.spec.ts

Test Suites: 0 of 2 total
Tests:       0 total
Snapshots:   0 total
Time:        3 s, estimated 15 s
```

```
 PASS  src/chaza/application/use-cases/create-chaza.use-case.spec.ts (18.798 s)
  CreateChazaUseCase
    √ debería crear una chaza exitosamente (39 ms)
```

</td></tr>
<tr><td>Test #2</td><td>
<table>
<tr><th>Integrante</th><th>Tipo de Prueba</th><th>Descripción</th></tr>
<tr><td>Ana María González Hernández</td><td>Test Unitario</td><td>Se realiza del test del caso de uso relacionado a la creación de un usuario que no hace parte de la base de datos, es decir, un nuevo usuario.El test verifica la correcta creación del usuario.</td></tr>
</table>
</td></tr>
<tr><td></td><td>Estructura general para escribir un test</td></tr>
</table>

Proyecto_IngeSoft > Proyecto > Backend > src > user > application > use-cases > register-user.use-cas.spec.ts > describe

```ts
import { Test, TestingModule } from '@nestjs/testing';
import { RegisterUserUseCase } from '../use-cases/register-user.use-case';

const mockUserRepository = {
  findUserByEmail: jest.fn(),
  createUser: jest.fn(),
};

const mockAuthService = {
  hashPassword: jest.fn(),
  generateToken: jest.fn(),
};

describe('RegisterUserUseCase', () => {
  let useCase: RegisterUserUseCase;

  beforeEach(async () => {
    const module: TestingModule = await Test.createTestingModule({
      providers: [
        RegisterUserUseCase,
        { provide: 'IUserRepository', useValue: mockUserRepository },
        { provide: 'AuthService', useValue: mockAuthService },
      ],
    }).compile();

    useCase = module.get<RegisterUserUseCase>(RegisterUserUseCase);
  });

  afterEach(() => {
    jest.clearAllMocks();
  });
```

Prueba unitaria específica

```ts
  it('debería registrar un usuario correctamente', async () => {
    const mockEmail = 'test@example.com';
    const mockPassword = 'password123';
    const hashedPassword = 'hashedPassword123';
    const mockUserId = '12345';  // Agregar un ID simulado
    const mockToken = 'mockToken123';

    mockUserRepository.findUserByEmail.mockResolvedValue(null);
    mockAuthService.hashPassword.mockResolvedValue(hashedPassword);

    // Simulamos el usuario con un ID
    mockUserRepository.createUser.mockImplementation((user) => {
      user.id = mockUserId; // Se asigna manualmente el ID
      return Promise.resolve();
    });

    mockAuthService.generateToken.mockReturnValue(mockToken);

    const result = await useCase.execute(mockEmail, mockPassword);

    expect(mockUserRepository.findUserByEmail).toHaveBeenCalledWith(mockEmail);
    expect(mockAuthService.hashPassword).toHaveBeenCalledWith(mockPassword);
    expect(mockUserRepository.createUser).toHaveBeenCalled();
    expect(mockAuthService.generateToken).toHaveBeenCalledWith(mockUserId);
    expect(result).toEqual({
      user: {
        email: mockEmail,
        userid: mockUserId, // Ahora si deberia estar definido
      },
      token: mockToken,
    });
  });
```

Resultado de la ejecución

```
PASS  src/user/application/use-cases/register-user.use-cas.spec.ts (6.005 s)
  RegisterUserUseCase
    ✓ debería registrar un usuario correctamente (20 ms)
```

<table>
<tr><td rowspan="5">Test #3</td><td>Integrante</td><td>Tipo de Prueba</td><td>Descripción</td></tr>
<tr><td>Juan José Medina Guerrero</td><td>Test Unitario</td><td>Se realiza el test del caso de uso que ocurre cuando un usuario no existe en la base de datos en la creación de una chaza, lo cual nos debería arrojar un error, lo cual es correcto, ya que no se puede vincular una chaza a un usuario que no existe.</td></tr>
<tr><td colspan="3">Estructura general para escribir un test</td></tr>
<tr><td colspan="3">

```
const mockChazaRepository = {
  createChaza: jest.fn(),
};

const mockUserRepository = {
  findUserById: jest.fn(),
};

describe('CreateChazaUseCase', () => {
  let useCase: CreateChazaUseCase;

  beforeEach(async () => {
    const module: TestingModule = await Test.createTestingModule({
      providers: [
        CreateChazaUseCase,
        { provide: 'IChazaRepository', useValue: mockChazaRepository },
        { provide: 'IUserRepository', useValue: mockUserRepository },
      ],
    }).compile();

    useCase = module.get<CreateChazaUseCase>(CreateChazaUseCase);
  });

  afterEach(() => {
    jest.clearAllMocks();
  });
```

</td></tr>
<tr><td colspan="3">Prueba unitaria específica</td></tr>
<tr><td colspan="3">

```
it('debería lanzar un error si el usuario no existe', async () => {
    mockUserRepository.findUserById.mockResolvedValue(null);

    await expect(
      useCase.execute('Chaza 1', 'Desc', 'Ubicacion', 1, '123')
    ).rejects.toThrow('User not found');

    expect(mockUserRepository.findUserById).toHaveBeenCalledWith('123');
    expect(mockChazaRepository.createChaza).not.toHaveBeenCalled();
  });
```

</td></tr>
<tr><td colspan="3">Resultado de la ejecución</td></tr>
<tr><td colspan="3">

```
 PASS  src/chaza/application/use-cases/create-chaza.use-case.spec.ts (25.84 s)
  CreateChazaUseCase
    √ debería crear una chaza exitosamente (21 ms)
    √ debería lanzar un error si el usuario no existe (13 ms)
```

</td></tr>
<tr><td colspan="4">Testing Frontend</td></tr>
<tr><td>Herramienta o framework usado</td><td colspan="3">El test está escrito en Flutter Test, un framework de pruebas unitarias para Flutter que permite validar el comportamiento de funciones y widgets de la aplicación sin necesidad de ejecutarla en un dispositivo. Se usa group para organizar las pruebas y test para definir casos específicos.</td></tr>
</table>

Test #4

Integrante	Tipo de Prueba	Descripción
Samuel Josué Vargas Castro	Test Unitario	El componente probado es la función isEmailValid, que valida si un correo electrónico tiene un formato correcto. Se utiliza en la autenticación para asegurar que los usuarios ingresen una dirección de email válida antes de continuar con el proceso de inicio de sesión o registro.

Estructura general para escribir un test

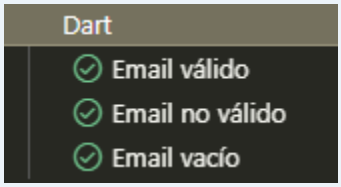
```
void main() {
  Run | Debug
  group("Grupo de tests", () {
    Run | Debug
    test("Caso de prueba 1", () {
      // Código de prueba
      const value1 = 1;
      const value2 = 2;

      // Verificación
      expect(value1 + value2, 3);
    });

    Run | Debug
    test("Caso de prueba 2", () {
      // Código de prueba
      const value1 = 2;
      const value2 = 2;

      // Verificación
      expect(value1 + value2, 4);
    });
  });
}
```

	Prueba unitaria específica

```dart
void main() {
  Run | Debug
  group("Test de email", () {
    Run | Debug
    test("Email válido", () {
      const testEmail = "samuevarga@gmail.com";

      expect(isEmailValid(testEmail), true);
    });

    Run | Debug
    test("Email no válido", () {
      const testEmail = "samuevarga@gmail";

      expect(isEmailValid(testEmail), false);
    });

    Run | Debug
    test("Email vacío", () {
      const testEmail = "";

      expect(isEmailValid(testEmail), false);
    });
  });
}
```

Resultado de la ejecución

```
PS C:\Users\sjvc5\Proyectos Programacion\Proyecto_IngeSoft\Proyecto\Frontend\chazapp> flutter test
00:04 +4: All tests passed!
```



Lecciones aprendidas y dificultades

Nos dimos cuenta de que los tests no solo sirven para verificar si algo funciona como en el caso de los métodos de los use-cases, sino también para detectar problemas en la estructura del código. Una de las mayores dificultades fue lidiar con las dependencias y los mocks, especialmente cuando Jest nos tiraba errores porque no encontraba módulos o los repositorios simulados no se comportaban como esperábamos, esto que nos llevó a tener que crear el archivo de configuración de jest (jest.config.ts) e incluso modificar cosas del package.json porque sin este y algunas configuraciones extra no reconocía los imports cuando empezaban por “src/” y mostraba test fallido aunque el Back-end corriera correctamente. Después de varios intentos y correcciones, entendimos que testear también nos ayuda a escribir mejor código desde el inicio. Además de que si los datos que esperamos en el test no coinciden exactamente con los generados, las pruebas fallan, incluso si todo parece estar bien en el código.