



**Carrera de
Software**

Universidad de las Fuerzas Armadas ESPE

Departamento: Ciencias de la Computación

Carrera: Ingeniería de Software

Taller académico N^o: 3

Asignatura: Análisis y Diseño de Software

Apellidos y nombres de los estudiantes:

Coloma Kevin
Granda Carlos
Moreira Erick

NRC: 23305

Fecha de realización: 30/7/2025

1. Objetivo

Implementar y documentar los casos de pruebas unitarias en nuestro proyecto de La Panadería “La Italiana”, de esta manera se busca asegurar los distintos componentes del sistema estén funcionando de la manera correcta, buscando detectar fallas mediante la aparición de errores. La realización de estas pruebas nos van a permitir validar la lógica de negocio, verificar la funcionalidad de las rutas de la API, y evitar fallos antes de presentar el software terminado al cliente.

2. Herramientas Utilizadas

- **Jest:** Jest es un framework de testing para JavaScript centrado en la simplicidad, funciona con proyectos que utilizan: Babel, TypeScript, Node, React, Angular, Vue y más. Es ideal para probar las funciones y la lógica de los controladores, middlewares y los utils. También su elección se debe a que es fácil de configurar y tiene un reporte para generar documentación de pruebas.
- **Supertest:** Supertest es una biblioteca de Node.js que nos va a servir para probar las rutas HTTP que se encuentran en nuestra carpeta routes, también para conocer cómo se comportan los controladores ante peticiones reales, ya que esta herramienta es utilizada principalmente para la integración o pruebas integrales de aplicaciones Node.js y sus API.

3. Desarrollo

Módulo o Componente	Entrada de Prueba	Resultado Esperado	Resultado Obtenido (en caso de ejecución)	Estado de la prueba (aprobada/fallida)
GET /productos	Request GET sin parámetros	200 OK, lista de productos	200 OK, lista recibida	Aprobada
GET /productos/buscar	buscar = pan	200 OK, lista filtrada (puede ser vacía en caso que no haya el producto)	200 OK, lista recibida	Aprobada

A continuación se muestra el código para la realización de las pruebas

```
productos.test.js X
backend-panaderia > tests > productos.test.js > ...
1  const request = require('supertest');
2  const { app } = require('../src/app');
3
4  describe('Productos - Endpoints API', () => {
5    it('GET /productos debería retornar todos los productos (status 200)', async () => {
6      const res = await request(app).get('/productos');
7
8      expect(res.statusCode).toBe(200);
9      expect(Array.isArray(res.body)).toBe(true);
10   });
11
12   it('GET /productos/buscar?buscar=pan debería funcionar (status 200)', async () => {
13     const res = await request(app).get('/productos/buscar?buscar=pan');
14
15     expect(res.statusCode).toBe(200);
16     expect(Array.isArray(res.body)).toBe(true);
17   });
18 });
19
```

Y el resultado obtenido, donde se observa que en este caso ambas pruebas pasaron el test

```
PS D:\7mo\Pruebas\3PARCIAL\ProyectoPanaderia\backend-panaderia> npm test

> backend-panaderia@1.0.0 test
> jest

console.log
  ✂ Cargando ► src/app.js
    at Object.log (src/app.js:1:9)

console.log
  🍌 [auth.controller] cargado
    at Object.log (src/controllers/auth.controller.js:1:9)

console.log
  🚚 GET /productos body= undefined
    at log (src/app.js:20:11)

console.log
  🚚 GET /productos/buscar?buscar=pan body= undefined
    at log (src/app.js:20:11)

PASS tests/productos.test.js
  Productos - Endpoints API
    ✓ GET /productos debería retornar todos los productos (status 200) (100 ms)
    ✓ GET /productos/buscar?buscar=pan debería funcionar (status 200) (18 ms)

Test Suites: 1 passed, 1 total
Tests: 2 passed, 2 total
Snapshots: 0 total
Time: 1.401 s
Ran all test suites.
```

Módulo o Componente	Entrada de Prueba	Resultado Esperado	Resultado Obtenido (en caso de ejecución)	Estado de la prueba (aprobada/fallida)
POST /productos	Campos del formulario + imagen + token admin	201 Created, objeto con id y nombre	id y nombre 201 Created, producto insertado	Aprobada

A continuación se muestra el código para la realización de la prueba, donde se verifica con la generación del token a través de las credenciales correspondientes y los campos para agregar un producto

```

productosPOST.test.js x
backend-panaderia > tests > productosPOST.test.js > describe('Productos - POST /productos') callback > it('debería crear un producto nuevo con imagen') callback > res
1  const request = require('supertest');
2  const path = require('path');
3  const { app } = require('../src/app');
4
5  let token;
6
7  beforeAll(async () => {
8    const res = await request(app)
9      .post('/auth/login')
10     .send({ usuario: 'admin', contrasena: '1234' });
11
12    token = res.body.token;
13  });
14
15  describe('Productos - POST /productos', () => {
16    it('debería crear un producto nuevo con imagen', async () => {
17      const res = await request(app)
18        .post('/productos')
19        .set('Authorization', `Bearer ${token}`)
20        .field('nombre', 'Pan test')
21        .field('descripcion', 'Este es un pan de prueba')
22        .field('precio', 1.5)
23        .field('fecha_hora_salida', new Date().toISOString())
24        .field('fecha_hora_expedicion', new Date().toISOString())
25        .field('stock', 10)
26        .field('categoria_id', 1)
27        .attach('foto', path.join(__dirname, 'assets', 'test-img.jpg'));
28
29      expect(res.statusCode).toBe(201);
30      expect(res.body).toHaveProperty('id');
31      expect(res.body.nombre).toBe('Pan test');
32    });
33  });
34

```

Y el resultado obtenido, donde se observa que en este caso ambas pruebas pasaron el test, agregando el producto de manera correcta

PASS tests/productosPOST.test.js

• Console

console.log

🔗 Cargando ▶ src/app.js

at Object.log (src/app.js:1:9)

console.log

👉 [auth.controller] cargado

at Object.log (src/controllers/auth.controller.js:1:9)

console.log

🔗 POST /auth/login body= { usuario: 'admin', contrasena: '1234' }

🔗 POST /auth/login body= { usuario: 'admin', contrasena: '1234' }

at log (src/app.js:20:11)

console.log

👉 [login] body recibido: { usuario: 'admin', contrasena: '1234' }

at log (src/app.js:20:11)

console.log

👉 [login] body recibido: { usuario: 'admin', contrasena: '1234' }

at log (src/controllers/auth.controller.js:41:11)

console.log

console.log

👉 [login] body recibido: { usuario: 'admin', contrasena: '1234' }

at log (src/controllers/auth.controller.js:41:11)

console.log

at log (src/controllers/auth.controller.js:41:11)

console.log

🔗 POST /productos body= undefined

Pan test



\$1.50

Stock: 10

Ver más

Añadir al carrito

Módulo o Componente	Entrada de Prueba	Resultado Esperado	Resultado Obtenido (en caso de ejecución)	Estado de la prueba (aprobada/fallida)
GET /usuarios	Token admin	200 OK, lista de usuarios	200 OK, lista de usuarios	Aprobada
PUT /usuarios/:id	ID 1, body con nuevo nombre y rol	200 OK si existe, 404 sí no	200 o 404	Aprobada
DELETE /usuarios/:id	Usuario de prueba creado en el test	200 OK con mensaje de eliminación	200 OK, usuario eliminado	Aprobada

A continuación se muestra el código para la realización de las pruebas

```

usuarios.test.js X
backend-panaderia > tests > usuarios.test.js > beforeAll() callback
1  const request = require('supertest');
2  const { app } = require('../src/app');
3
4  let token;
5
6  beforeAll(async () => {
7    const res = await request(app)
8      .post('/auth/login')
9      .send({ usuario: 'admin', contrasena: '1234' });
10
11    token = res.body.token;
12  });
13
14  describe('Usuarios - Endpoints protegidos', () => {
15    it('GET /usuarios debería retornar lista de usuarios (solo admin)', async () => {
16      const res = await request(app)
17        .get('/usuarios')
18        .set('Authorization', `Bearer ${token}`);
19
20      expect(res.statusCode).toBe(200);
21      expect(Array.isArray(res.body)).toBe(true);
22    });
23
24    it('PUT /usuarios/:id debería actualizar usuario', async () => {
25      const res = await request(app)
26        .put('/usuarios/3') // ⚠ Asegúrate que el ID 3 exista
27        .set('Authorization', `Bearer ${token}`)
28        .send({
29          usuario: 'admin_actualizado',
30          rol: 'admin'
31        });
32
33      expect([200, 404]).toContain(res.statusCode);
34      if (res.statusCode === 200) {
35        expect(res.body.usuario).toBe('admin_actualizado');
36      } else {
37        console.log('⚠ Usuario no encontrado para actualizar');
38      }
39    });
40
41    it('DELETE /usuarios/:id debería eliminar un usuario', async () => {
42      // Crear usuario temporal para eliminar
43      const nuevo = await request(app)
44        .post('/auth/register') // asumimos que existe esta ruta
45        .send({ usuario: 'tempuser', contrasena: '1234', rol: 'usuario' });

```

```
usuarios.test.js X
backend-panaderia > tests > usuarios.test.js > beforeAll() callback
14 describe('Usuarios - Endpoints protegidos', () => {
41   it('DELETE /usuarios/:id debería eliminar un usuario', async () => {
43     const nuevo = await request(app)
45     .send({ usuario: 'tempuser', contrasena: '1234', rol: 'usuario' });
46
47     const nuevoId = nuevo.body.usuario.id;
48
49     const res = await request(app)
50     .delete(`/usuarios/${nuevoId}`)
51     .set('Authorization', `Bearer ${token}`);
52
53     expect([200, 404]).toContain(res.statusCode);
54     if (res.statusCode === 200) {
55       expect(res.body.message).toBe('Usuario eliminado');
56     } else {
57       console.log('⚠ Usuario ya no existía');
58     }
59   });
60 }
61
```

Y el resultado obtenido, donde se observa que en este caso las pruebas pasaron el test, obteniendo todos los usuarios mediante la petición GET, editando un usuario y su rol mediante la petición PUT a través de su id, y finalmente eliminando este usuario de prueba con la petición DELETE nuevamente a través de su id.

```
PASS tests/usuarios.test.js
• Console

console.log
  ✖ Cargando ▶ src/app.js
    at Object.log (src/app.js:1:9)

console.log
  ⚡ [auth.controller] cargado
    at Object.log (src/controllers/auth.controller.js:1:9)

console.log
  📡 POST /auth/login body= { usuario: 'admin', contrasena: '1234' }
    at log (src/app.js:20:11)

console.log
  📡 [login] body recibido: { usuario: 'admin', contrasena: '1234' }
    at log (src/controllers/auth.controller.js:41:11)

console.log
  📡 GET /usuarios body= undefined
    at log (src/app.js:20:11)

console.log
  📡 PUT /usuarios/3 body= { usuario: 'admin_actualizado', rol: 'admin' }
    at log (src/app.js:20:11)

console.log
  📡 POST /auth/register body= { usuario: 'tempuser', contrasena: '1234', rol: 'usuario' }
    at log (src/app.js:20:11)

console.log
    at log (src/app.js:20:11)

console.log
  📡 PUT /usuarios/3 body= { usuario: 'admin_actualizado', rol: 'admin' }
    at log (src/app.js:20:11)

console.log
  📡 POST /auth/register body= { usuario: 'tempuser', contrasena: '1234', rol: 'usuario' }
```

```

    at log (src/app.js:20:11)

console.log
  🔥 PUT /usuarios/3 body= { usuario: 'admin_actualizado', rol: 'admin' }

    at log (src/app.js:20:11)

console.log
  🔥 POST /auth/register body= { usuario: 'tempuser', contrasena: '1234', rol: 'usuario' }

    at log (src/app.js:20:11)

console.log
  🔥 [register] body recibido: { usuario: 'tempuser', contrasena: '1234', rol: 'usuario' }
    at log (src/app.js:20:11)

console.log
  🔥 POST /auth/register body= { usuario: 'tempuser', contrasena: '1234', rol: 'usuario' }

    at log (src/app.js:20:11)

console.log
  🔥 [register] body recibido: { usuario: 'tempuser', contrasena: '1234', rol: 'usuario' }
    at log (src/controllers/auth.controller.js:10:11)
    at log (src/app.js:20:11)

console.log
  🔥 [register] body recibido: { usuario: 'tempuser', contrasena: '1234', rol: 'usuario' }
    at log (src/controllers/auth.controller.js:10:11)

  🔥 [register] body recibido: { usuario: 'tempuser', contrasena: '1234', rol: 'usuario' }
    at log (src/controllers/auth.controller.js:10:11)

    at log (src/controllers/auth.controller.js:10:11)

console.log
  🔥 DELETE /usuarios/9 body= undefined

console.log
  🔥 DELETE /usuarios/9 body= undefined

    at log (src/app.js:20:11)

```

Módulo o Componente	Entrada de Prueba	Resultado Esperado	Resultado Obtenido (en caso de ejecución)	Estado de la prueba (aprobada/fallida)
GET /categorias	Sin token	200 OK, lista de categorías	200 OK	Aprobada
POST /categorias	Nombre + desc + imagen (con token admin)	201 Created, objeto con id	201 Created	Aprobada
PUT /categorias/:id	ID existente + nuevos valores + imagen	200 OK, categoría modificada	200 OK	Aprobada
DELETE /categorias/:id	DELETE /categorias/:id	200 OK, categoría eliminada	200 OK	Aprobada

A continuación se muestra el código para la realización de las pruebas

```
categorias.test.js X
backend-panaderia > tests > categorias.test.js > beforeAll() callback > res
1  const request = require('supertest');
2  const path = require('path');
3  const { app } = require('../src/app');
4
5  let token;
6  let categoriaIdCreada;
7
8  beforeAll(async () => {
9    const res = await request(app)
10     .post('/auth/login')
11     .send({ usuario: 'admin', contrasena: '1234' });
12
13    token = res.body.token;
14  });
15
16  describe('Categorías - Pruebas completas', () => {
17    it('GET /categorias debe devolver lista sin token', async () => {
18      const res = await request(app).get('/categorias');
19
20      expect(res.statusCode).toBe(200);
21      expect(Array.isArray(res.body)).toBe(true);
22    });
23
24    it('POST /categorias debe crear una nueva categoría', async () => {
25      const res = await request(app)
26        .post('/categorias')
27        .set('Authorization', `Bearer ${token}`)
28        .field('nombre', 'Categoría Test')
29        .field('descripcion', 'Descripción de prueba')
30        .attach('foto', path.join(__dirname, 'assets', 'categoriatest.jpg'));
31
32      expect(res.statusCode).toBe(201);
33      expect(res.body).toHaveProperty('id');
34      expect(res.body.nombre).toBe('Categoría Test');
35
36      categoriaIdCreada = res.body.id;
37    });
38
39    it('PUT /categorias/:id debe actualizar la categoría creada', async () => {
40      const res = await request(app)
41        .put(`/categorias/${categoriaIdCreada}`)
```

```
categorias.test.js x
backend-panaderia > tests > categorias.test.js > beforeAll() callback > res
16 describe('Categorías - Pruebas completas', () => {
37 });
38
39 it('PUT /categorias/:id debe actualizar la categoría creada', async () => {
40   const res = await request(app)
41     .put(`/categorias/${categoriaIdCreada}`)
42     .set('Authorization', `Bearer ${token}`)
43     .field('nombre', 'Categoría Modificada')
44     .field('descripcion', 'Desc modificada')
45     .attach('foto', path.join(__dirname, 'assets', 'test-img.jpg'));
46
47   expect([200, 404]).toContain(res.statusCode);
48   if (res.statusCode === 200) {
49     expect(res.body.nombre).toBe('Categoría Modificada');
50   }
51 });
52
53 it('DELETE /categorias/:id debe eliminar la categoría creada', async () => {
54   const res = await request(app)
55     .delete(`/categorias/${categoriaIdCreada}`)
56     .set('Authorization', `Bearer ${token}`);
57
58   expect([200, 404]).toContain(res.statusCode);
59   if (res.statusCode === 200) {
60     expect(res.body.message).toBe('Categoría eliminada');
61   }
62 });
63 });
64
```

Y el resultado obtenido, donde se observa que en este caso las pruebas pasaron el test, obteniendo todas las categorías mediante la petición GET sin requerir un Token, agregando una nueva categoría con la petición POST y los atributos necesarios, editando una categoría mediante la petición PUT a través de su id, y finalmente eliminando la categoría de prueba con la petición DELETE nuevamente a través de su id.

PASS tests/categorias.test.js

- Console

console.log

🔗 Cargando ▶ src/app.js

at Object.log (src/app.js:1:9)

console.log

📦 [auth.controller] cargado

at Object.log (src/controllers/auth.controller.js:1:9)

console.log

📡 POST /auth/login body= { usuario: 'admin', contrasena: '1234' }

at log (src/app.js:20:11)

console.log

🔑 [login] body recibido: { usuario: 'admin', contrasena: '1234' }

at log (src/controllers/auth.controller.js:41:11)

console.log

📡 POST /auth/login body= { usuario: 'admin', contrasena: '1234' }

at log (src/app.js:20:11)

console.log

🔑 [login] body recibido: { usuario: 'admin', contrasena: '1234' }

at log (src/controllers/auth.controller.js:41:11)

console.log

at log (src/app.js:20:11)

console.log

🔑 [login] body recibido: { usuario: 'admin', contrasena: '1234' }

at log (src/controllers/auth.controller.js:41:11)

console.log

at log (src/app.js:20:11)

console.log

🔑 [login] body recibido: { usuario: 'admin', contrasena: '1234' }

at log (src/controllers/auth.controller.js:41:11)

```

console.log
  PUT /categorias/6 body= undefined
  at log (src/app.js:20:11)

console.log
  PUT /categorias/6 body= undefined
  at log (src/app.js:20:11)

  at log (src/app.js:20:11)

console.log
  DELETE /categorias/6 body= undefined
  at log (src/app.js:20:11)
  at log (src/app.js:20:11)

console.log
  DELETE /categorias/6 body= undefined
  at log (src/app.js:20:11)

A worker process has failed to exit gracefully and has been force exited. This is likely caused by tests leaking due to improper teardown. Try running with --
detectOpenHandles to find leaks. Active timers can also cause this, ensure that .unref() was called on them.

Test Suites: 4 passed, 4 total
console.log
  DELETE /categorias/6 body= undefined
  at log (src/app.js:20:11)

A worker process has failed to exit gracefully and has been force exited. This is likely caused by tests leaking due to improper teardown. Try running with --
detectOpenHandles to find leaks. Active timers can also cause this, ensure that .unref() was called on them.

Test Suites: 4 passed, 4 total
  DELETE /categorias/6 body= undefined
  at log (src/app.js:20:11)

A worker process has failed to exit gracefully and has been force exited. This is likely caused by tests leaking due to improper teardown. Try running with --
detectOpenHandles to find leaks. Active timers can also cause this, ensure that .unref() was called on them.

Test Suites: 4 passed, 4 total

A worker process has failed to exit gracefully and has been force exited. This is likely caused by tests leaking due to improper teardown. Try running with --
detectOpenHandles to find leaks. Active timers can also cause this, ensure that .unref() was called on them.

```

4. Conclusiones de los Resultados Obtenidos

Como se pudo observar, se realizaron las pruebas unitarias de los distintos módulos de nuestro proyecto de manera exitosa bajo condiciones normales de uso, las pruebas fueron realizadas utilizando Jest y Supertest sobre el backend de la panadería que fue desarrollado utilizando Node.js y Express con PostgreSQL como base de datos.

En Resumen, las pruebas automatizadas han permitido validar la integridad de los procesos CRUD, el control de acceso y la consistencia de los datos. Esto sienta una base sólida para continuar con confianza hacia la puesta en producción del sistema.

5. Referencias

- Meta. (2025). Jest Documentation. Recuperado de <https://jestjs.io/docs/getting-started>
- Holowaychuk, T. & colaboradores. (2025). Supertest. GitHub. Recuperado de <https://github.com/visionmedia/supertest>
- Express.js Contributors. (2025). Express.js - API Reference. Recuperado de <https://expressjs.com/en/api.html>