Universidad de las Fuerzas Armadas ESPE

Departamento: Ciencias de la Computación

Carrera: Ingeniería de Software

Taller académico Nª: 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

backend-panaderia > tests > productos.test.js > ...

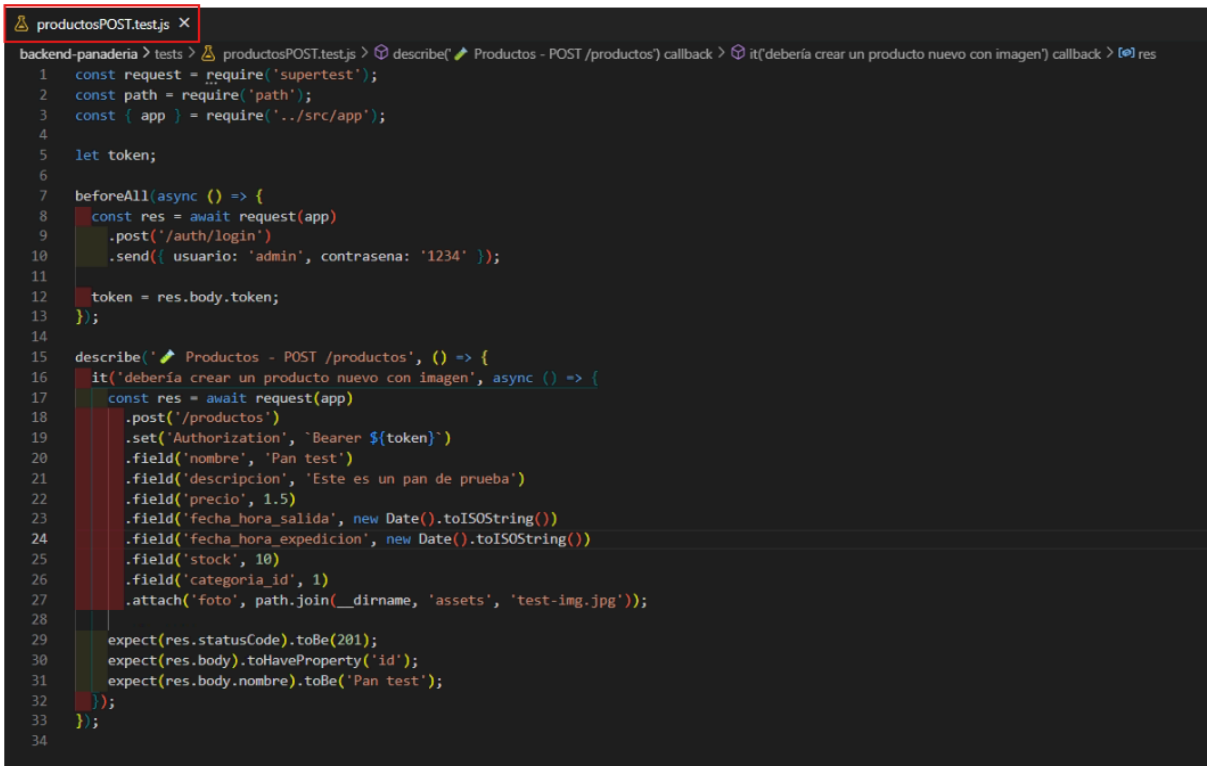
```
1   const request = require('supertest');
2   const { app } = require('../src/app');
3
4   describe('Productos - Endpoints API', () => {
5     it('GET /productos debería retornar todos los productos (status 200)', async () => {
6       const res = await request(app).get('/productos');
7
8       expect(res.statusCode).toBe(200);
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\ProyectoPanadería\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.
```

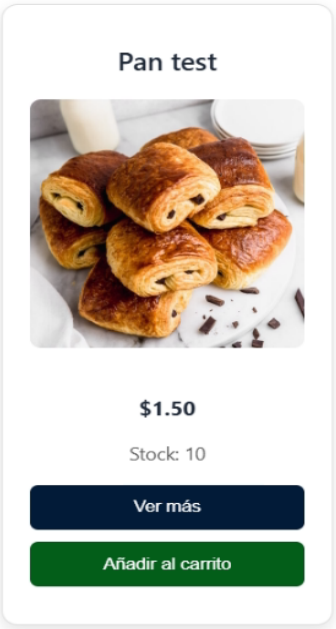
Módulo o Componente	Entrada de Prueba	Resultado Esperado	Resultado Obtenido (en caso de ejecución)	Estado de la prueba (aprobada/fallida)
POST /productos	Campos del formulario + imagen + token admin	201 Created, objeto con id y nombre	id y nombre 201 Created, producto insertado	Aprobada

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



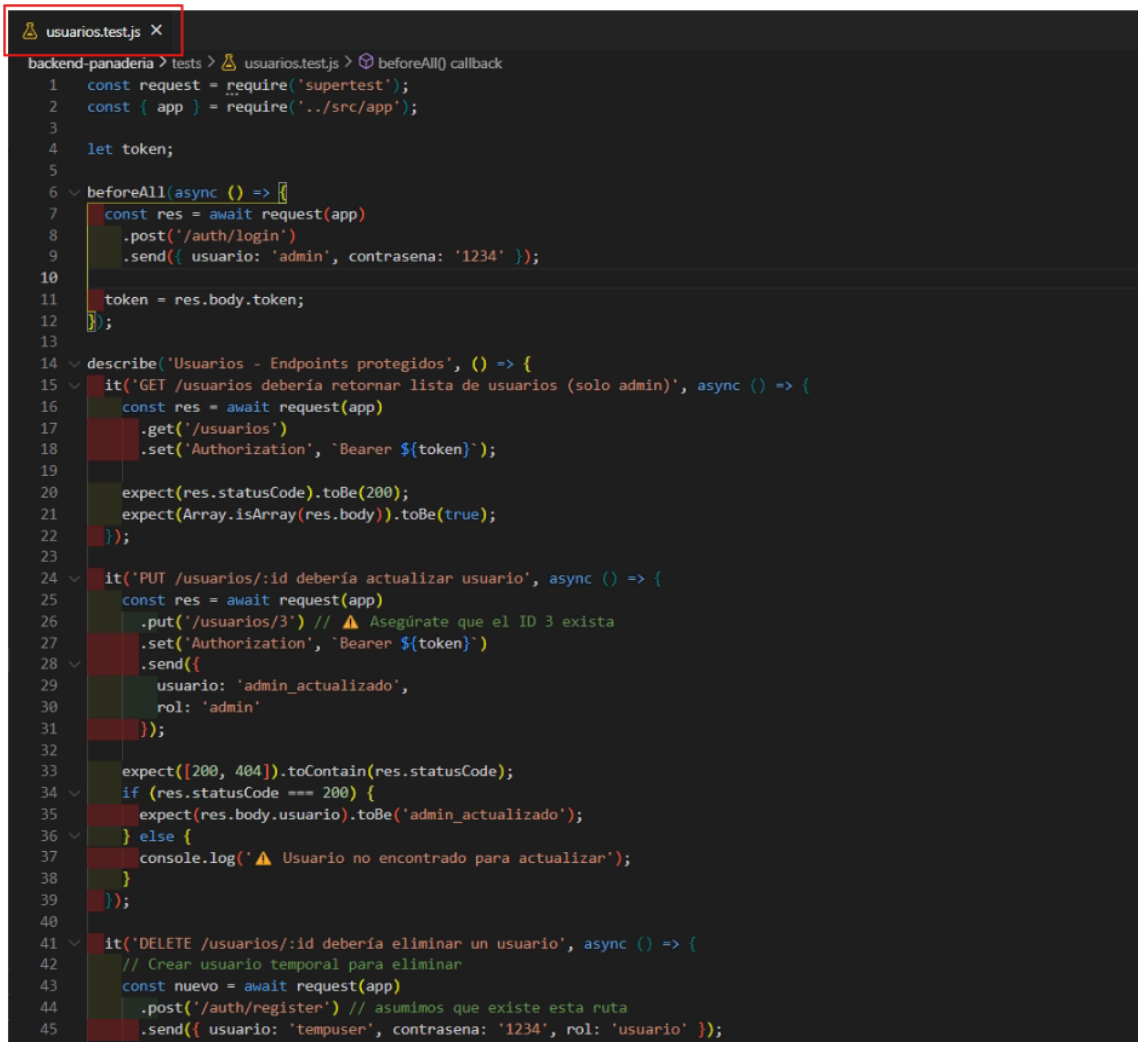
Y el resultado obtenido, donde se observa que en este caso ambas pruebas pasaron el test, agregando el producto de manera correcta

```
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
```



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



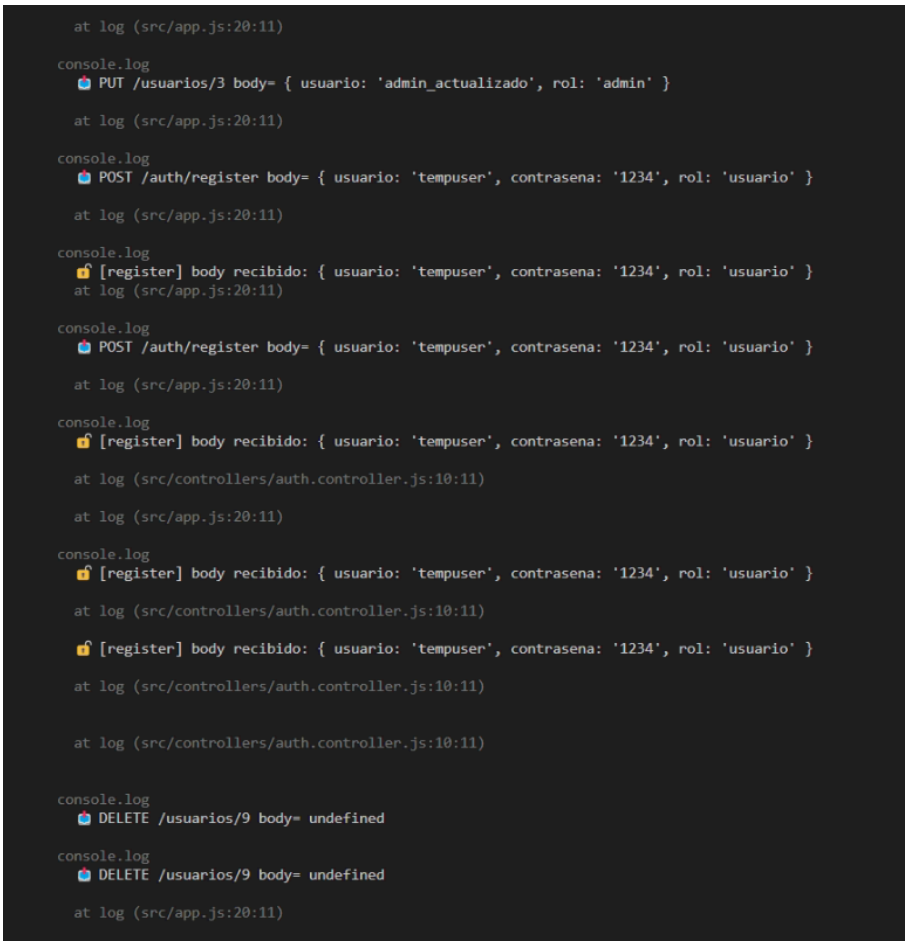
usuarios.test.js

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
```

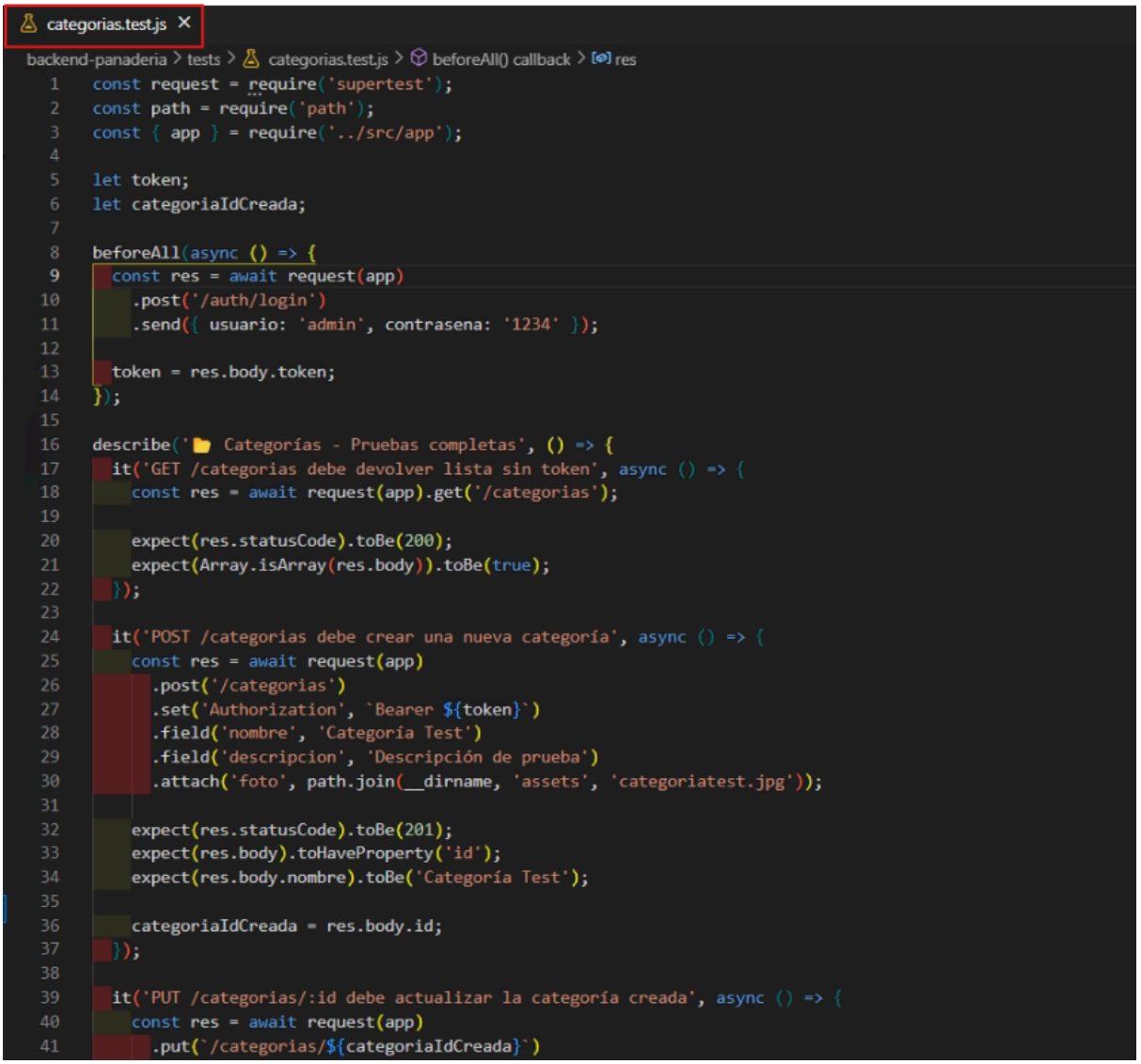
Y el resultado obtenido, donde se observa que en este caso las pruebas pasaron el test, obteniendo todos los usuarios mediante la petición GET, editando un usuario y su rol mediante la petición PUT a través de su id, y finalmente eliminando este usuario de prueba con la petición DELETE nuevamente a través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



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

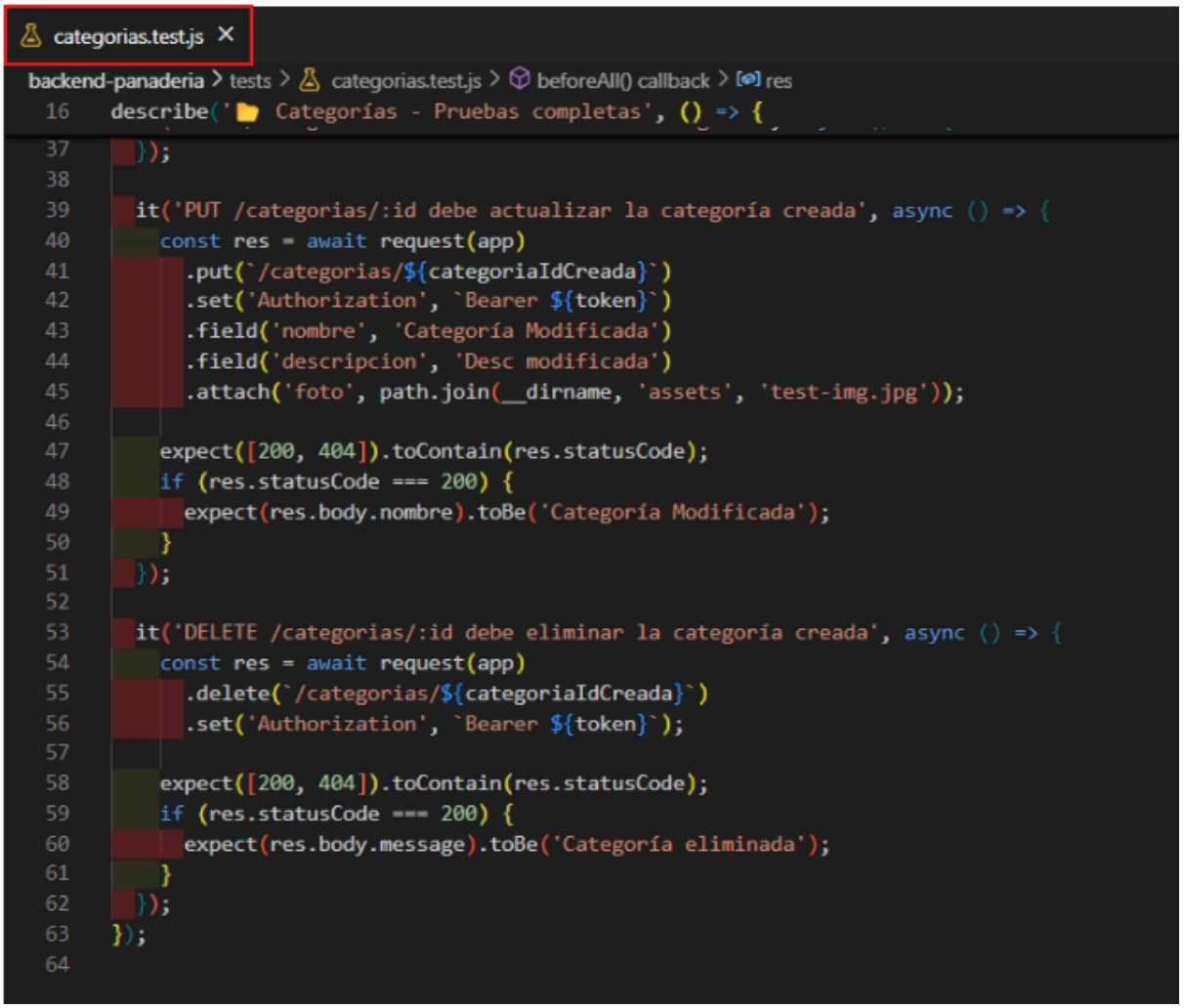
A continuación se muestra el código para la realización de las pruebas



categorias.test.js

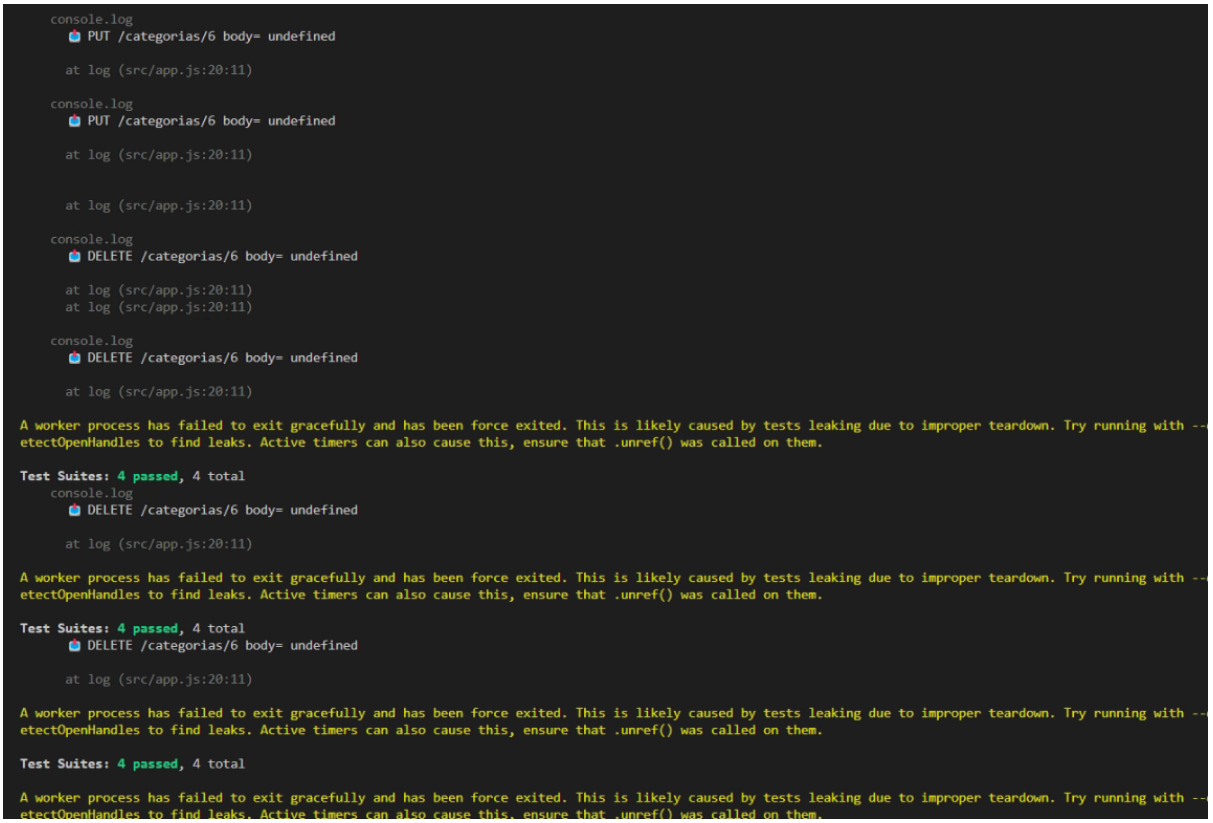
backend-panaderia > tests > categorias.test.js > beforeAll() callback > res

```
const request = require('supertest');
const path = require('path');
const { app } = require('../src/app');

let token;
let categoriaIdCreada;

beforeAll(async () => {
  const res = await request(app)
    .post('/auth/login')
    .send({ usuario: 'admin', contrasena: '1234' });

  token = res.body.token;
});

describe('📁 Categorías - Pruebas completas', () => {
  it('GET /categorias debe devolver lista sin token', async () => {
    const res = await request(app).get('/categorias');

    expect(res.statusCode).toBe(200);
    expect(Array.isArray(res.body)).toBe(true);
  });

  it('POST /categorias debe crear una nueva categoría', async () => {
    const res = await request(app)
      .post('/categorias')
      .set('Authorization', `Bearer ${token}`)
      .field('nombre', 'Categoría Test')
      .field('descripcion', 'Descripción de prueba')
      .attach('foto', path.join(__dirname, 'assets', 'categoriatest.jpg'));

    expect(res.statusCode).toBe(201);
    expect(res.body).toHaveProperty('id');
    expect(res.body.nombre).toBe('Categoría Test');

    categoriaIdCreada = res.body.id;
  });

  it('PUT /categorias/:id debe actualizar la categoría creada', async () => {
    const res = await request(app)
      .put(`/categorias/${categoriaIdCreada}`)
```



Y el resultado obtenido, donde se observa que en este caso las pruebas pasaron el test, obteniendo todos 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)
```



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