Universidad Nacional de Colombia - sede Bogotá
Facultad de Ingeniería
Departamento de Sistemas e Industrial
Curso: Ingeniería de Software 1 (2016701)

TALLER 3 - Testing

Prueba #1

- Nombre del integrante: Alvaro Andres Romero Castro
- Tipo de prueba realizada: Integración
- Descripción breve del componente probado: Creación de espacios a la base de datos por administradores registrados
- Herramienta o framework usado: jest y supertest
- Screenshot del código del test

```
it('debería retornar error 404 si no existe el usuario', async () => {
  const response = await request(app)
    .post('/api/spaces/new')
    .send({
      owner_id: '11111', //usuario no existe
      location: 'test',
      name: 'test',
      capacity: 10
    });

  expect(response.status).toBe(404);
  expect(response.body).toHaveProperty('status', 'Not Found');
  expect(response.body.message).toContain('Owner not found');
});

it('debería retornar error 403 si falla por usuario sin role "admin"', async () => {
  const response = await request(app)
    .post('/api/spaces/new')
    .send({
      owner_id: '54321', //usuario test con rol normal
      location: 'test',
      name: 'test',
      capacity: 10
    });

  expect(response.status).toBe(403);
  expect(response.body).toHaveProperty('status', 'Forbidden');
  expect(response.body.message).toContain('Owner does not have admin privileges');
});
});
```

```
describe('POST /api/users', () => {
  it('debería crear un espacio y retornar status 201', async () => {
    const response = await request(app)
      .post('/api/spaces/new')
      .send({
          owner_id: '12345', //usuario test con rol admin
          location: 'test',
          name: 'test',
          capacity: 10
    });

    expect(response.status).toBe(201);
    expect(response.body).toHaveProperty('message', 'Space created correctly');
    expect(response.body).toHaveProperty('space_id','fakeSpaceId123');
    expect(response.body).toHaveProperty('status', 'success');
  });

  //hay 16 combinaciones de los campos obligatorios, y no nos interesa el caso donde estan todos (test de arriba)
  for (var i=0; i<15;i++) {
    var load = {};
    if (i&1) load.owner_id = '12345';
    if (i&2) load.capacity = 10;
    if (i&4) load.location = 'Test location';
    if (i&8) load.name = 'Test name';
    load.equipment = 'Test equipment'
    load.description = 'Test description'
    it('debería retornar error 400 si faltan campos requeridos', async () => {
      const response = await request(app)
      .post('/api/spaces/new')
      .send(load);

      expect(response.status).toBe(400);
      expect(response.body).toHaveProperty('message', 'Owner_id, name, capacity or location missing');
    });
  }
```

- Resultado de la ejecución:

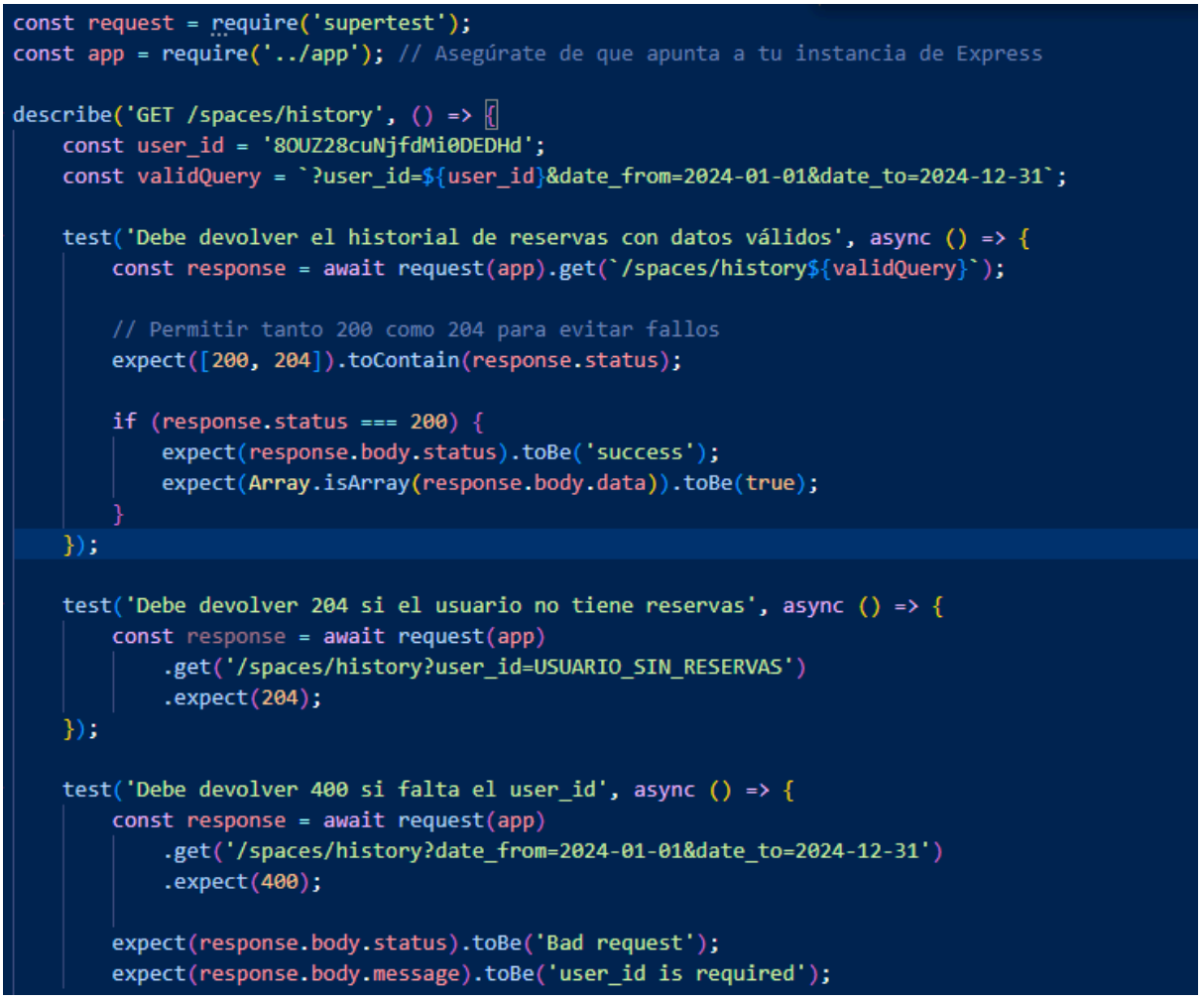
```
PASS tests/c_space.test.js
  POST /api/users
    √ debería crear un espacio y retornar status 201 (29 ms)
    √ debería retornar error 400 si faltan campos requeridos (6 ms)
    √ debería retornar error 400 si faltan campos requeridos (4 ms)
    √ debería retornar error 400 si faltan campos requeridos (4 ms)
    √ debería retornar error 400 si faltan campos requeridos (4 ms)
    √ debería retornar error 400 si faltan campos requeridos (3 ms)
    √ debería retornar error 400 si faltan campos requeridos (4 ms)
    √ debería retornar error 400 si faltan campos requeridos (3 ms)
    √ debería retornar error 400 si faltan campos requeridos (4 ms)
    √ debería retornar error 400 si faltan campos requeridos (3 ms)
    √ debería retornar error 400 si faltan campos requeridos (3 ms)
    √ debería retornar error 400 si faltan campos requeridos (4 ms)
    √ debería retornar error 400 si faltan campos requeridos (3 ms)
    √ debería retornar error 400 si faltan campos requeridos (3 ms)
    √ debería retornar error 400 si faltan campos requeridos (2 ms)
    √ debería retornar error 400 si faltan campos requeridos (3 ms)
    √ debería retornar error 404 si no existe el usuario (4 ms)
    √ debería retornar error 403 si falla por usuario sin role "admin" (3 ms)

Test Suites: 1 passed, 1 total
Tests:       18 passed, 18 total
Snapshots:   0 total
Time:        0.659 s, estimated 1 s
Ran all test suites matching /c_space.test.js/i.
```

- Lecciones aprendidas y dificultades:

Aunque el chequeo de los resultados con jest sea fácil, simular las funcionalidades de firebase con jest es difícil porque toca tener en cuenta cómo es que cada componente devuelve información. Sobre todo, porque se suelen encadenar varios métodos unos detrás de otros, por lo que toca simular cada uno de forma que el siguiente tenga la información del anterior.

Prueba #2

- Nombre del integrante: Tomás Sebastian Vallejo Fonseca
- Tipo de prueba realizada: Prueba de integración de endpoint (GET /spaces/history)
- Descripción breve del componente probado: Se probó el endpoint GET /spaces/history, el cual devuelve el historial de reservas de un usuario dentro de un rango de fechas.
- Herramienta o framework usado: Jest y Superjet
- Screenshot del código del test:



```
const request = require('supertest');
const app = require('../app'); // Asegúrate de que apunta a tu instancia de Express

describe('GET /spaces/history', () => {
    const user_id = '8OUZ28cuNjfdMi0DEDHd';
    const validQuery = `?user_id=${user_id}&date_from=2024-01-01&date_to=2024-12-31`;

    test('Debe devolver el historial de reservas con datos válidos', async () => {
        const response = await request(app).get(`/spaces/history${validQuery}`);

        // Permitir tanto 200 como 204 para evitar fallos
        expect([200, 204]).toContain(response.status);

        if (response.status === 200) {
            expect(response.body.status).toBe('success');
            expect(Array.isArray(response.body.data)).toBe(true);
        }
    });

    test('Debe devolver 204 si el usuario no tiene reservas', async () => {
        const response = await request(app)
            .get('/spaces/history?user_id=USUARIO_SIN_RESERVAS')
            .expect(204);
    });

    test('Debe devolver 400 si falta el user_id', async () => {
        const response = await request(app)
            .get('/spaces/history?date_from=2024-01-01&date_to=2024-12-31')
            .expect(400);

        expect(response.body.status).toBe('Bad request');
        expect(response.body.message).toBe('user_id is required');
```

```
        expect(response.body.message).toBe('user_id is required');
    });

    test('Debe devolver 500 si hay un error interno en la base de datos', async () => {
        jest.spyOn(require('../config/firebase').admin.database(), 'ref').mockImplementation(() => {
            throw new Error('Simulación de error en Firebase');
        });

        const response = await request(app)
            .get(`/spaces/history${validQuery}`)
            .expect(500);

        expect(response.body.status).toBe('Internal Server Error');
        expect(response.body.message).toContain('Simulación de error en Firebase');
    });
});
```

- Resultado de la ejecución:

```
PASS tests/history.test.js
  GET /spaces/history
    √ Debe devolver el historial de reservas con datos válidos (1729 ms)
    √ Debe devolver 204 si el usuario no tiene reservas (132 ms)
    √ Debe devolver 400 si falta el user_id (9 ms)
    √ Debe devolver 500 si hay un error interno en la base de datos (79 ms)

Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        4.402 s, estimated 5 s
Ran all test suites matching /tests\\history.test.js/i.
```

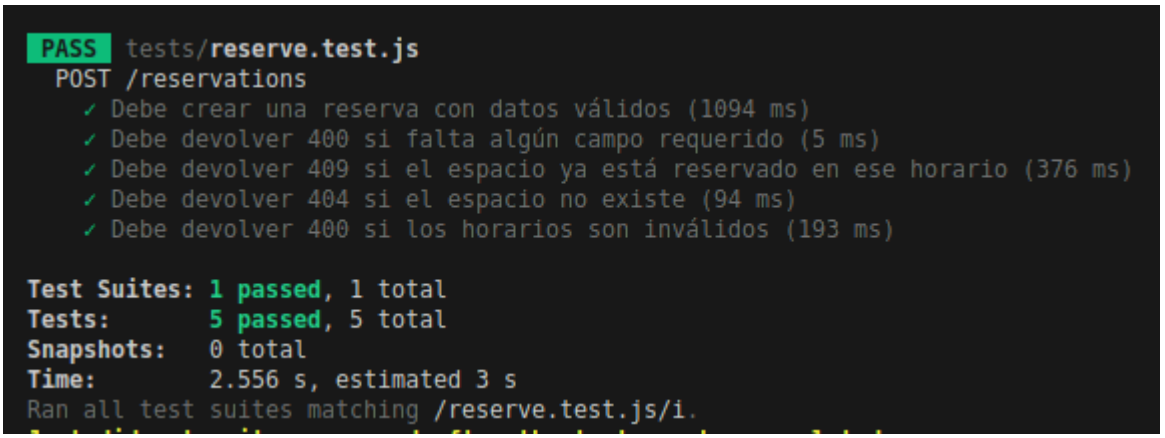
- Lecciones aprendidas y dificultades:
  Es importante manejar respuestas alternativas como 204 No Content en pruebas automatizadas para evitar falsos negativos.
  Supertest es una herramienta útil para validar respuestas de APIs.
  Hubo algunas dificultades en la conexión con firebase pero se logró solucionar

Prueba #3

- Nombre del integrante: Javier Santiago Giraldo Jimenez
- Tipo de prueba realizada: Pruebas de API (Pruebas de integración) para verificar la creación de reservas en el sistema.
- Descripción breve del componente probado: El componente probado es un endpoint en Express (`POST /spaces/reservations`) que permite a los usuarios realizar reservas de espacios. Se valida que las solicitudes con datos correctos sean procesadas correctamente y que aquellas con datos faltantes sean rechazadas con un código de error adecuado.
- Herramienta o framework usado: Jest y Supertest
- Screenshot del código del test:

```
const request = require('supertest');
const app = require('../app');

describe('POST /reservations', () => {
    const validReservation = {
        space_id: '-OKCh4Nq-ctluXb5S6pJ',
        start_time: '2025-03-15T09:00:00Z',
        end_time: '2025-03-15T11:00:00Z',
        user_id: '8OUZ28cuNjfdMi0DEDHd'
    };

    test('Debe crear una reserva con datos válidos', async () => {
        const response = await request(app)
            .post('/spaces/reservations')
            .send(validReservation)
            .expect(201);

        expect(response.body.status).toBe('success');
        expect(response.body.data).toHaveProperty('reservation_id');
        expect(response.body.data).toHaveProperty('pdf_url');
    });

    test('Debe devolver 400 si falta algún campo requerido', async () => {
        const incompleteReservation = { ...validReservation };
        delete incompleteReservation.start_time;

        const response = await request(app)
            .post('/spaces/reservations')
            .send(incompleteReservation)
            .expect(400);

        expect(response.body.status).toBe('Bad request');
    });

    test('Debe devolver 409 si el espacio ya está reservado en ese horario', async () => {
        await request(app).post('/spaces/reservations').send(validReservation);
        const response = await request(app)
            .post('/spaces/reservations')
            .send(validReservation)
            .expect(409);

        expect(response.body.status).toBe('Conflict');
    });

    test('Debe devolver 404 si el espacio no existe', async () => {
        const response = await request(app)
            .post('/spaces/reservations')
            .send({ ...validReservation, space_id: 'ID_INEXISTENTE' })
            .expect(404);

        expect(response.body.status).toBe('Not found');
    });

    test('Debe devolver 400 si los horarios son inválidos', async () => {
        const invalidReservation = {
            ...validReservation,
            start_time: '2025-03-15T12:00:00Z',
            end_time: '2025-03-15T10:00:00Z'
        };
        const response = await request(app)
            .post('/spaces/reservations')
            .send(invalidReservation)
            .expect(400);

        expect(response.body.status).toBe('Bad request');
    });
});
```

- Resultado de la ejecución:



- Lecciones aprendidas y dificultades:
    - Se validó que el endpoint maneja correctamente los casos de éxito y error.
    - Se identificó que la API devuelve respuestas claras cuando hay errores en los datos enviados.
    - Dificultad en el uso de IDs en la base de datos ya que están previamente cifrados.
    - Dificultad en el uso de firebase dado que cuenta con dos tipos de bases de datos (realtime Database y Firestore database), la sincronización entre ambas no es automática lo cual puede crear conflicto al acceder a un dato que no está presente en la BD instaciada.

Prueba #4

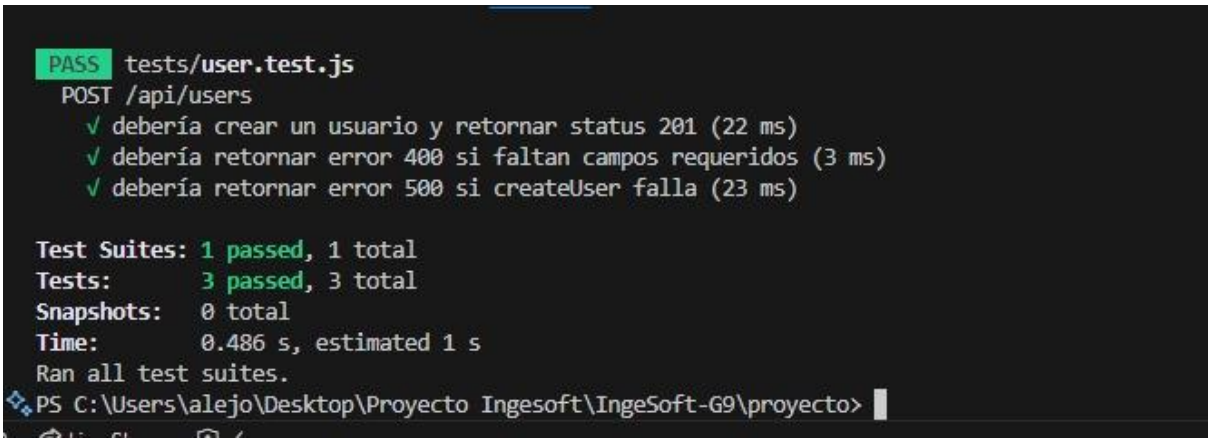
- Nombre del integrante: Manuel Alejandro Navas Bohorquez
- Tipo de prueba realizada :Unitaria
- Descripción breve del componente probado:
    - Creación de usuarios en la base de datos
- Herramienta o framework usado:jest, supertest

- Screenshot del código del test:

```
You, 11 hours ago | 1 author (You)
const request = require('supertest');  140.5k (gzipped: 43.3k)
const express = require('express');
const bodyParser = require('body-parser');  487.4k (gzipped: 212.1k)

jest.mock('../config/firebase', () => {

  const createUserMock = jest.fn().mockResolvedValue({
    uid: 'fakeUid123',
    displayName: 'Test User'
  });

  const setMock = jest.fn().mockResolvedValue(true);
  const docMock = jest.fn(() => ({ set: setMock }));
  const collectionMock = jest.fn(() => ({ doc: docMock }));

  const firestore = () => ({
    collection: collectionMock,
    FieldValue: {
      serverTimestamp: jest.fn(() => "timestamp")
    }
  });

  const admin = {
    auth: jest.fn(() => ({ createUser: createUserMock })),
    firestore: firestore
  };

  return { admin };
});

// traer router pa testear
const userRouter = require('../routes/CreateUser');

// abrirlo para que pueda llamar de api
const app = express();
app.use(bodyParser.json());
app.use('/api/users', userRouter);
```

```
//test 1 crear usuario todo ok
describe('POST /api/users', () => {
  it('debería crear un usuario y retornar status 201', async () => {
    const response = await request(app)
      .post('/api/users')
      .send({
        email: 'test@example.com',      You, 11 hours ago • Porfin gonorea, ya conectamos la base
        password: 'secret',
        nombre: 'Test User',
        rol: 'admin'
      });

    expect(response.status).toBe(201);
    expect(response.body).toHaveProperty('userId', 'fakeUid123');
    expect(response.body).toHaveProperty('status', 'success');
  });

  //test 2 faltan los parametros o alguno deberia dar error

  it('debería retornar error 400 si faltan campos requeridos', async () => {
    const response = await request(app)
      .post('/api/users')
      .send({
        email: 'test@example.com',
        password: 'secret'
        // Falta nombre y rol
      });

    expect(response.status).toBe(400);
    expect(response.body).toHaveProperty('message', 'Email, password, nombre, or rol missing');
  });

  // test 3 lo que deberia hacer si firebase falla, deberia dar error

  it('debería retornar error 500 si createUser falla', async () => {

    const { admin } = require('../config/firebase');
    admin.auth.mockImplementationOnce(() => ({
      createUser: jest.fn().mockRejectedValue(new Error('Firebase auth error'))
    }));

    const response = await request(app)
      .post('/api/users')
      .send({
        email: 'fail@example.com',
        password: 'secret',
        nombre: 'Fail User',
        rol: 'admin'
      });

    expect(response.status).toBe(500);
    expect(response.body).toHaveProperty('status', 'Internal Server Error');
    expect(response.body.message).toContain('Firebase auth error');
  });
});
```

- Resultado de la ejecución:



- Lecciones aprendidas y dificultades: Simular una conexión de firebase con jest requiere varios pasos donde requerimos la ayuda de la ia, los test a veces no salian porque no tenia bien la determinación de parametros fallidos, en firebase ese debe ser muy específico de que herramienta y como se debe usar, lecciones aprendidas es mejor primero saber que esperar y empezar a desarrollar que desarrollar y ver que sale