

**Proyecto: Informe testing**

Sebastian Andrade Cedano  
Cristian Steven Motta Ojeda  
Juan Esteban Santacruz Corredor

Ingeniería de software I

Universidad Nacional de Colombia

Bogotá D.C.  
2025

## Índice

Índice.....	2
1. Introducción.....	3
2. Test #1.....	3
3. Test #2.....	5
4. Test #3.....	6
5. Reflexión.....	8

## 1. Introducción

"Codes" es una plataforma en línea dedicada a la práctica de programación competitiva. Ofrece a los usuarios acceso a una amplia variedad de problemas, permitiéndoles enviar sus soluciones para que sean validadas automáticamente por la aplicación. Además, permite a los "problemsetters" (personas encargadas de crear los problemas) añadir y editar problemas de manera intuitiva.

En el apartado técnico, la aplicación está construida haciendo uso de una arquitectura de microservicios, contando una REST API principal que permite gestionar la información almacenada en la base de datos, un microservicio adicional que permite compilar y correr el código enviado por los usuarios, y una aplicación web encargada de permitir a los usuarios interactuar con la plataforma de manera fácil e intuitiva.

## 2. Test #1

- **Integrante**

Cristian Steven Motta Ojeda

- **Tipo de prueba**

Prueba unitaria

- **Descripción del componente**

Este test verifica el correcto funcionamiento del endpoint /run, que se encarga de ejecutar código en C++ enviado por el usuario. Evalúa si el código se compila y ejecuta correctamente, detecta errores de compilación, maneja bucles infinitos mediante un límite de tiempo y responde apropiadamente ante entradas inválidas.

- **Herramienta**

1. *Jest*: Framework de pruebas para JavaScript.
2. *Supertest*: Librería para hacer peticiones HTTP a la API en un entorno de prueba.
3. *ts-jest*: Preprocesador para ejecutar pruebas en TypeScript con Jest.

- **Código del test**

```

1 describe("POST /run", () => {
2   it("should successfully execute a valid C++ program", async () => {
3     const response = await request(app).post("/run").send({
4       id: 1,
5       code: `#include <iostream>\nusing namespace std;\nint main() { int a, b; cin >> a >> b; cout << a + b << endl; return 0; }`,
6       input: "3 5",
7       time_limit: 2
8     });
9
10    expect(response.status).toBe(200);
11    expect(response.body.status).toBe("OK");
12    expect(response.body.output).toBe("8\n");
13    expect(parseFloat(response.body.execution_time)).toBeGreaterThan(0);
14  });
15
16   it("should return COMPILATION_ERROR for invalid C++ code", async () => {
17     const response = await request(app).post("/run").send({
18       id: 2,
19       code: `#include <iostream>\nusing namespace std;\nint main() { int a, b cin >> a >> b; cout << a + b << endl; return 0; }`, // Missing semicolon
20       input: "3 5",
21       time_limit: 2
22     });
23
24    expect(response.status).toBe(200);
25    expect(response.body.status).toBe("COMPILATION_ERROR");
26  });
27
28   it("should return TIME_LIMIT_EXCEEDED for an infinite loop", async () => {
29     const response = await request(app).post("/run").send({
30       id: 3,
31       code: `#include <iostream>\nusing namespace std;\nint main() { while(true) {} return 0; }`,
32       input: "",
33       time_limit: 2
34     });
35
36    expect(response.status).toBe(200);
37    expect(response.body.status).toBe("TIME_LIMIT_EXCEEDED");
38  });
39
40   it("should handle server errors gracefully", async () => {
41     jest.spyOn(console, "error").mockImplementation(() => {});
42
43     const response = await request(app).post("/run").send({
44       id: "invalid_id",
45       code: "int main() { return 0; }",
46       input: "",
47       time_limit: 2
48     });
49
50    expect(response.status).toBe(400);
51    expect(response.body.error).toBe("Invalid input parameters");
52  });
53 });

```

Imagen 1: Código del test para el runner

## - Resultado

```

● cristian@Cristians-MacBook code_runner % npm test

> code_runner@1.0.0 test
> jest

PASS tests/run.test.ts
  POST /run
    ✓ should successfully execute a valid C++ program (1434 ms)
    ✓ should return COMPILATION_ERROR for invalid C++ code (202 ms)
    ✓ should return TIME_LIMIT_EXCEEDED for an infinite loop (2228 ms)
    ✓ should handle server errors gracefully (192 ms)

Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        4.776 s, estimated 5 s
Ran all test suites.

```

Imagen 2: Resultado del test para el runner

### 3. Test #2

- **Integrante**

Sebastian Andrade Cedano

- **Tipo de prueba**

Prueba unitaria

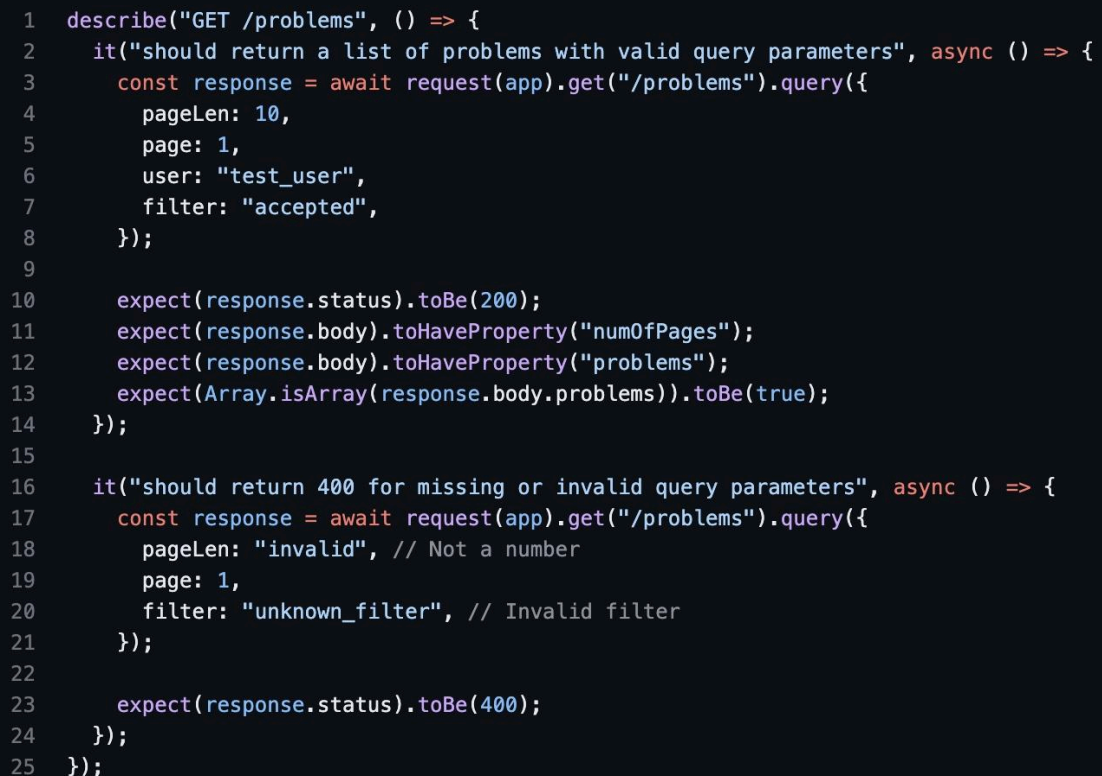
- **Descripción del componente**

Este test valida la funcionalidad del endpoint `/problems`, que devuelve una lista paginada de problemas de programación. Comprueba si la API responde correctamente cuando se proporcionan parámetros válidos y si maneja adecuadamente los errores cuando los parámetros son inválidos o faltan.

- **Herramienta**

1. *Jest*: Framework de pruebas para JavaScript.
2. *Supertest*: Librería para hacer peticiones HTTP a la API en un entorno de prueba.
3. *ts-jest*: Preprocesador para ejecutar pruebas en TypeScript con Jest.

- **Código del test**



```
1 describe("GET /problems", () => {
2   it("should return a list of problems with valid query parameters", async () => {
3     const response = await request(app).get("/problems").query({
4       pageLen: 10,
5       page: 1,
6       user: "test_user",
7       filter: "accepted",
8     });
9
10    expect(response.status).toBe(200);
11    expect(response.body).toHaveProperty("numOfPages");
12    expect(response.body).toHaveProperty("problems");
13    expect(Array.isArray(response.body.problems)).toBe(true);
14  });
15
16  it("should return 400 for missing or invalid query parameters", async () => {
17    const response = await request(app).get("/problems").query({
18      pageLen: "invalid", // Not a number
19      page: 1,
20      filter: "unknown_filter", // Invalid filter
21    });
22
23    expect(response.status).toBe(400);
24  });
25 });
```

Imagen 3: Código del test para obtener problemas

## - Resultado

```
● cristian@Cristians-MacBook main % npm test

> main@1.0.0 test
> jest

PASS test/problems.test.ts
  GET /problems
    ✓ should return a list of problems with valid query parameters (443 ms)
    ✓ should return 400 for missing or invalid query parameters (15 ms)

Test Suites: 1 passed, 1 total
Tests: 2 passed, 2 total
Snapshots: 0 total
Time: 1.891 s, estimated 2 s
Ran all test suites.
```

Imagen 4: Resultado del test para obtener problemas

## 4. Test #3

### - Integrante

Juan Esteban Santacruz Corredor

### - Tipo de prueba

Prueba unitaria

### - Descripción del componente

Este test verifica la creación de nuevos problemas de programación a través del endpoint /problems. Se asegura de que el problema se registre exitosamente cuando los datos son completos y válidos, y también evalúa el comportamiento de la API cuando se intenta crear un problema con información incompleta.

### - Herramienta

4. *Jest*: Framework de pruebas para JavaScript.
5. *Supertest*: Librería para hacer peticiones HTTP a la API en un entorno de prueba.
6. *ts-jest*: Preprocesador para ejecutar pruebas en TypeScript con Jest.

### - Código del test

```

1  describe("POST /problems", () => {
2    it("should successfully create a problem with valid input", async () => {
3      const response = await request(app).post("/problems").send({
4        name: "Sample Problem",
5        statement: "Solve this problem...",
6        editorial: "Editorial explanation...",
7        time_limit_seconds: 2,
8        memory_limit_mb: 256,
9        problemsetter_handle: "shollyero",
10       input: "1 2",
11       output: "3",
12     });
13
14     expect(response.status).toBe(201);
15     expect(response.body.message).toBe("OK");
16   });
17
18   it("should return 400 if required fields are missing", async () => {
19     const response = await request(app).post("/problems").send({
20       name: "Incomplete Problem",
21       statement: "This problem is missing fields",
22       // Missing editorial, time_limit_seconds, memory_limit_mb, etc.
23     });
24
25     expect(response.status).toBe(400);
26     expect(response.body.message).toBe("There is not enough data to create the problem.");
27   });
28 });

```

Imagen 5: Código del test para crear problemas

## - Resultado

```

ⓧ cristian@Cristians-MacBook main % npm test

> main@1.0.0 test
> jest

PASS test/problems.test.ts
  POST /problems
    ✓ should successfully create a problem with valid input (31 ms)
    ✓ should return 400 if required fields are missing (2 ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        0.843 s, estimated 2 s
Ran all test suites.

```

Imagen 6: Resultado del test para crear problemas

## **5. Reflexión**

Como estudiantes de pregrado, el desarrollo de nuestros proyectos está alejado (en algunos aspectos) de las dinámicas que se trabajan en un entorno laboral real. Muchas veces aprendemos la importancia de estas herramientas después de habernos graduado, lo que puede dificultar nuestro desempeño como profesionales. Dentro de estas dinámicas se encuentra el testing.

El testing es una herramienta fundamental que se aplica para detectar errores antes de la implementación, garantizar la calidad del software y prevenir fallos. Debido a nuestra inexperiencia, muchas veces pasamos por alto estas herramientas, que no afectan directamente el funcionamiento de nuestra aplicación, pero sí pueden facilitar el desarrollo del mismo. Gracias a este ejercicio, pudimos evidenciar la importancia del testing en el desarrollo de software y aprendimos a implementar tests en nuestras aplicaciones, lo que nos acercó más a las dinámicas que se utilizan en un ambiente laboral.