



UNIVERSIDAD  
**NACIONAL**  
DE COLOMBIA

Universidad Nacional de Colombia - sede Bogotá

Facultad de Ingeniería

Departamento de Sistemas e Industrial

Curso: Ingeniería de Software 1 (2016701)

# Testing unitario

*Nicolás Andrés Castellanos Rico*

*Juan Felipe Quiroga Medina*

*Erfán Andrés Triana Duque*

*Juan David Yopasa Maldonado*

UBudget

## Herramientas de pruebas

Para las pruebas en Backend, se usó Jest. Para las pruebas en Frontend, se usó Vitest, que funciona de forma muy parecida a Jest, con algunas ventajas

# Pruebas unitarias por integrante

## 3.1 Pruebas realizadas por Nicolás Andrés Castellanos Rico

Descripción de la funcionalidad probada: Login, Registro y Cambio de contraseña

Código del test:

```
import request from "supertest";
import app from "../src/server";
import { AppDataSource } from "../src/database";

jest.mock("../src/server"); // Prevent server.ts from auto-initializing DB
or starting server

beforeAll(async () => {
  await AppDataSource.initialize();
});

afterAll(async () => {
  await AppDataSource.destroy();
});

describe("Auth Endpoints", () => {
  const email = "testuser@example.com";
  const originalPassword = "Password123!";
  const newPassword = "NewPass456!";
  let resetToken = "";

  it("should register a new user", async () => {
    const res = await request(app)
      .post("/auth/register")
      .send({
        name: "testuser",
        email,
        password: originalPassword,
      });
  });
});
```

```
    expect(res.statusCode).toBe(201);
    expect(res.body).toHaveProperty(
      "message",
      "User registered successfully."
    );
  });

it("should login an existing user", async () => {
  const res = await request(app)
    .post("/auth/login")
    .send({
      email,
      password: originalPassword,
    });

  expect(res.statusCode).toBe(200);
  expect(res.body).toHaveProperty("token");
});

it("should request a password reset", async () => {
  const res = await request(app)
    .post("/auth/request-reset")
    .send({
      email,
    });

  expect(res.statusCode).toBe(200);
  expect(res.body).toHaveProperty("token");
  expect(res.body).toHaveProperty("message", "Reset token generated");

  resetToken = res.body.token;
  expect(resetToken).toBeTruthy();
});

it("should reset the password using the token", async () => {
  const res = await request(app)
    .post("/auth/reset-password")
    .send({
      token: resetToken,
      newPassword,
    });
});
```

```

    expect(res.statusCode).toBe(200);
    expect(res.body).toHaveProperty(
      "message",
      "Password updated successfully"
    );
  });

  it("should login with the new password", async () => {
    const res = await request(app)
      .post("/auth/login")
      .send({
        email,
        password: newPassword,
      });

    expect(res.statusCode).toBe(200);
    expect(res.body).toHaveProperty("token");
  });
});

// --- CASOS LÍMITE NEGATIVOS ---

it("should fail to register an existing user (case: existing user)",
  async () => {
    const res = await request(app)
      .post("/auth/register")
      .send({
        name: "testuser",
        email,
        password: originalPassword,
      });

    (Conflict),
    expect(res.statusCode).toBe(400);
    expect(res.body).toHaveProperty("message");
    expect(res.body.message).toContain("already exists");
  });

  it("should login an existing user", async () => {
    const res = await request(app)
      .post("/auth/login")
      .send({

```

```

        email,
        password: originalPassword,
    });

    expect(res.statusCode).toBe(200);
    expect(res.body).toHaveProperty("token");
});

it("should fail to login with incorrect password (case: incorrect password)", async () => {
    const res = await request(app)
        .post("/auth/login")
        .send({
            email,
            password: "WrongPassword123!",
        });

    expect(res.statusCode).toBe(401);
    expect(res.body).toHaveProperty(
        "message",
        "Invalid credentials"
    );
});

it("should fail to login with an unregistered email (case: unregistered user)", async () => {
    const res = await request(app)
        .post("/auth/login")
        .send({
            email: "nonexistent@example.com",
            password: originalPassword,
        });

    // Se espera un código 401 (Unauthorized) por usuario no encontrado o
    // credenciales inválidas.
    expect(res.statusCode).toBe(401);
    expect(res.body).toHaveProperty(
        "message",
        "Invalid credentials"
    );
});

```

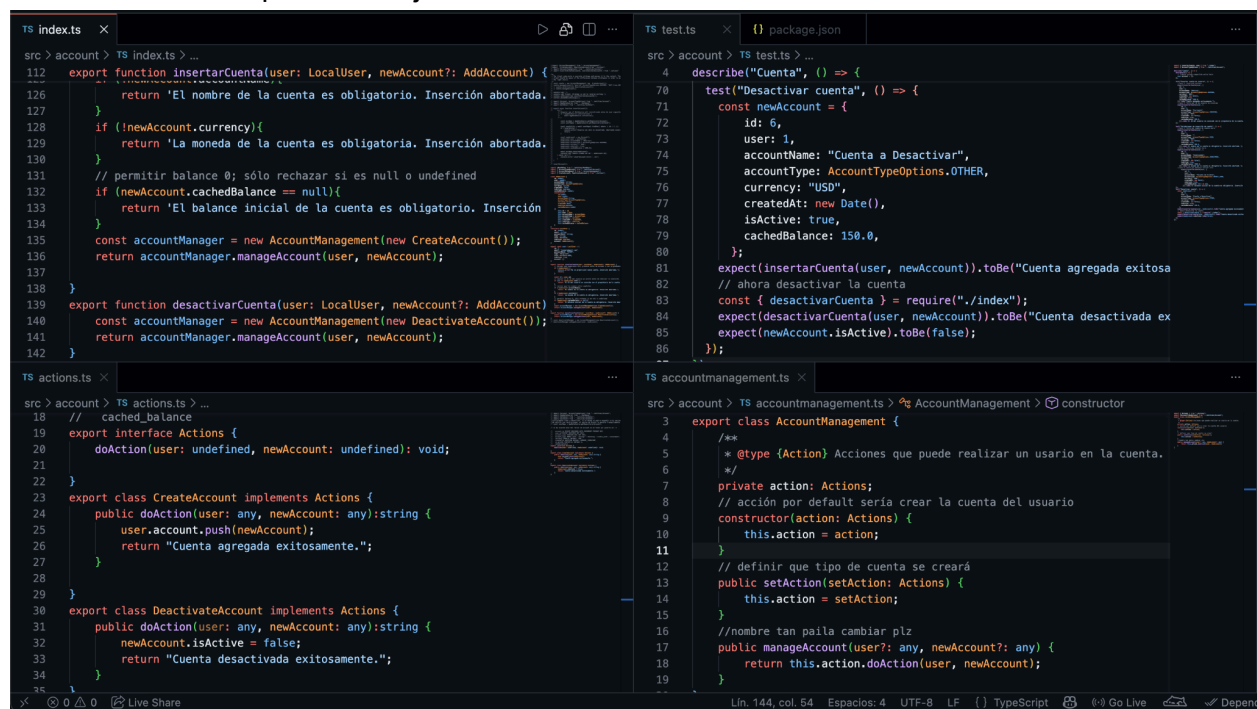
Casos límite documentados:

- Caso 1: Registro de usuario existente
- Caso 2: Login con contraseña incorrecta
- Caso 3: Login de usuario no registrado

## 3.2 Pruebas realizadas por Juan Felipe Quiroga Medina

Descripción de la funcionalidad probada: Agregar una nueva cuenta relacionada a un único usuario, se implementa el patrón de diseño Strategy.

Módulos sobre los que se trabajó:



```
TS index.ts
src > account > TS index.ts > ...
112 export function insertarCuenta(user: LocalUser, newAccount?: AddAccount) {
126   return "El nombre de la cuenta es obligatorio. Inserción abortada."
127 }
128 if (!newAccount.currency){
129   return "La moneda de la cuenta es obligatoria. Inserción abortada."
130 }
131 // permitir balance 0; sólo rechazar si es null o undefined
132 if (newAccount.cachedBalance == null){
133   return "El balance inicial de la cuenta es obligatorio. Inserción
134 }
135 const accountManager = new AccountManagement(new CreateAccount());
136 return accountManager.manageAccount(user, newAccount);
137 }
138 }
139 export function desactivarCuenta(user: LocalUser, newAccount?: AddAccount)
140 const accountManager = new AccountManagement(new DeactivateAccount());
141 return accountManager.manageAccount(user, newAccount);
142 }

TS test.ts
src > account > TS test.ts > ...
4 describe("Cuenta", () => {
70 test("Desactivar cuenta", () => {
71   const newAccount = {
72     id: 6,
73     user: 1,
74     accountName: "Cuenta a Desactivar",
75     accountType: AccountTypeOptions.OTHER,
76     currency: "USD",
77     createdAt: new Date(),
78     isActive: true,
79     cachedBalance: 150.0,
80   };
81   expect(insertarCuenta(user, newAccount)).toBe("Cuenta agregada exitosa
82   // ahora desactivar la cuenta
83   const { desactivarCuenta } = require("../index");
84   expect(desactivarCuenta(user, newAccount)).toBe("Cuenta desactivada ex
85   expect(newAccount.isActive).toBe(false);
86 });

TS actions.ts
src > account > TS actions.ts > ...
18 // cached_balance
19 export interface Actions {
20   doAction(user: undefined, newAccount: undefined): void;
21 }
22 }
23 export class CreateAccount implements Actions {
24   public doAction(user: any, newAccount: any):string {
25     user.account.push(newAccount);
26     return "Cuenta agregada exitosamente.";
27   }
28 }
29 }
30 export class DeactivateAccount implements Actions {
31   public doAction(user: any, newAccount: any):string {
32     newAccount.isActive = false;
33     return "Cuenta desactivada exitosamente.";
34   }
35 }

TS accountmanagement.ts
src > account > TS accountmanagement.ts > AccountManagement > constructor
3 export class AccountManagement {
4   /**
5    * @type {Action} Acciones que puede realizar un usuario en la cuenta.
6    */
7   private action: Actions;
8   // acción por default seria crear la cuenta del usuario
9   constructor(action: Actions) {
10     this.action = action;
11   }
12   // definir que tipo de cuenta se creará
13   public setAction(setAction: Actions) {
14     this.action = setAction;
15   }
16   //nombre tan paila cambiar plz
17   public manageAccount(user?: any, newAccount?: any) {
18     return this.action.doAction(user, newAccount);
19   }
20 }
```

Para tener paridad con lo esperado del test, se agregaron unos return tipo string con el mensaje esperado, con el fin de comprobar que se fue por el camino deseado. Serán retirados más adelante.

Código del test:

```
import { insertarCuenta, user } from "../index";
import { AccountTypeOptions } from "../entities/Account";

describe("Cuenta", () => {
  beforeEach(() => {
    // limpiar estado compartido entre tests
  })
})
```

```

    user.account = [];
  });

test("Insertar cuenta en usuario", () => {
  //caso 1: inserción exitosa
  expect(insertarCuenta(user, {
    id: 1,
    user: 1,
    accountName: "Ahorros",
    accountType: AccountTypeOptions.SAVINGS,
    currency: "USD",
    createdAt: new Date(),
    isActive: true,
    cachedBalance: 1000.0,
  })).toBe("Cuenta agregada exitosamente.");
  //caso 2: fallo por id de usuario no coincide
  expect(insertarCuenta(user, {
    id: 2,
    user: 2,
    accountName: "Corriente",
    accountType: AccountTypeOptions.CHECKING,
    currency: "USD",
    createdAt: new Date(),
    isActive: true,
    cachedBalance: 500.0,
  })).toBe('El ID del usuario no coincide con el propietario de la cuenta. Inserción abortada.');
```

```

  });

test("Validaciones de inserción de cuenta", () => {
  //caso 1: fallo por nombre de cuenta vacío
  expect(insertarCuenta(user, {
    id: 3,
    user: 1,
    accountName: "",
    accountType: AccountTypeOptions.CASH,
    currency: "USD",
    createdAt: new Date(),
    isActive: true,
    cachedBalance: 300.0,
  })).toBe('El nombre de la cuenta es obligatorio. Inserción abortada.');
```

```

  //caso 2: fallo por moneda vacía
  expect(insertarCuenta(user, {
    id: 4,
    user: 1,
    accountName: "Inversiones",
    accountType: AccountTypeOptions.INVESTMENT,
    currency: "",
    createdAt: new Date(),
    isActive: true,
    cachedBalance: 2000.0,
  })).toBe('La moneda de la cuenta es obligatoria. Inserción abortada.');
```

```

//caso 3: fallo por balance inicial nulo
expect(insertarCuenta(user, {
  id: 5,
  user: 1,
  accountName: "Tarjeta de Crédito",
  accountType: AccountTypeOptions.CREDIT_CARD,
  currency: "USD",
  createdAt: new Date(),
  isActive: true,
  cachedBalance: null as any,
})).toBe('El balance inicial de la cuenta es obligatorio. Inserción abortada.');
```

```

});
test("Desactivar cuenta", () => {
  const newAccount = {
    id: 6,
    user: 1,
    accountName: "Cuenta a Desactivar",
    accountType: AccountTypeOptions.OTHER,
    currency: "USD",
    createdAt: new Date(),
    isActive: true,
    cachedBalance: 150.0,
  };
  expect(insertarCuenta(user, newAccount)).toBe("Cuenta agregada exitosamente.");
  // ahora desactivar la cuenta
  const { desactivarCuenta } = require("../index");
  expect(desactivarCuenta(user, newAccount)).toBe("Cuenta desactivada exitosamente.");
  expect(newAccount.isActive).toBe(false);
});
});

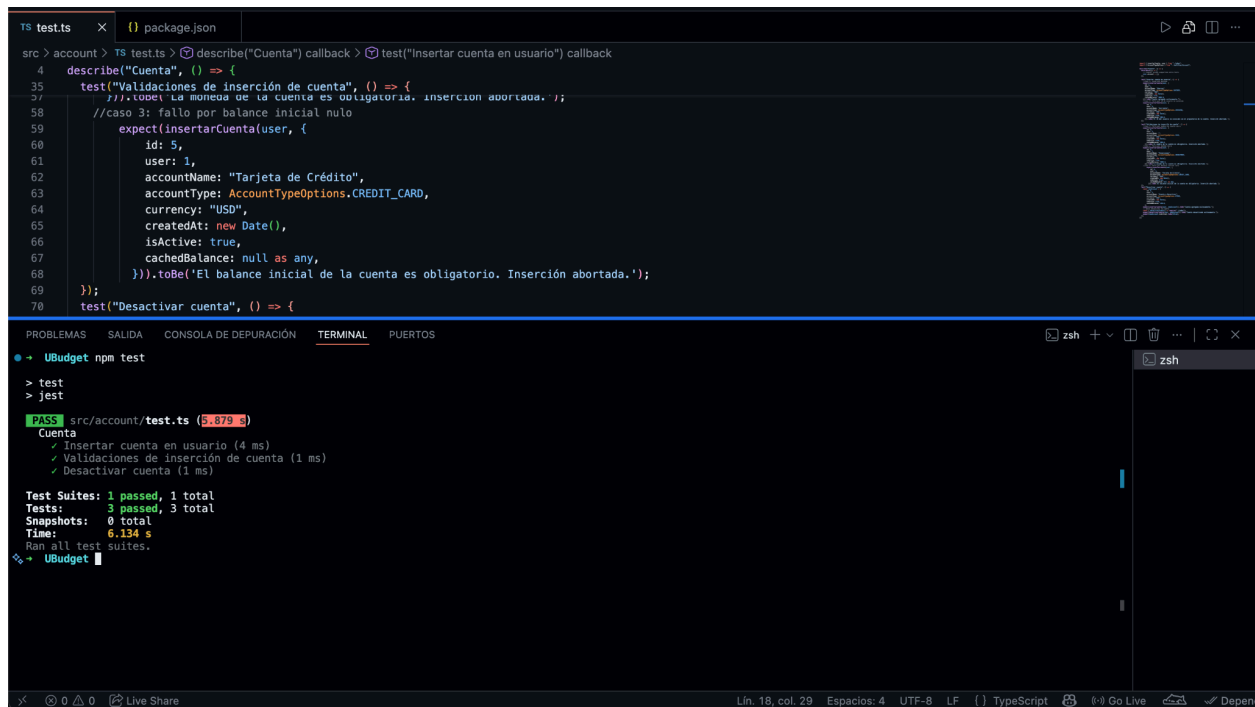
```

Casos límite documentados:

- Caso 1: Inserción de una cuenta tipo Ahorros, comprobar que el usuario ya exista.
- Caso 2: Comprobar que los campos se completen correctamente al momento de crear la cuenta.
- Caso 3: Desactivar una cuenta ya existente.



Evidencia de ejecución:



The image shows a VS Code editor with a TypeScript test file named `test.ts` and its execution results in the terminal. The test file contains the following code:

```
src > account > TS test.ts > describe("Cuenta") callback > test("Insertar cuenta en usuario") callback
4 describe("Cuenta", () => {
35 test("Validaciones de inserción de cuenta", () => {
37 // La moneda de la cuenta es obligatoria. insercion abortada. ;
58 // caso 3: fallo por balance inicial nulo
59 expect(insertarCuenta(user, {
60 id: 5,
61 user: 1,
62 accountName: "Tarjeta de Crédito",
63 accountType: AccountTypeOptions.CREDIT_CARD,
64 currency: "USD",
65 createdAt: new Date(),
66 isActive: true,
67 cachedBalance: null as any,
68 })).toBe('El balance inicial de la cuenta es obligatorio. Inserción abortada.');
```

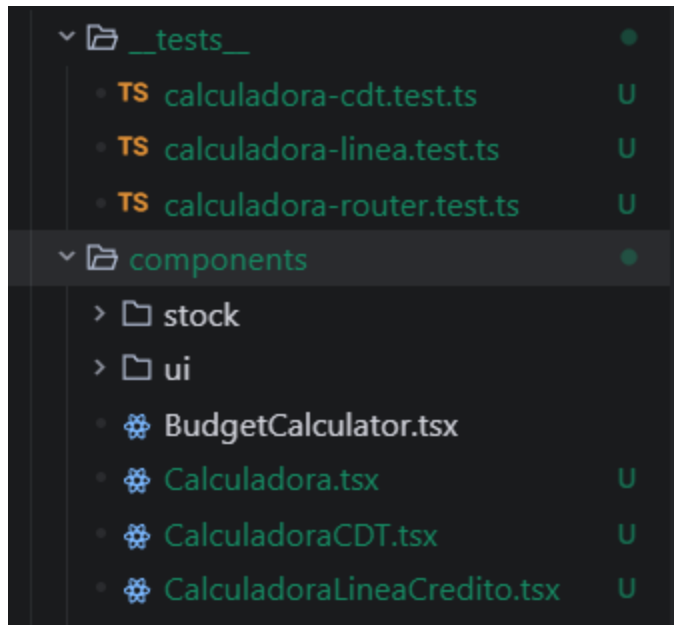
The terminal output shows the test results:

```
PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS
+ UBudget npm test
> test
> jest
PASS src/account/test.ts (5.879 s)
  Cuenta
    ✓ Insertar cuenta en usuario (4 ms)
    ✓ Validaciones de inserción de cuenta (1 ms)
    ✓ Desactivar cuenta (1 ms)
Test Suites: 1 passed, 1 total
Tests: 3 passed, 3 total
Snapshots: 0 total
Time: 6.134 s
Ran all test suites.
+ UBudget
```

### 3.3 Pruebas realizadas por Erfán Andrés Triana Duque

Descripción de la funcionalidad probada: Ventana de calculadora multifuncional, y correcto funcionamiento de calculadora CDT y calculadora de línea de crédito.

La pruebas se construyen para los archivos componentes ***Calculadora.tsx***, ***CalculadoraCDT.tsx*** y ***CalculadoraLineaCredito.tsx***



Código del test:

#### ***calculadora-cdt.test.ts***

```
import { describe, it, expect } from 'vitest'
import { calcularCDTNeto, tasaMensualDesdeEA } from
'../components/CalculadoraCDT'

describe("Calculadora de CDT", () => {
  it('calcula neto al vencimiento con capitalización y retención', () => {
    const res = calcularCDTNeto('vencimiento', 5_000_000, 12, 12, 4)
    expect(res.interes).toBeGreaterThan(550_000)
    expect(res.interes).toBeLessThan(610_000)
    expect(res.neto).toBeGreaterThan(5_560_000)
    expect(res.neto).toBeLessThan(5_590_000)
  })

  it('en pago periódico el neto es menor que al vencimiento (sin
capitalizar)', () => {
    const vto = calcularCDTNeto('vencimiento', 5_000_000, 12, 12, 4)
    const per = calcularCDTNeto('periodico', 5_000_000, 12, 12, 4)
    expect(per.neto).toBeLessThan(vto.neto)
  })

  it('la retención corresponde al porcentaje sobre el interés', () => {
    const tm = tasaMensualDesdeEA(10)
    const res = calcularCDTNeto('periodico', 4_000_000, 10, 6, 4)
```

```

    const interesEsperado = 4_000_000 * tm * 6
    expect(Math.abs(res.interes - interesEsperado)).toBeLessThan(1)
    expect(Math.abs(res.retencion - interesEsperado *
0.04)).toBeLessThan(1)
  })
})

describe("Calculadora de CDT - Casos Límite", () => { // Caso Límite 1:Cero
Capital

it('maneja capital inicial de 0', () => { const res =
calcularCDTNeto('vencimiento', 0, 12, 12, 4) expect(res.interes).toBe(0)
expect(res.retencion).toBe(0) expect(res.neto).toBe(0) }) // Caso Límite 2:
Cero Meses

it('maneja plazo de 0 meses', () => { const res =
calcularCDTNeto('periodico', 5_000_000, 12, 0, 4)
expect(res.interes).toBe(0) expect(res.retencion).toBe(0)
expect(res.neto).toBe(5_000_000) // Solo se devuelve el capital }) // Caso
Límite 3: Cero Tasa EA (para tasaMensualDesdeEA)
it('tasaMensualDesdeEA devuelve 0 para TasaEA de 0', () => {
expect(tasaMensualDesdeEA(0)).toBe(0) })
}

```

### **calculadora-linea.test.ts**

```

import { describe, it, expect } from 'vitest'
import { calcularCuotaCredito, generarAmortizacionCredito } from
'../components/CalculadoraLineaCredito'

describe('Calculadora de Línea de Crédito', () => {
  it('calcula la cuota mensual incluyendo amortización francesa', () => {
    const cuota = calcularCuotaCredito(10_000_000, 24, 36, 0)
    expect(cuota).toBeGreaterThan(350_000)
    expect(cuota).toBeLessThan(410_000)
  })

  it('genera tabla de amortización con saldo final cercano a 0', () => {
    const cuota = calcularCuotaCredito(10_000_000, 24, 36, 0)
    const schedule = generarAmortizacionCredito(10_000_000, 24, 36, cuota,
0)
    const last = schedule[schedule.length - 1]
    expect(schedule.length).toBe(36)
    expect(last.saldoFinal).toBeLessThan(100)
  })
}

```

```

    })

    it('mantiene coherencia entre total pagado e intereses + capital', () => {
      const cuota = calcularCuotaCredito(5_000_000, 18, 24, 0)
      const schedule = generarAmortizacionCredito(5_000_000, 18, 24, cuota, 0)
      const totalPagado = cuota * 24
      const totalInteres = schedule.reduce((s, r) => s + r.interes, 0)
      const capitalPagado = 5_000_000 - schedule[schedule.length - 1].saldoFinal
      expect(Math.abs(totalPagado - (totalInteres + capitalPagado))).toBeLessThan(200)
    })
  })
})

```

#### ***calculadora-router.test.ts***

```

import { describe, it, expect } from 'vitest'
import { resolveCalculadoraTarget } from '../components/Calculadora'

describe('Navegación de Calculadora', () => {
  it('resuelve acción línea de crédito', () => {
    expect(resolveCalculadoraTarget('linea')).toBe('linea')
  })

  it("resuelve acción CDT", () => {
    expect(resolveCalculadoraTarget('cdt')).toBe('cdt')
  })

  it('default para acciones desconocidas', () => {
    expect(resolveCalculadoraTarget('otra')).toBe('none')
  })
})

```

Casos límite documentados:

- Caso 1: Cero Capital
- Caso 2: Cero Meses
- Caso 3: Cero Tasa EA

Evidencia de ejecución:

```
✓ src/__tests__/calculadora-cdt.test.ts (3)
✓ src/__tests__/calculadora-linea.test.ts (3)
✓ src/__tests__/calculadora-router.test.ts (3)

Test Files  3 passed (3)
Tests       9 passed (9)
Start at    22:04:08
Duration    2.33s (transform 384ms, setup 0ms, collect 1.94s, tests 18ms, envi
```

### 3.4 Pruebas realizadas por Juan David Yopasa Maldonado

Descripción de la funcionalidad probada:

Se agregaron 3 funciones que ayudan a procesar textos sobre noticias financieras, la primera funcion transforma los titulos a formatos mas legibles de leer, la segunda se en carga de encontrar los tickers de las empresas, y la tercera para generar el resumen y saber si el stock subio , bajo o permaneció estable.

Código del test:

```
//capitaliza el titulo
export function capitalizarTitulo(texto:string):string {
  return texto
    .split(" ")
    .map(palabra => palabra.charAt(0).toUpperCase() +
palabra.slice(1).toLowerCase())
    .join(" ");
}
//extrae los tickers validos e ignora los q no son comunes
export function obtenerTickers(texto:string):string[] {
  const listaProhibida = new Set([
    "sube", "suben", "cae", "hoy", "y", "sin", "tickers",
    "por", "ventas", "demanda", "estable"
  ]);
  return (texto.match(/\b[a-zA-Z]{2,5}\b/g) || [])
    .filter(palabra => !listaProhibida.has(palabra.toLowerCase()))
    .map(palabra => palabra.toUpperCase());
}
//resume cada titular
export function resumirTitulos(titulares:string[]):string {
  return titulares
    .map(t => {
```

```

    const sube = t.toLowerCase().includes("sube");
    const cae = t.toLowerCase().includes("cae");
    const ticker = (t.match(/\b[A-Za-z]{2,5}\b/) || [""])[0].toUpperCase();
    if (!ticker) return "";
    if (sube) return `${ticker} subio`;
    if (cae) return `${ticker} cayo`;
    return `${ticker}=`;
  })
  .filter(Boolean)
  .join(" ");
}

```

Casos límite documentados:

- Caso 1 (títulos cortos):

Revisar como reacciona las funciones cuando no se tiene mucha información, así que es para asegurar de no falle con entradas mínimas

- Caso 2 (tickers falsos):

Ayuda a comprobar que sea capaz de detectar el ticker correctamente, pues hay palabras que cumplen el patron de un ticker pero en realidad son palabras de lenguajes natural

- Caso 3 (tirulares ambiguos o incompletos):

Se prueba q pasa cuando el titular no indica claramente si el stock subio o bajo, el objetivo es validar que no asuma un movimiento y devuelva un resultado erróneo cuando el titular no es explicito

Evidencia de ejecución:

```

Test Suites: 1 passed, 1 total
Tests:      3 passed, 3 total
Snapshots:  0 total
Time:       30.725 s
Ran all test suites.

```