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

Informe Testing

Juan Esteban Muñoz Muñoz
Omar Chaparro Flechas
Felipe Duque Jimenez

Introducción

El testing es una forma de asegurarnos de que nuestro juego funciona bien y sin errores antes de que los jugadores lo usen. En este taller, vamos a hacer pruebas en nuestro juego tipo Wordle para entender cómo probar diferentes partes del código y asegurarnos de que todo marche correctamente.

Vamos a revisar cosas como si el tablero se muestra bien, si las palabras se eligen de forma correcta y si el juego responde como esperamos. Para esto, usaremos herramientas como Jest y React Testing Library, que nos ayudan a comprobar que nuestro código hace lo que debe.

El objetivo es aprender más sobre testing, mejorar nuestro código y sentirnos más cómodos probando nuestras aplicaciones.

Test 1

Nombre	Tipo de prueba	Descripción breve del componente probado	Herramienta o framework usado
Omar Chaparro	Unitario	Este conjunto de pruebas verifica la función VerifyWord, que evalúa un intento de palabra en comparación con la palabra objetivo en un juego. Se prueban diferentes casos, incluyendo coincidencias exactas, letras en posiciones incorrectas, letras ausentes	Jest

Código test:

```javascript
import verifyWord from "./gameLogic";

describe("Función verifyWord", () => {
	test("Todas las letras correctas y en la posición correcta", () => {
	expect(verifyWord("apple", "apple")).toEqual([2, 2, 2, 2, 2]);
	});

	test("Todas las letras incorrectas", () => {
	expect(verifyWord("apple", "storm")).toEqual([0, 0, 0, 0, 0]);
	});

	test("Algunas letras en la posición correcta, otras incorrectas", ()
=> {
	expect(verifyWord("apple", "apron")).toEqual([2, 2, 0, 0, 0]);
	});

	test("Algunas letras en la posición incorrecta", () => {
	expect(verifyWord("apple", "pleap")).toEqual([1, 1, 1, 1, 1]);
	});
});
```

Resultado:

```
console.log
  [ 2, 2, 2, 2, 2 ]

    at verifyWord (src/firebase/gameLogic.js:29:13)

console.log
  [ 0, 0, 0, 0, 0 ]

    at verifyWord (src/firebase/gameLogic.js:29:13)

console.log
  [ 2, 2, 0, 0, 0 ]

    at verifyWord (src/firebase/gameLogic.js:29:13)

console.log
  [ 1, 1, 1, 1, 1 ]

    at verifyWord (src/firebase/gameLogic.js:29:13)

PASS src/firebase/gameLogic.test.js
Función verifyWord
  ✓ Todas las letras correctas y en la posición correcta (24 ms)
  ✓ Todas las letras incorrectas (2 ms)
  ✓ Algunas letras en la posición correcta, otras incorrectas (2 ms)
  ✓ Algunas letras en la posición incorrecta (1 ms)
```

Test 2

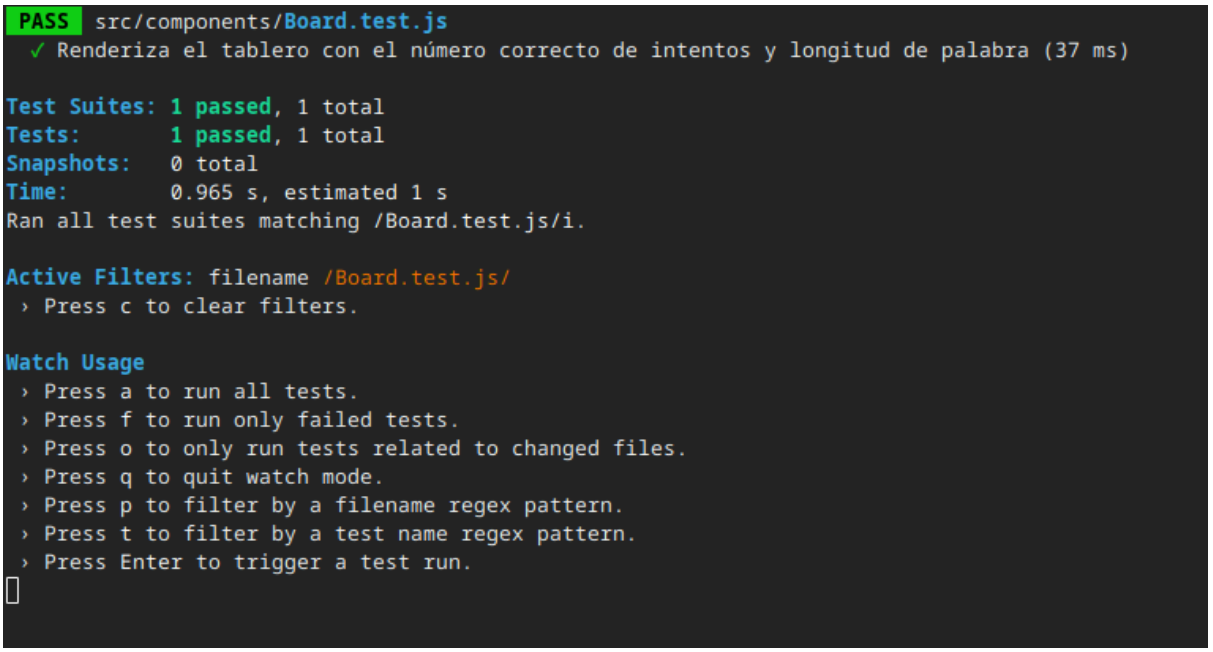
Nombre	Tipo de prueba	Descripción breve del componente probado	Herramienta o framework usado
Felipe Duque	Unitario	Este test verifica que el componente Board renderiza correctamente la cantidad esperada de intentos (filas) y que cada intento contiene el número adecuado de celdas según la longitud de la palabra.	Jest, React Testing Library

Código Test:

```javascript
import { render, screen } from '@testing-library/react';
import Board from './Board';

test('Renderiza el tablero con el número correcto de intentos y longitud de palabra', () => {
	const wordLength = 5;
	const { container } = render(
	<Board
		wordLength={wordLength}
		checkWord={() => {}}
		setPlayerAttempts={() => {}}
		setBoardActions={() => {}}
		enemyColors={[]}
	/>
	);

	const rows = container.querySelectorAll('.row');
	expect(rows.length).toBe(6);

	rows.forEach(row => {
	const cells = row.children;
	expect(cells.length).toBe(wordLength);
```

```
        });
  });
```

Resultado:


PASS src/components/Board.test.js
✓ Renderiza el tablero con el número correcto de intentos y longitud de palabra (37 ms)
Test Suites: 1 passed, 1 total
Tests: 1 passed, 1 total
Snapshots: 0 total
Time: 0.965 s, estimated 1 s
Ran all test suites matching /Board.test.js/i.
Active Filters: filename /Board.test.js/
› Press c to clear filters.
Watch Usage
› Press a to run all tests.
› Press f to run only failed tests.
› Press o to only run tests related to changed files.
› Press q to quit watch mode.
› Press p to filter by a filename regex pattern.
› Press t to filter by a test name regex pattern.
› Press Enter to trigger a test run.

Test 3

Nombre	Tipo de prueba	Descripción breve del componente probado	Herramienta o framework usado
Juan Esteban Muñoz	Unitario	Prueba la función getRandomWord, que obtiene una palabra aleatoria de Firestore según el modo de juego seleccionado. Se verifican los siguientes escenarios: • Devuelve una palabra válida si Firestore contiene datos. • Retorna null si no hay coincidencias en la base de datos. • Maneja errores correctamente y retorna null.	Jest

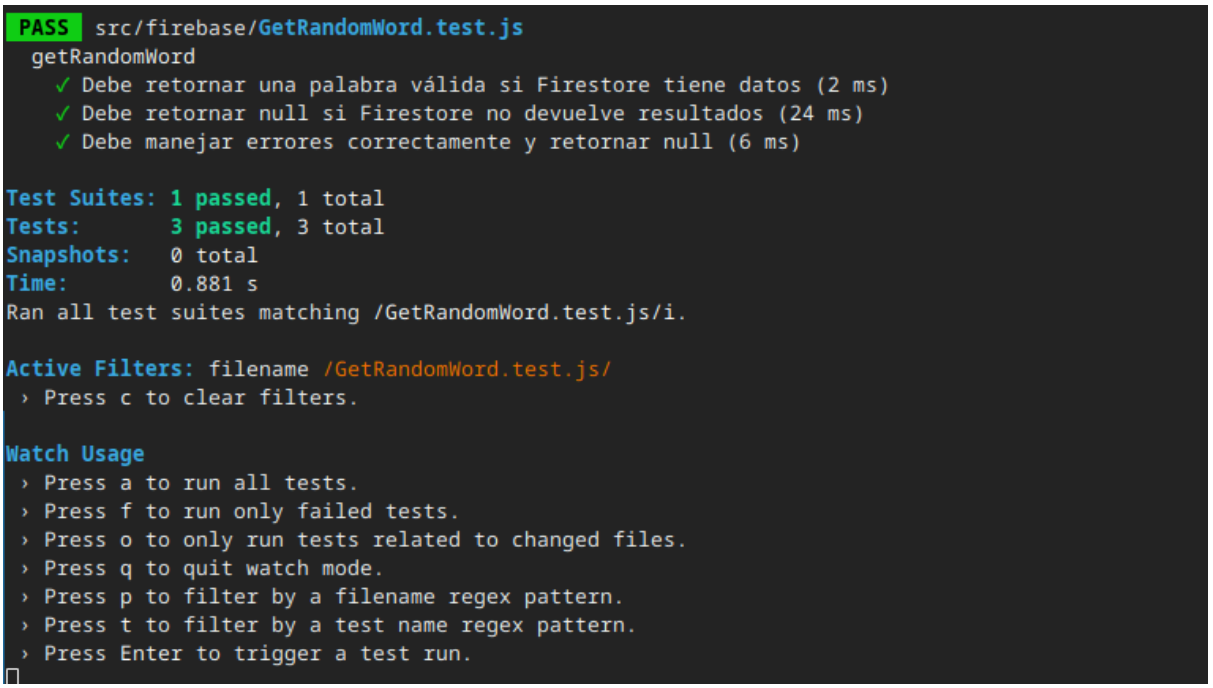
Código Test:

JavaScript

```
import { getRandomWord } from './GetRandomWord';
import { getDocs } from 'firebase/firestore';

// Mock de Firebase Firestore
jest.mock('./config', () => ({
	db: {} // Mock del objeto db para evitar el error
}));

jest.mock('firebase/firestore', () => ({
	collection: jest.fn(),
	query: jest.fn(),
	where: jest.fn(),
	getDocs: jest.fn()
}));

describe('getRandomWord', () => {
	beforeEach(() => {
	jest.clearAllMocks();
	});

	test('Debe retornar una palabra válida si Firestore tiene datos',
async () => {
	const mockDocs = [{ data: () => ({ word: "prueba" }) }];
	getDocs.mockResolvedValue({ empty: false, docs: mockDocs });

	const word = await getRandomWord("NORMAL");
```

```
        expect(word).toBe("prueba");
        });

        test('Debe retornar null si Firestore no devuelve resultados', async
  () => {
        getDocs.mockResolvedValue({ empty: true, docs: [] });

        const word = await getRandomWord("NORMAL");
        expect(word).toBeNull();
        });

        test('Debe manejar errores correctamente y retornar null', async () =>
  {
        getDocs.mockRejectedValue(new Error("Error de conexión"));

        const word = await getRandomWord("NORMAL");
        expect(word).toBeNull();
        });
  });
```

Resultado



Lecciones aprendidas y dificultades

Lecciones Aprendidas:

- **Mocks para Firebase:** Probar funciones que dependen de Firestore sin **mocks** generó errores y ralentizó las pruebas. Aprendimos a simular respuestas para evitar depender de la base de datos real.
- **Tests simples son mejores:** Intentamos hacer pruebas muy grandes y complejas, pero dividirlas en tests pequeños y específicos resultó más efectivo.
- **Importancia del diseño de tests:** Un buen diseño desde el inicio facilita la depuración y mejora la calidad del código.
- **Pruebas en el board:** Testear la actualización en tiempo real del **board** fue un reto, ya que debía reflejar correctamente los cambios de estado.
- **Selección aleatoria de palabras:** Testear la función que obtiene palabras aleatorias fue complicado porque el resultado cambia cada vez. Usar un **mock** permitió verificar que se obtienen palabras dentro de los criterios definidos.

Dificultades:

- **Errores inesperados:** Algunos tests fallaban sin razón aparente, lo que nos llevó a revisar y mejorar la lógica del código.
- **Configuración de Firebase:** Problemas con valores faltantes como projectId hicieron que los tests no corrieran al inicio.
- **Dependencia de la base de datos:** Probar la selección de palabras aleatorias directamente en Firestore fue lento y poco confiable. Usar **mocks** solucionó este problema.
- **Simulación de eventos en el juego:** Testear la interacción del usuario con el **board** y la actualización en tiempo real resultó más complejo de lo esperado.