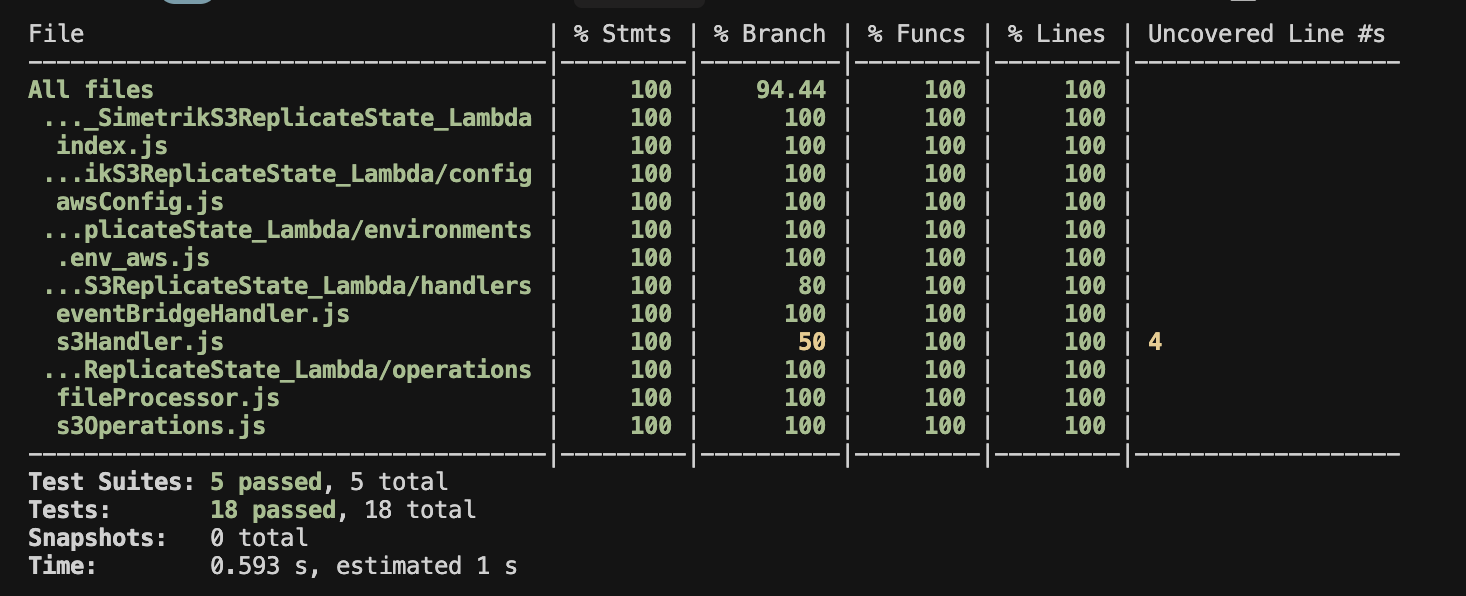
****

**1 XX**

**Solución Implementada**

**1. Creé constantes al inicio del archivo:**

**javascript**

**Apply to index.test.j...**

*// Constantes para mensajes de respuesta*

const SUCCESS\_MESSAGE = "Operación completada con éxito.";

const UNKNOWN\_EVENT\_MESSAGE = "Evento desconocido.";

const ERROR\_PROCESSING\_MESSAGE = "Error al procesar el evento.";

**2. Reemplacé todas las ocurrencias duplicadas:**

* "Operación completada con éxito." → SUCCESS\_MESSAGE
* "Evento desconocido." → UNKNOWN\_EVENT\_MESSAGE
* "Error al procesar el evento." → ERROR\_PROCESSING\_MESSAGE

**Beneficios de esta solución:**

✅ **Elimina la duplicación** - Los mensajes ahora están definidos en un solo lugar✅ **Facilita el mantenimiento** - Si necesitas cambiar un mensaje, solo lo cambias en la constante✅ **Mejora la legibilidad** - Los nombres de las constantes son más descriptivos✅ **Reduce errores** - No hay riesgo de escribir mal el mensaje en diferentes lugares

**2 XX**

**Creé constantes para códigos de estado HTTP:**

**javascript**

**Apply to index.test.j...**

*// Constantes para códigos de estado HTTP*

const HTTP\_STATUS\_OK = 200;

const HTTP\_STATUS\_BAD\_REQUEST = 400;

const HTTP\_STATUS\_INTERNAL\_SERVER\_ERROR = 500;

**2. Reemplacé todos los números mágicos:**

* 200 → HTTP\_STATUS\_OK
* 400 → HTTP\_STATUS\_BAD\_REQUEST
* 500 → HTTP\_STATUS\_INTERNAL\_SERVER\_ERROR

**Beneficios de esta solución:**

✅ **Elimina números mágicos** - Los códigos HTTP ahora tienen nombres descriptivos✅ **Mejora la legibilidad** - Es más claro qué significa cada código✅ **Facilita el mantenimiento** - Si necesitas cambiar un código, solo lo cambias en la constante✅ **Reduce errores** - No hay riesgo de escribir mal un número✅ **Sigue estándares** - Los nombres de las constantes siguen convenciones comunes

Estructura final el archivo

// Constantes para códigos de estado HTTP

const HTTP\_STATUS\_OK = 200;

const HTTP\_STATUS\_BAD\_REQUEST = 400;

const HTTP\_STATUS\_INTERNAL\_SERVER\_ERROR = 500;

// Constantes para mensajes de respuesta

const SUCCESS\_MESSAGE = "Operación completada con éxito.";

const UNKNOWN\_EVENT\_MESSAGE = "Evento desconocido.";

const ERROR\_PROCESSING\_MESSAGE = "Error al procesar el evento.";

3XXXX

**Solución Implementada**

**1. Creé una nueva constante:**

**javascript**

**Apply to index.test.j...**

const MOCK\_ERROR\_MESSAGE = "Error simulado";

**2. Reemplacé todas las ocurrencias duplicadas:**

* new Error("Error simulado") → new Error(MOCK\_ERROR\_MESSAGE)
* "Error simulado" en buildResponse → MOCK\_ERROR\_MESSAGE
* error: "Error simulado" → error: MOCK\_ERROR\_MESSAGE

**Beneficios de esta solución:**

✅ **Elimina la duplicación** - El mensaje de error ahora está definido en un solo lugar✅ **Facilita el mantenimiento** - Si necesitas cambiar el mensaje de error, solo lo cambias en la constante✅ **Mejora la consistencia** - Asegura que el mismo mensaje se use en todos los lugares✅ **Reduce errores** - No hay riesgo de escribir mal el mensaje en diferentes lugares

**Estructura final de constantes:**

*// Constantes para códigos de estado HTTP*

const HTTP\_STATUS\_OK = 200;

const HTTP\_STATUS\_BAD\_REQUEST = 400;

const HTTP\_STATUS\_INTERNAL\_SERVER\_ERROR = 500;

*// Constantes para mensajes de respuesta*

const SUCCESS\_MESSAGE = "Operación completada con éxito.";

const UNKNOWN\_EVENT\_MESSAGE = "Evento desconocido.";

const ERROR\_PROCESSING\_MESSAGE = "Error al procesar el evento.";

const MOCK\_ERROR\_MESSAGE = "Error simulado";

XXXX S3handler

**Solución Implementada**

**1. Creé constantes para los valores de prueba:**

**javascript**

**Apply to s3Handler.te...**

*// Constantes para valores de prueba*

const TEST\_BUCKET\_NAME = "test-bucket";

const TEST\_FILE\_KEY = "folder/file1.txt";

**2. Reemplacé todas las ocurrencias duplicadas:**

* "test-bucket" → TEST\_BUCKET\_NAME (6 veces)
* "folder/file1.txt" → TEST\_FILE\_KEY (4 veces)

**Beneficios de esta solución:**

✅ **Elimina la duplicación** - Los valores de prueba ahora están definidos en un solo lugar✅ **Facilita el mantenimiento** - Si necesitas cambiar el nombre del bucket o archivo de prueba, solo lo cambias en las constantes✅ **Mejora la consistencia** - Asegura que los mismos valores se usen en todos los tests✅ **Reduce errores** - No hay riesgo de escribir mal los valores en diferentes lugares✅ **Mejora la legibilidad** - Los nombres de las constantes son más descriptivos

**Estructura final del archivo:**

**javascript**

**Apply to s3Handler.te...**

*// Constantes para valores de prueba*

const TEST\_BUCKET\_NAME = "test-bucket";

const TEST\_FILE\_KEY = "folder/file1.txt";

**Verificación:**

Ahora puedes ejecutar los tests para asegurarte de que todo funciona correctamente:

bash

Apply to s3Handler.te...

Run

npm test

¿Te gustaría que revise si hay otros literales duplicados en otros archivos del proyecto?

FileProcessor.test

XXX

**Solución Implementada**

**1. Creé funciones auxiliares para verificaciones comunes:**

**javascript**

**Apply to fileProcesso...**

*// Funciones auxiliares para verificaciones comunes*

const verifyGetFileReplicationStatusCalled = () => {

  expect(getFileReplicationStatus).toHaveBeenCalledTimes(1);

  expect(getFileReplicationStatus).toHaveBeenCalledWith(bucketName, objectKey);

};

const verifyUpdateFileMetadataCalled = () => {

  expect(updateFileMetadata).toHaveBeenCalledTimes(1);

  expect(updateFileMetadata).toHaveBeenCalledWith(bucketName, objectKey);

};

const verifyUpdateFileMetadataNotCalled = () => {

  expect(updateFileMetadata).not.toHaveBeenCalled();

};

**2. Organicé los tests en grupos lógicos usando describe anidados:**

* **"cuando el estado de replicación es 'REPLICA'"** - Tests para el caso exitoso
* **"cuando el estado de replicación no es 'REPLICA'"** - Tests para el caso donde no se actualiza
* **"manejo de errores"** - Tests para casos de error

**3. Simplifiqué cada test individual:**

* Cada test ahora es más corto y enfocado
* Usa las funciones auxiliares para las verificaciones
* Elimina la duplicación de código

**Beneficios de esta refactorización:**

✅ **Cumple con el límite de líneas** - Cada función ahora tiene menos de 35 líneas✅ **Mejora la legibilidad** - El código está mejor organizado y es más fácil de entender✅ **Reduce la duplicación** - Las verificaciones comunes están en funciones reutilizables✅ **Facilita el mantenimiento** - Es más fácil agregar nuevos tests o modificar los existentes✅ **Mejora la estructura** - Los tests están agrupados lógicamente por funcionalidad

**Estructura final:**

* **Función principal**: describe("processFile", () => { ... }) - Ahora más corta
* **Funciones auxiliares**: Para verificaciones comunes
* **Grupos de tests**: Organizados por escenarios
* **Tests individuales**: Más cortos y enfocados

XXXXXX

Index

**Solución Implementada**

**1. Creé funciones auxiliares para crear eventos de prueba:**

**javascript**

**Apply to index.test.j...**

*// Funciones auxiliares para crear eventos de prueba*

const createS3Event = () => ({ ... });

const createEventBridgeEvent = () => ({ ... });

const createUnknownEvent = () => ({ ... });

**2. Creé funciones auxiliares para verificaciones comunes:**

**javascript**

**Apply to index.test.j...**

*// Funciones auxiliares para verificaciones comunes*

const verifySuccessResponse = (*result*) => { ... };

const verifyErrorResponse = (*result*, *statusCode*, *message*, *error* = undefined) => { ... };

**3. Organicé los tests en grupos lógicos usando describe anidados:**

* **"manejo de eventos S3"** - Tests para eventos de S3
* **"manejo de eventos EventBridge"** - Tests para eventos de EventBridge
* **"manejo de eventos desconocidos"** - Tests para eventos no reconocidos
* **"manejo de errores"** - Tests para casos de error

**4. Simplifiqué cada test individual:**

* Cada test ahora es más corto y enfocado
* Usa las funciones auxiliares para crear eventos y verificaciones
* Elimina la duplicación de código

**Beneficios de esta refactorización:**

✅ **Cumple con el límite de líneas** - Cada función ahora tiene menos de 35 líneas✅ **Mejora la legibilidad** - El código está mejor organizado y es más fácil de entender✅ **Reduce la duplicación** - Los eventos de prueba y verificaciones están en funciones reutilizables✅ **Facilita el mantenimiento** - Es más fácil agregar nuevos tests o modificar los existentes✅ **Mejora la estructura** - Los tests están agrupados lógicamente por tipo de evento✅ **Aumenta la reutilización** - Las funciones auxiliares pueden usarse en otros tests

**Estructura final:**

* **Función principal**: describe("handler function", () => { ... }) - Ahora más corta
* **Funciones auxiliares**: Para crear eventos y verificaciones
* **Grupos de tests**: Organizados por tipo de evento
* **Tests individuales**: Más cortos y enfocados

XXXXXX