**Test Doubles - Equipo 1**

1. Resumen

Implementamos los cinco tipos de Test Doubles utilizando Jest: Dummy, Stub, Mock, Fake y Spy, aplicando ejemplos reales y personalizados a partir de nuestro proyecto.

1. Enlace al repositorio

<https://github.com/Sofiazugasti/test_doubles_equipo1>

1. ¿Qué Test Doubles se implementaron?

En el proyecto implementamos los cinco tipos de Test Doubles: Dummy, Stub, Mock, Fake y Spy. Para el caso del Dummy, creamos una clase sencilla que solo cumple con la interfaz mínima requerida por el sistema, sin agregar lógica adicional. En específico, usamos un Dummy para simular una organización al crear un proyecto solidario, permitiendo así enfocar la prueba únicamente en la lógica del proyecto.

🟠 Dummy Utilizamos un Dummy para simular una organización al momento de crear un proyecto solidario. Este objeto no tenía lógica interna, pero cumplía con la estructura mínima requerida por el sistema. Esto nos permitió centrarnos en probar la lógica del proyecto sin necesidad de definir toda la lógica de la organización. Aprendimos que los Dummies son útiles cuando solo se necesita "rellenar" un parámetro sin afectar el flujo de la prueba.

🟡 Stub Creamos un Stub que devolvía valores específicos como si consultara una base de datos, por ejemplo, cuando simulamos que ya existía un proyecto con el mismo nombre. Gracias a esto, pudimos probar cómo responde el sistema ante condiciones controladas. Esto nos ayudó a entender cómo los Stubs son ideales para validar rutas específicas de ejecución sin lógica adicional.

🔵 Mock Implementamos un Mock para simular la lógica de una base de datos, con funciones como hasDuplicateApplication, validateFields y saveApplication. Con esto pudimos verificar si se llamaban correctamente, con los parámetros esperados, y cuántas veces. Aprendimos que los Mocks nos permiten validar interacciones específicas y son esenciales cuando queremos comprobar cómo se comunican nuestras clases entre sí.

🟢 Fake El Fake que desarrollamos fue un servicio de base de datos simulado con datos reales en memoria. Implementaba una lógica funcional (por ejemplo, getProjectById) y fue clave para probar si había cupo disponible en un proyecto solidario sin conectarnos a Supabase. Esto nos enseñó que los Fakes son útiles cuando queremos ejecutar pruebas más completas, sin necesidad de infraestructura externa.

🟣 Spy Usamos Spies con jest.fn() para observar funciones como getApplication y saveApplication, con el fin de verificar si se ejecutaban correctamente durante el proceso de postulación. Esto nos permitió confirmar que ciertas acciones no ocurrían cuando no debían, como evitar duplicaciones. Aprendimos que los Spies son ideales para confirmar el comportamiento indirecto de funciones dentro del flujo.

1. ¿Qué aprendimos al integrarlos?

🟠 Dummy Aprendimos que un Dummy no necesita lógica interna, solo cumplir con la forma de un objeto real. Nos ayudó a enfocarnos en probar la clase principal sin preocuparnos por dependencias irrelevantes en ese momento.

🟡 Stub Al usar Stubs, entendimos cómo simular respuestas específicas para probar distintos caminos de ejecución sin conectarnos a sistemas externos. Fue útil para simular proyectos duplicados o respuestas controladas.

🔵 Mock Con los Mocks vimos cómo validar que una función haya sido llamada correctamente. Aprendimos a verificar interacciones entre objetos y asegurar que se llamen funciones clave bajo ciertas condiciones.

🟢 Fake El Fake nos permitió simular un servicio funcional (como una base de datos en memoria) con una implementación realista. Comprendimos que es ideal para pruebas más completas sin usar servicios externos.

🟣 Spy Los Spies nos enseñaron a observar si ciertas funciones se ejecutan (o no) durante el flujo. Aprendimos que es útil cuando queremos verificar interacciones sin necesidad de reemplazar toda la lógica.

1. ¿Qué problemas enfrentaron y cómo los resolvieron?

🟠 Dummy Tuvimos problemas al principio por no incluir los atributos mínimos requeridos en el Dummy, lo cual causaba fallos. Lo resolvimos revisando la clase principal y ajustando el Dummy con lo esencial.

🟡 Stub Nos equivocamos al no usar mockResolvedValue() para funciones asíncronas, lo que causaba errores. Aprendimos que incluso stubs deben imitar correctamente el comportamiento real, incluyendo promesas.

🔵 Mock Al inicio confundimos los mocks con los spies y usábamos mocks donde solo queríamos observar. Lo resolvimos diferenciando cuándo debíamos validar llamadas y cuándo simular comportamiento.

🟢 Fake El reto fue entender que un Fake debe tener lógica realista, pero no completa. Nos ayudó a practicar cómo diseñar servicios con dependencias inyectadas y simular escenarios más complejos sin conectarnos a Supabase.

🟣 Spy Fallamos al principio en entender que los spies no devuelven valores automáticamente. Lo resolvimos combinando jest.fn() con funciones predefinidas y entendiendo cuándo es mejor un spy que un stub o mock.

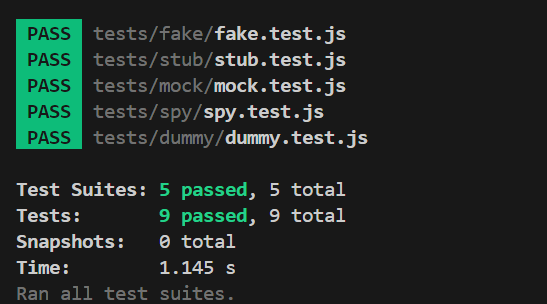


Figura 1. Evidencia de pruebas ejecutadas exitosamente.