

**Universidad Tecnológica Nacional**

**Facultad Regional Rosario**

Especialidad: Ing. en Sistemas de Información.

Asignatura: Desarrollo de software

Comisión: 302

Fecha: 04/10/24

**TRABAJO PRÁCTICO:**

**Testing Library**

|  |  |
| --- | --- |
| **Alumno** | **Legajo** |
| Mondino, Juan Cruz | 51922 |
| Giampietro, Gustavo | 50671 |
| Dequelli, Gabriel | 44073 |

Índice

[Introducción 2](#_Toc178154670)

[Objetivo 2](#_Toc178154671)

[Tema y alcance 2](#_Toc178154672)

[Definir una propuesta 3](#_Toc178154673)

[Propósito 3](#_Toc178154674)

[¿Cómo lo hace? 3](#_Toc178154675)

[Ventajas de Testing Library 4](#_Toc178154676)

[¿Cómo se usa? 4](#_Toc178154677)

[Conclusión 4](#_Toc178154678)

[Establecer características de evaluación 4](#_Toc178154679)

[Investigar las distintas tecnologías 5](#_Toc178154680)

[Implementar un proyecto de pequeña escala para comparar entre las tecnologías que investiga cada grupo 5](#_Toc178154681)

[Conclusión 5](#_Toc178154682)

# Introducción

Este trabajo se realiza con el fin de comparar diferentes Tests Suitets o Frameworks, para tener la capacidad de elegir el que mejor se ajuste a las necesidades de cada aplicación.

A este trabajo se lo denomina PoC (Proof of Concept), este es un estudio de viabilidad que se lleva adelante previo al desarrollo o implementación de una nueva idea, proyecto o producto. En particular en el área de desarrollo de software esto puede aplicar a un nuevo producto de software o también a la aplicación de nuevas técnica, metodología o tecnologías de desarrollo de software. Una PoC sirve a todos los involucrados y stakeholders para evaluar la conveniencia práctica de realizar un proyecto o implementar nuevas técnicas, metodologías o tecnologías; más allá de las conveniencias teóricas.

La prueba de concepto en una fase temprana permite:

* Identificar los posibles riesgos y obstáculos.
* Determinar la viabilidad del proyecto.
* Proporcionar pruebas evidentes de funcionalidad a las partes interesadas antes de invertir más tiempo, esfuerzo y dinero.

# Objetivo

Llevar adelante una investigación y Prueba de Concepto (PoC) de implementación de tecnologías de desarrollo de software, de forma intergrupal e implementarlo en un desarrollo o demostración técnica para evaluar su conveniencia.

# Tema y alcance

Los grupos deberán elegir de la lista propuesta o proponer una tecnología para investigar.

Las tecnologías propuestas se agrupan según el tema y los grupos de un mismo tema deberán trabajar de manera conjunta para llevar adelante la PoC.

Etapas:

1. Definir una propuesta.
2. Establecer características de evaluación.
3. Investigar las distintas tecnologías.
4. Implementar un proyecto de pequeña escala para comparar entre las tecnologías que investiga cada grupo.
5. Redactar un informe y conclusión conjunta de la PoC.
6. Realizar una presentación conjunta frente al curso en base a la implementación y al informe.

## Definir una propuesta

Nosotros elegimos Testing Library y compararemos esta tecnología con: Mock Service Worker, Mocha, Vitest y Jest. El objetivo de esta comparación es, como se aclaró anteriormente, conocer el funcionamiento de estas distintas tecnologías de testeo para tener la capacidad de decidir cuál es la mejor en cada caso.

Testing Library es una colección de bibliotecas enfocadas en facilitar la escritura de pruebas unitarias y de integración para aplicaciones web. Se utiliza principalmente con frameworks de JavaScript, como React, Angular, y Vue, pero tiene variantes para diferentes entornos y plataformas.

### Propósito

El objetivo principal de Testing Library es ayudar a los desarrolladores a realizar pruebas de componentes o aplicaciones basadas en la interacción del usuario, asegurando que las pruebas sean lo más cercanas posible al uso real de la interfaz. En lugar de centrarse en la implementación interna de los componentes, fomenta una filosofía de prueba orientada al comportamiento y la accesibilidad.

### ¿Cómo lo hace?

Esta librería está centrada en la interacción con la interfaz de usuario (UI) de manera similar a como lo haría un usuario real. Esto se logra mediante la selección de elementos basados en atributos semánticos y accesibles, como `role`, `label`, `text`, entre otros. Por lo tanto, Testing Library enfatiza el uso de consultas que reflejan cómo un usuario interactuaría con la aplicación.

Ejemplo de funciones comunes:

1. **Renderizado del componente:** Se utiliza `render` para montar un componente en un entorno de prueba. Render es una función que simula el montaje de un componente dentro de un entorno de pruebas. Esto permite realizar pruebas sobre el componente como si estuviera siendo visualizado por el usuario real, aunque el renderizado ocurre en un entorno de simulación sin necesidad de un navegador real.

2. **Búsqueda de elementos:** Provee métodos como `getByText`, `getByRole`, `getByLabelText`, entre otros, para seleccionar los elementos con los que se desea interactuar.

3. **Interacción del usuario:** Para simular eventos del usuario, se utiliza `userEvent`, que permite realizar acciones como clics, escritura de texto, etc.

4. **Afirmaciones:** Una vez realizadas las interacciones, se verifica el estado final de la UI mediante afirmaciones (`expect`) con `Jest` u otra herramienta de aserción. Una herramienta de aserción (o framework de aserción) es una biblioteca o conjunto de funciones que se utilizan en el contexto de pruebas automatizadas para verificar si un resultado esperado coincide con el resultado real producido por el código. Las aserciones son declaraciones que evalúan una condición lógica y determinan si una prueba pasa o falla en función de si la condición es verdadera o falsa. Las herramientas de aserción son esenciales porque permiten a los desarrolladores automatizar la validación del comportamiento esperado del código. Esto facilita la identificación de errores, mejora la confianza en los cambios de código y ayuda a garantizar la calidad del software.

### Ventajas de Testing Library

1. Pruebas centradas en el usuario: Las pruebas están orientadas a cómo el usuario interactúa con la aplicación, lo que mejora la fiabilidad y el valor de las pruebas.

2. Accesibilidad: Fomenta el uso de atributos accesibles.

3. Mantenimiento más sencillo: Al evitar pruebas basadas en la implementación interna de los componentes, las pruebas suelen ser más resistentes a cambios en el código.

4. Desacople de la implementación: No se preocupa por los detalles específicos de cómo se implementa un componente, sino por cómo este es visible y usable desde el punto de vista del usuario.

### ¿Cómo se usa?

Testing Library se utiliza principalmente dentro de un entorno de pruebas, en combinación con herramientas como Jest (para correr pruebas en JavaScript/TypeScript) o Karma (usado en Angular). Un ejemplo de uso en Angular sería:

#### Javascript

import { render, screen } from '@testing-library/angular';

import { MyComponent } from './my-component';

test('debería mostrar el texto', async () => {

await render(MyComponent, { componentProperties: { text: 'Hola Mundo' } });

const element = screen.getByText('Hola Mundo');

expect(element).toBeInTheDocument();

});

En este ejemplo, `render` monta el componente, luego `screen.getByText` selecciona el texto visible, y finalmente `expect` verifica que el texto está en el DOM.

### Conclusión

Testing Library es una herramienta poderosa y flexible para garantizar que los componentes funcionen correctamente desde la perspectiva del usuario final, lo que resulta en una experiencia de usuario de alta calidad y accesible.

## Establecer características de evaluación

Las pruebas que realizamos cubren una amplia gama de escenarios a la hora de la utilización de la aplicación:

* Verificación de texto.
* Verificación de la presencia del logo.
* Verificación de enlaces con target = "\_blank".
* Verificación del atributo href correcto en un enlace.
* Verificación de que el componente se renderiza sin errores.
* Verificación de clases CSS aplicadas.
* Simulación de escritura en un campo de texto.
* Verificación de que el texto enviado se muestra después de enviar el formulario.
* Verificación de que el envío del formulario vacío no genera errores.
* Verificación de que el botón está deshabilitado cuando el campo de texto está vacío.

## Investigar las distintas tecnologías

Comparación entre Testing Library con Jest, Vitest, Mocha y Mock Service Worker:

Testing Library vs Jest

- Propósito:

- Testing Library: Enfocado en la interacción del usuario, promueve pruebas que simulan el comportamiento real de la aplicación y el usuario.

- Jest: Un framework de pruebas completo para JavaScript, que incluye un motor de pruebas, un corredor de pruebas y herramientas de aserción. Es usado tanto para pruebas unitarias como de integración.

- Ámbito de uso:

- Testing Library: Se centra en pruebas de interfaz de usuario (UI) para frameworks como React, Angular y Vue, y busca asegurar que los componentes funcionen correctamente desde el punto de vista del usuario.

- Jest: Se usa tanto en el backend como en el frontend, y puede probar desde simples funciones hasta aplicaciones completas.

- Similitudes:

- Testing Library a menudo usa Jest como herramienta de aserción y para ejecutar las pruebas.

- Diferencias:

- Testing Library se enfoca más en pruebas funcionales y accesibilidad, mientras que Jest cubre pruebas a un nivel más amplio, incluyendo pruebas unitarias e integración sin estar limitado a la UI.

Testing Library vs Vitest

- Propósito:

- Testing Library: Simula cómo un usuario interactúa con la interfaz de usuario.

- Vitest: Un framework de pruebas moderno diseñado para trabajar específicamente con Vite. Es rápido y permite pruebas instantáneas, similar a Jest, pero optimizado para el ecosistema Vite.

- Ámbito de uso:

- Testing Library: Enfoque UI y pruebas de componentes en aplicaciones web.

- Vitest: Pruebas de aplicaciones frontend y backend, optimizado para aplicaciones construidas con Vite, pero no se limita a pruebas de UI.

- Similitudes:

- Ambas pueden trabajar en el frontend, pero Testing Library depende de Vitest (o Jest) para ejecutar las pruebas.

- Diferencias:

- Vitest está más orientado a la velocidad en el entorno de desarrollo y es particularmente útil para proyectos que utilizan Vite como herramienta de construcción.

Testing Library vs Mocha

- Propósito:

- Testing Library: Enfocado en pruebas de UI, validando interacciones del usuario.

- Mocha: Un framework de pruebas flexible para JavaScript que no tiene muchas dependencias integradas. Es usado para pruebas tanto del frontend como del backend.

- Ámbito de uso:

- Testing Library: Validación de componentes y simulación de la experiencia del usuario en interfaces gráficas.

- Mocha: Usado en aplicaciones de Node.js y en proyectos JavaScript generales, permitiendo flexibilidad en el entorno y elección de herramientas complementarias como bibliotecas de aserción.

- Similitudes:

- Ambos pueden ser utilizados en el frontend, y Mocha puede integrarse con Testing Library para ejecutar pruebas.

- Diferencias:

- Mocha no incluye aserciones ni simulaciones (mocks) por defecto, mientras que Testing Library se enfoca en proporcionar herramientas centradas en el usuario para pruebas de UI.

Testing Library vs Mock Service Worker (MSW)

- Propósito:

- Testing Library: Pruebas centradas en la UI, simulando la interacción con el DOM.

- Mock Service Worker (MSW): Se utiliza para simular APIs en pruebas y desarrollo, permitiendo interceptar solicitudes de red y responder con datos mock. Los datos mock son datos falsos o simulados que se crean para probar el comportamiento de una aplicación o sistema sin depender de fuentes de datos reales, como bases de datos o APIs externas. Estos datos imitan el formato y la estructura de los datos reales, permitiendo verificar cómo la aplicación manejaría las respuestas y errores de un servicio sin necesidad de conectarse a él.

- Ámbito de uso:

- Testing Library: Simula la interacción del usuario con la interfaz gráfica, verificando que los componentes se comporten correctamente.

- MSW: Simula interacciones con APIs sin necesidad de acceder a servicios reales, usado para probar cómo las aplicaciones manejan las respuestas del servidor.

- Similitudes:

- Ambos son útiles en pruebas de frontend: Testing Library prueba la UI, y MSW facilita la simulación de respuestas de API en dichas pruebas.

- Diferencias:

- MSW se centra en la simulación de redes y backend, mientras que Testing Library se enfoca en cómo los usuarios interactúan con la UI.

Resumen

- Testing Library está más orientado a la simulación del comportamiento del usuario y pruebas de UI.

- Jest y Vitest son frameworks más completos que permiten probar tanto frontend como backend, pero Testing Library a menudo depende de ellos para correr las pruebas.

- Mocha es un framework más flexible, que requiere bibliotecas adicionales para realizar ciertas tareas, como aserciones o mocks, y es menos enfocado en pruebas de UI que Testing Library.

- Mock Service Worker (MSW) complementa Testing Library cuando es necesario simular APIs en las pruebas, mientras que Testing Library se enfoca en la interacción visual.

En conjunto, Testing Library está especializado en pruebas de interfaces gráficas centradas en la accesibilidad y el comportamiento del usuario, mientras que Jest, Vitest y Mocha son frameworks de pruebas más generales que pueden ser utilizados tanto para el frontend como el backend. MSW, por su parte, ofrece una forma de simular interacciones de red y API, complementando herramientas como Testing Library.

## Implementar un proyecto de pequeña escala para comparar entre las tecnologías que investiga cada grupo

El proyecto de pequeña escala en el que se realiza el testeo es una aplicación web de prueba, generada por react con el comando:

npx create-react-app my-test-app

Por más que sea simple, tiene las funcionalidades más comunes (enumeradas anteriormente) de una página web.

# Conclusión