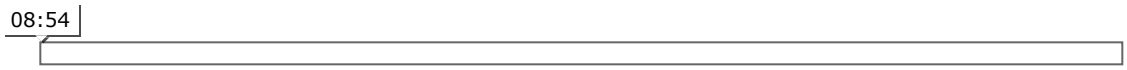
Watch: 🧮 **Las bases de Testing Library: testeando un componente simple**



Puedes encontrar el código de este vídeo en GitHub.

En este vídeo vemos el ejemplo de una pequeña aplicación hecha con React. Tenemos un componente `Posts`, que renderiza la lista de posts creados. Desde este componente pasamos al componente hijo, `Form`, la función para crear posts, que hace una petición POST a una API externa.

Vamos a escribir un test que testee el componente `Form`. Como hemos dicho, necesitamos pasar la función que ejecutará cuando se haga submit, así que crearemos una función mock con Jest:

```
const submit = jest.fn();
render(<Form submit={submit} />);
```

A partir de aquí, tenemos el componente renderizado y podemos hacer queries con el objeto `screen`. Como lo haría un usuario, buscaremos los inputs a través de su label:

```
const titleInput = screen.getByLabelText(/title/i);
```

Seguramente te encontrarás muchos ejemplos hechos con `fireEvent`. El problema es que cuando queremos lanzar determinados eventos, tenemos que incluir el objeto de evento con su propiedad target y value. Por ejemplo, para cambiar el valor del input de texto que hemos encontrado antes, tendríamos que hacer lo siguiente:

```
fireEvent.change(titleInput, {
  target: { value: "My awesome post" },
});
```

Como vemos, esto está peligrosamente cerca de testear un detalle de implementación, y puede que nos encontremos con un bug que no detecte el test si realmente el evento no se lanza. Un usuario nunca emite un evento de JavaScript directamente, así que los desarrolladores de Testing Library han añadido una librería más, userEvent. Con ella conseguimos lanzar eventos de una forma mucho más semántica:

```
userEvent.type(title, "My awesome post");
```

Finalmente, el test completo nos quedaría así:

```
import { render, screen } from "@testing-library/react";
import userEvent from "@testing-library/user-event";
import Form from "../components/Form";

describe("When user fills in and submits form", () => {
  it("submit method is called with title and content", () => {
    const submit = jest.fn();
    render(<Form submit={submit} />);

    const titleInput = screen.getByLabelText(/title/i);
    userEvent.type(titleInput, "My awesome post");
```

```
    const contentInput = screen.getByLabelText(/content/i);
    userEvent.type(contentInput, "Lorem ipsum dolor sit amet");

    const button = screen.getByRole("button", { name: /submit/i
});
    userEvent.click(button);

    expect(submit).toHaveBeenCalledWith({
      title: "My awesome post",
      content: "Lorem ipsum dolor sit amet",
    });
  });
});
```

Tenemos un test bastante semántico y que cubre bien el componente Form, pero vemos que realmente hay algunos problemas con este enfoque. Estamos testeando si una función se llama correctamente, algo que un usuario no haría. Al estar testeando el componente `Form` sin el contexto del componente `Posts`, nos aporta poco valor. Por lo tanto, vemos que aunque Testing Library nos ayude a testear como lo haría un usuario, como desarrolladores tenemos que preocuparnos por el enfoque de nuestros tests más allá de la librería que utilicemos. ¡Esto es lo que veremos a lo largo del curso!

Links relacionados:

- Queries de Testing Library
- UserEvent