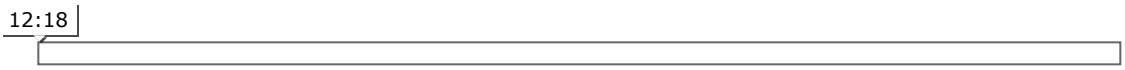
Watch: 📸 **Snapshot testing, ¿sí o no?**



Puedes encontrar el código de este vídeo en GitHub.

Jest nos ofrece la opción de crear tests de snapshots, es decir, comparar el HTML generado por nuestro componente con una versión anterior de ese HTML que sabemos que es correcta.

Esto parece una forma rápida de testear la presentación de nuestros componentes, pero tiene sus contras. En primer lugar, si decidimos hacer un cambio por ejemplo para mejorar la semántica de nuestro HTML, el test no pasará y tendremos que actualizar los snapshots. También hemos visto en el curso que testear componentes presentacionales no aporta mucho valor, así que el caso de uso más relevante de los snapshots nos aporta poco.

Si intentamos usar snapshot testing en componentes más complejos, por ejemplo, en un componente que cargue datos de forma asíncrona, tendremos que asegurarnos que comparamos el HTML generado en el punto adecuado:

```
describe('Comments component', () => {
  it('should render the list of comments', async () => {
    const { container } = await render(CommentsComponent, {
      declarations: [CommentComponent],
      componentProviders: [
        {
          provide: CommentRepositoryService,
          useValue: {
            async getAll() {
              return await [
                {
                  id: 1,
                  date: '2020-01-01',
                  content: 'Lorem ipsum',
                  user: {
                    id: 1,
                    name: 'Núria',
                    avatar: './nuria.png',
                  },
                },
              ];
            },
          },
        },
      ],
    });

    await screen.findAllByRole('article');

    expect(container).toMatchSnapshot();
  });
});
```

Además, al comparar HTML no podemos usar factories con datos aleatorios. Si comparamos el test anterior con un test que no use snapshots, vemos que realmente usar snapshots no nos agiliza realmente el proceso de testing, y es mucho más fácil de leer este test:

```
describe("Comments component", () => {
  it("should render the list of comments", async () => {
    const commentsList = CommentFactory.buildList(1);

    await render(CommentsComponent, {
      declarations: [CommentComponent],
      componentProviders: [
        {
          provide: CommentRepositoryService,
          useValue: {
            async getAll() {
              return await commentsList;
            },
          },
        },
      ],
    });

    const comment = await
screen.findByText(commentsList[0].content)

    expect(comment).toBeInTheDocument();
  });
});
```