TEXT CLASS REVIEW

TEMAS A TRATAR EN LA CUE

- ¿Qué es testing y para qué sirve?
- ¿Cómo se hace?
- Pruebas unitarias y sus características.
- Test end to end.
- TDD y sus características.
- Testeo unitario.

INTRODUCCIÓN

Por tu gran trabajo realizado en el Drill anterior, ya has aprendido los principales conceptos de `Vue.js`. Ahora, comenzaremos a ahondar en cómo lograr que nuestro código esté libre de errores o "bugs". En primer lugar, veremos una introducción general a los principales tópicos sobre testeo, y cómo preparar nuestro entorno para llevar a cabo esta misión.

¿QUÉ ES TESTING Y PARA QUÉ SIRVE?

Es el proceso de comprobar, mediante pruebas, que una aplicación, o parte de ella, funciona correctamente. Un buen testeo sirve para mejorar la velocidad de desarrollo, lo hace más escalable, minimiza los errores, y aumenta la calidad del código.

¿CÓMO SE HACE?

Creando todas las pruebas necesarias para que la aplicación pueda experimentarse de manera rápida, y repetitiva. Existen dos enfoques principales sobre la forma de testear:

- **Testeo manual:** lo explicaremos con un ejemplo; creas la función suma, y una vez la tengas, abres el navegador y pruebas con dos números que la suma se realice correctamente. De

eso se trata el testeo manual. Pero, imagínense ahora haciendo ese proceso para una gran aplicación, ¿es poco eficiente cierto?

- **Testeo automático:** se trabaja con programas, que se encargan de ejecutar las pruebas creadas por los desarrolladores, y así verificar que nuestra aplicación funciona correctamente.

Existen varios tipos de test, pero nos enfocaremos en los dos más requeridos por los desarrolladores Front end, éstos son: `test unitarios y end to end`.

PRUEBAS UNITARIAS Y SUS CARACTERISTICAS

Son aquellas que se realizan sobre una pequeña parte de la aplicación. Generalmente, corresponde a funciones, que llaman y prueban métodos de nuestro código, de manera aislada, para comprobar su correcto funcionamiento. Se caracterizan por ser rápidas, fáciles de escribir y entender, y por esto sirven de documentación; son repetibles, pues deberían seguir funcionando siempre igual, son automatizables e independientes, confirman que pequeñas partes de nuestro código funcionen bien, pero no garantizan que lo hagan bien estando todos juntos.

Ahora, mencionaremos dos tipos de pruebas unitarias, que se usan en aplicaciones construidas con `Vue`:

- Pruebas de componentes: consisten en renderizar el componente, y verificar que el `HTML` coincide con su estado.
- Snapshot testing: consisten en procesar los componentes de la interfaz de usuario, tomar una captura de pantalla, y luego compararla con una imagen referencial guardada al lado del test.

Las pruebas unitarias no usan la conexión real a una base de datos, sino que simulan la conexión a través de objetos de prueba.

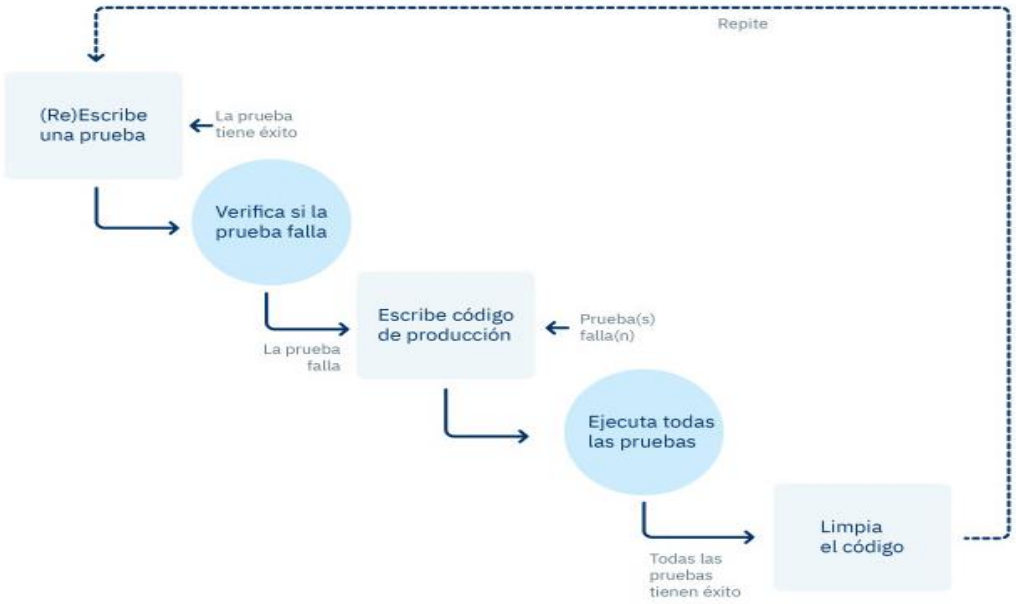
TEST END TO END

Se usan para comprobar que una aplicación funciona correctamente, desde la perspectiva del usuario, de principio a fin. Éstas utilizan un navegador automatizado, simulando un escenario real. Sirven para identificar las dependencias del sistema, ahorrar tiempo, y garantizar que la información se transmita correctamente entre los diversos componentes y sistemas involucrados. Sin embargo, son lentos, frágiles y, surge una última complicación cuando el test creado pasa, pero al volver a ejecutarlo, falla, y viceversa, incluso cuando no se han hecho cambios en el código. Esto es conocido como `“flaky”` test.

A pesar de que los test `"end to end"` son bastante prácticos, no son suficientes por sí solos. Debido a esto, el uso combinado con test unitarios, es una buena aproximación para verificar que nuestro código funcione correctamente.

TDD Y SUS CARACTERÍSTICAS

El desarrollo guiado por pruebas, o `Test-driven development (TDD)`, es una práctica de programación que se basa en combinar 2 metodologías: `Test-first development` (escribir las pruebas primero), y `Refactoring` (refactorización de código).



Con esta práctica, se persigue conseguir una mayor rapidez en el desarrollo, lograr que el código cumpla con los requisitos solicitados, sea robusto, seguro, limpio, legible y, mantenible. Sin embargo, existe una complicación asociada al uso de bases de datos, ya que para la creación de las pruebas, se necesita tener ciertos datos conocidos, para así poder usarlos de prueba. Esto se puede lograr con el uso de objetos de prueba, como los `mocks`.

TESTEO UNITARIO

- **Set up de herramientas con Vue CLI.**

  Antes de comenzar a trabajar en el desarrollo dirigido por pruebas, es necesario revisar las herramientas que usaremos para lograrlo.

  - Hardware: reprocesador Intel I3 o similar mínimo, RAM 8GB mínimo.
  - Arquitectura por defecto de la aplicación: `Vue CLI`.
  - Ejecución de tests mediante línea de comandos (CLI): `NodeJS`.
  - Framework de pruebas unitarias: `Jest`.
  - Biblioteca para afirmaciones: `Jest`.
  - Biblioteca para stubs y mocks: `Jest`.

  Cuando comenzamos un proyecto con `Vue CLI` usando la línea de comandos `NodeJS`, existe la posibilidad de seleccionar manualmente las herramientas que vamos a emplear. La opción para realizar testeo unitario, viene con dos alternativas: `Jest o Mocha + Chai`. Cualquiera que seleccionemos, funciona de manera inmediata. A través de `Vue CLI`. también podemos instalar `Vue Test Utils`, el paquete oficial para testeo de `Vue`.

- **Entorno de pruebas Vue test utils para Vue.**

  `Vue.js` posee su propia librería para realizar pruebas: `Vue Test Utils`. Aquí encontramos diversos métodos para testeo de una aplicación creada con `Vue`, que facilitarán nuestra tarea a la hora de montar componentes, simular eventos de usuario, hacer renderizado superficial, llevar a cabo modificaciones de estado y las `props` de componentes, entre otros. Esta biblioteca se instala automáticamente cuando se inicia un proyecto con `Vue CLI`, y elegimos, manualmente, la opción que nos permite realizar pruebas unitarias.