

Node JS, CI/CD, Terminal y GIT



NODE JS

Permite implementar y ejecutar scripts, que se definen dentro del archivo `package.json` los cuales pueden correr pruebas unitarias o algunas otras tareas que se pueden automatizar.

Para poder ejecutar los scripts es necesario escribir el comando directo en la terminal con el comando `npm run <nombre_script>`.

Existen herramientas que permiten la ejecución de scripts de manera automática, de tal manera que se puede "simular" un ambiente de CI/CD localmente.

Debido a que los **scripts** son altamente personalizables, es posible implementar y reforzar buenas prácticas por medio de los **hooks** para automatizar y optimizar prácticamente cualquier aspecto del **workflow** de desarrollo.

Esto se logra usando **git-hooks**, los cuales son eventos que el servidor **git** expone localmente y a los que se puede acceder por medio de los **hooks**.

Las pruebas unitarias que se realizan mayormente son pruebas de funcionalidad y son parte importante del ciclo **CI/CD** y del desarrollo ágil, estas consisten en aislar un componente u objeto y validar que funcione correctamente, simulando escenarios de trabajo donde se consuma su interfaz.

Por la parte del **frontend**, **React** ya integra una suite de pruebas unitarias dentro de la librería `react-dom/test-utils`, esta implementa internamente la librería de **unit testing jest**, la cual acepta los 3 elementos principales de las pruebas unitarias:

1. Arrange

La organización de las pruebas unitarias permite definir los requisitos que debe cumplir el código; prepara el ambiente de pruebas para simular un ambiente real y probar los componentes u objetos en ese ambiente simulado.

Liberá los recursos usados para la simulación una vez que las pruebas se hayan realizado.

Para definirlo se usan principalmente los métodos `beforeAll`, `before`, `after` y `afterAll`.

2. Act:

Es la ejecución de la prueba.

Usa principalmente los métodos `describe` el cual define la estructura interna y el flujo de ejecución que se seguirá para realizar las pruebas, las cuales se definen individualmente con el método `it`.

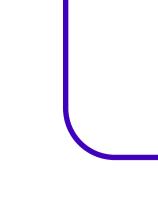
3. Assert:

Es el resultado esperado por la prueba, donde se comprueba si los datos obtenidos son los esperados, si es así, se sigue adelante con la ejecución de las pruebas, si no, la prueba no es válida por lo que el código debe ser corregido.

Usa los métodos `expect` y una gran variedad de **assertions** que se conjugan para validar tipos de datos.

Aunque React sea una tecnología frontend, lo recomendable es ejecutar las pruebas unitarias en un entorno no productivo del lado del servidor, donde se simula el frontend pero corriendo en el servidor, la librería jest hace esto posible y su integración con NodeJS es estable:

Como se puede observar, las pruebas unitarias deben ser parte importante del ciclo de desarrollo.



Referencias:

- Expect · Jest. (s.f)
De <https://jestjs.io/docs/en/expect>

- Git Hooks - Git. (s.f)
De <https://git-scm.com/book/en/v2/Customizing-Git-Git-Hooks>

- Jest · Delightful JavaScript Testing. (s.f)
De <https://jestjs.io/>

- Prueba unitaria - Wikipedia, la enciclopedia libre.(s.f)
De https://es.wikipedia.org/wiki/Prueba_unitaria

- Test Utilities - React.(s.f)
De <https://reactjs.org/docs/test-utils.htm>

- Web semántica, qué es y cómo mejora tu web - Webtematica. (s.f)
De <https://webtematica.com/que-es-la-web-semanitica>