



Intro a Testing

DEV.F
DESARROLLAMOS(PERSONAS);

dev

The logo consists of the text 'DEV.F.' in a bold, white, sans-serif font. The 'F' is stylized with three small squares at its top right corner. The logo is centered within a dark blue diamond shape.

DEV.F.

DISCUSIÓN EN CLASE

***¿POR QUÉ ES IMPORTANTE
REALIZAR PRUEBAS A
NUESTRO SOFTWARE?***

dev



¿Por qué es importante el testing?

En un principio, puede parecer una tontería: ¿para qué comprobar, si lo que acabas de hacer funciona? En teoría, lo acabas de hacer y lo has probado, es más, podría parecer una pérdida de tiempo escribiendo código que comprueba el código, pudiendo ahorrártelo y sacar tu aplicación antes, ¿no?

En cierto modo puede ser así; puede ser que, si no implementas ningún tipo de testing, podrías acabar antes el desarrollo.



Impacto de un Software con defectos (1)

Comunicaciones: Pérdida o corrupción de los medios de comunicación, no entrega de datos.

Aplicaciones espaciales: Pérdida de vidas, retrasos en el lanzamiento.

Defensa y guerra: Identificación errónea de amigos o enemigos, Información táctica incorrecta.



El software llamado MCAS causó la caída de 2 aviones Boeing 737 MAX, causando la muerte de 346 personas

<https://www.youtube.com/watch?v=nurVW3f7FHs>

Impacto de un Software con defectos (2)

Transporte: Muertes, retrasos, aceleraciones repentinas, incapacidad para frenar.

Aplicaciones críticas para la seguridad: Muerte, lesiones.

Energía eléctrica: Muerte, lesiones, cortes de energía, riesgos para la salud a largo plazo (radiación).



BBVA ✓

30 min · 🌐



BBVA México informa:

Con respecto a los comentarios reportados por algunos clientes en redes sociales, la institución precisa:

Algunas operaciones de compra realizadas recientemente con tarjetas de débito no fueron cargadas a las respectivas cuentas y esos montos, por un error humano, fueron liberados por lo que se reflejan en las propias cuentas como saldos disponibles.

Esta situación será normalizada en el transcurso de las próximas horas.

La institución bancaria ofrece a los clientes que se encuentren en esta situación una sincera disculpa por la confusión que pueda estarse generando con los saldos de sus cuentas.



6.3 mil

2.4 mil comentarios 3.9 mil veces compartido



Me gusta



Comentar



Compartir

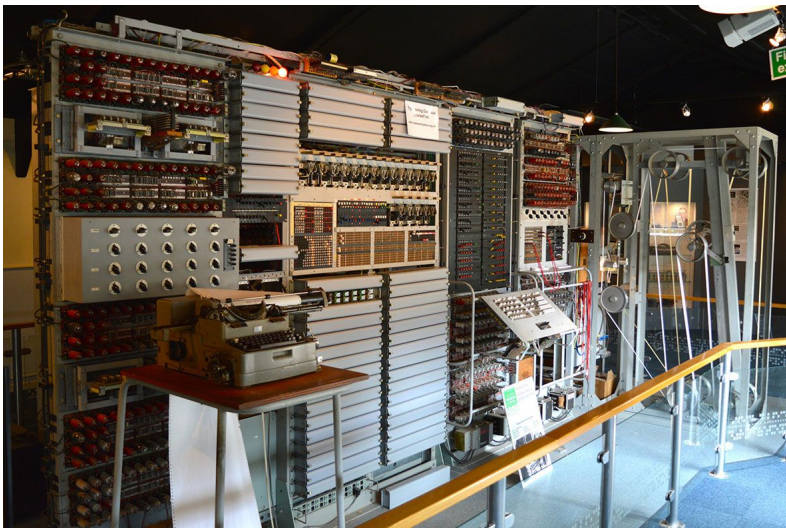
Impacto de un Software con defectos (3)

Gestión del dinero: Fraude, violación de la privacidad, cierre de bolsas y bancos, tipos de interés negativos.

Control de las elecciones: Resultados erróneos (intencionados o no).

Control de las cárceles: Intentos y éxitos de fuga con ayuda de la tecnología, liberación accidental de reclusos, fallos en las cerraduras controladas por software.

Aplicación de la ley: Detenciones y encarcelamientos falsos.



Un poco de historia sobre los bugs...

En 1947, la Universidad de Harvard utilizaba una computadora del tamaño de una habitación llamada Mark II.

- Relays mecánicos
- Tubos de vacío incandescentes
- Los técnicos programan el ordenador reconfigurándolo y tenían que cambiar algún que otro tubo de vacío.

Una polilla entró volando en la computadora y fue electrocutada por el alto voltaje cuando se posó en un relay

Literal, de ahí surgió el primer “bug” informático.

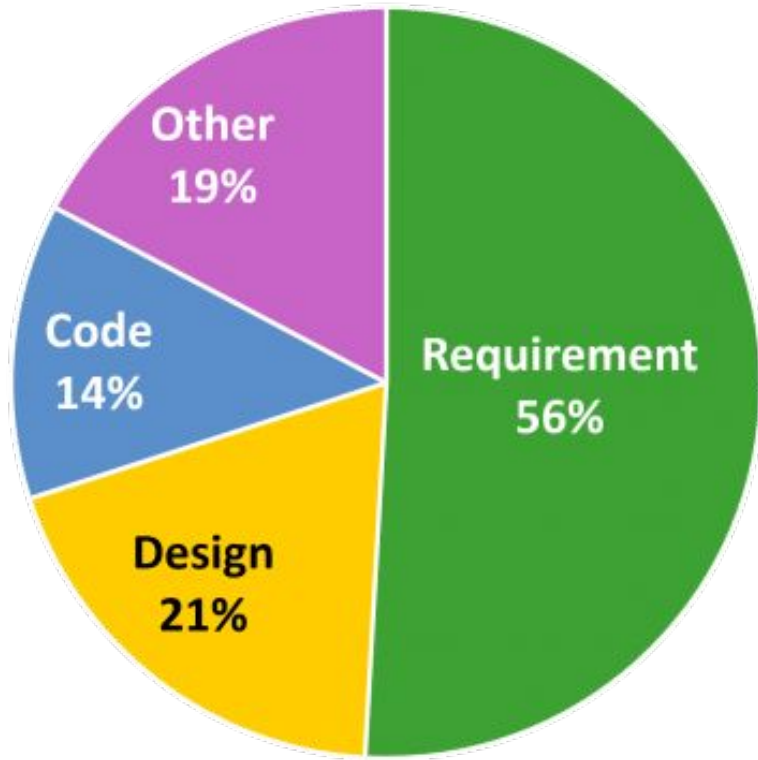


Causas de los Defectos en el Software

Veamos primero las estadísticas sobre las causas de los errores en los proyectos de software. Aquí vemos que la mayoría de los errores que se cometen en un proyecto tienen su raíz en los requisitos.

Los requisitos son la interfaz entre el cliente (cliente/usuario/etc.) y el contratista (proveedor de servicios/equipo de desarrollo/etc.). Se trata de una interfaz sólo entre personas.

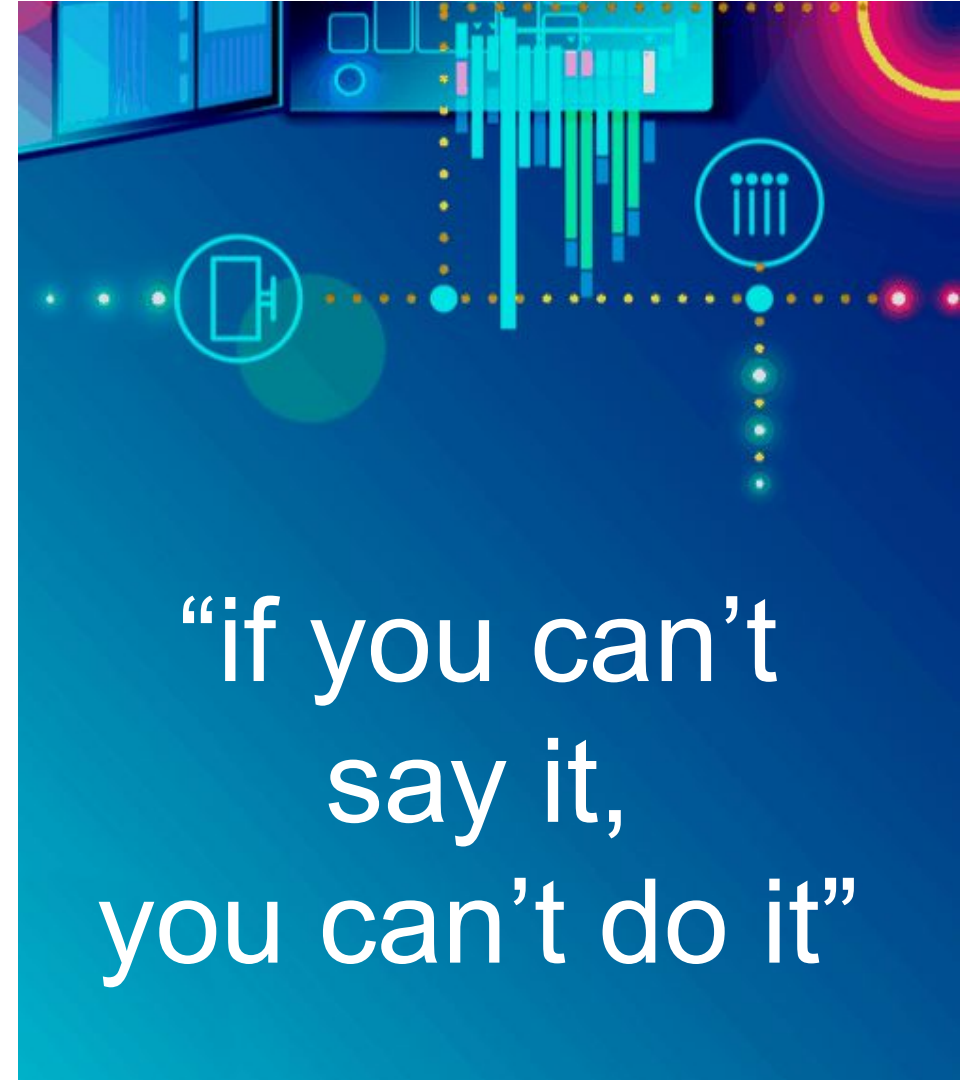
El único medio de comunicación disponible entre humanos es el lenguaje natural, con todos los defectos que esto conlleva, por eso es fácil introducir errores.



The logo consists of the text 'DEV.F.' in a bold, white, sans-serif font. The 'F' is stylized with three small squares at its base, suggesting a circuit or digital theme. The logo is centered within a dark blue diamond shape.

Up to 35% Production Bugs are from Poor Requirements

*En enero de 2021, Accenture reconoció en el **Software Intelligence Forum** que **hasta el 35% de los defectos de producción están causados por problemas de requisitos**, según sus datos basados en 1000 proyectos. Lo que esto significa es que en la mayoría de los proyectos, el trabajo de aseguramiento de la calidad de los requisitos está fallando.*



“if you can’t
say it,
you can’t do it”

Especificaciones

Hay que saber qué es el producto antes de poder decir si tiene un fallo.

Una especificación define el producto que se está creando e incluye:

Requisitos funcionales que describen las características que soportará el producto. Por ejemplo, en un procesador de textos (Guardar, imprimir, comprobar la ortografía, etc ...)

Los requisitos no funcionales son restricciones inherentes del producto. Por ejemplo seguridad, fiabilidad, facilidad de uso, plataforma, tiempo de carga, etc.



Un bug de software ocurre sí:

- El software no hace algo que la especificación dice que debe hacer.
- El software hace algo que la especificación dice que no debe hacer.
- El software hace algo que la especificación no menciona.
- El software no hace algo que la especificación del producto no menciona pero que debería hacer.
- El software es difícil de entender, difícil de usar, lento...



DEV.F.

TESTING

Es el proceso de testear o de probar que algo funciona como debería.

Probar mi código con más código.

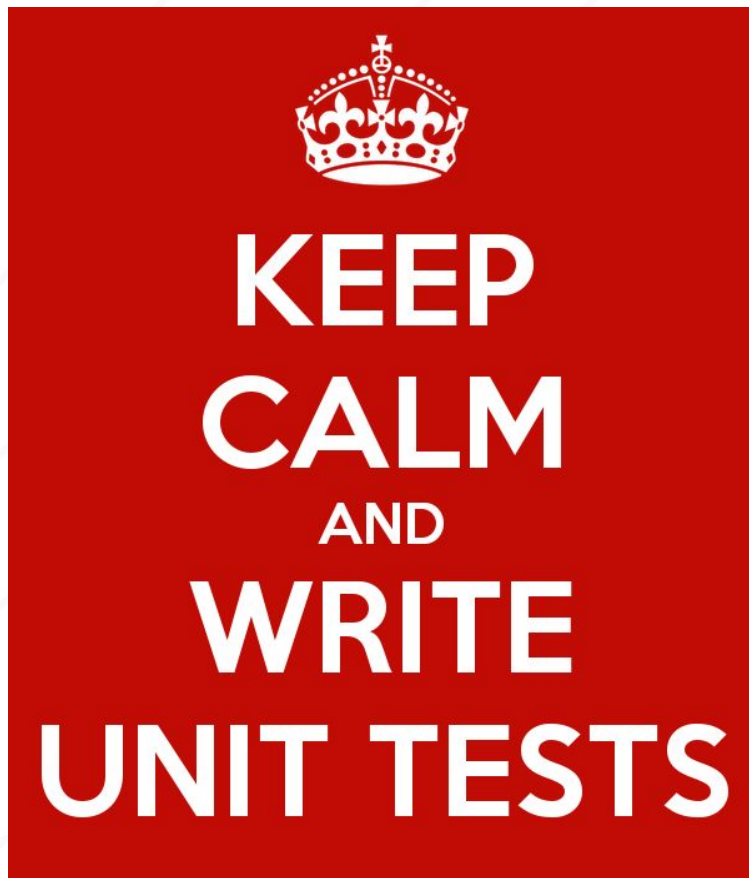
Ventajas de hacer Testing

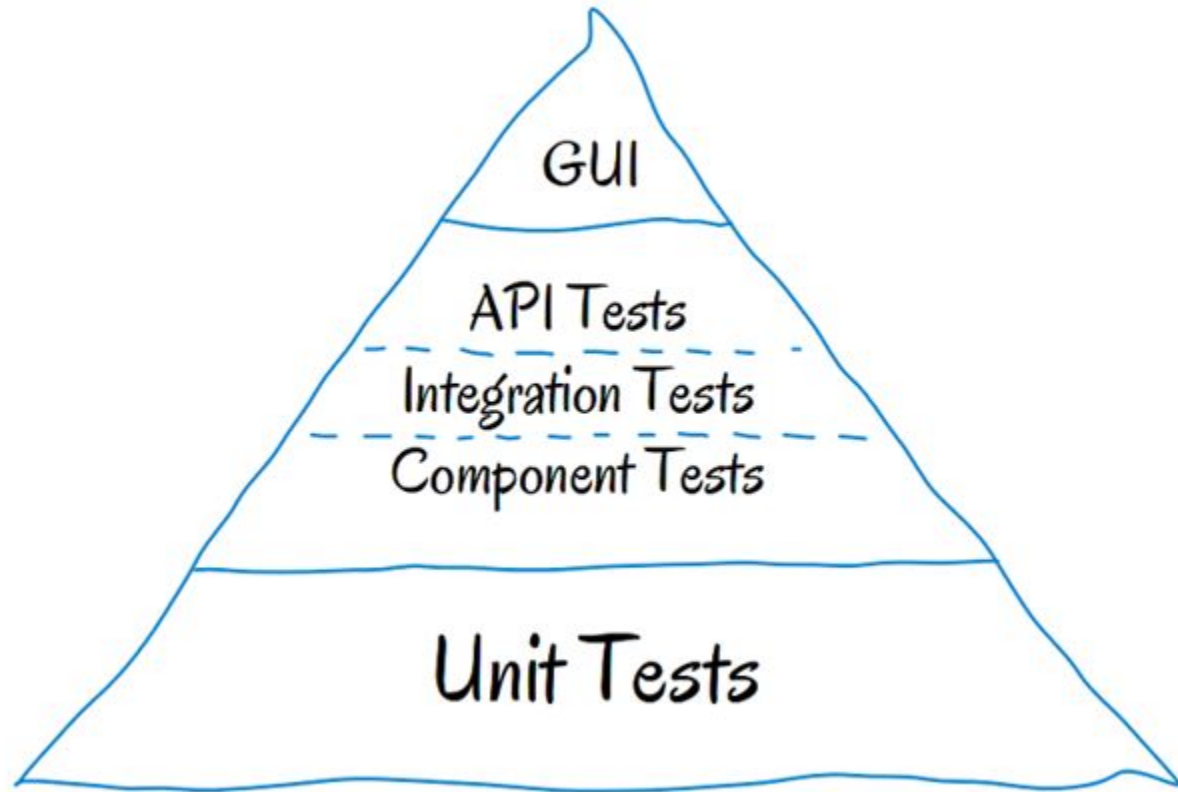
- Ahorro de tiempo
- Detectar dónde están los errores
- Asegurar que la app funciona en cualquier dispositivo
- Ofrece una base sólida al refactorizar tu código
- ¡Y mucho más! 🎉



Tipos de Testing

- Unit Testing
- Integration Test
- Functional Test
- End-to-end Test
- Acceptance Test
- Performance Test
- Stress Test

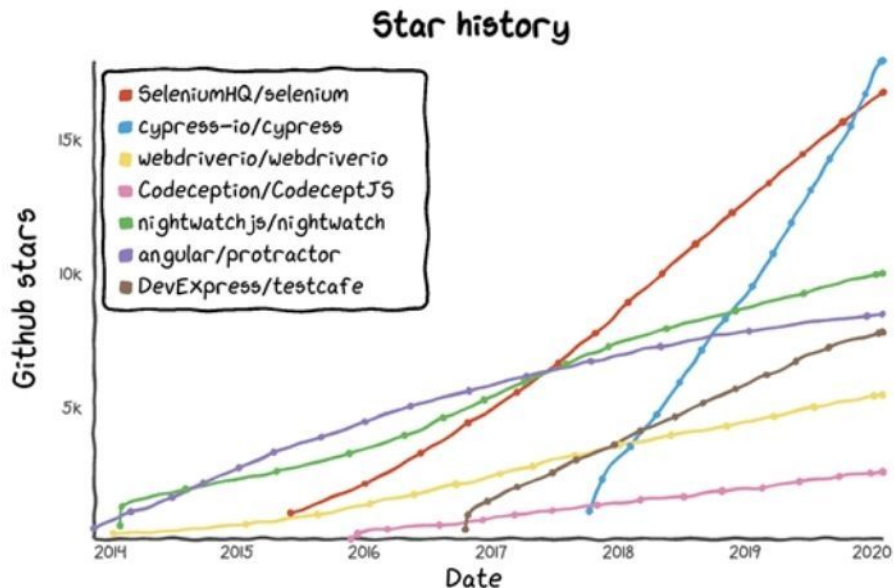






DEV.F
DESARROLLAMOS(PERSONAS);

dev



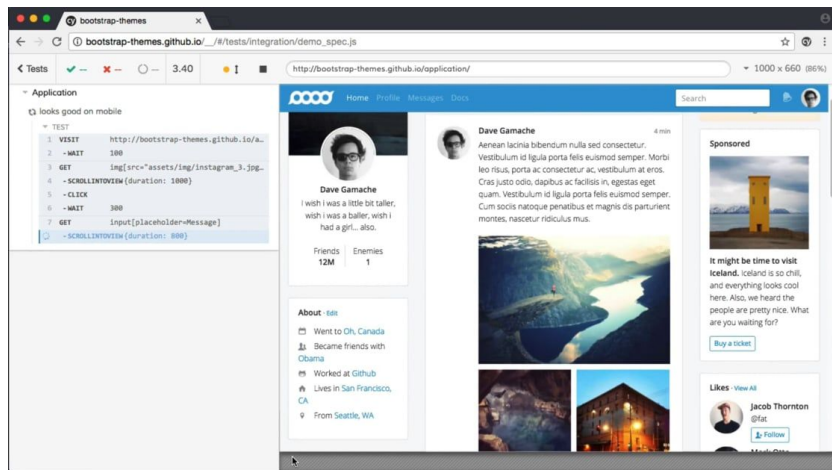
<https://www.cypress.io/>

¿Qué es Cypress?

Cypress es un framework de testing moderno y todo en uno.

Es rápido, fácil de usar y permite ejecutar pruebas sobre cualquier aplicación web.

En poco más de 2 años desde su lanzamiento de la versión 1.0.0 se ha convertido en una de las herramientas más populares de testing.



Test your code, not your patience.

Cypress is the new standard in front-end testing that every developer and QA engineer needs.

<https://www.cypress.io/>

Beneficios de Cypress

Cypress no utiliza Selenium, las pruebas se ejecutan dentro del navegador controlándolo a través de un proceso backend en NodeJS. Además, funciona a nivel de red permitiendo la interceptación de todo el tráfico entrante y saliente de nuestra aplicación.

Como las pruebas se ejecutan dentro del navegador, dispone del acceso nativo a todas las APIs como window, document, etc. Además la ejecución es muchísimo más rápida que en otras herramientas alternativas.

Su instalación es muy simple, solo se necesita agregar una sola dependencia con NPM.



Instalación y Configuración de Cypress en Vite JS

DEV.FX
DESARROLLAMOS(PERSONAS);

dev

Instalación de Cypress



```
npm install cypress --save-dev
```

<https://www.npmjs.com/package/cypress>

package.json

```
"scripts": {  
  "dev": "vite",  
  "build": "vite build",  
  "preview": "vite preview",  
  "cypress": "cypress open"  
},
```

Configurando el Script de Cypress

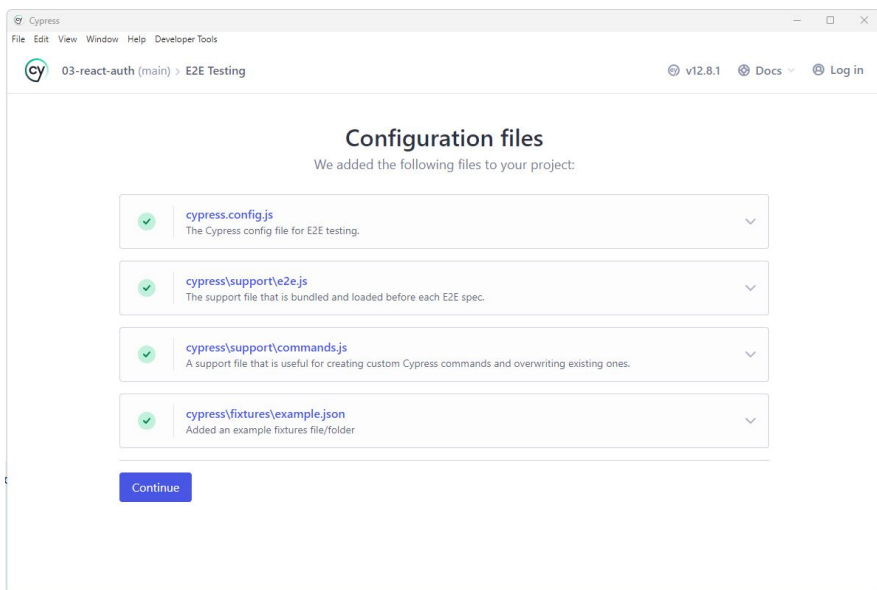
En el apartado de scripts del archivo **package.json**, añadir una nueva línea para añadir a cypress:

"cypress": "cypress open"

Probando Cypress

Una vez añadido el script de Cypress, procederemos a probar en la consola, ejecutando el comando:

npm run cypress





cypress.config.js



```
import { defineConfig } from 'cypress'

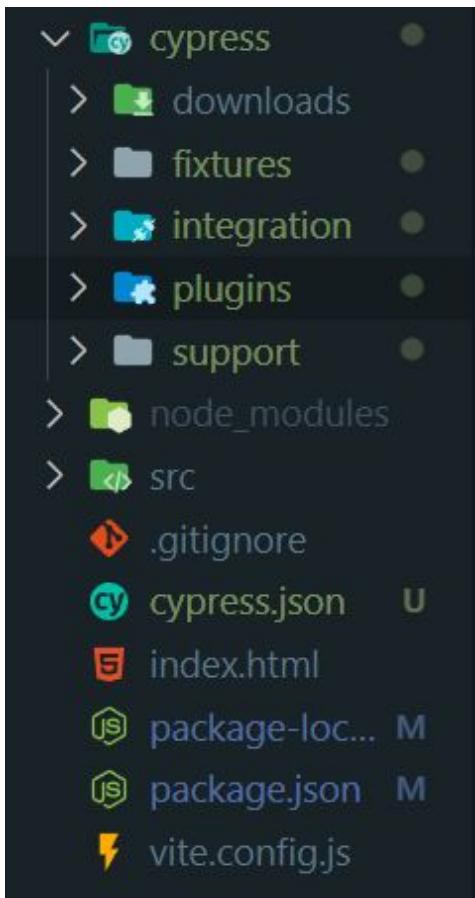
export default defineConfig({
  e2e: {
    setupNodeEvents (on, config) {
      // implement node event listeners here
    },
    baseUrl: 'http://localhost:5173'
  }
})
```

Configurando Cypress.config.js

Si estamos en la versión 10+ de Cypress, ahora se usa el archivo **cypress.config.js** (en versiones anteriores era Cypress.json)

Le indicaremos una URL base en la configuración para permitirnos usar rutas relativas en los test.

Nota: Si usamos Vite 3+, el puerto por defecto será 5173 en vez de 3000.



Notamos nuevas carpetas

Después de instalar Cypress, notaremos que se crean nuevas carpetas y archivos, principalmente fijarse en las siguientes:

cypress/plugins: Aquí se configuran los plugins de Cypress.

cypress/integration: Aquí guardaremos los archivos de nuestras pruebas de Integración.

cypress.config.json: Archivo de configuración de Cypress



Pruebas de Integración con Cypress

DEV.FX
DESARROLLAMOS(PERSONAS);

dev

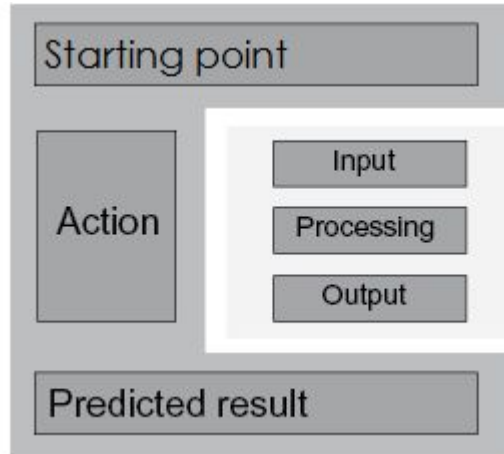
Estructura de un Test (Teoría)

Una prueba sólida suele abarcar 3 fases:

1. Establecer el estado de la aplicación.
2. Realizar una acción / acciones.
3. Hacer una afirmación sobre el estado resultante de la aplicación.

Primero se pone la aplicación en un estado específico, luego se realiza alguna acción en la aplicación que hace que cambie, y finalmente se comprueba el estado resultante de la aplicación.

Test case



(System)
behaviour

Estructura de un Test (En Código)

describe(): Es simplemente una forma de agrupar nuestras pruebas. Toma dos argumentos, el primero es el nombre del grupo de pruebas, y el segundo es una función callback.

it(): Lo utilizamos para un caso de prueba individual. Toma dos argumentos, una cadena que explica lo que debe hacer la prueba, y una función callback que contiene nuestra prueba real. Dentro de cada it() deberemos establecer el estado de la aplicación, realizar acciones y esperar un resultado.

expect/constains/have: Aquí es donde especificamos el resultado esperado después de ejecutar las acciones anteriores.

```
describe('Nombre del Grupo de Pruebas', ()=>{
  it('Nombre Prueba #1', ()=>{
    //acción #1
    //acción #2...n
    expect('resultado 1')
  })

  it('Nombre Prueba #2', ()=>{
    //acción #1
    //acción #2...n
    expect('resultado 2')
  })
})
```

Browser Commands

cy.visit	visit url
cy.go("back")	click on browser's "back" button
cy.go("forward")	click on browser's "forward" button
cy.reload	refresh the page
cy.viewport	change window size

Selection Commands

cy.get	select based on HTML tag attrs
cy.contains	select based value within opening and closing tags
.first	select first matching element
.last	select last matching element
.next	select next matching element
.prev	select previous matching element

Action Commands

.click	click on an element
.dblclick	double click on an element
.rightclick	right click on an element

Acciones que podemos realizar

Podemos invocar a los métodos de Cypress a través del objeto **"cy"**

Por ejemplo podemos usar el método **"visit"** para navegar en una página web

Cypress también provee un método **"get"** para encontrar un elemento web en el DOM y realizar acciones sobre él.

Puede consultarse la documentación oficial en:

<https://docs.cypress.io/>



Configuración de Linter

DEV.F
DESARROLLAMOS(PERSONAS);

Instalar Plugin de Cypress para Eslint



```
npm install eslint-plugin-cypress --save-dev
```

<https://github.com/cypress-io/eslint-plugin-cypress>

Añadir plugin en configuración de package.json



package.json

```
"eslintConfig": {  
  "extends": [  
    "./node_modules/standard/eslintrc.json",  
    "plugin:cypress/recommended"  
  ]  
}
```

<https://github.com/cypress-io/eslint-plugin-cypress>