



TOMA  
FUERZA.

TU PROPÓSITO

HAZLO  
CONTINENTAL<sup>®</sup>

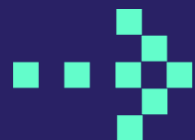
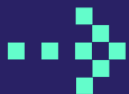




# DEBUGG Y PLUGINS EN CYPRESS



Semana 11



Docente: Dr. Arana Caparachin Maglioni

# **Delimitaciones de Búsqueda de Elementos con Cypress**

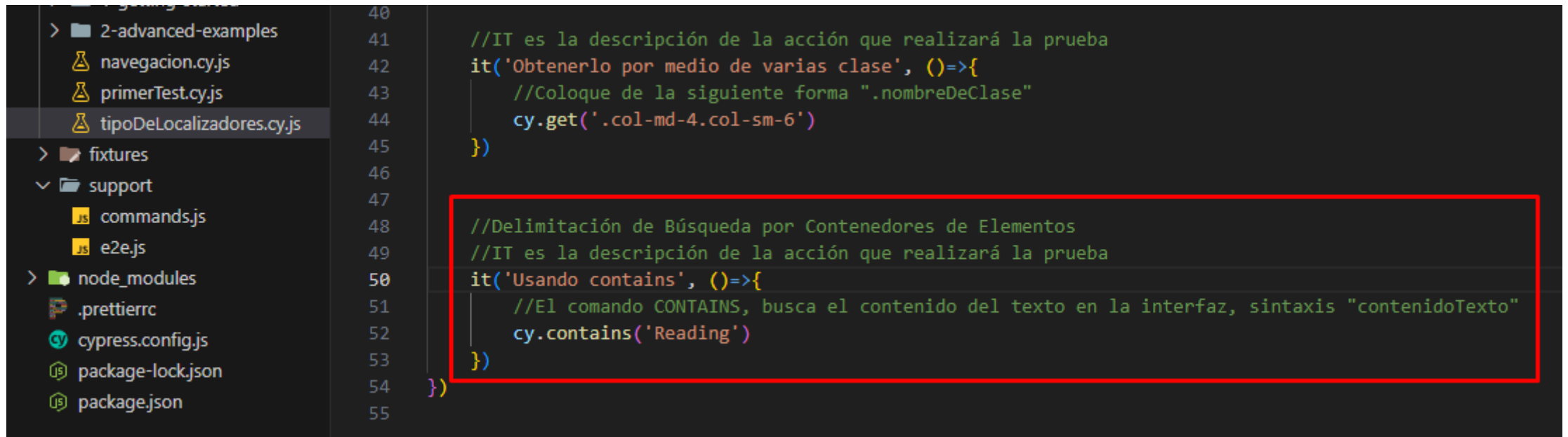
Práctica 05

# DELIMITACIÓN DE BÚSQUEDA

- Hasta el momento ya conocemos como localizar los elementos en el FrontEnd de un proyecto con Cypress, pero ahora, vamos a delimitar mucho más la búsqueda, para rastrear los elementos mediante las pruebas.
- Para ello, en el mismo archivo de “**tipoDeLocalizadores.cy.js**”, continúe con el código compartido.

# DELIMITACIÓN DE BÚSQUEDA

- Vamos a crear una búsqueda de elemento rastreando los **Contenedores**, en anterior ocasión buscábamos por identificadores, pero ahora buscaremos por el texto de etiqueta que se muestra en la interfaz.



```
40
41 //IT es la descripción de la acción que realizará la prueba
42 it('Obtenerlo por medio de varias clase', ()=>{
43   //Coloque de la siguiente forma ".nombreDeClase"
44   cy.get('.col-md-4.col-sm-6')
45 })
46
47
48 //Delimitación de Búsqueda por Contenedores de Elementos
49 //IT es la descripción de la acción que realizará la prueba
50 it('Usando contains', ()=>{
51   //El comando CONTAINS, busca el contenido del texto en la interfaz, sintaxis "contenidoTexto"
52   cy.contains('Reading')
53 })
54 })
55
```

# DELIMITACIÓN DE BÚSQUEDA

- Verifique que realmente localice un elemento por medio del **Contenido del Texto**.

The screenshot displays a web browser window with a 'Practice Form' and a Playwright test runner interface. The test runner shows a list of test steps, with '-contains Reading' highlighted. A red arrow points from this step to the 'Reading' radio button in the 'Student Registration Form'.

**Test Runner Interface:**

- Specs: 7 tests, 0 failures, 0 warnings, 0 errors.
- tipoDeLocalizadores.cjs: 00:13
- Obtenerlo por medio de un atributo
- Obtenerlo por medio de una Etiqueta y un Atributo
- Obtenerlo por medio de su ID
- Obtenerlo por medio de una clase
- Obtenerlo por medio de varias clase
- Usando contains**
- TEST BODY
- contains Reading**

**Practice Form:**

Student Registration Form

Name: First Name, Last Name

Email: name@example.com

Gender: ☐ Male ☐ Female ☐ Other

Mobile: Mobile Number

Date of Birth: 28 Oct 2024

Subjects: ☐ Sports ☒ **Reading** ☐ Music

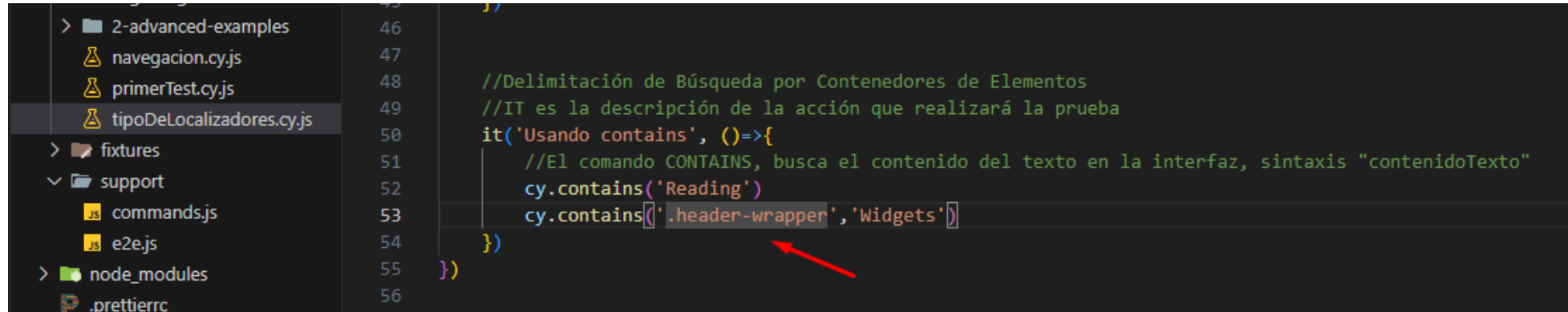
Hobbies: ☐ Sports ☒ **Reading** ☐ Music

Picture: Select picture, Examinar... No se ha seleccionado ningún archivo.

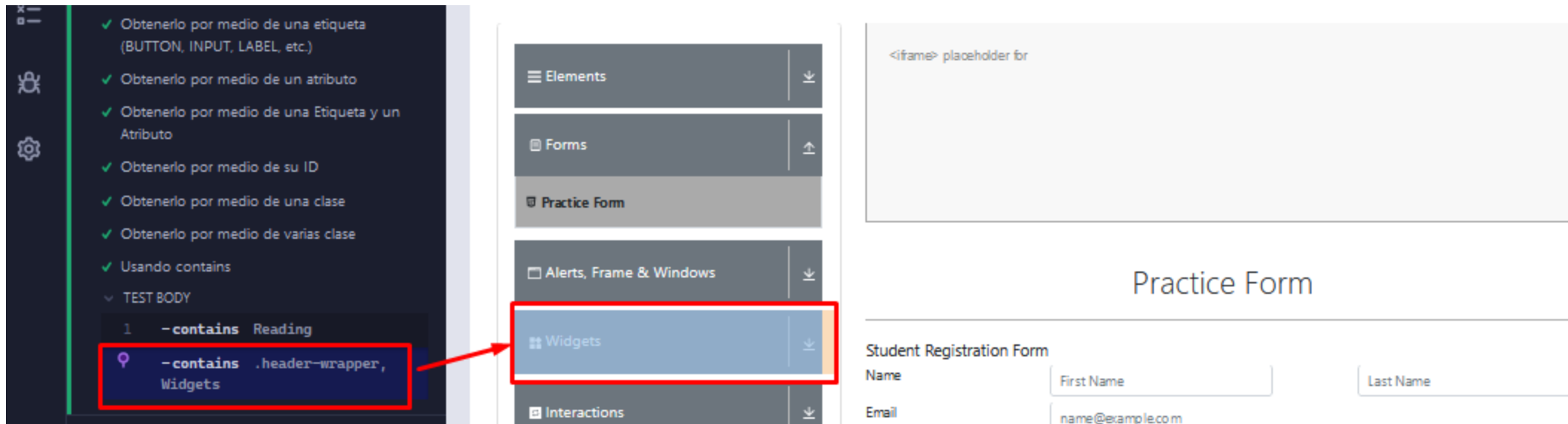
Current Address: Current Address

# DELIMITACIÓN DE BÚSQUEDA

- También puede unir los tipos de búsqueda (por clase y por contenido).



```
//Delimitación de Búsqueda por Contenedores de Elementos
//IT es la descripción de la acción que realizará la prueba
it('Usando contains', ()=>{
  //El comando CONTAINS, busca el contenido del texto en la interfaz, sintaxis "contenidoTexto"
  cy.contains('Reading')
  cy.contains('.header-wrapper', 'Widgets')
})
```



Obtenerlo por medio de una etiqueta (BUTTON, INPUT, LABEL, etc.)

Obtenerlo por medio de un atributo

Obtenerlo por medio de una Etiqueta y un Atributo

Obtenerlo por medio de su ID

Obtenerlo por medio de una clase

Obtenerlo por medio de varias clase

Usando contains

TEST BODY

- 1 - contains Reading
- contains .header-wrapper, Widgets

Elements

Forms

Practice Form

Alerts, Frame & Windows

Widgets

Interactions

<iframe> placeholder for

Practice Form

Student Registration Form

Name

First Name

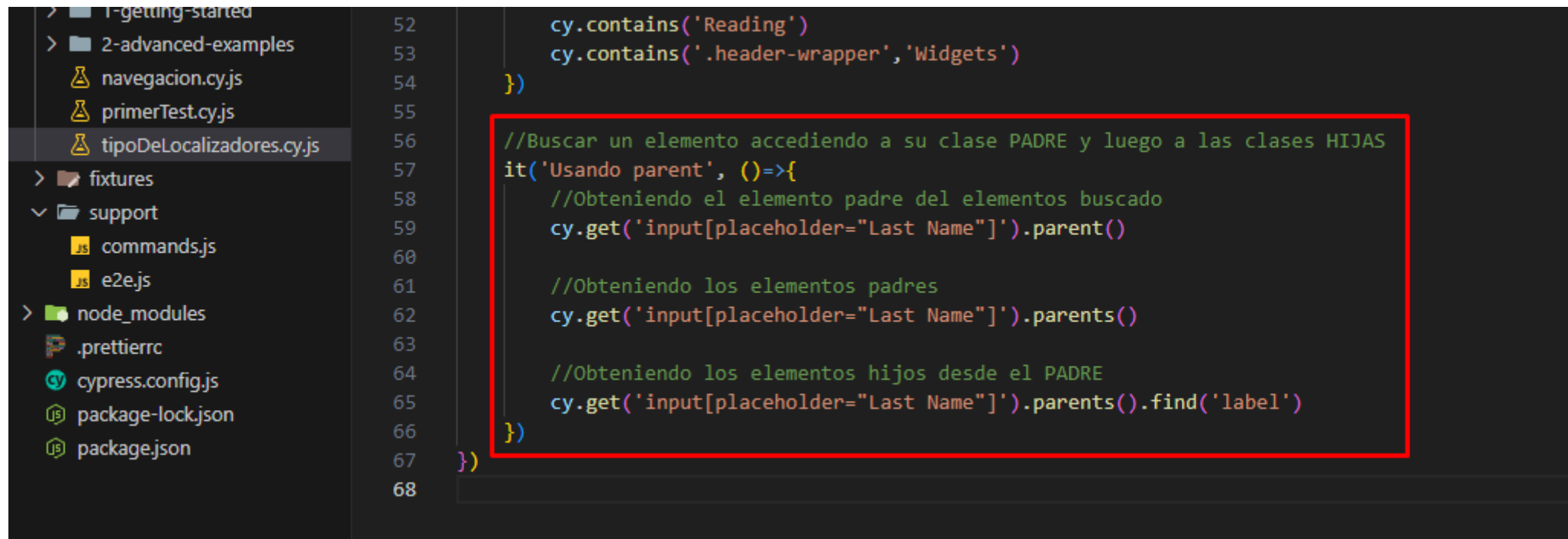
Last Name

Email

name@example.com

# DELIMITACIÓN DE BÚSQUEDA

- Ahora, ¿Cómo trabajamos cuando nuestros elementos no tiene identificadores?
- Para eso se trabaja con las búsquedas de elementos mediante clases PADRE-HIJOS



```
52 cy.contains('Reading')
53 cy.contains('.header-wrapper', 'Widgets')
54 })
55
56 //Buscar un elemento accediendo a su clase PADRE y luego a las clases HIJAS
57 it('Usando parent', ()=>{
58   //Obteniendo el elemento padre del elementos buscado
59   cy.get('input[placeholder="Last Name"]').parent()
60
61   //Obteniendo los elementos padres
62   cy.get('input[placeholder="Last Name"]').parents()
63
64   //Obteniendo los elementos hijos desde el PADRE
65   cy.get('input[placeholder="Last Name"]').parents().find('label')
66 })
67
68
```



# DELIMITACIÓN DE BÚSQUEDA

- Lo que tendrá como resultado lo siguiente:

The screenshot displays a Playwright test runner interface. On the left, a sidebar shows a list of test steps under the heading 'TEST BODY'. The steps are:

- 1. `get input[placeholder="Last Name"]`
- 2. `parent`
- 3. `get input[placeholder="Last Name"]`
- 4. `parents` (11)
- 5. `get input[placeholder="Last Name"]`
- 6. `parents` (11)
- 7. `find label` (16) - This step is highlighted with a red box.

The main area shows a 'Practice Form' titled 'Student Registration Form'. The form contains the following fields and elements:

- Name:** First Name, Last Name
- Email:** name@example.com
- Gender:** Male, Female, Other
- Mobile:** Mobile Number
- Date of Birth:** 28 Oct 2024
- Subjects:**
- Hobbies:** Sports, Reading, Music
- Picture:** Select picture
- Current Address:** Examinar... No se ha seleccionado ningún archivo.
- State and City:** Select State, Select City

A 'Submit' button is located at the bottom right of the form. On the far right, there is a 'ZeroStep' advertisement with the text 'Build Playwright Tests using AI' and a 'Try for free' button.

# **Guardando Elementos con Cypress**

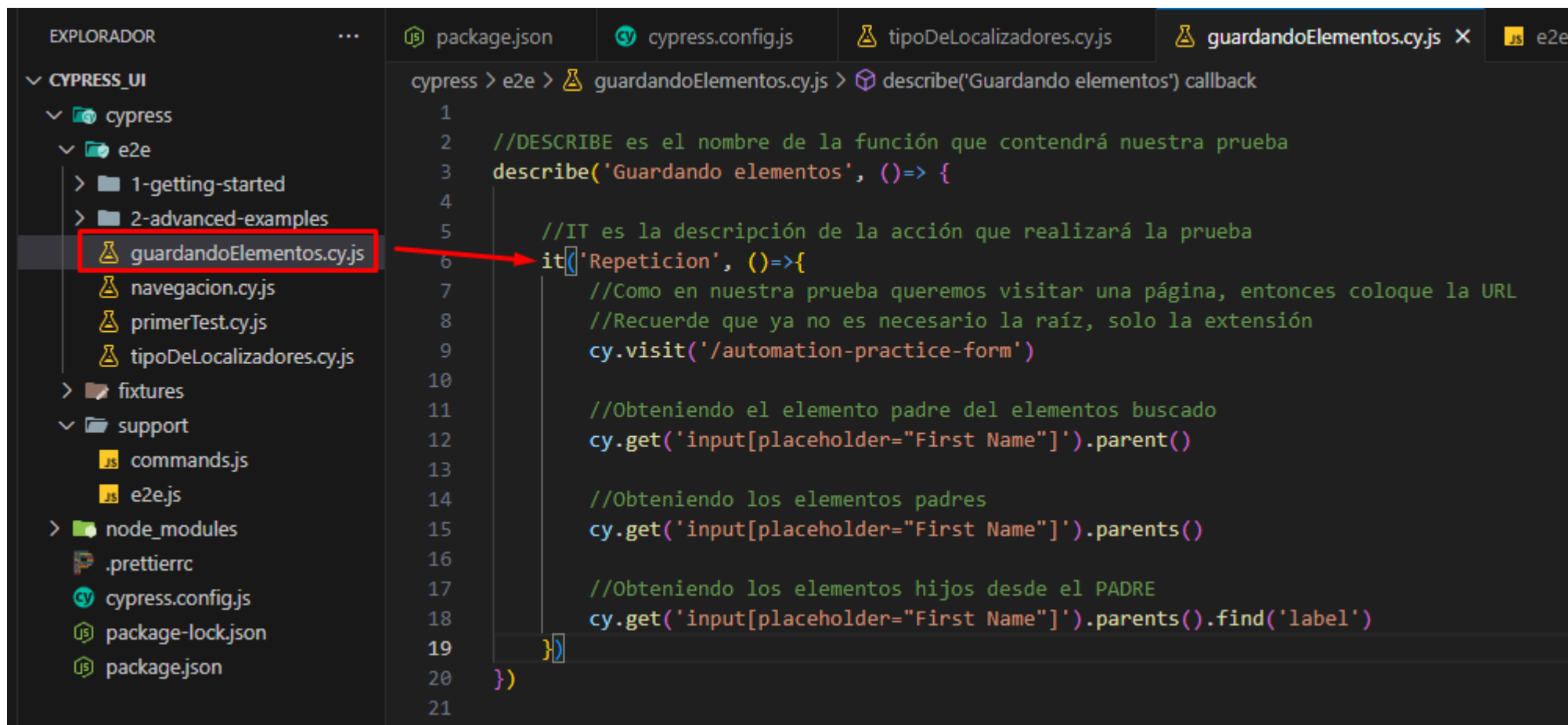
Práctica 06

# GUARDANDO ELEMENTOS CON CYPRESS

- Hasta este momento hemos digitalizado cada prueba, pero aún no la hemos automatizado, recuerde que Cypress es una herramienta de automatización de pruebas.
- Para ello, comenzaremos a guardar estas pequeñas pruebas de búsqueda de elementos que hemos estado realizando.
- Para comenzar, cree un nuevo archivo de nombre: “**guardandoElementos.cy.js**”

# GUARDANDO ELEMENTOS CON CYPRESS

- Primero, verifiquemos como se ve algunas pruebas cuando se repiten líneas de código:

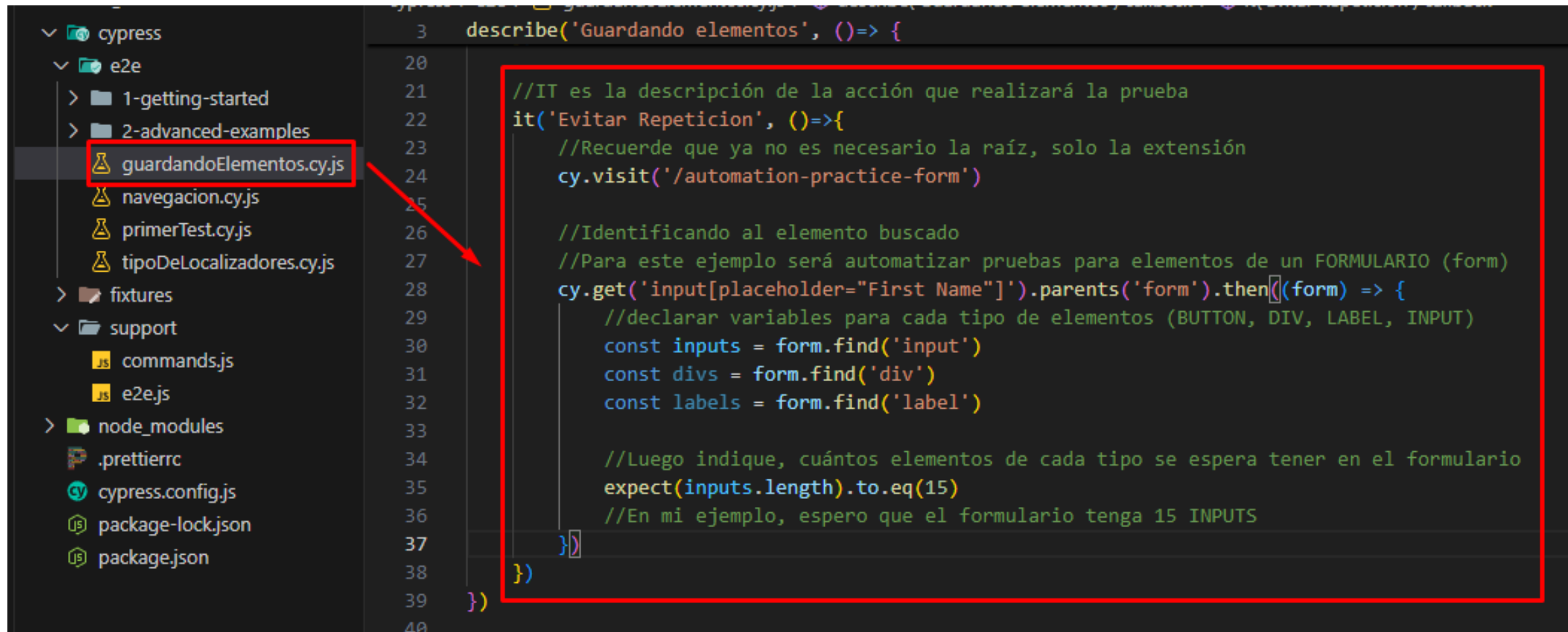


The screenshot shows the Visual Studio Code interface. On the left, the 'EXPLORADOR' (Explorer) sidebar displays the project structure. Under the 'CYPRESS\_UI' folder, the 'e2e' folder is expanded, and the file 'guardandoElementos.cy.js' is selected and highlighted with a red rectangle. A red arrow points from this file to the main editor. The main editor shows the content of 'guardandoElementos.cy.js' with the following code:

```
1  cypress > e2e > guardandoElementos.cy.js > describe('Guardando elementos') callback
2  //DESCRIBE es el nombre de la función que contendrá nuestra prueba
3  describe('Guardando elementos', () => {
4
5      //IT es la descripción de la acción que realizará la prueba
6      it('Repeticion', () => {
7          //Como en nuestra prueba queremos visitar una página, entonces coloque la URL
8          //Recuerde que ya no es necesario la raíz, solo la extensión
9          cy.visit('/automation-practice-form')
10
11         //Obteniendo el elemento padre del elementos buscado
12         cy.get('input[placeholder="First Name"]').parent()
13
14         //Obteniendo los elementos padres
15         cy.get('input[placeholder="First Name"]').parents()
16
17         //Obteniendo los elementos hijos desde el PADRE
18         cy.get('input[placeholder="First Name"]').parents().find('label')
19     })
20 })
21
```

# GUARDANDO ELEMENTOS CON CYPRESS

- Ahora, veamos el cambio cuando decidimos automatizar estas pruebas:



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with a 'cypress' folder containing an 'e2e' subfolder. Inside 'e2e', there is a '2-advanced-examples' folder, and within it, a file named 'guardandoElementos.cy.js' is highlighted with a red box. A red arrow points from this file to the code editor. The code editor shows the content of 'guardandoElementos.cy.js', which is a Cypress test file. The code is written in JavaScript and includes comments in Spanish. The code is as follows:

```
3 describe('Guardando elementos', () => {
20
21 //IT es la descripción de la acción que realizará la prueba
22 it('Evitar Repeticion', ()=>{
23 //Recuerde que ya no es necesario la raíz, solo la extensión
24 cy.visit('/automation-practice-form')
25
26 //Identificando al elemento buscado
27 //Para este ejemplo será automatizar pruebas para elementos de un FORMULARIO (form)
28 cy.get('input[placeholder="First Name"]').parents('form').then((form) => {
29 //declarar variables para cada tipo de elementos (BUTTON, DIV, LABEL, INPUT)
30 const inputs = form.find('input')
31 const divs = form.find('div')
32 const labels = form.find('label')
33
34 //Luego indique, cuántos elementos de cada tipo se espera tener en el formulario
35 expect(inputs.length).to.eq(15)
36 //En mi ejemplo, espero que el formulario tenga 15 INPUTS
37 })
38 })
39 })
40
```

# GUARDANDO ELEMENTOS CON CYPRESS

- Vea como reporta Cypress esta prueba automatizada:

The image is a composite of three screenshots from the Cypress test runner interface. The leftmost screenshot shows the 'Specs' panel on the left sidebar, with a red box around the 'Specs' header and a red arrow pointing to the 'guardandoElementos.cy.js' file listed below. The middle screenshot shows the command log for the 'guardandoElementos' spec, with a red box around the command `-assert expected 15 to equal 15`. The rightmost screenshot shows the application under test, which is a 'Student Registration Form' with fields for Name, Email, Gender, Mobile Number, Date of Birth, Subjects, Hobbies, Picture, and Current Address.

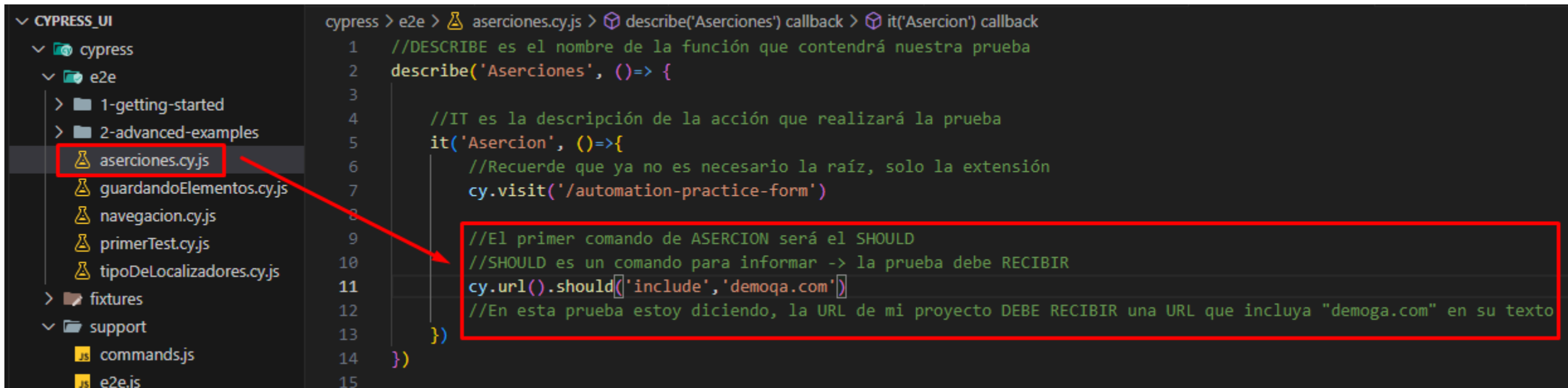
- Verifique usted el resultado al colocar otro valor diferente a 15. Con esto acabamos de automatizar nuestra primera prueba.

# Aserciones con Cypress

Práctica 07

# ASERCIONES CON CYPRESS

- Cuando nos referimos a una **aserción**, nos estamos refiriendo a “**un hecho o verdad esperado y comprobado**”.
- Para la práctica, vuelva a crear un nuevo archivo con el nombre “**aserciones.cy.js**”



```
cypress > e2e > aserciones.cy.js > describe('Aserciones') callback > it('Asercion') callback
1 //DESCRIBE es el nombre de la función que contendrá nuestra prueba
2 describe('Aserciones', ()=> {
3
4     //IT es la descripción de la acción que realizará la prueba
5     it('Asercion', ()=>{
6         //Recuerde que ya no es necesario la raíz, solo la extensión
7         cy.visit('/automation-practice-form')
8
9         //El primer comando de ASERCION será el SHOULD
10        //SHOULD es un comando para informar -> la prueba debe RECIBIR
11        cy.url().should('include', 'demoqa.com')
12        //En esta prueba estoy diciendo, la URL de mi proyecto DEBE RECIBIR una URL que incluya "demoqa.com" en su texto
13    })
14 })
15
```



# ASERCIONES CON CYPRESS

- Si verificamos CYPRESS, verá que la prueba es correcta:

The image shows the Cypress test runner interface. On the left, the 'Specs' tab is selected. The main area displays the test file 'aserciones.cy.js' with a green checkmark and the number '1', indicating a successful test run. The console shows the following output:

```
Printed output to your console
- assert expected https://demoqa.com/automation-practice-form to include demoqa.com
```

The browser window on the right shows the URL <https://demoqa.com/automation-practice-form>. The page contains a 'Student Registration Form' with fields for Name, Email, Gender, Mobile, and Date of Birth.

# ASERCIONES CON CYPRESS

- Recuerde que SHOULD es una función que puede agregarse a lo aprendido anteriormente, para muestra, queremos verificar que el elementos que tiene el ID “firstName” esté visible.

```
> 2-advanced-examples
  aserciones.cy.js
  guardandoElementos.cy.js
  navegacion.cy.js
  primerTest.cy.js
  tipoDeLocalizadores.cy.js
> fixtures
  support
    commands.js
    e2e.js
> node_modules
  .prettierrc

4 //IT es la descripción de la acción que realizará la prueba
5 it('Asercion', ()=>{
6   //Recuerde que ya no es necesario la raíz, solo la extensión
7   cy.visit('/automation-practice-form')
8
9   //El primer comando de ASERCION será el SHOULD
10  //SHOULD es un comando para informar -> la prueba debe RECIBIR
11  cy.url().should('include', 'demoqa.com')
12  //En esta prueba estoy diciendo, la URL de mi proyecto DEBE RECIBIR una URL que incluya "demoqa.com" en su texto
13
14  cy.get('#firstName').should('be.visible')
15  //En esta prueba estoy diciendo, el elemento con el ID firstName DEBE RECIBIR un VISIBLE en su característica
16
17 })
18
```

```
2 url
3 -assert expected https://demoqa.com/automation-practice-form to include demoqa.com
4 get #firstName
5 -assert expected <input#firstName..sm-2.form-control> to be visible
(xhr) GET 200 https://
```

Interactions

Book Store Application

First Name	
Email	name@example.com
Gender	<input type="radio"/> Male <input type="radio"/> Female <input type="radio"/> Other
Mobile (no digit)	Mobile Number
Date of Birth	28 Oct 2024
Subjects	
Hobbies	<input type="checkbox"/> Sports <input type="checkbox"/> Reading <input type="checkbox"/> Music
Picture	Select picture [Examinar...] No se ha seleccionado ningún archivo.
Current Address	Current Address

# ASERCIONES CON CYPRESS

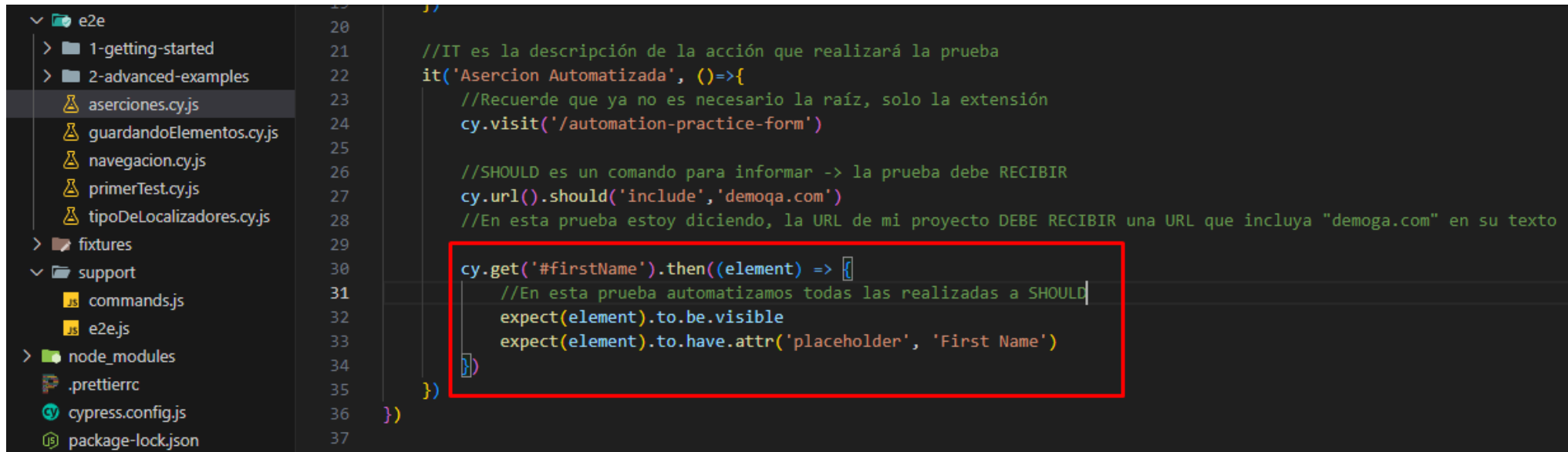
- De igual forma, también puede concatenar varios SHOULD para realizar varias pruebas en simultaneo a un solo elemento.

```
1  > 1-getting-started
2  > 2-advanced-examples
3  aserciones.cy.js
4  guardandoElementos.cy.js
5  navegacion.cy.js
6  primerTest.cy.js
7  tipoDeLocalizadores.cy.js
8
9  > fixtures
10 > support
11   commands.js
12   e2e.js
13
14 > node_modules
15 .prettierrc
16 cypress.config.js
17 package-lock.json
18
19 5  it('Asercion', ()=>{
20 6   //Recuerde que ya no es necesario la raíz, solo la extensión
21 7   cy.visit('/automation-practice-form')
22
23   //El primer comando de ASERCIION será el SHOULD
24   //SHOULD es un comando para informar -> la prueba debe RECIBIR
25 10  cy.url().should('include','demoqa.com')
26 11  //En esta prueba estoy diciendo, la URL de mi proyecto DEBE RECIBIR una URL que incluya "demoqa.com" en su texto
27 12
28 13
29 14  cy.get('#firstName').should('be.visible')
30 15  //En esta prueba estoy diciendo, el elemento con el ID firstName DEBE RECIBIR un VISIBLE en su característica
31 16
32 17  cy.get('#firstName').should('be.visible').should('have.attr', 'placeholder', 'First Name')
33 18  //En esta prueba estoy diciendo, que el elemento DEBE RECIBIR un atributo "placeholder" con el contenido de "First Name"
34 19  })
35 20  })
```

```
2  url
3  - assert expected https://
demoqa.com/automation-
practice-form to include
demoqa.com
4  get #firstName
5  - assert expected
<input#firstName..mr-
sm-2.form-control> to be
visible
6  get #firstName
7  - assert expected
<input#firstName..mr-
sm-2.form-control> to be
visible
8  - assert expected
<input#firstName..mr-
sm-2.form-control> to have
attribute placeholder with
the value First Name
(xhr) ● GET 200 https://
```

# ASERCIONES CON CYPRESS

- También podemos automatizar las pruebas realizadas con ASERCIONES, recuerde primero hacer la búsqueda del elemento que desea probar y luego:



```
20
21 //IT es la descripción de la acción que realizará la prueba
22 it('Asercion Automatizada', ()=>{
23   //Recuerde que ya no es necesario la raíz, solo la extensión
24   cy.visit('/automation-practice-form')
25
26   //SHOULD es un comando para informar -> la prueba debe RECIBIR
27   cy.url().should('include','demoqa.com')
28   //En esta prueba estoy diciendo, la URL de mi proyecto DEBE RECIBIR una URL que incluya "demoqa.com" en su texto
29
30   cy.get('#firstName').then((element) => {
31     //En esta prueba automatizamos todas las realizadas a SHOULD
32     expect(element).to.be.visible
33     expect(element).to.have.attr('placeholder', 'First Name')
34   })
35 })
36
37
```

- Felicitaciones, ya tiene 2 pruebas automatizadas.
- Para conocer a detalle todas las Aserciones y como utilizar la búsqueda de elementos, visite la página web:

<https://docs.cypress.io/app/references/assertions>

# Hooks con Cypress

Práctica 08

# HOOKS CON CYPRESS

- Los **hooks** o **GANCHOS** son proveedores que permiten, mediante condicionales, realizar operaciones antes/después de un conjunto de pruebas.
- La versatilidad de su uso es para encapsular bloques operativos como flujos de trabajo.
- Recordemos que Cypress es un framework que opera en diferentes capas, unitarias/integración E2E, que se puede complementar con librerías como Testing Library / Jest, por lo que dependiendo de su intención, podríamos atacar con mayor certeza casos de uso o requerimientos.

# HOOKS CON CYPRESS

- ¿Para qué debo instalar un HOOK en mi automatización de pruebas?
- Bueno, para ello vamos a hacer un ejemplo:

```
1 //DESCRIBE es el nombre de la función que contendrá nuestra prueba
2 describe('Aserciones', ()=> {
3
4   //IT es la descripción de la acción que realizará la prueba
5   it('Asercion', ()=>{
6     //Recuerde que ya no es necesario la raíz, solo la extensión
7     cy.visit('/automation-practice-form')
8
9     //El primer comando de ASERCIION será el SHOULD
10    //SHOULD es un comando para informar -> la prueba debe RECIBIR
11    cy.url().should('include', 'demoqa.com')
12    //En esta prueba estoy diciendo, la URL de mi proyecto DEBE RECIBIR una URL que incluya "demoqa.com" en su texto
13
14    cy.get('#firstName').should('be.visible')
15    //En esta prueba estoy diciendo, el elemento con el ID firstName DEBE RECIBIR un VISIBLE en su característica
16
17    cy.get('#firstName').should('be.visible').should('have.attr', 'placeholder', 'First Name')
18    //En esta prueba estoy diciendo, que el elemento DEBE RECIBIR un atributo "placeholder" con el contenido de "First Name"
19  })
20
```

# HOOKS CON CYPRESS

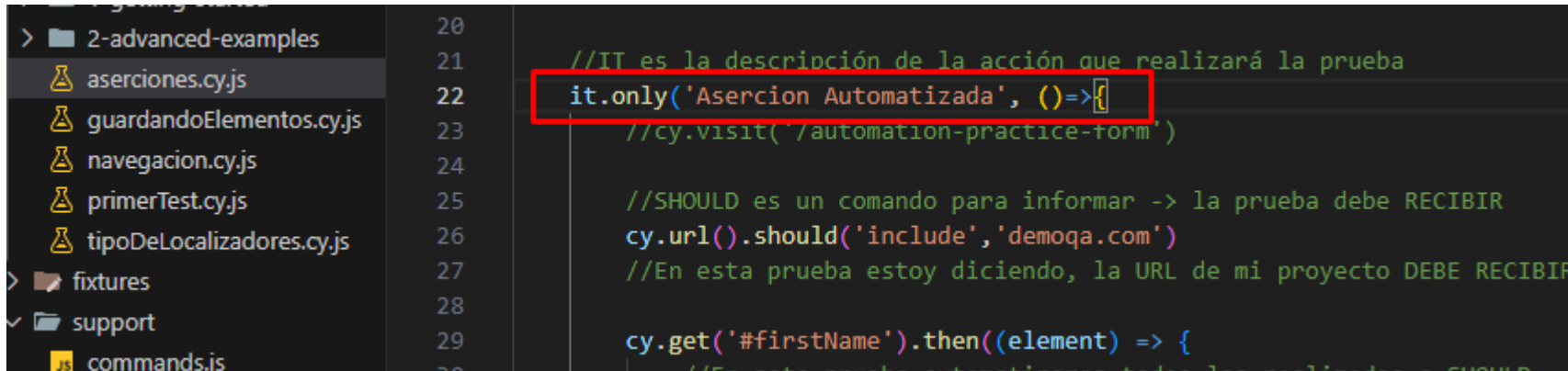
- Si observamos en nuestro proyecto de ASERSIONES, verá como la línea de VISIT se repite en las 3 pruebas, y eso hace que al momento de ejecutarse cada prueba, tenga que cargar la página en cada una de ellas.
- Lo que en cuestión de eficiencia, no lo es, pues solo es necesario visitar la página una vez y ya.
- Entonces, podríamos **eliminar** esa línea de código en las siguientes pruebas:

```
20
21 //IT es la descripción de la acción que realizará la prueba
22 it('Asercion Automatizada', ()=>{
23     //cy.visit('/automation-practice-form') ←
24
25     //SHOULD es un comando para informar -> la prueba debe RECIBIR
26     cy.url().should('include', 'demoqa.com')
```

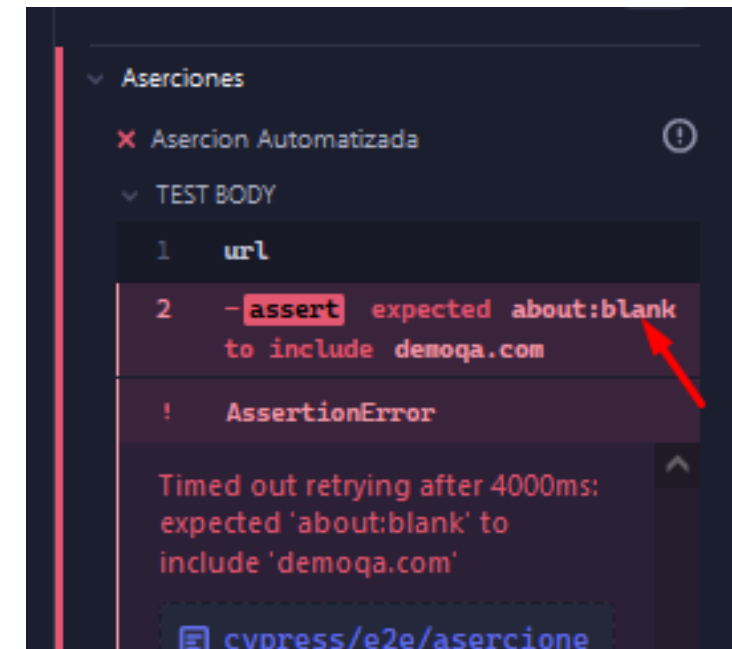


# HOOKS CON CYPRESS

- Si usted verifica en CYPRESS, verá como la ejecución tarda menos y no hay problemas, pero que pasa si, siguiendo el ejemplo, en realidad solo quiero ejecutar una de las pruebas y no todas a la vez.
- Para ello, agregue el comando ONLY luego del IT:



```
20
21 //IT es la descripción de la acción que realizará la prueba
22 it.only('Asercion Automatizada', ()=>{
23   //cy.visit('/automation-practice-form')
24
25   //SHOULD es un comando para informar -> la prueba debe RECIBIR
26   cy.url().should('include','demoqa.com')
27   //En esta prueba estoy diciendo, la URL de mi proyecto DEBE RECIBIR
28
29   cy.get('#firstName').then((element) => {
30     //En esta prueba automatizada, todos los elementos a GROUP
```



# HOOKS CON CYPRESS

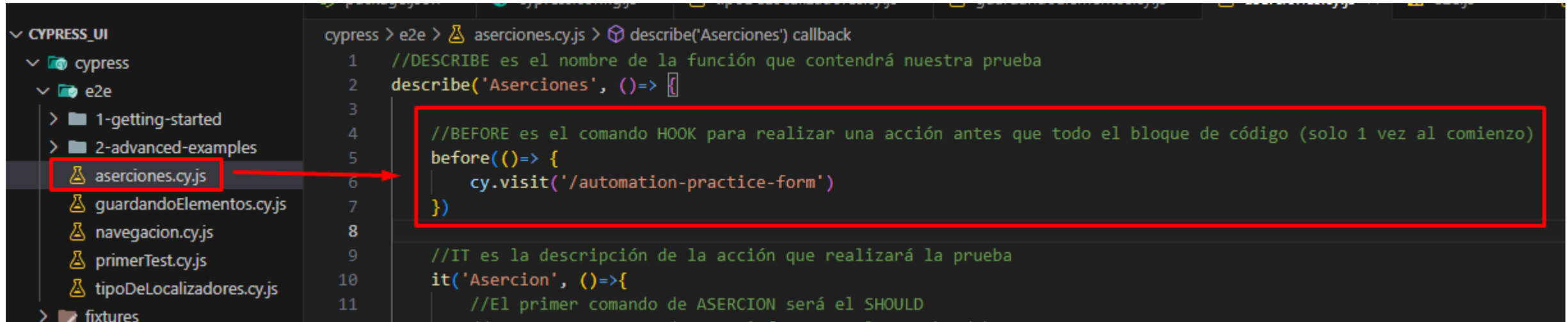
- De inmediato se genera el error de que no se ha visitado ninguna URL, pero ¿por qué? Si previamente yo había visitado la URL en la “**anterior**” prueba.
- Justamente para eso existen los HOOKS, para que este tipo de errores de ejecución no existan.

Entre los comandos de “tiempo” se encuentran:

- ✓ Before (antes que todo el bloque)
- ✓ After (después de todo el bloque)
- ✓ BeforeEach (antes de cada bloque)
- ✓ AfterEach (después de cada bloque)

# HOOKS CON CYPRESS

- Para nuestro ejemplo, como solo queremos que se visite la página al comienzo (antes de comenzar con TODO el bloque de pruebas), entonces utilizaré solamente BEFORE.



The screenshot shows the VS Code interface with the Cypress project structure on the left and a code editor on the right. In the left sidebar, the file `aserciones.cy.js` is highlighted with a red box, and a red arrow points from it to the code editor. The code editor displays the following code:

```
cypress > e2e > aserciones.cy.js > describe('Aserciones') callback
1 //DESCRIBE es el nombre de la función que contendrá nuestra prueba
2 describe('Aserciones', ()=> {
3
4   //BEFORE es el comando HOOK para realizar una acción antes que todo el bloque de código (solo 1 vez al comienzo)
5   before(()=> {
6     cy.visit('/automation-practice-form')
7   })
8
9   //IT es la descripción de la acción que realizará la prueba
10  it('Asercion', ()=>{
11    //El primer comando de ASERCIION será el SHOULD
```

# HOOKS CON CYPRESS

- Y ahora, al ejecutar SOLO una de las pruebas, verá como esta vez ya no existe el error:

The image shows a Cypress test runner interface on the left and a web application on the right.

**Cypress Test Runner (Left):**

- Specs: 1 passed, 0 failed, 0 pending.
- File: `aserciones.cyps` (00:16).
- Test Suite: `Aserciones`.
- Test: `Asercion Automatizada`.
- Hook: `> BEFORE ALL` (highlighted with a red box).
- Test Body (highlighted with a red box):

```
1 url
2 - assert expected https://demoqa.com/automation-practice-form to include demoqa.com
3 get #firstName
4 - assert expected <input#firstName..mr-sm-2.form-control> to be visible
5 - assert expected <input#firstName..mr-sm-2.form-control> to have attribute placeholder with the value First Name
```

**Web Application (Right):**

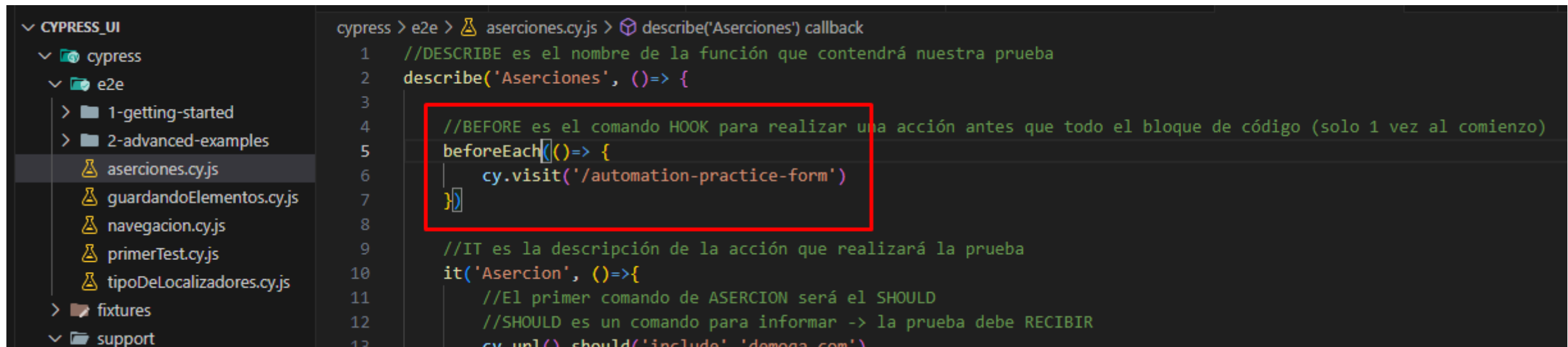
- URL: `https://demoqa.com/automation-practice-form`.
- Browser: Firefox 131.
- Page Title: `Practice Form`.
- Form Fields:
  - Name: First Name, Last Name
  - Email: name@example.com
  - Gender: Male, Female, Other
  - Mobile: Mobile Number
  - Date of Birth: 28 Oct 2024
  - Subjects:
  - Hobbies: Sports, Reading, Music
  - Picture: Select picture
- Footer: `Examinar... No se ha seleccionado ningún archivo.`

**ZeroStep Advertisement (Right):**

- Build Playwright Tests using AI
- Try for free →

# HOOKS CON CYPRESS

- Ahora, ¿Qué pasa si es necesario que en cada bloque de pruebas, si o si debe actualizar o “visitar” la URL indicada?
- Entonces, para ese caso solo coloque BEFOREEACH:



The screenshot shows the VS Code interface with the Cypress project structure on the left and a code editor on the right. The file explorer on the left shows the following structure:

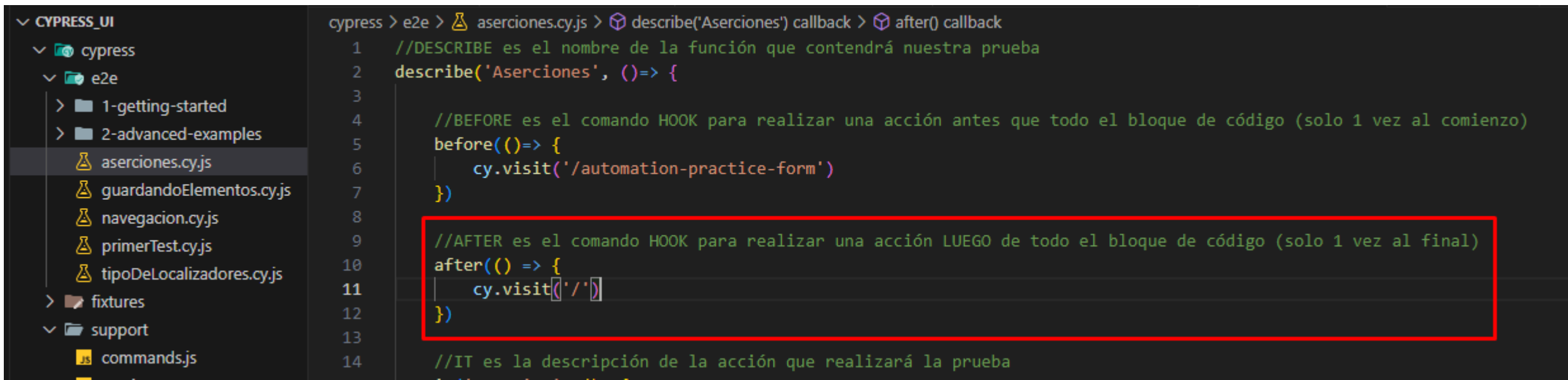
- ▼ CYPRESS\_UI
  - ▼ cypress
    - ▼ e2e
      - > 1-getting-started
      - > 2-advanced-examples
      - aserciones.cy.js
      - guardandoElementos.cy.js
      - navegacion.cy.js
      - primerTest.cy.js
      - tipoDeLocalizadores.cy.js
    - > fixtures
    - ▼ support

The code editor on the right shows the content of `aserciones.cy.js` with the following code:

```
cypress > e2e > aserciones.cy.js > describe('Aserciones') callback
1 //DESCRIBE es el nombre de la función que contendrá nuestra prueba
2 describe('Aserciones', ()=> {
3
4     //BEFORE es el comando HOOK para realizar una acción antes que todo el bloque de código (solo 1 vez al comienzo)
5     beforeEach(()=> {
6         cy.visit('/automation-practice-form')
7     })
8
9     //IT es la descripción de la acción que realizará la prueba
10    it('Asercion', ()=>{
11        //El primer comando de ASERCIION será el SHOULD
12        //SHOULD es un comando para informar -> la prueba debe RECIBIR
13        cy.url().should('include', 'demoga.com')
```

# HOOKS CON CYPRESS

- Ahora, ¿Y si deseo que luego de realizar todas las pruebas, se redirija automáticamente a la interfaz principal?
- Entonces, para ese caso solo coloque AFTER:

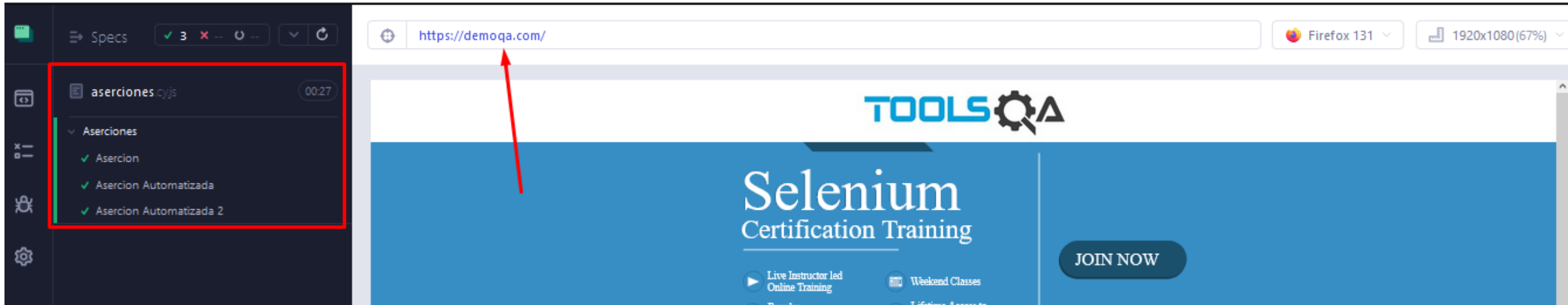


The screenshot shows the Cypress IDE interface. On the left, a file explorer displays the project structure under 'CYPRESS\_UI', including 'cypress', 'e2e', and 'fixtures' folders. The 'e2e' folder is expanded, showing several test files, with 'aserciones.cy.js' selected. The main editor area shows the content of 'aserciones.cy.js'. The code includes a 'describe' block for 'Aserciones'. Inside this block, there is a 'before' hook that visits the URL '/automation-practice-form'. Below it, an 'after' hook is highlighted with a red rectangle; this hook visits the root URL '/'. The code is written in a dark theme with syntax highlighting.

```
cyress > e2e > aserciones.cy.js > describe('Aserciones') callback > after() callback
1 //DESCRIBE es el nombre de la función que contendrá nuestra prueba
2 describe('Aserciones', ()=> {
3
4     //BEFORE es el comando HOOK para realizar una acción antes que todo el bloque de código (solo 1 vez al comienzo)
5     before(()=> {
6         cy.visit('/automation-practice-form')
7     })
8
9     //AFTER es el comando HOOK para realizar una acción LUEGO de todo el bloque de código (solo 1 vez al final)
10    after(() => {
11        cy.visit('/')
12    })
13
14    //IT es la descripción de la acción que realizará la prueba
```

# HOOKS CON CYPRESS

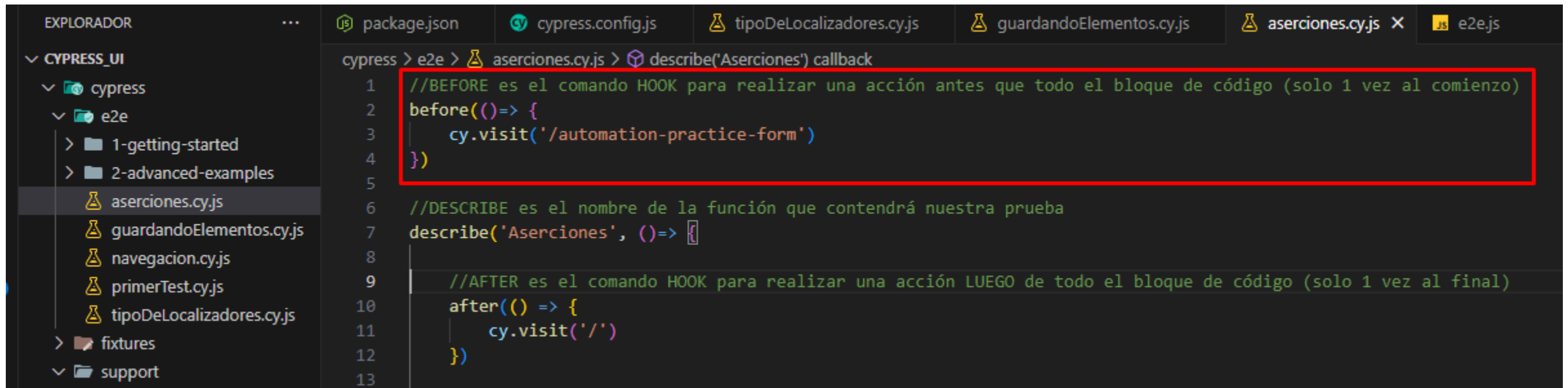
- Y mire como culmina la prueba:



- Respecto a AFTEREACH, recuerde que funciona igual a BEFOREEACH, solo que después de cada IT, por ende, es usted quién debe indicar que HOOKS colocar y como hacerlo.
- Recuerde que los HOOKS están para minimizar líneas de código repetitivas en cada prueba IT.

# HOOKS CON CYPRESS

- Una aclaración final, si observa bien, los comandos HOOKS se encuentran ahora dentro del DESCRIBE, pero usted puede colocar fuera del DESCRIBE si así lo considera necesario:



```
EXPLORADOR  ...  package.json  cypress.config.js  tipoDeLocalizadores.cy.js  guardandoElementos.cy.js  aserciones.cy.js X  e2e.js

v CYPRESS_UI
v cypress
v e2e
  > 1-getting-started
  > 2-advanced-examples
  aserciones.cy.js
  guardandoElementos.cy.js
  navegacion.cy.js
  primerTest.cy.js
  tipoDeLocalizadores.cy.js
  fixtures
  support

cypress > e2e > aserciones.cy.js > describe('Aserciones') callback
1  //BEFORE es el comando HOOK para realizar una acción antes que todo el bloque de código (solo 1 vez al comienzo)
2  before(()=> {
3    cy.visit('/automation-practice-form')
4  })
5
6  //DESCRIBE es el nombre de la función que contendrá nuestra prueba
7  describe('Aserciones', ()=> {
8
9    //AFTER es el comando HOOK para realizar una acción LUEGO de todo el bloque de código (solo 1 vez al final)
10   after(() => {
11     cy.visit('/')
12   })
13 }
```

- Obviamente con esta acción, tenga en cuenta que el flujo de ejecución cambiará.

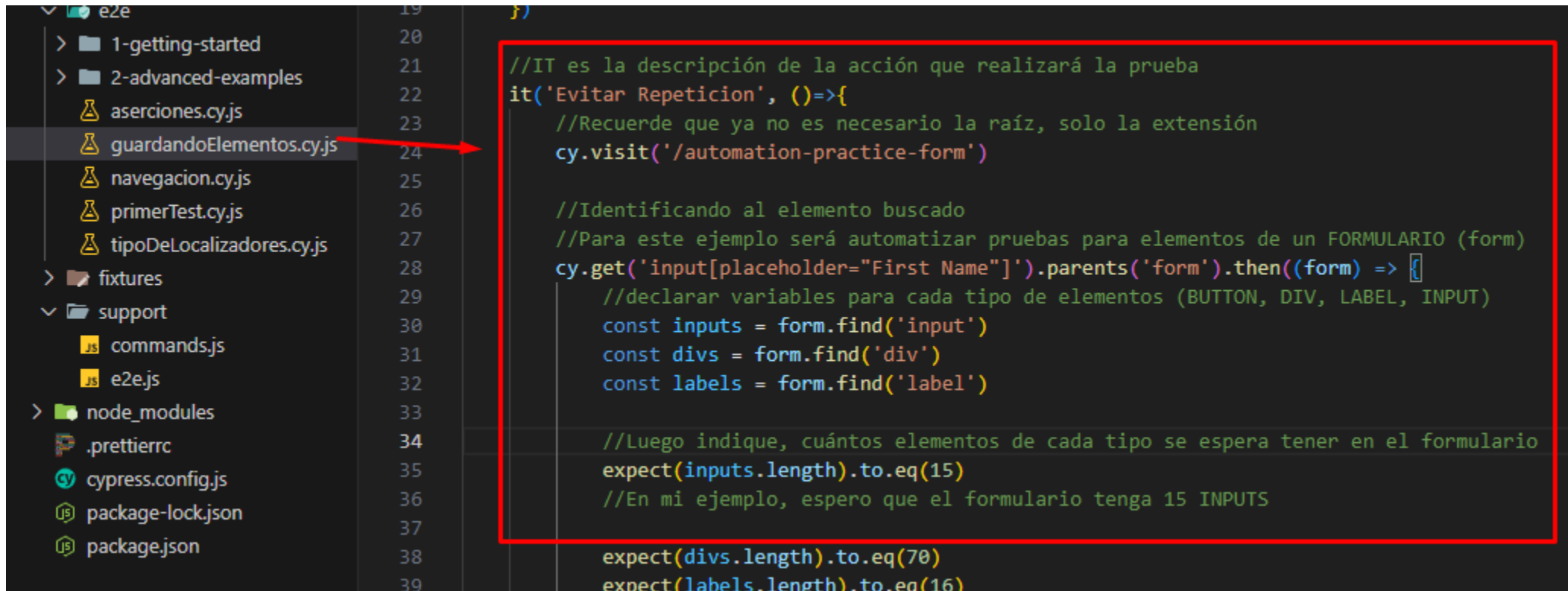


# **Debugger con Cypress**

Práctica 09

# DEBUGGER CON CYPRESS

- Para hacer el DEBUGG de nuestro proyecto, primero vayamos al archivo de nombre **guardandoElementos.cy.js**, dentro de este archivo, llegue a la prueba “Evitar Repetición”.



```
19  })
20
21  //IT es la descripción de la acción que realizará la prueba
22  it('Evitar Repetición', ()=>{
23    //Recuerde que ya no es necesario la raíz, solo la extensión
24    cy.visit('/automation-practice-form')
25
26    //Identificando al elemento buscado
27    //Para este ejemplo será automatizar pruebas para elementos de un FORMULARIO (form)
28    cy.get('input[placeholder="First Name"]').parents('form').then((form) => {
29      //declarar variables para cada tipo de elementos (BUTTON, DIV, LABEL, INPUT)
30      const inputs = form.find('input')
31      const divs = form.find('div')
32      const labels = form.find('label')
33
34      //Luego indique, cuántos elementos de cada tipo se espera tener en el formulario
35      expect(inputs.length).to.eq(15)
36      //En mi ejemplo, espero que el formulario tenga 15 INPUTS
37
38      expect(divs.length).to.eq(70)
39      expect(labels.length).to.eq(16)
```

# DEBUGGER CON CYPRESS

- El proceso de inspección de un código (**Debugger**) nos permite analizar procesos secuenciales como estados, valores de retorno, interacciones, etc.
- Como está descrita en su documentación, Cypress nos presenta varias piezas de información que ocurren cuando sucede un error.
  - ✓ **Nombre del error** - Describiendo el tipo de error
  - ✓ **Mensaje de error** - Describiendo en alto nivel el hecho o fallo, donde adicionalmente veremos una sugerencia para arreglar al mismo error
  - ✓ **Enlace adicional** - Enlace directo a la documentación de Cypress como referencia
  - ✓ **Sección de código** - Marcando la línea origen de error
  - ✓ **Bloque de código** - Fracción de código del origen de error
  - ✓ **Trazo de error** - Seguimiento del camino del error
  - ✓ **DevTools** - Botón que permite visualizar en DevTools nuestro asunto

# DEBUGGER CON CYPRESS

- Dentro de CYPRESS, para realizar el DEBBUG, primero debemos instalar un PLUGGIN en nuestro proyecto. Para ello, solo es necesario dirigirse al archivo de configuración y hacer el llamado desde ahí.

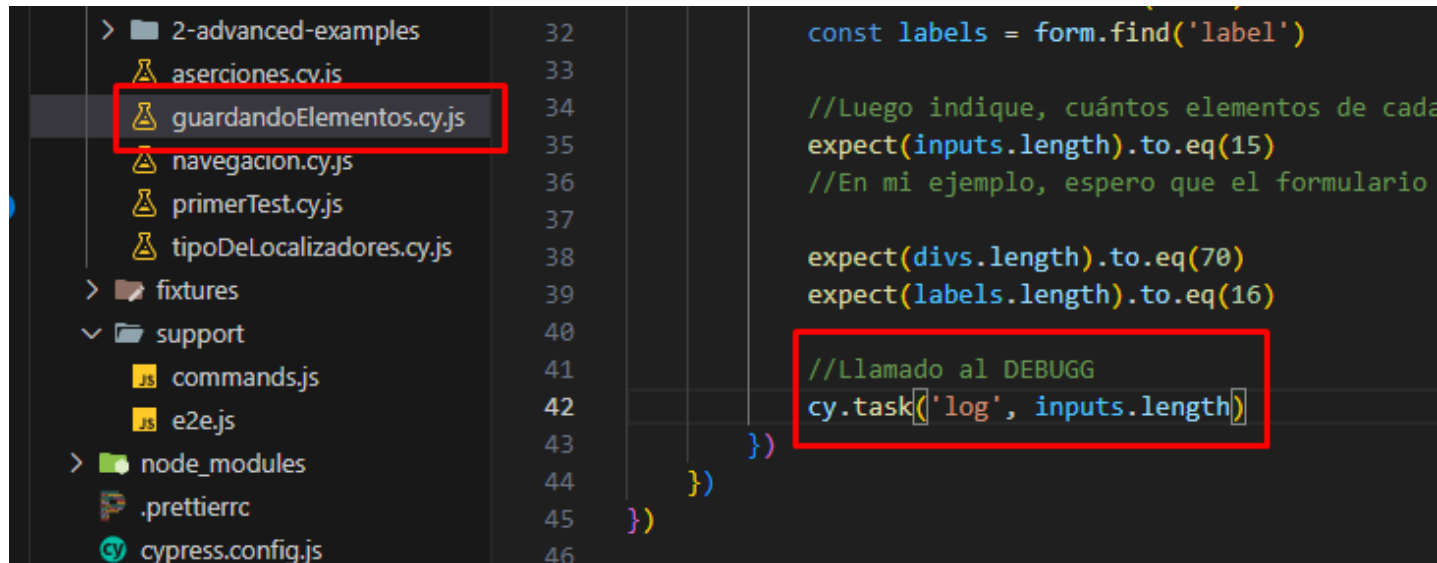


The screenshot shows the Visual Studio Code editor with the Cypress configuration file (`cypress.config.js`) open. The file explorer on the left shows the project structure, with `cypress.config.js` highlighted. The main editor displays the configuration code, and a red box highlights a custom plugin function `setupNodeEvents` that logs messages to the console.

```
1  const { defineConfig } = require("cypress");
2
3  module.exports = defineConfig({
4    e2e: {
5      baseUrl: 'https://demoqa.com/',
6
7      setupNodeEvents(on, config) {
8        //Plugin para imprimir en la consola la terminal
9        on('task', {
10          log(message) {
11            console.log('Mensaje del console log del task ' + message)
12            return null
13          }
14        })
15      },
16
17      excludeSpecPattern:[
18        "**/1-getting-started/*.js",
```

# DEBUGGER CON CYPRESS

- Para hacer el DEBUGG de nuestro proyecto, primero vayamos al archivo de nombre **guardandoElementos.cy.js**, dentro de este archivo, llegue a la prueba “Evitar Repetición”.
- Para el ejemplo de como funciona el PLUGIN instalado, vamos a pedir que nos muestre en consola el valor de la cantidad de INPUTS (recuerde que nuestra prueba asume que son 15, pero al hacer el DEBUGG, verificaremos cuánto es el valor realmente).



```
32 > 2-advanced-examples
33   aserciones.cy.js
34   guardandoElementos.cy.js
35   navegacion.cy.js
36   primerTest.cy.js
37   tipoDeLocalizadores.cy.js
38
39 > fixtures
40
41 > support
42   commands.js
43   e2e.js
44
45 > node_modules
46   .prettierrc
47   cypress.config.js
```

```
32 const labels = form.find('label')
33
34 //Luego indique, cuántos elementos de cada
35 expect(inputs.length).to.eq(15)
36 //En mi ejemplo, espero que el formulario
37
38 expect(divs.length).to.eq(70)
39 expect(labels.length).to.eq(16)
40
41 //Llamado al DEBUGG
42 cy.task('log', inputs.length)
43
44 })
45
46 }
```

# DEBUGGER CON CYPRESS

- Ahora vamos a la consola de VSC de nuestro proyecto. Para ver los resultados, tendrá que reiniciar el puerto de ejecución, para ello:
  - ✓ Simplemente, puede cerrar el navegador de ejecución.
  - ✓ O eliminar la terminal abierta (esto es más recomendable).



The image shows a screenshot of the Visual Studio Code (VSC) interface, specifically the terminal panel. The terminal is titled 'node' and shows the following commands and output:

```
PS D:\xampp\htdocs\cypress_ui> npm run test

> cypress_ui@1.0.0 test
> cypress open
```

A red arrow points to the close button (trash icon) in the top right corner of the terminal panel, indicating the recommended action to close the terminal.

# DEBUGGER CON CYPRESS

- Verifiquemos el resultado dentro de CYPRESS, en teoría nada debería cambiar allí:

The screenshot displays the Cypress test runner interface. On the left, a dark sidebar shows the test suite 'guardandoElementos' with a duration of 00:03. A red box highlights the 'Guardando elementos' section, which contains two items: 'Repeticion' and 'Evitar Repeticion', both marked with green checkmarks. The main area on the right shows the web application being tested, 'Smartindale Martindale Tester', with a 'Practice Form' section. The form includes a 'Student Registration Form' with input fields for 'First Name' and 'Last Name'.

# DEBUGGER CON CYPRESS

- Vuelva a ver la consola de VSC de nuestro proyecto:

```
2 get input[placeholder="First  
   Name"]  
3 parents form  
4 -assert expected 15 to equal 15  
5 -assert expected 70 to equal 70  
6 -assert expected 16 to equal 16  
   task log, 15  
(xhr) GET 200 https://  
      epl.adtrafficquality.google/
```

```
TERMINAL  PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  PUERTOS  node + v [icon] [icon] ... ^ x  
  
7&aiict=1&aiapm=0.3221&aiapmi=0.33938&aiombap=1&aiopts=1&aief=1&dt=1730148209312&bpp=1&bdt=524&idt=167&shv=r20241023&mjsv=m202410230101&ptt=9&saldra=  
a&cookie=ID%3D5732b521ca5474c6%3AT%3D1729543893%3ART%3D1730148196%3AS%3DALNI_MZRvcwXqsioCZiRyFVzWwVawDUKg&gpic=UID%3D00000a604c168c7b%3AT%3D17295438  
93%3ART%3D1730148196%3AS%3DALNI_MaH0iAt0P2T30Gl-ERJ6KLTYM7aZQ&eo_id_str=ID%3Da9001a088629177c%3AT%3D1729543893%3ART%3D1730148196%3AS%3DAA-AfjZXAFRZYn  
5jAx6wu2zeQUuc&nras=1&correlator=2365209555730&frm=23&ifc=1&pv=1&nhd=1&u_tz=-300&u_his=3&u_h=960&u_w=1708&u_ah=923&u_aw=1708&u_cd=24&u_sd=0.8&adx=-12  
245933&ady=-12245933&biw=1708&bih=816&isw=1899&ish=1080&ifk=1819281066&scr_x=0&scr_y=0&eid=44759876%2C44759927%2C42532524%2C95344190%2C95345281%2C310  
88451%2C95345688%2C95345789&oid=2&pvsid=2280320383150544&tmod=860869999&uas=0&nvt=1&fsapi=1&fc=1664&brdim=-10%2C-10%2C-10%2C-10%2C1708%2C0%2C1728%2C9  
43%2C1920%2C1080&vis=1&rsz=%7C%7Cs%7C&abl=NS&fu=32772&bc=31&bz=1.01&ifi=1&uci=1.md8it4fwuo0r&fsb=1&dtd=178 200 58.285 ms - -  
GET /pagead/gen_204?id=ach_evt&tn=DIV&id=main-pane&cls=flex&ign=false&pw=1707&ph=816&x=853.5&y=676.8 204 43.667 ms - -  
GET /pagead/gen_204?id=ach_evt&tn=DIV&ign=true&pw=1707&ph=816&x=0&y=0 204 45.169 ms - -  
GET /pagead/gen_204?id=ach_evt&tn=DIV&id=main-pane&cls=flex&ign=false&pw=1707&ph=816&x=853.5&y=0 204 46.103 ms - -  
GET /safeframe/1-0-40/html/container.html?n=1 200 280.009 ms - -  
POST /__cypress/add_verified_command 204 0.293 ms - -  
Mensaje del console log del task 15  
GET /gampad/ads?pvsid=2280320383150544&correlator=3863669141768929&eid=31088532%2C95344208&output=ldjh&gdfp_req=1&vrg=202410240101&ptt=17&impl=fif&iu  
_parts=21849154601%3A22343295815%2CAd.Plus-Anchor&enc_prev_ius=%2F0%2F1&prev_iu_szs=320x100&ifi=2&didk=3917016210&sfv=1-0-40&sc=1&cookie=ID%3D5732b52
```

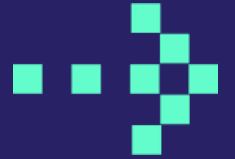


# DEBUGGER CON CYPRESS

- La finalidad de hacer el DEBUGG; es para conocer los resultados sin la necesidad de tener el navegador al lado.
- Pero estas finalidades la veremos en siguientes sesiones (cuando usemos Cypress con Backend).



**Muchas gracias**



TOMA  
FUERZA.

TU PROPÓSITO

HAZLO  
CONTINENTAL<sup>®</sup>

