

EXERCISES QUE TRABAJAREMOS EN LA CUE

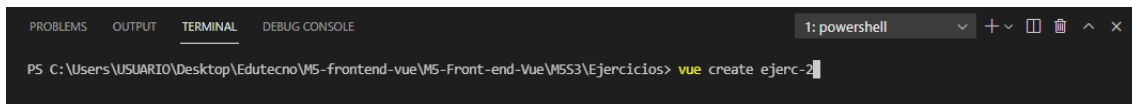
- EXERCISE 1: CONFIGURANDO AMBIENTE.
- EXERCISE 2: CORRIENDO EL PRIMER TEST E2E.
- EXERCISE 3: CREANDO UN TEST END TO END.

EXERCISE 1: CONFIGURANDO AMBIENTE

INTRODUCCIÓN

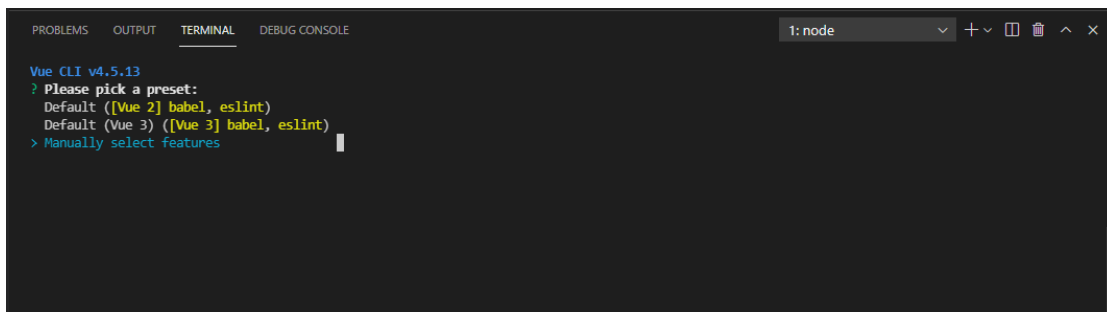
Usando la interfaz de línea de comandos (command line interface o CLI), **Vue** nos permite iniciar proyectos empleando todas las herramientas necesarias para desarrollar y testear. De esta forma, podemos agregar las dependencias necesarias que se ocupan en proyectos **Vue**, tales como: **Vue-Test-Utils**, **Mocha**, **Chai**, **Cypress**, **Vue router**, **vuex**, entre otros, con solo seleccionar unas opciones.

Para comenzar un proyecto con **Vue**, debemos previamente tener instalado **Vue-CLI**. Como en el Drill anterior ya lo instalamos, vamos a usarlo nuevamente. Para crearlo, debemos posicionarnos en la carpeta donde queremos trabajar, y escribir en la terminal el siguiente comando: **“vue create nombre_proyecto”**. Aquí debes reemplazar “nombre_proyecto”, por el nombre que quieras darle.



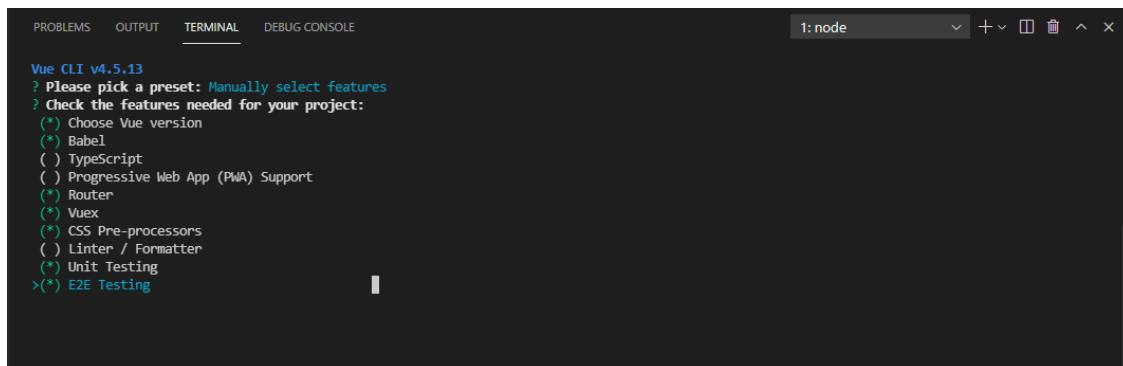
```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
1: powershell
PS C:\Users\USUARIO\Desktop\EduTecno\MS-frontend-vue\MS-Front-end-Vue\MS53\Ejercicios> vue create ejerc-2
```

Cuando presionamos “enter”, comienzan a aparecer diferentes vistas. En la primera vamos a elegir, utilizando para ello las flechas del teclado arriba y abajo, **“Manually select features”**, así podremos seleccionar, según nuestras necesidades, las herramientas que instalaremos, y de nuevo tecla “enter”.



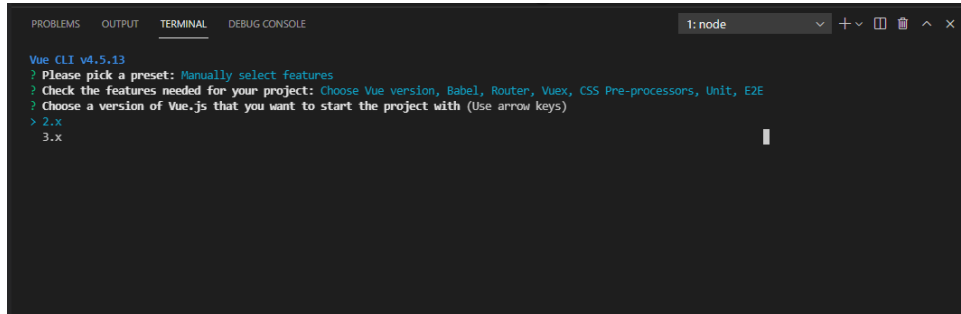
```
Vue CLI v4.5.13
? Please pick a preset:
  Default ([Vue 2] babel, eslint)
  Default (Vue 3) ([Vue 3] babel, eslint)
> Manually select features
```

En la vista que sigue, para seleccionar una dependencia a instalar, debemos presionar la tecla “espacio”. En este caso, agregaremos: **Babel**, **Router**, **Vuex**, **CSS Pre-procesor**, **Unit Testing** y **E2E**. Una vez marcadas nuestras preferencias, presionamos “enter”.



```
Vue CLI v4.5.13
? Please pick a preset: Manually select features
? Check the features needed for your project:
  (*) Choose Vue version
  (*) Babel
  ( ) TypeScript
  ( ) Progressive Web App (PWA) Support
  (*) Router
  (*) Vuex
  (*) CSS Pre-processors
  ( ) Linter / Formatter
  (*) Unit Testing
> (*) E2E Testing
```

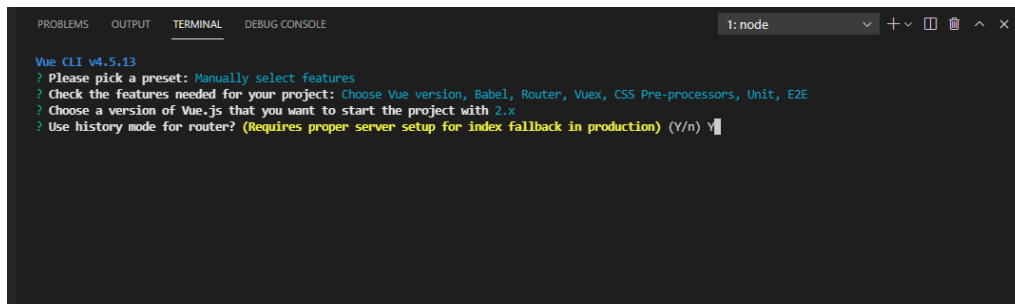
A continuación, vamos a seleccionar la versión de **“vue 2.x”**, y de inmediato presionamos “enter”.



```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
1: node

Vue CLI v4.5.13
? Please pick a preset: Manually select features
? Check the features needed for your project: Choose Vue version, Babel, Router, Vuex, CSS Pre-processors, Unit, E2E
? Choose a version of Vue.js that you want to start the project with (Use arrow keys)
> 2.x
  3.x
```

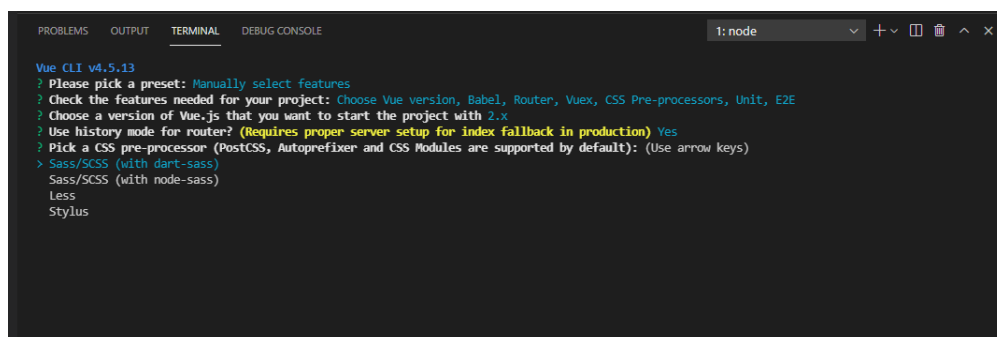
Ahora nos pregunta: **“Use history mode for router?”**, le escribimos **“Y”**, y presionamos “enter”.



```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
1: node

Vue CLI v4.5.13
? Please pick a preset: Manually select features
? Check the features needed for your project: Choose Vue version, Babel, Router, Vuex, CSS Pre-processors, Unit, E2E
? Choose a version of Vue.js that you want to start the project with 2.x
? Use history mode for router? (Requires proper server setup for index fallback in production) (Y/n) Y
```

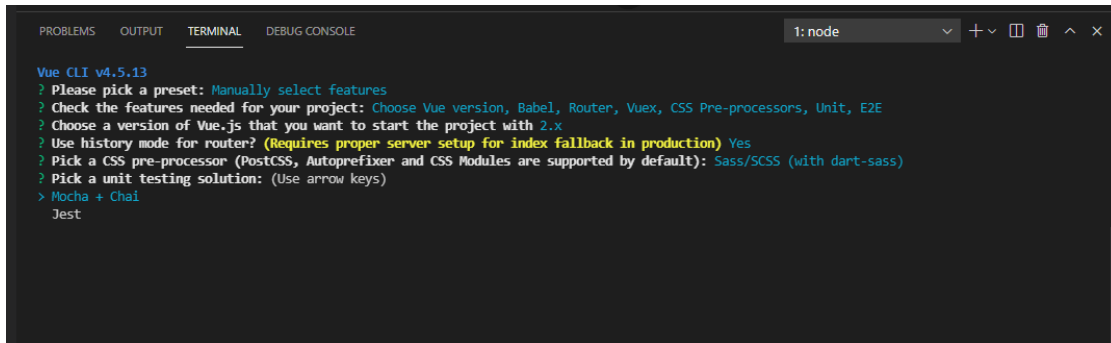
En la siguiente pregunta nos dan a elegir entre distintos preprocesadores de **CSS**. Seleccionamos: **Sass/SCSS (with dart-sass)**, y presionamos “enter”.



```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
1: node

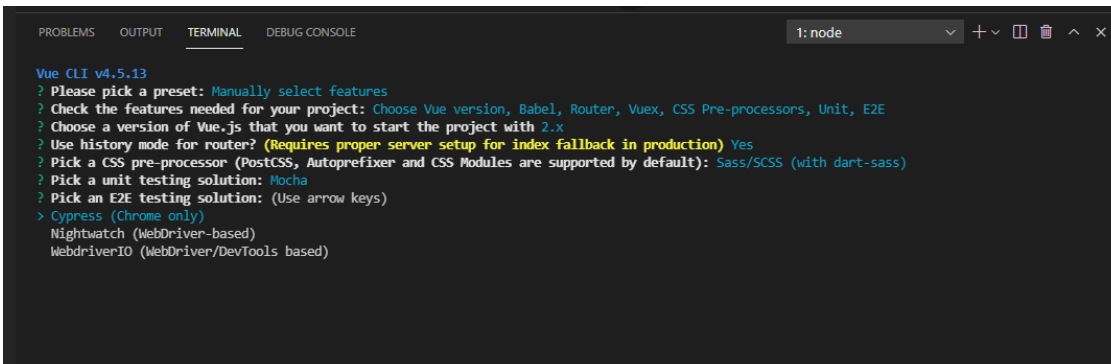
Vue CLI v4.5.13
? Please pick a preset: Manually select features
? Check the features needed for your project: Choose Vue version, Babel, Router, Vuex, CSS Pre-processors, Unit, E2E
? Choose a version of Vue.js that you want to start the project with 2.x
? Use history mode for router? (Requires proper server setup for index fallback in production) Yes
? Pick a CSS pre-processor (PostCSS, Autoprefixer and CSS Modules are supported by default): (Use arrow keys)
> Sass/SCSS (with dart-sass)
  Sass/SCSS (with node-sass)
  Less
  Stylus
```

En la siguiente pregunta nos permiten elegir entre usar **Mocha + Chai**, o **Jest**, para realizar las pruebas. Vamos a seleccionar: **Mocha + Chai**, y presionamos “enter”.



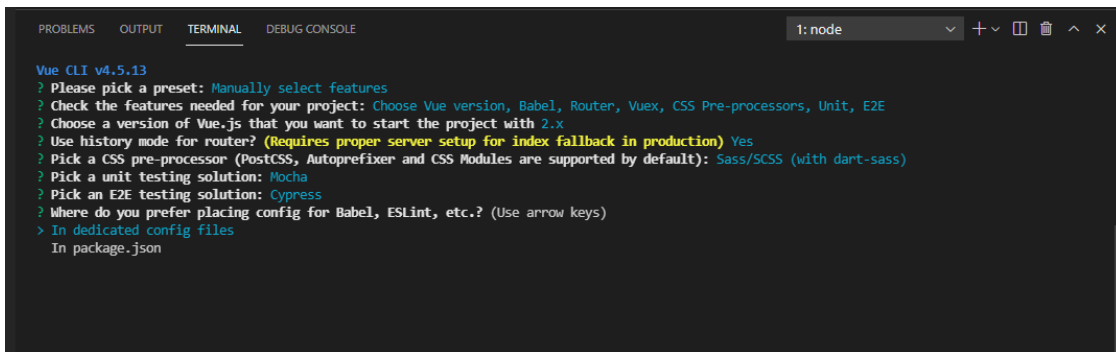
```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE 1: node
Vue CLI v4.5.13
? Please pick a preset: Manually select features
? Check the features needed for your project: Choose Vue version, Babel, Router, Vuex, CSS Pre-processors, Unit, E2E
? Choose a version of Vue.js that you want to start the project with 2.x
? Use history mode for router? (Requires proper server setup for index fallback in production) Yes
? Pick a CSS pre-processor (PostCSS, Autoprefixer and CSS Modules are supported by default): Sass/SCSS (with dart-sass)
? Pick a unit testing solution: (Use arrow keys)
> Mocha + Chai
  Jest
```

En la siguiente pregunta podemos elegir entre usar: **Cypress**, **Nightwatch**, o **WebdriverIO**, para realizar las pruebas **E2E**. Vamos a seleccionar **Cypress**, y presionamos “enter”.



```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE 1: node
Vue CLI v4.5.13
? Please pick a preset: Manually select features
? Check the features needed for your project: Choose Vue version, Babel, Router, Vuex, CSS Pre-processors, Unit, E2E
? Choose a version of Vue.js that you want to start the project with 2.x
? Use history mode for router? (Requires proper server setup for index fallback in production) Yes
? Pick a CSS pre-processor (PostCSS, Autoprefixer and CSS Modules are supported by default): Sass/SCSS (with dart-sass)
? Pick a unit testing solution: Mocha
? Pick an E2E testing solution: (Use arrow keys)
> Cypress (Chrome only)
  Nightwatch (WebDriver-based)
  WebdriverIO (WebDriver/DevTools based)
```

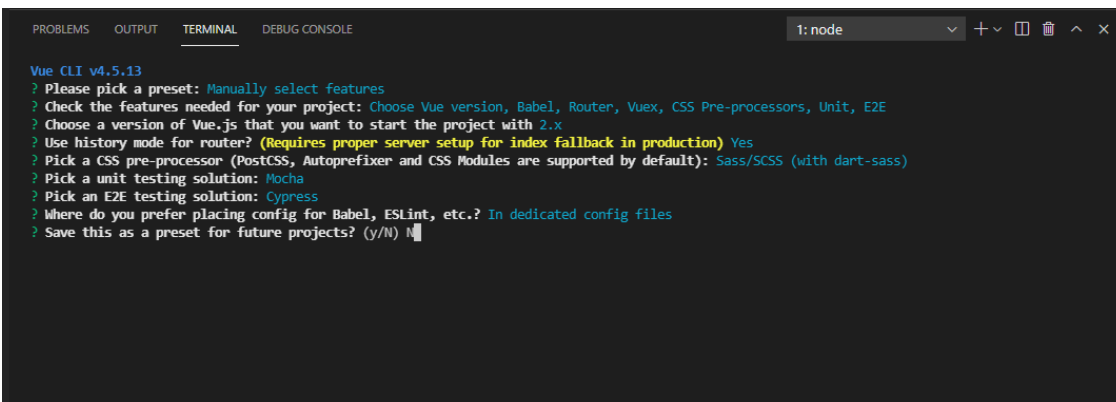
En la siguiente pregunta, hay que dejar seleccionada la alternativa: **“In dedicated config files”**, y presionamos “enter”.



```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
1: node

Vue CLI v4.5.13
? Please pick a preset: Manually select features
? Check the features needed for your project: Choose Vue version, Babel, Router, Vuex, CSS Pre-processors, Unit, E2E
? Choose a version of Vue.js that you want to start the project with 2.x
? Use history mode for router? (Requires proper server setup for index fallback in production) Yes
? Pick a CSS pre-processor (PostCSS, Autoprefixer and CSS Modules are supported by default): Sass/SCSS (with dart-sass)
? Pick a unit testing solution: Mocha
? Pick an E2E testing solution: Cypress
? Where do you prefer placing config for Babel, ESLint, etc.? (Use arrow keys)
> In dedicated config files
  In package.json
```

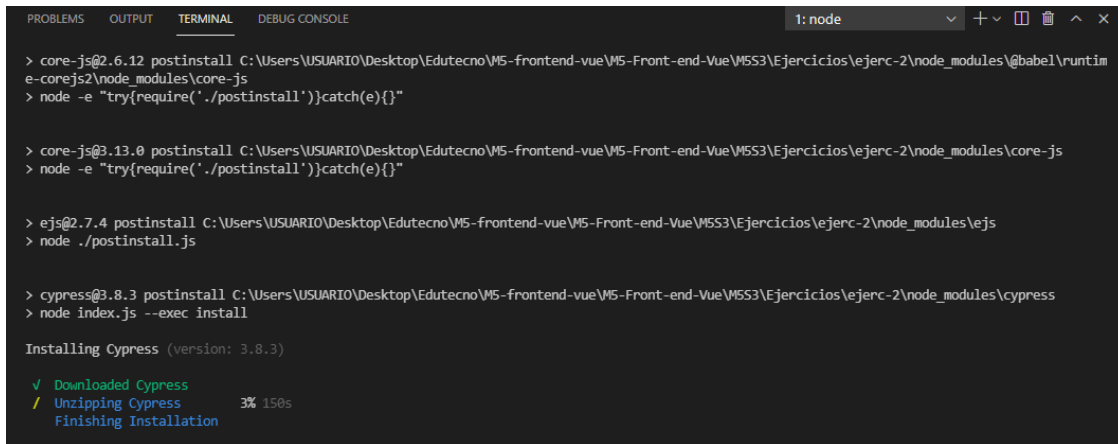
Por último, nos pregunta si queremos conservar la configuración para los próximos proyectos, le ponemos una **“N”** de No, ya que así podremos seguir seleccionando lo que queramos según nuestras necesidades, y presionamos “enter”.



```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
1: node

Vue CLI v4.5.13
? Please pick a preset: Manually select features
? Check the features needed for your project: Choose Vue version, Babel, Router, Vuex, CSS Pre-processors, Unit, E2E
? Choose a version of Vue.js that you want to start the project with 2.x
? Use history mode for router? (Requires proper server setup for index fallback in production) Yes
? Pick a CSS pre-processor (PostCSS, Autoprefixer and CSS Modules are supported by default): Sass/SCSS (with dart-sass)
? Pick a unit testing solution: Mocha
? Pick an E2E testing solution: Cypress
? Where do you prefer placing config for Babel, ESLint, etc.? In dedicated config files
? Save this as a preset for future projects? (y/N) N
```

Como vemos, se está empezando a crear e instalar el proyecto. El tiempo que le tome dependerá de nuestra conexión a internet, y del equipo con el que estemos trabajando. También podemos ver que ya descargó **Cypress**, y como lo empieza a descomprimir para instalarlo.



```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE 1: node
> core-js@2.6.12 postinstall C:\Users\USUARIO\Desktop\Edutecno\W5-frontend-vue\W5-Front-end-Vue\W5S3\Ejercicios\ejerc-2\node_modules\@babel\runtim
e-corejs2\node_modules\core-js
> node -e "try{require('./postinstall')}catch(e){}"


> core-js@3.13.0 postinstall C:\Users\USUARIO\Desktop\Edutecno\W5-frontend-vue\W5-Front-end-Vue\W5S3\Ejercicios\ejerc-2\node_modules\core-js
> node -e "try{require('./postinstall')}catch(e){}"

> ejs@2.7.4 postinstall C:\Users\USUARIO\Desktop\Edutecno\W5-frontend-vue\W5-Front-end-Vue\W5S3\Ejercicios\ejerc-2\node_modules\ejs
> node ./postinstall.js

> cypress@3.8.3 postinstall C:\Users\USUARIO\Desktop\Edutecno\W5-frontend-vue\W5-Front-end-Vue\W5S3\Ejercicios\ejerc-2\node_modules\cypress
> node index.js --exec install

Installing Cypress (version: 3.8.3)
✓ Downloaded Cypress
/ Unzipping Cypress 3% 150s
  Finishing Installation
```

Cuando termina, podemos ver dos comandos: el primero, que nos indica que debemos ingresar a nuestra carpeta del proyecto **"cd nombre_proyecto"**, y el segundo, para compilar y levantar el proyecto **"npm run serve"**.



```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE 1: powershell
71 packages are looking for funding
  run `npm fund` for details

✓ Invoking generators...
📦 Installing additional dependencies...

added 35 packages from 49 contributors in 13.224s

72 packages are looking for funding
  run `npm fund` for details

🔗 Running completion hooks...

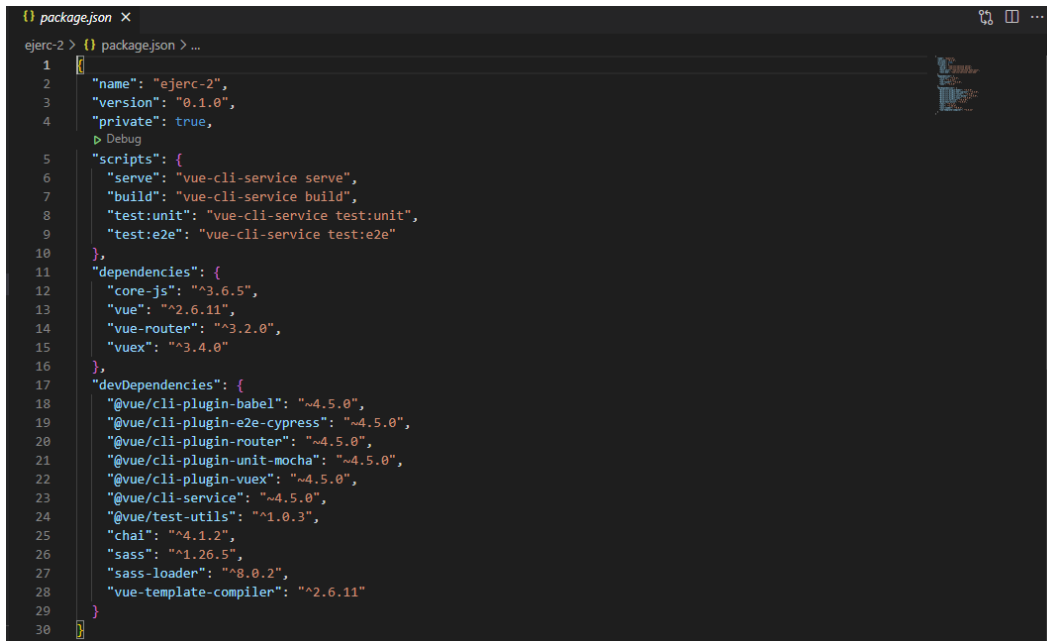
📄 Generating README.md...

🎉 Successfully created project ejerc-2.
👉 Get started with the following commands:

$ cd ejerc-2
$ npm run serve

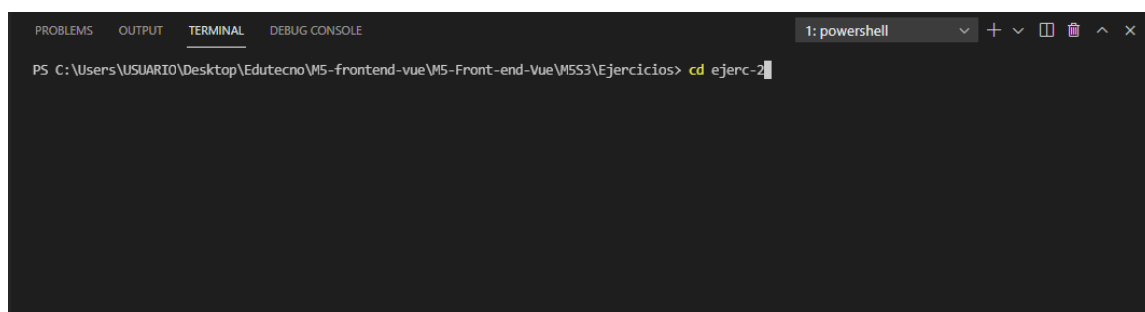
PS C:\Users\USUARIO\Desktop\Edutecno\W5-frontend-vue\W5-Front-end-Vue\W5S3\Ejercicios>
```

Revisemos primero la estructura del proyecto. En el archivo `package.json`, están todas nuestras dependencias. Aquí podemos ver que ya estás instalados: `Cypress`, `Mocha`, `Vue-Test-Utils`, pues `CLI` lo instala de manera automática cuando seleccionamos `Unit Testing` y `E2E`.



```
1 {
2   "name": "ejerc-2",
3   "version": "0.1.0",
4   "private": true,
5   "scripts": {
6     "serve": "vue-cli-service serve",
7     "build": "vue-cli-service build",
8     "test:unit": "vue-cli-service test:unit",
9     "test:e2e": "vue-cli-service test:e2e"
10  },
11  "dependencies": {
12    "core-js": "^3.6.5",
13    "vue": "^2.6.11",
14    "vue-router": "^3.2.0",
15    "vuex": "^3.4.0"
16  },
17  "devDependencies": {
18    "@vue/cli-plugin-babel": "~4.5.0",
19    "@vue/cli-plugin-e2e-cypress": "~4.5.0",
20    "@vue/cli-plugin-router": "~4.5.0",
21    "@vue/cli-plugin-unit-mocha": "~4.5.0",
22    "@vue/cli-plugin-vuex": "~4.5.0",
23    "@vue/cli-service": "~4.5.0",
24    "@vue/test-utils": "^1.0.3",
25    "chai": "^4.1.2",
26    "sass": "^1.26.5",
27    "sass-loader": "^8.0.2",
28    "vue-template-compiler": "^2.6.11"
29  }
30 }
```

De inmediato nos dirigimos a nuestra carpeta del proyecto, escribiendo en la terminal el comando: `"cd nombre_proyecto"`, y presionamos "enter".



```
1: powershell
PS C:\Users\USUARIO\Desktop\Edutecno\MS-front-end-vue\MS-Front-end-Vue\MS53\Ejercicios> cd ejerc-2
```

Ahora, podemos escribir en la consola `"npm run serve"`, y presionamos "enter". Así estamos compilando el proyecto para levantarlo.

```
PS C:\Users\USUARIO\Desktop\Edutecno\MS-frontent-vue\MS-Front-end-Vue\MS3\Ejercicios\ejerc-2> npm run serve
```

Una vez terminada la compilación, nos da el enlace para verlo en un servidor local. Si abrimos ese enlace en un navegador, podremos ver algo como esto:

[Home](#) | [About](#)



Welcome to Your Vue.js App

For a guide and recipes on how to configure / customize this project,
check out the [vue-cli documentation](#).

Installed CLI Plugins

[babel](#) [router](#) [vuex](#) [unit-mocha](#) [e2e-cypress](#)

Essential Links

[Core Docs](#) [Forum](#) [Community Chat](#) [Twitter](#) [News](#)

Ecosystem

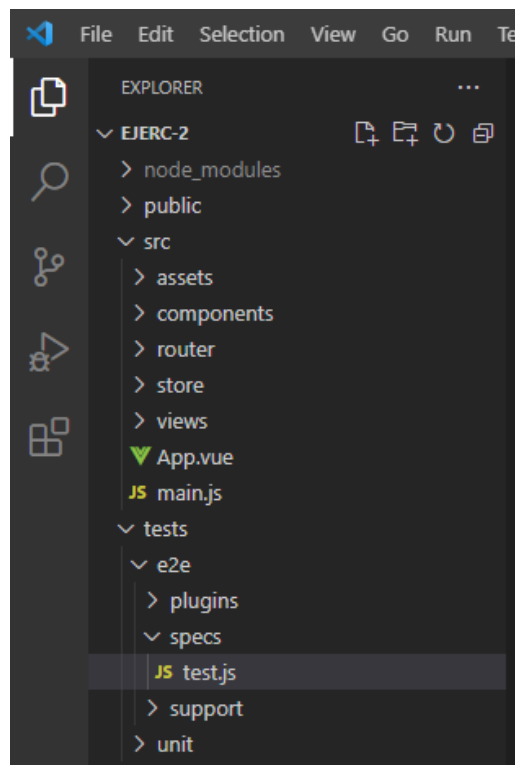
EXERCISE 2: CORRIENDO EL PRIMER TEST E2E

INTRODUCCIÓN

Ahora que hemos logrado crear nuestro proyecto, y confirmar que se está visualizando de manera correcta en el navegador, vamos a revisarlo.

ESTRUCTURA DE LA CARPETA TEST

Dentro de la carpeta **"test"**, existen dos carpetas: **"e2e"** y **"unit"**. Vamos a revisar la primera. En su interior, encontramos que tiene tres carpetas: **"plugins"**, **"specs"** y **"support"**. En **"specs"**, es donde se almacenan nuestras pruebas de extremo a extremo, y ahí encontramos un ejemplo de test. Lo revisaremos rápidamente.



ESTRUCTURA DE LA PRUEBA PREDISEÑADA

En la línea 1, encontramos comentado el enlace para acceder a la documentación de **Cypress**.

```
JS test.js x
tests > e2e > specs > JS test.js > ...
1 // https://docs.cypress.io/api/introduction/api.html
2
```

Mientras que en la línea 3, encontramos la palabra **“describe”**: un método de **Mocha** que permite crear un bloque para agrupar varias pruebas relacionadas. Recibe dos argumentos: el primero es el nombre del grupo de pruebas, y el segundo es una función **callback**. En este caso, el **string** corresponde a una descripción simbólica del test.

```
3 describe('My First Test', () => {
```

En la siguiente línea, encontramos la variable **“it”**, perteneciente a **Mocha**, que permite crear una prueba unitaria. Recibe dos argumentos: una cadena que explica qué debe hacer la prueba, y una función **callback**. En este caso, el **string** corresponde a una descripción de lo que hará la prueba, que es visitar la **URL** raíz de la aplicación.

```
4 it('Visits the app root url', () => {
```

En la línea 5, encontramos **“cy.visit(//)”**, perteneciente a **Cypress**. Sirve para cargar una **URL**, en este caso, nuestra “Home”.

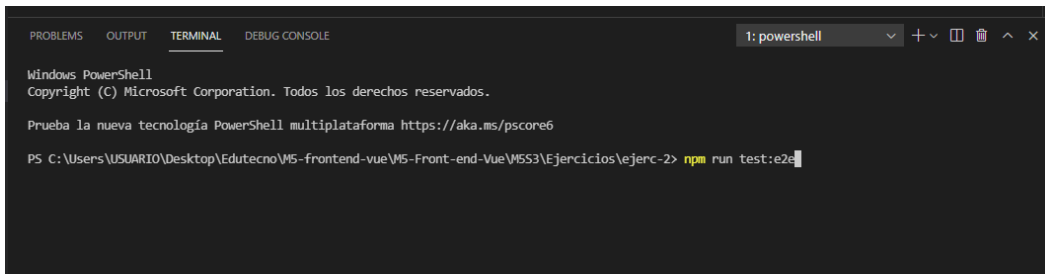
```
5 cy.visit('/')
6
```

En la línea 6, encontramos: `cy.contains('h1', 'Welcome to Your Vue.js App')`. Esta función permite seleccionar, en el primer parámetro, según el selector, y en el segundo, para comparar el contenido.

```
6 | cy.contains('h1', 'Welcome to Your Vue.js App')
```

CORRIENDO LA PRUEBA

Como ya terminamos de analizar la prueba, vamos a correrla. Para ello, escribiremos en una nueva terminal: `npm run test:e2e`, y presionamos “enter”.



```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
1: powershell
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pscore6

PS C:\Users\USUARIO\Desktop\Edutecno\MS-front-end-Vue\MS3\Ejercicios\ejerc-2> npm run test:e2e
```

De inmediato, vemos que comienza el proceso de compilado, que tardará según las capacidades de nuestro computador. Una vez terminada la compilación, se empezará a ejecutar el proceso de correr la prueba `end to end`.

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
1: node

> ejerc-2@0.1.0 test:e2e C:\Users\USUARIO\Desktop\Edutecno\M5-frontend-vue\M5-Front-end-Vue\M53\Ejercicios\ejerc-2
> vue-cli-service test:e2e

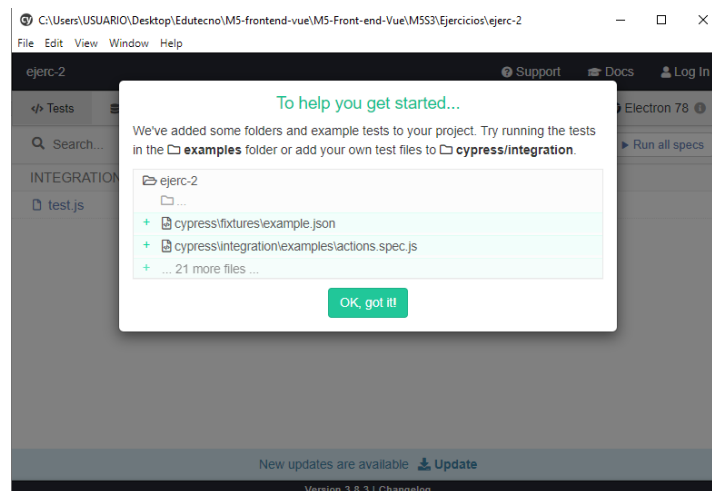
[INFO] Starting e2e tests...
[INFO] Starting development server...

[DONE] Compiled successfully in 2204ms

App running at:
- Local: http://localhost:8080/
- Network: http://192.168.0.23:8080/

App is served in production mode.
Note this is for preview or E2E testing only.
```

Aunque aún no ha terminado de ejecutarse la prueba, vemos que se abre la interfaz de usuario de **Cypress**. Hay una ventana modal, que nos da un par de recomendaciones sobre el uso. Hacemos clic en el botón **“Ok, got it”**, y podemos observar la pantalla. Esto puede parecer lento, pero recordemos que **Cypress** es la opción más rápida de testeo para pruebas de punta a punta, por lo que se debe esperar.



Después de algunos minutos, podemos ver el registro que sigue. Finalmente, nuestra prueba ha terminado de correr.

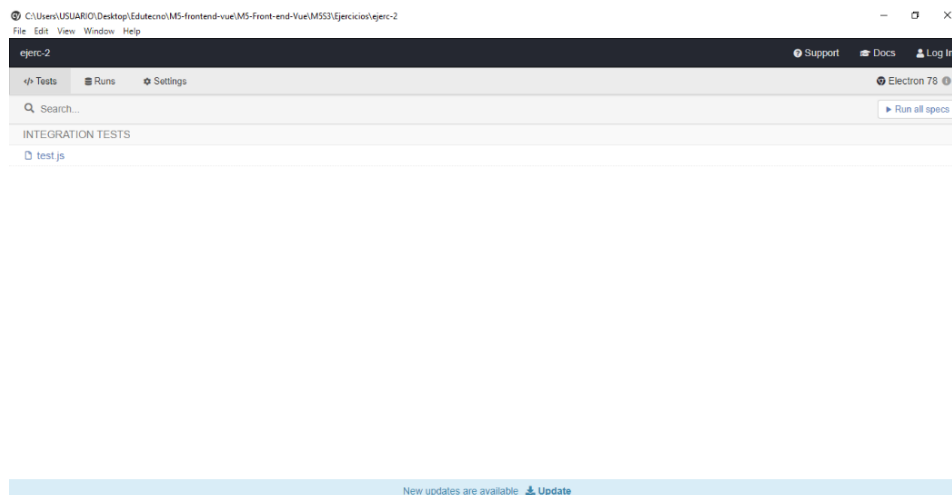
```

PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
1: node

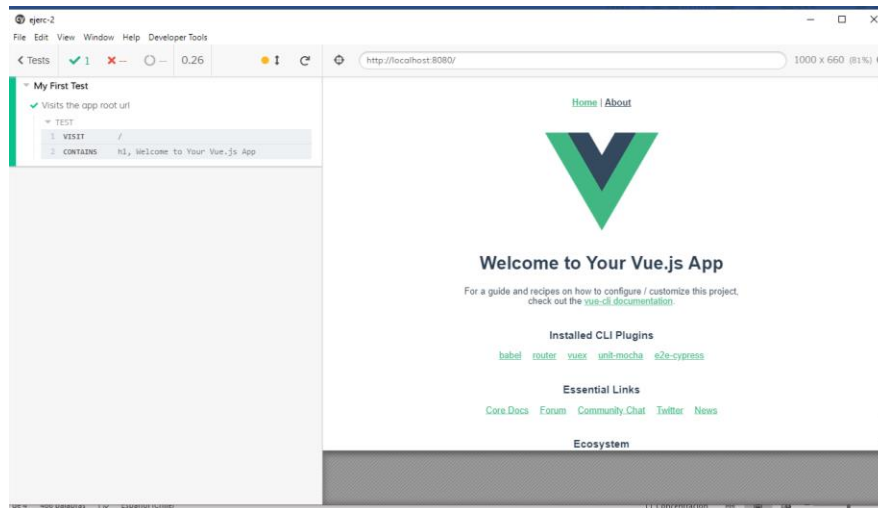
Note this is for preview or E2E testing only.

GET /_ 200 16.523 ms - -
GET /_cypress/runner/cypress_runner.css 200 9.800 ms - -
GET /_cypress/runner/cypress_runner.js 200 414.699 ms - -
GET /_cypress/runner/integration/test.js 200 13.288 ms - 725
GET /_cypress/runner/fonts/fa-solid-900.woff2 200 8.146 ms - 75728
GET /_cypress/tests?p=tests%5Ce2e%5Csupport%5Cindex.js-411 - - ms - -
GET /_cypress/tests?p=tests%5Ce2e%5Cspecs%5Ctest.js-887 - - ms - -
Browserslist: caniuse-lite is outdated. Please run the following command: `npm update`
GET /_ 200 2.180 ms - -
GET /_cypress/runner/cypress_runner.css 200 3.559 ms - -
GET /_cypress/runner/cypress_runner.js 200 1.115 ms - -
GET /_cypress/runner/integration/test.js 200 2.618 ms - 725
GET /_cypress/runner/fonts/fa-solid-900.woff2 200 1.005 ms - 75728
GET /_cypress/tests?p=tests%5Ce2e%5Cspecs%5Ctest.js-145 200 71.648 ms - 736
GET /_cypress/tests?p=tests%5Ce2e%5Csupport%5Cindex.js-045 200 114.326 ms - -
GET /_ 200 120.126 ms - -
GET /css/app.574e90c1.css 200 15.258 ms - -
GET /js/app.8694c59f.js 200 30.907 ms - -
GET /js/chunk-vendors.d3ab0cc2.js 200 30.694 ms - -
GET /img/logo.82b9c7a5.png 200 10.315 ms - -
  
```

Ahora, podemos volver a la interfaz de **Cypress**, y seleccionar la prueba para ver los detalles.



Apenas le damos clic, se despliega una ventana mostrando los resultados. En el costado izquierdo, podemos ver un panel con los detalles de la prueba, y en el lado derecho, la imagen que muestra el navegador.

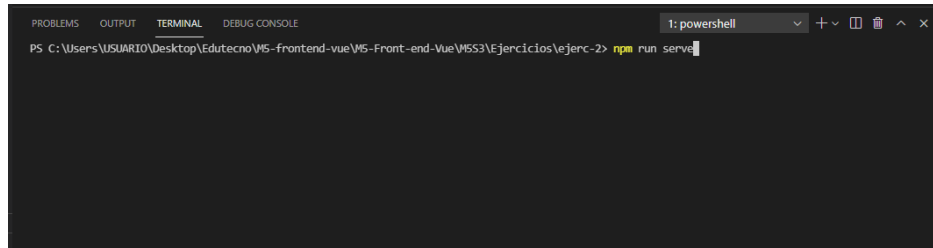


Con eso, terminamos la ejecución de una primera prueba **end to end**, utilizando **Cypress** con **Mocha**.

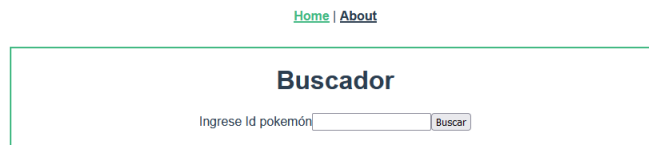
EXERCISE 3: CREANDO UN TEST END TO END

INTRODUCCIÓN

En el siguiente ejercicio, crearemos un test **end to end**. Reciclaremos la base de nuestro código, de uno de los ejercicios realizados en el Drill anterior, el cual hemos dejado disponible en este Exercise. Una vez descargado, es necesario que lo descompriman en la carpeta que quieren destinar para realizar el ejercicio. Con eso realizado, pueden ir a Visual Studio Code, o el editor de texto de su preferencia. Abren la carpeta del ejercicio, la terminal, y escriben el comando: **"npm install"**. Con esto, ya pueden instalar todas las dependencias del proyecto, pues generalmente en la versión comprimida, no incluimos la carpeta **"node modules"**. Una vez que la instalación de dependencias ha finalizado, se levanta el proyecto para confirmar que todo está en orden, escribiendo la terminal: **"npm run serve"** seguido de tecla enter.

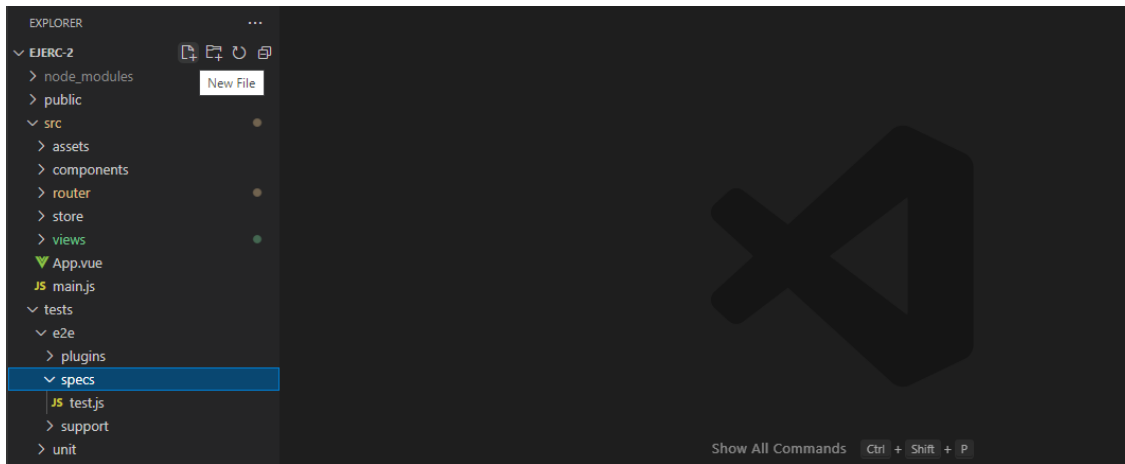


Una vez que termine de compilar, diríjase al navegador.

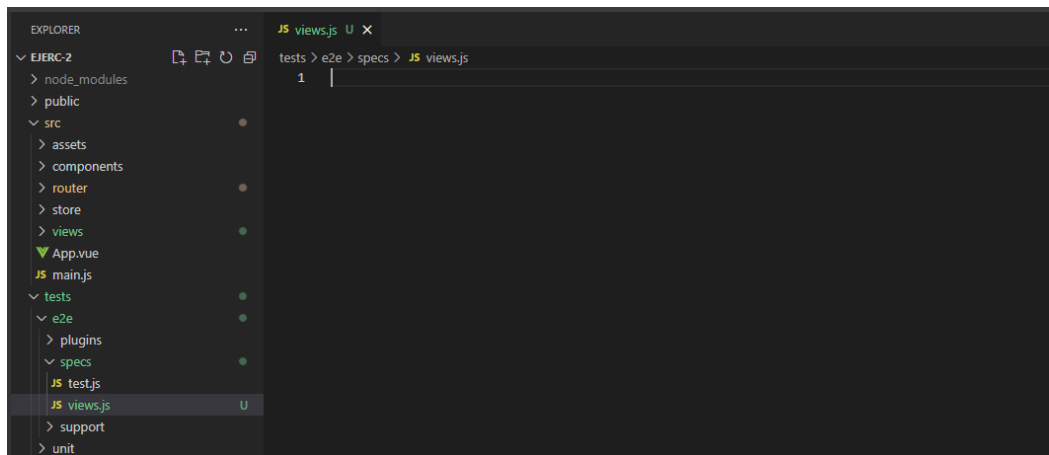


CREANDO LA PRUEBA

Ya teniendo confirmado que funciona bien, comenzamos a trabajar juntos. Nos dirigimos a la carpeta `"tests">"e2e">"specs"`, y creamos un nuevo archivo.



Lo llamaremos “views.js”. Si se puede notar, acá no es necesario agregar el “.spec” en el nombre del documento, como sucedía en **Jest**.



Como pauta para construir nuestra prueba, vamos a usar el código del archivo **“test.js”**, así que nos dirigimos al documento, lo copiamos, y lo pegamos en nuestra prueba **“views.js”**. Ahora podemos comenzar a editar.


```
JS views.js U
tests > e2e > specs > JS views.js > ...
1 describe('My First Test', () => {
2   it('Visits the app root url', () => {
3     cy.visit('/')
4     cy.contains('h1', 'Welcome to Your Vue.js App')
5   })
6 })
```

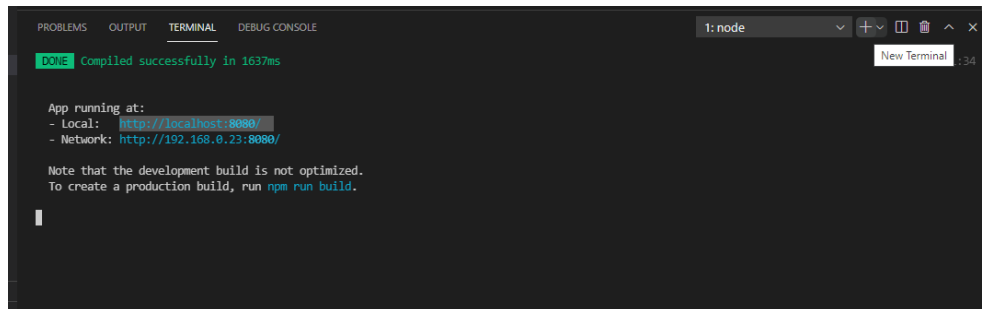
Lo primero que haremos, es cambiar el nombre de nuestra suite de pruebas, a: **'testing the views'**. Conservaremos la línea **"cy.visit('/')"**, que nos permitirá probar si funciona la ruta de acceso a la URL raíz del proyecto.

```
1 describe('Testing the views', () => {
2   it('Visits the app root url', () => {
3     cy.visit('/')
4     cy.contains('h1', 'Welcome to Your Vue.js App')
5   })
6 })
```

Para confirmar que efectivamente llegamos a nuestro "Home", vamos a reciclar también la línea que sigue, y reemplazaremos el texto por **"Buscador"**, quedando de la siguiente forma:

```
1 describe('Testing the views', () => {
2   it('Visits the app root url', () => {
3     cy.visit('/')
4     cy.contains('h1', 'Buscador')
5   })
6 })
```

De inmediato, presionamos guardar, y abrimos una nueva terminal.

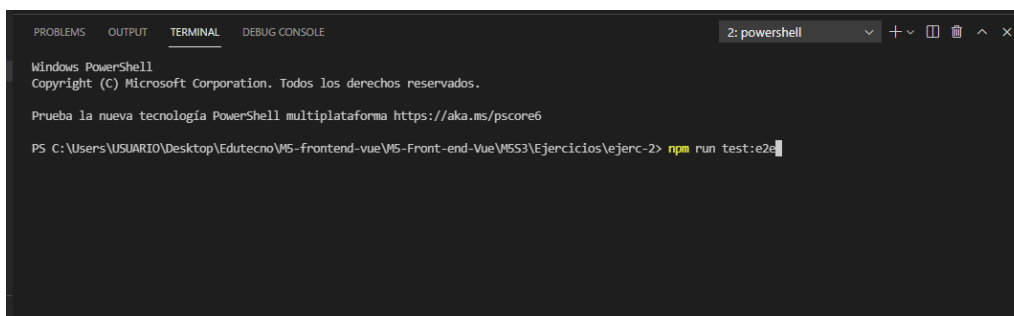


```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
1: node
DONE Compiled successfully in 1637ms
New Terminal 134

App running at:
- Local: http://localhost:8080/
- Network: http://192.168.0.23:8080/

Note that the development build is not optimized.
To create a production build, run npm run build.
```

Escribimos el comando: **“npm run test:e2e”**, y presionamos enter.



```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
2: powershell
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

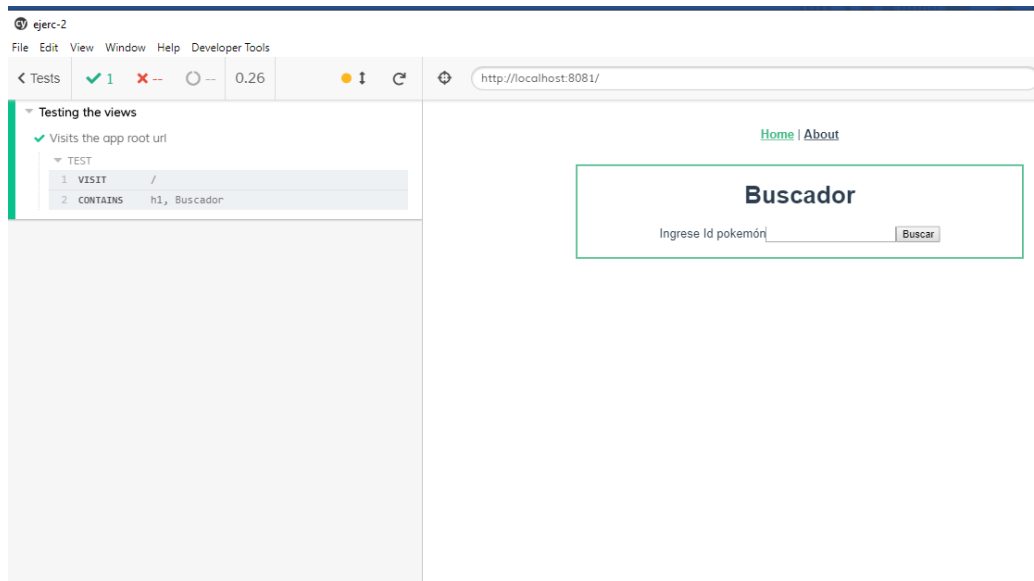
Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pscore6

PS C:\Users\USUARIO\Desktop\Edutecno\M5-frontend-vue\M5-Front-end-Vue\M5S3\Ejercicios\ejerc-2> npm run test:e2e
```

Vamos a la interfaz de **Cypress**. Seleccionamos el test **“view.js”**, y lo pinchamos para ejecutarlo.



Podemos ver la ventana que se abre, y cómo la prueba corre a toda velocidad, marcando todo como correcto.

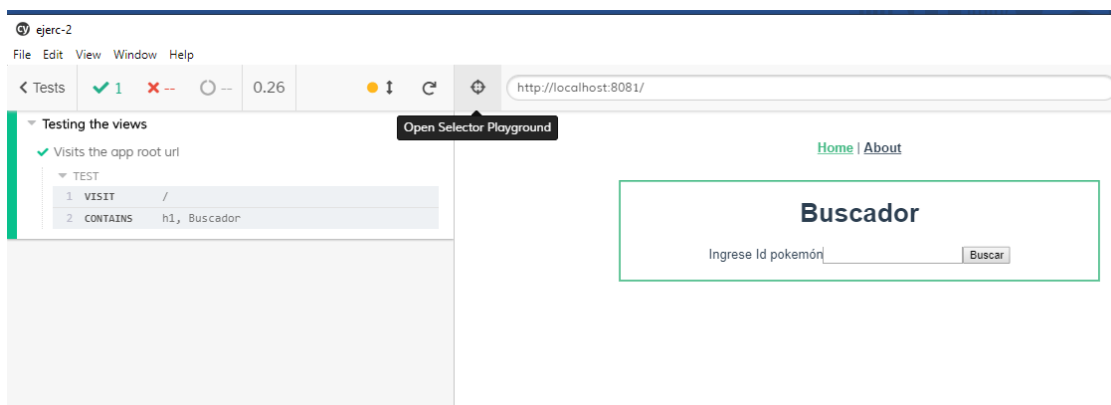


Volvemos a nuestro documento, y creamos una nueva prueba, escribiendo: `"it" ('', () => {})`. Llevará por nombre: `'Renders empty input at start'`, ya que lo que queremos probar es que el input esté vacío al iniciar la página.

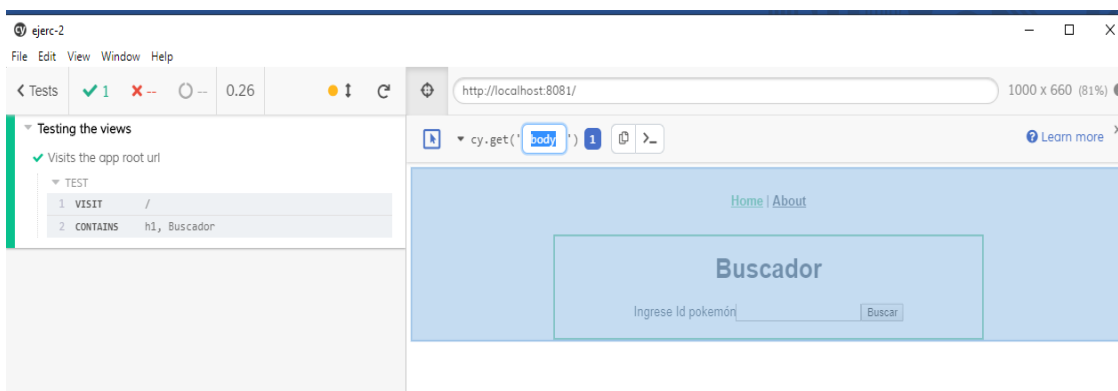
```
1 it('Renders empty input at start', () => {  
2  
3 })
```

USANDO “OPEN SELECTOR PLAYGROUND” DE CYPRESS

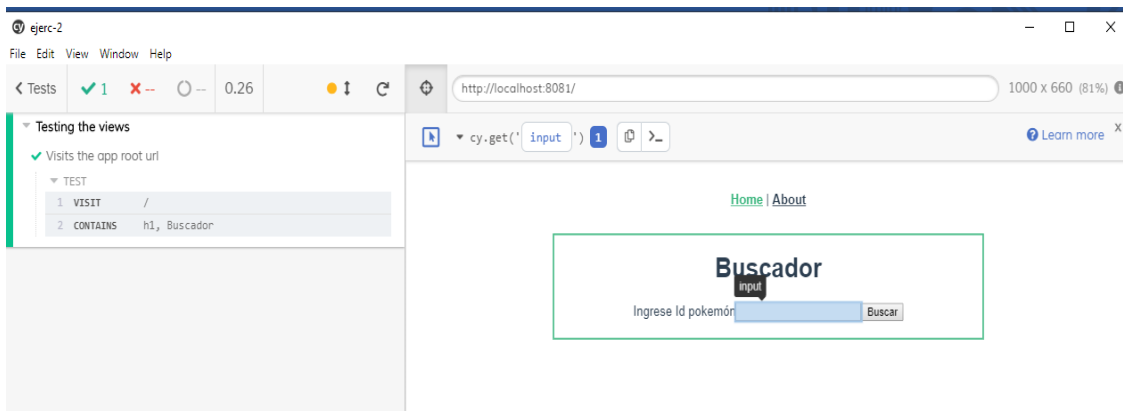
Cypress tiene una funcionalidad que permite seleccionar un elemento desde la interfaz, y nos entrega el código para emplearlo. Para aprender a usarla, volvemos a **Cypress**, y vamos a seleccionar el botón: **“Open selector Playground”**, que está en el costado izquierdo de la barra de direcciones de la interfaz, es el que tiene el símbolo de una mira.



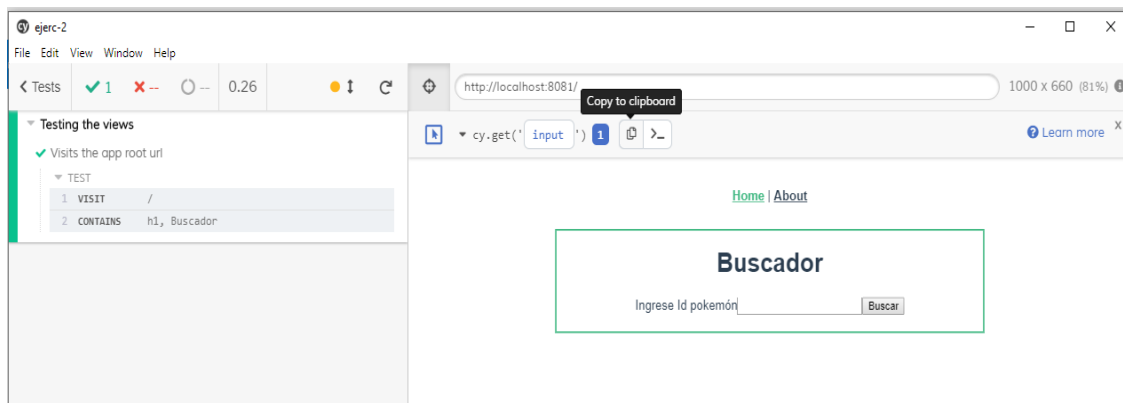
De inmediato, podemos ver cómo se marca todo el contenido de nuestra página “Home”.



Lo siguiente que queremos probar, es que el input esté vacío al iniciar la página. Así que vamos a seleccionarlo.



Una vez marcado, podemos ir arriba, y ver que nos muestra la sintaxis de `cy.get`, asociada a nuestro input. Presionemos el botón copiar.



Vamos a nuestra prueba. Le damos a pegar, y enseguida podemos ver que tenemos el código para seleccionar el input. Esta funcionalidad es muy útil, y la usaremos durante el desarrollo de la prueba.

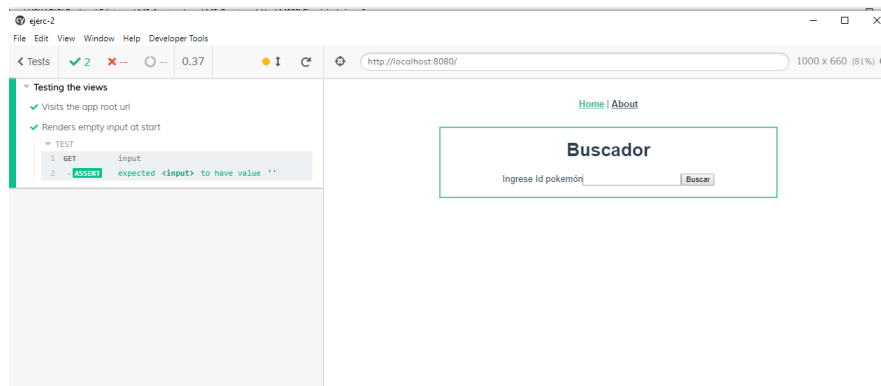
```
1 it('Renders empty input at start', () => {  
2   cy.get('input')  
3  
4 })
```

CONTINUANDO CON LA CONSTRUCCIÓN DE LA PRUEBA

“Get” permite obtener uno o más elementos **DOM**, por selector o alias. Con esa función, ya tenemos seleccionado nuestro elemento. Ahora, necesitamos confirmar que el campo está vacío. Para ello usaremos **“should”**. Esta permite crear una aserción, que intentará pasar una y otra vez hasta que lo logre o se agote. Una vez dentro, escribiremos: **“have.value”**, que nos permitirá encontrar el valor. Esto es porque esperamos que el input esté vacío. El código quedaría así:

```
1 it('Renders empty input at start', () => {  
2   cy.get('input').should('have.value', '')  
3  
4 })
```

De inmediato guardamos. Al ir a la interfaz de **Cypress**, vemos que la prueba corre automáticamente, y que la ha pasado sin problemas.



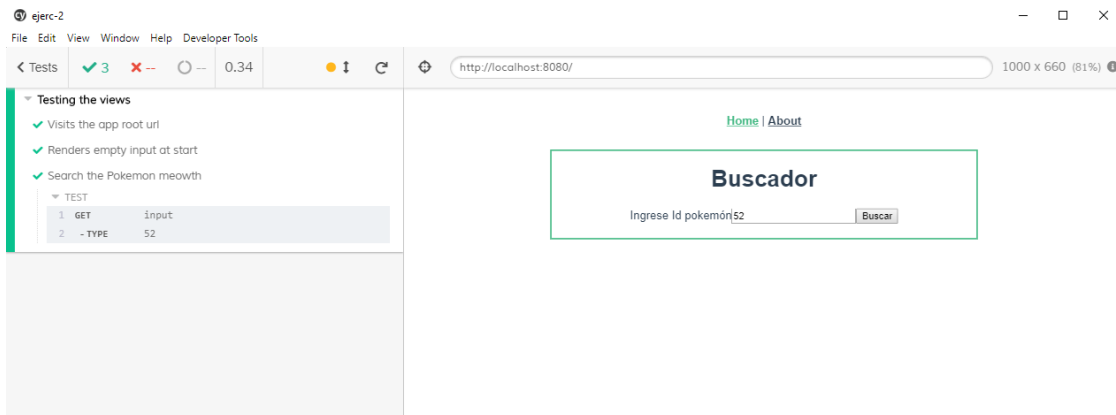
Lo siguiente que tenemos que probar, es si funciona la conexión con la **API**, y podemos obtener, a través del ID, la imagen y el nombre del Pokemon deseado. Así comenzaremos por escribir un nuevo **"it('', () => {})"**, y le daremos de nombre a la prueba: **'Search the Pokemon meowth'**, pues la indicación es que busque al Pokemon Meowth.

```
1 it('Search the Pokemon meowth', () => {
2   })
3 }
```

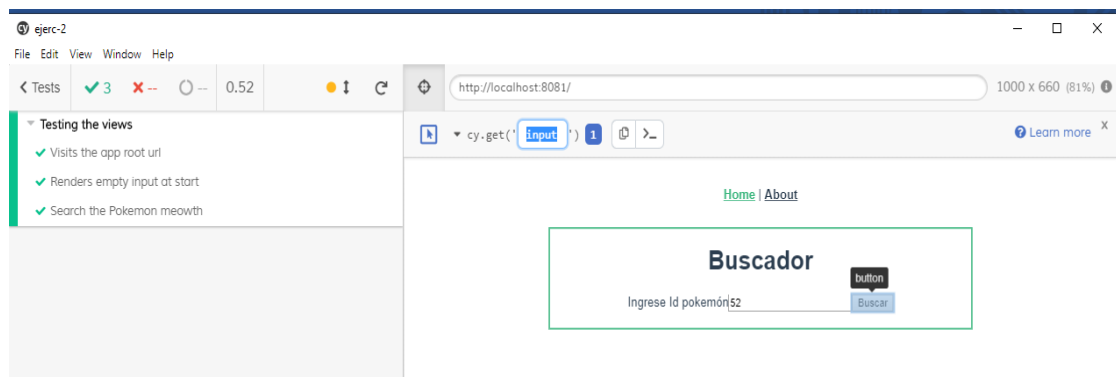
Adentro, comenzaremos a escribir todo lo necesario para nuestra prueba. En primer lugar, agregaremos el número de **ID** de Meowth. Así que copiamos el **cy.get('input')** de arriba, y lo pegamos. De inmediato, incluiremos la función **".type"**, que permite escribir texto dentro de un elemento **DOM**. Y le agregaremos el número "52".

```
1 it('Search the Pokemon meowth', () => {
2   cy.get('input').type('52')
3   })
```

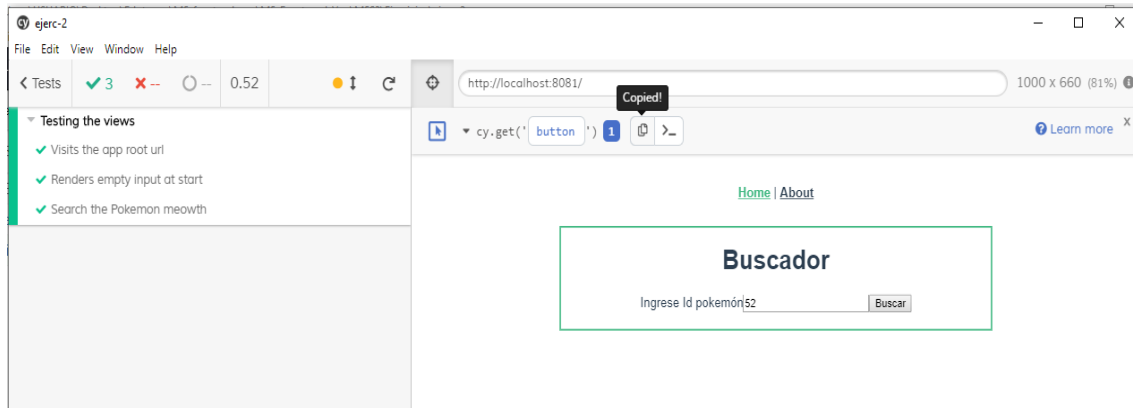
Podemos guardar, y verificar la corrida de la prueba en la interfaz de Cypress. Vemos que funciona, así que continuamos con el siguiente paso.



Como ya logramos escribir el número de **ID**, lo siguiente que tenemos que hacer es clic sobre el botón. Así que presionamos el botón **“Open selector Playground”**, y seleccionamos el botón de búsqueda.



Copiamos el código que nos disponibiliza más arriba.



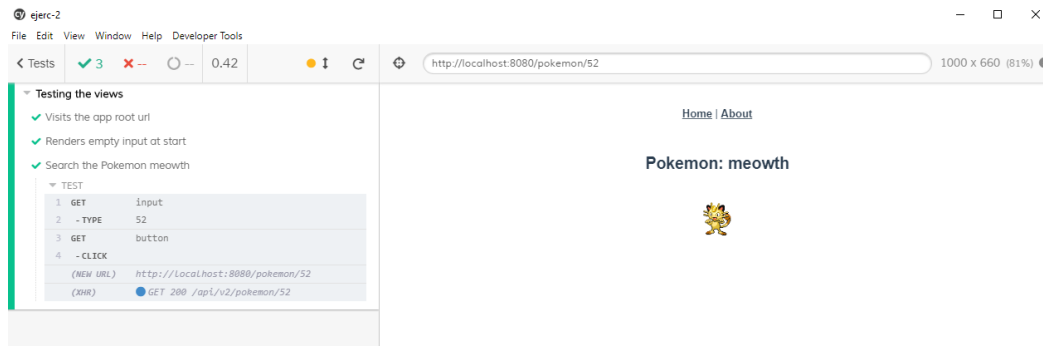
Volvemos a nuestra prueba, y lo pegamos.

```
1 it('Search the Pokemon meowth', () => {  
2   cy.get('input').type('52')  
3   cy.get('button')  
4  
5 })
```

Como ya logramos encontrar nuestro botón, nos falta agregar un método que sea capaz de captar el clic que gatilla la llamada a la **API**. Para eso, usaremos el método **“click()”**, que cliquea un elemento **DOM**.

```
1 it('Search the Pokemon meowth', () => {  
2   cy.get('input').type('52')  
3   cy.get('button').click()  
4  
5 })
```

Una vez más guardamos, y vamos a la interfaz de **Cypress** a revisar. Vemos que funciona, así que solo para confirmar que efectivamente está cargando el Pokemon correcto, crearemos una última aseveración.



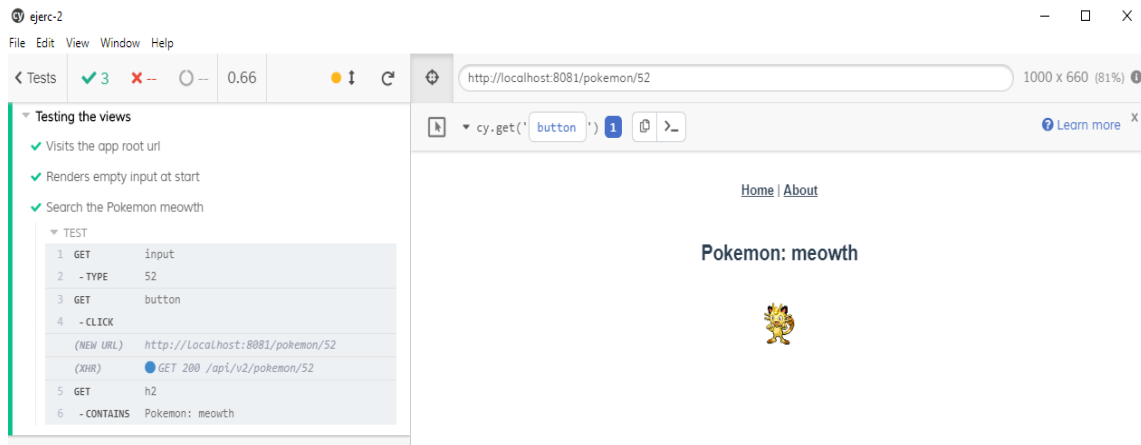
Presionamos el botón **“Open selector Playground”** de **Cypress**, y vamos a seleccionar el elemento que muestra el nombre del Pokemon. Lo copiamos, y lo pegamos en nuestra prueba.

```
1 it('Search the Pokemon meowth', () => {
2   cy.get('input').type('52')
3   cy.get('button').click()
4   cy.get('h2')
5
6 })
```

Ahora, nos falta comprobar el contenido. Para ello, usaremos **“.contains”**, que nos permite obtener el texto que contiene el elemento **DOM**. Los elementos DOM, pueden contener más texto del deseado, y aun así coincidir. Escribimos: **“.contains('Pokemon: meowth')"**.

```
1 it('Search the Pokemon meowth', () => {
2   cy.get('input').type('52')
3   cy.get('button').click()
4   cy.get('h2').contains('Pokemon: meowth')
5 })
```

Guardamos, vamos a **Cypress**, revisamos la corrida de la prueba y vemos que ha pasado. Con esto, ya podemos dar por terminada esa parte del test.



Todavía nos falta probar una última parte, que es confirmar si el menú de navegación nos lleva a la página "About". Vamos a escribir un nuevo **`it('', () => {})`**, y le pondremos por nombre: **`"Visits the about page"`**.

```
1 it('Visits the about page', () => {
2
3 })
```

Vamos al botón **`"Open selector Playground"`**, para poder seleccionar el enlace que nos debe llevar a la vista "About". Lo pinchamos, seleccionamos el enlace, y lo copiamos en nuestra prueba. Terminamos la prueba con **`"click()"`**.

```
1 it('Visits the about page', () => {
2   cy.get('[href="/about"]').click()
3 })
```

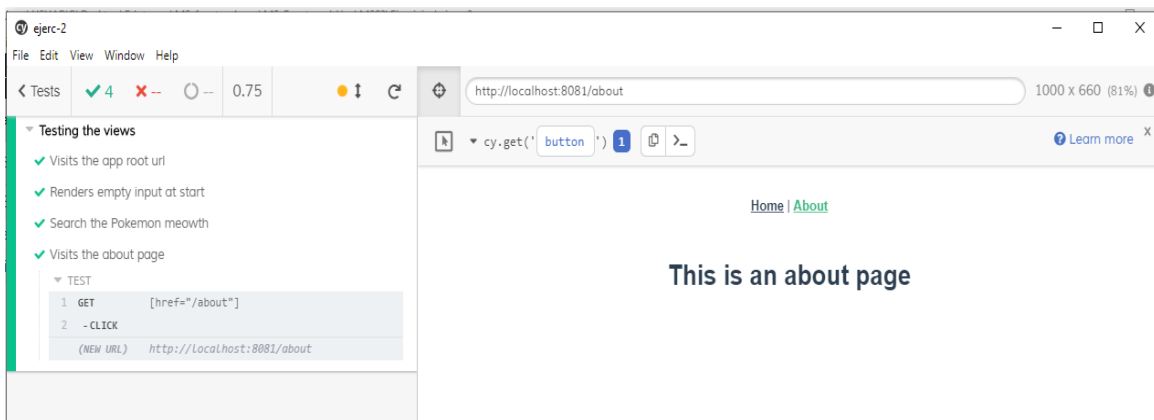


Guardamos, y vamos a **Cypress** a verificar el resultado de la prueba. Vemos que funciona bien, así que revisamos con una última confirmación.

Presionamos el botón **“Open selector Playground”**, y seleccionamos el elemento del título de la página. Copiamos el código, y lo pegamos en nuestra prueba. Complementamos con **“.contains()”**, y adentro le entregamos el texto contenido.

```
1 it('Visits the about page', () => {  
2   cy.get('[href="/about"]').click()  
3   cy.get('h1').contains('This is an about page')  
4 })
```

Guardamos, y vamos a **Cypress**. Nuevamente vemos que funciona.



Con esto, ya hemos terminado la creación de una prueba **end to end**.