Proceso QA – Automatización de pruebas Web

1. **Stack:**

- Framework de automatización: Cypress
- Lenguaje de programación: JavaScript
- Lenguaje para escritura de casos de prueba: Gherkin
- Lectura y ejecución de casos de prueba: Cucumber
- IDE (Recomendado): Aqua – Jetbrains
- Patrón de diseño: Page Object Model (POM)

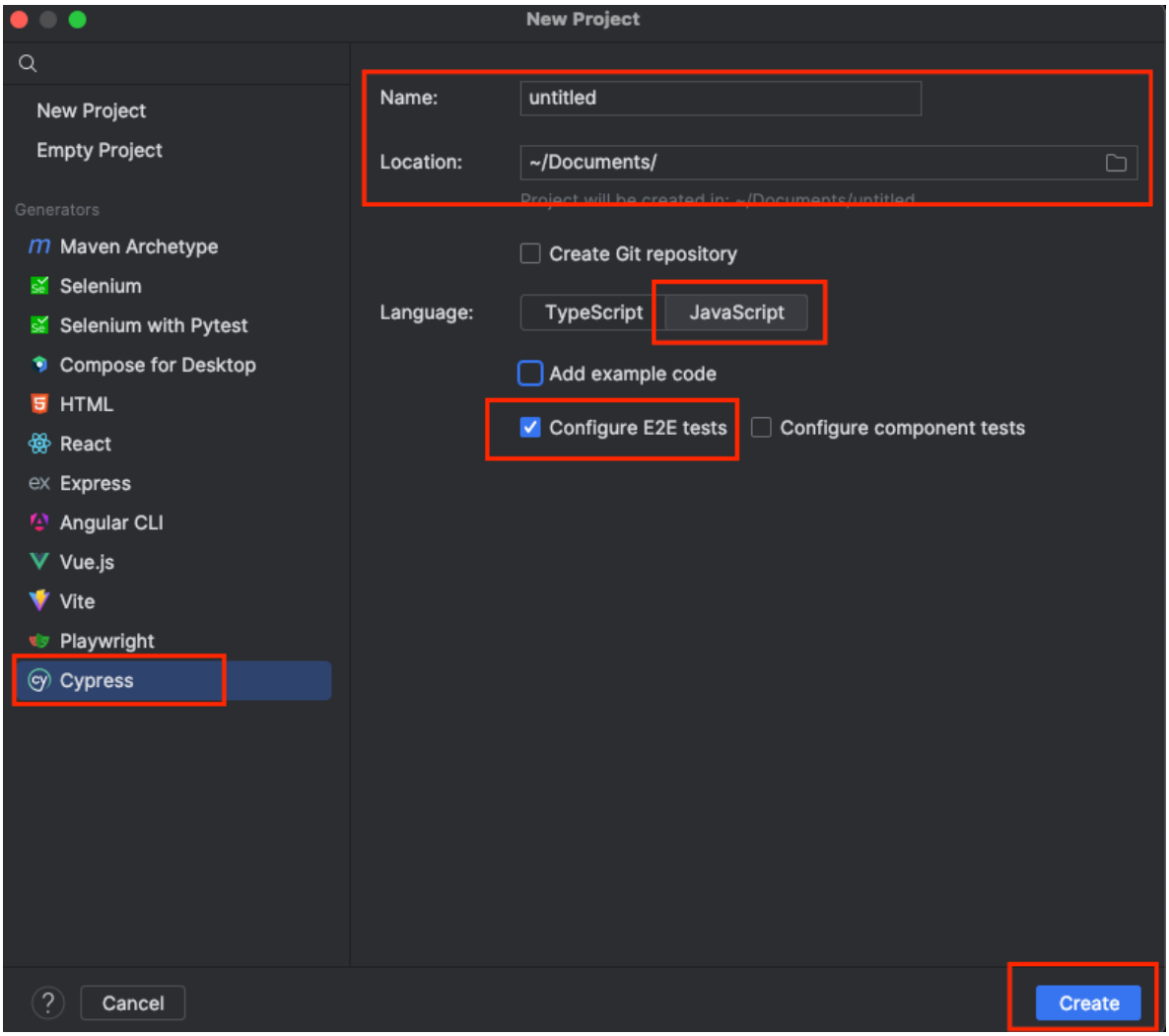
2. **Descargar herramientas:**

- Instalar node en el siguiente enlace: https://nodejs.org/en/download
- Se recomienda descargar la caja de herramientas de Jetbrains en el siguiente enlace: https://www.jetbrains.com/toolbox-app/
- Tener la instalación de toolbox permite obtener de forma rápida las ultimas actualizaciones y toda la suite de productos de Jetbrains
- Si solo desea descargar el IDE Aqua ir al siguiente enlace: https://www.jetbrains.com/aqua/

Nota: El IDE recomendado permite crear proyectos Cypress con la plantilla por defecto, esto con el fin de eliminar algunos pasos de estructuración del proyecto, sin embargo, cualquier IDE con soporte para lenguaje JavaScript se puede utilizar si es de su preferencia, teniendo en cuenta la necesidad de crear la estructura de forma manual.

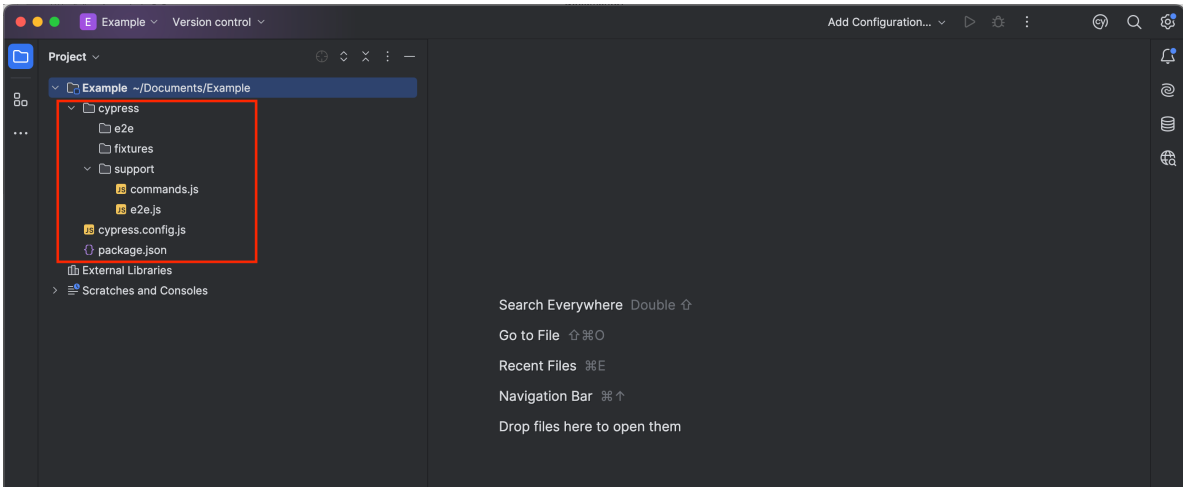
3. **Inicializar Proyecto:**
   (Para toda esta documentación se utilizará como referencia el IDE - Aqua)

- Abrir el IDE
- En la barra de herramientas ir a File → New → Project
- Seleccionar Generador → **Cypress**
- Seleccionar nuevo proyecto
- Asignar nombre del proyecto
- Asignar ubicación local
- Seleccionar lenguaje → JavaScript
- Marcar el check de **Configure E2E tests**
- Pulsar en el botón **Create**



4. Estructura del proyecto

Al realizar los pasos anteriores se creará un nuevo proyecto con la siguiente estructura:



Dentro del directorio **cypress** se incluyen las siguientes carpetas:

- **e2e**: En esta ubicación se guardan los test y su implementación, dentro de esta carpeta crear las siguientes subcarpetas
    - **features:** Allí se crearan todos los casos de prueba con la extensión **.feature** que indica que será utilizado el lenguaje gherkin y cucumber para interpretar los tests.
    - **steps-definitions**: En esta carpeta se crearan los archivos con extensión **.js** que contienen la implementación requerida para los pasos definidos en los casos de prueba.
- **fixtures**: En esta ubicación se pueden crear y guardar los archivos que contienen los datos de prueba y/o recursos necesarios para la ejecución de las pruebas.
- **support**: Este directorio contiene 2 archivos que son indispensables dentro de la estructura del proyecto, el primero llamado **commands.js** que contiene los comandos que se crean y se utilizan dentro del proyecto y el segundo llamado **e2e.js** que contiene la importación requerida por el archivo **commands.js** para que las funciones que allí se definan pueden ser llamadas dentro de cualquier implementación dentro del proyecto.

En la raíz del proyecto se crea el archivo **cypress.config.js** el cual contiene toda la configuración requerida en el proyecto y es indispensable para que se pueda ejecutar los features.

Finalmente en la raíz del proyecto se crea el archivo **package.json** que contiene las dependencias y la configuración general del proyecto, por defecto se incluye la dependencia de Cypress requerida así:

```
{
  "name": "Example",
  "version": "1.0.0",
  "devDependencies": {
    "cypress": "*"
  }
}
```

Nota: Al asignarle el valor ***** a la dependencia de cypress se va a utilizar por defecto la última versión disponible, si se requiere utilizar una versión especifica se debe asignar el valor asi:

```
{
  "name": "Example",
  "version": "1.0.0",
  "devDependencies": {
    "cypress": "^13.6.2"
  }
}
```

5. **Configuración del proyecto**
    - Agregar dependencias de Cucumber

- Cypress por defecto utiliza una sintaxis específica para la escritura de los casos de prueba, sin embargo, en este proyecto se utiliza la sintaxis definida por el lenguaje gherkin, por lo que se debe configurar el proyecto para que se adecue a la interpretación de casos de prueba escritos en lenguaje gherkin utilizando las capacidades de cucumber, para esto se debe agregar las siguientes dependencias dentro del archivo **package.json**:

```
"devDependencies": {
  "@badeball/cypress-cucumber-preprocessor": "^19.1.1",
  "@bahmutov/cypress-esbuild-preprocessor": "^2.2.0",
  "esbuild": "^0.19.7",
  "playwright-webkit": "^1.40.0"
}
```

- Agregar configuración de pruebas en Cucumber:
  - En la raíz del proyecto crear un archivo con el nombre **.cypress-cucumber-preprocessorrc.json**
  - Dentro de este archivo se implementan algunas configuraciones del proyecto con respecto a la dependencia de cucumber pero específicamente se debe indicar la ruta donde se encuentra la definición de los pasos de prueba, en este caso se debe indicar de acuerdo a la carpeta que previamente hemos creado dentro de la estructura del proyecto en el directorio **cypress/e2e**, así:

```
{
  "dryRun": true,
  "stepDefinitions": [
    "[filepath]/**/*.{js,ts}",
    "[filepath].{js,ts}",
    "cypress/e2e/steps/*.{js,ts}"
  ]
}
```

  - El atributo **dryRun** indica si se permite la ejecución del proyecto sin la definición de los pasos de prueba, en el caso de tener el valor **true** entonces creará una definición de ejemplo como salida en la terminal.

- Configurar el proyecto para el uso de las dependencias de cucumber y su configuración
  - En el archivo **cypress.config.js** importar las siguientes dependencias

```
const createBundler = require("@bahmutov/cypress-esbuild-
preprocessor");
const preprocessor = require("@badeball/cypress-cucumber-
preprocessor");
const createEsbuildPlugin = require("@badeball/cypress-cucumber-
preprocessor/esbuild");
```

- A continuación, agregar la siguiente función que se encarga de utilizar las capacidades de las dependencias de Cucumber para ser ejecutadas dentro de Cypress:

```
async function setupNodeEvents(on, config) {
    await preprocessor.addCucumberPreprocessorPlugin(on,
config);

    on(
        "file:preprocessor",
        createBundler({
            plugins: [createEsbuildPlugin.default(config)],
        })
    );

    return config;
}
```

- Incluir dentro del módulo de exportación la configuración definida del proyecto según la función **setupNodeEvents**, así:

```
module.exports = defineConfig({
    e2e: {
        setupNodeEvents
    }
});
```

- Incluir en la configuración el atributo **specPattern** que contiene el valor de la ruta donde se encuentras los archivos de prueba **.features** para ejecución:

```
module.exports = defineConfig({
    e2e: {
        setupNodeEvents,
        specPattern: [
            "cypress/e2e/features/*.feature",
        ],
    },
});
```

Nota: El atributo **specPattern** puede ser una lista o un valor definido, la lista es útil si se tiene varios casos de prueba en la misma ruta y se desea ejecutar unicamente algunos casos específicos, por ejemplo así:

```
module.exports = defineConfig({
    e2e: {
        setupNodeEvents,
        specPattern: [
            "cypress/e2e/features/signup.feature",
            "cypress/e2e/features/login.feature"
        ],
    },
});
```

- Aplicar cambios y configuraciones
  - Ejecutar en la terminal del IDE el comando **npm install,** esto instalará todas las dependencias definidas en el proyecto y de esta forma estará listo para iniciar la definición de casos de prueba y su implementación.

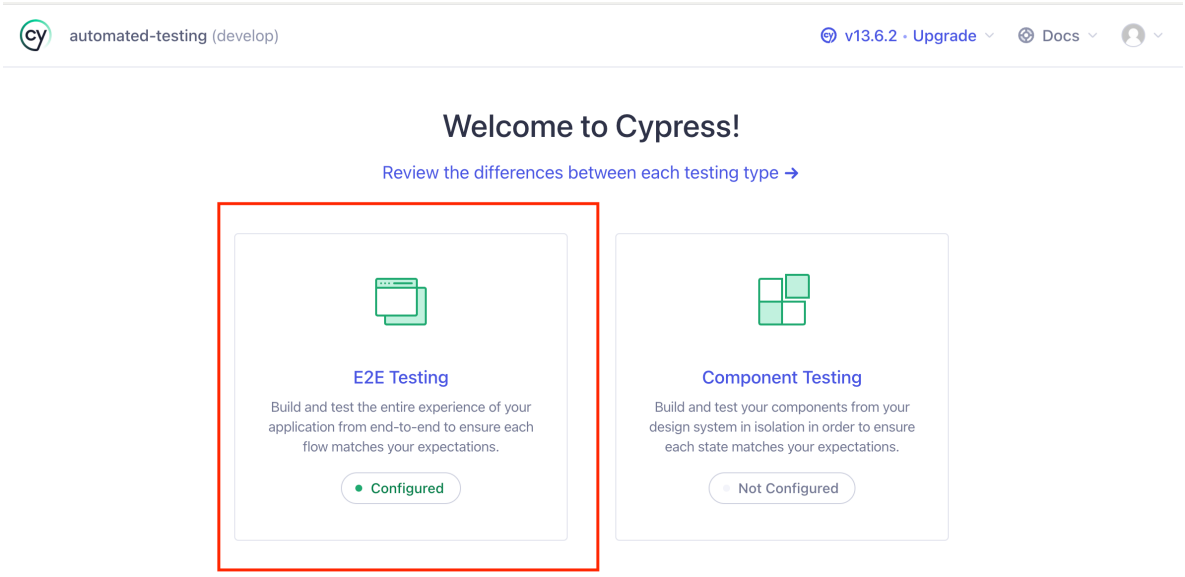
6. Añadir configuraciones personalizadas

- Comandos personalizados para ejecución:
  - Dentro del archivo **package.json** se puede incluir el atributo **scripts** el cual contiene los comandos que se quieran definir para la ejecución del proyecto, como en el siguiente ejemplo:

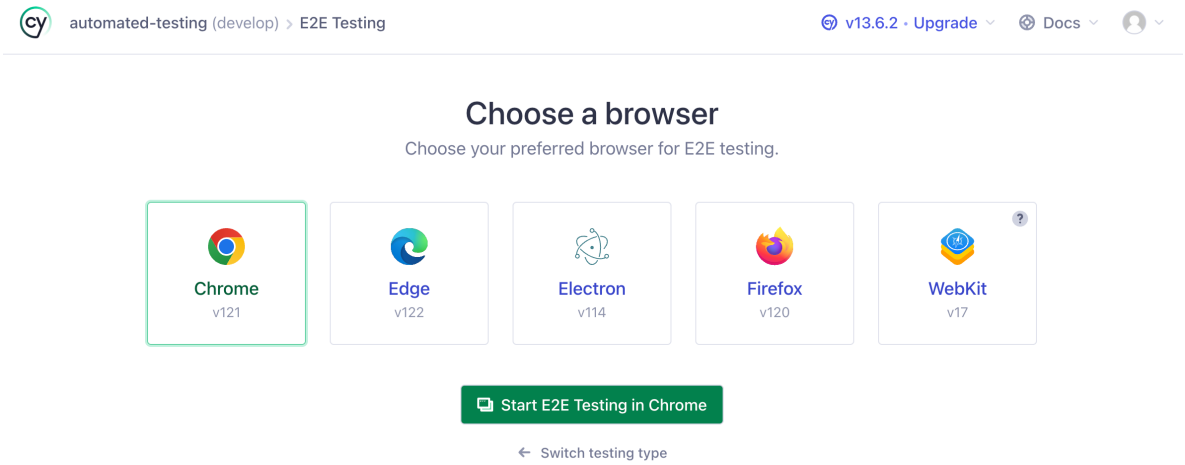
```
"scripts": {
  "cypress:runner": "cypress open --e2e -b chrome",
  "cypress:execution": "cypress run"
}
```

En este caso el comando **cypress:runner** ejecuta las siguientes funciones:

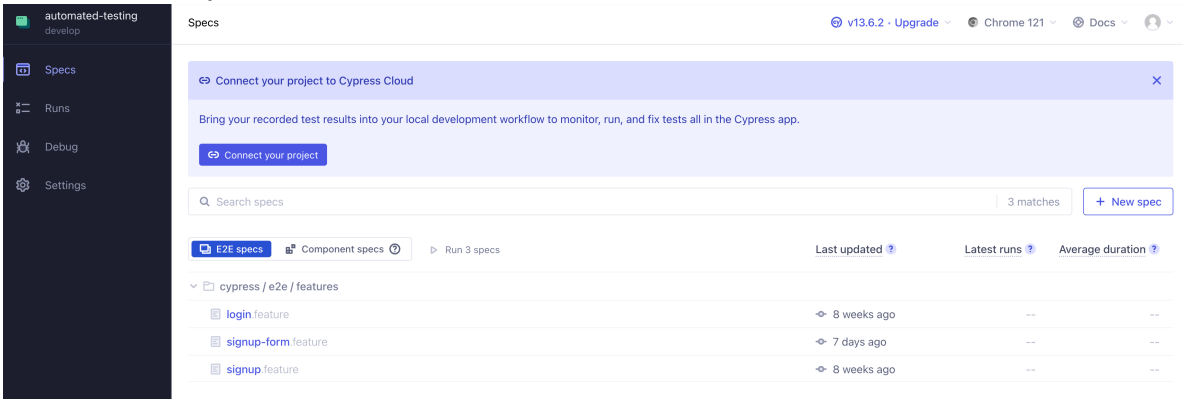
  - **cypress open:** Se encarga de abrir la interfaz en el navegador de cypress para la ejecución personalizada del proyecto, dentro de esta interfaz se puede seleccionar el tipo de pruebas a ejecutar si son test **E2E** o **Component Test**



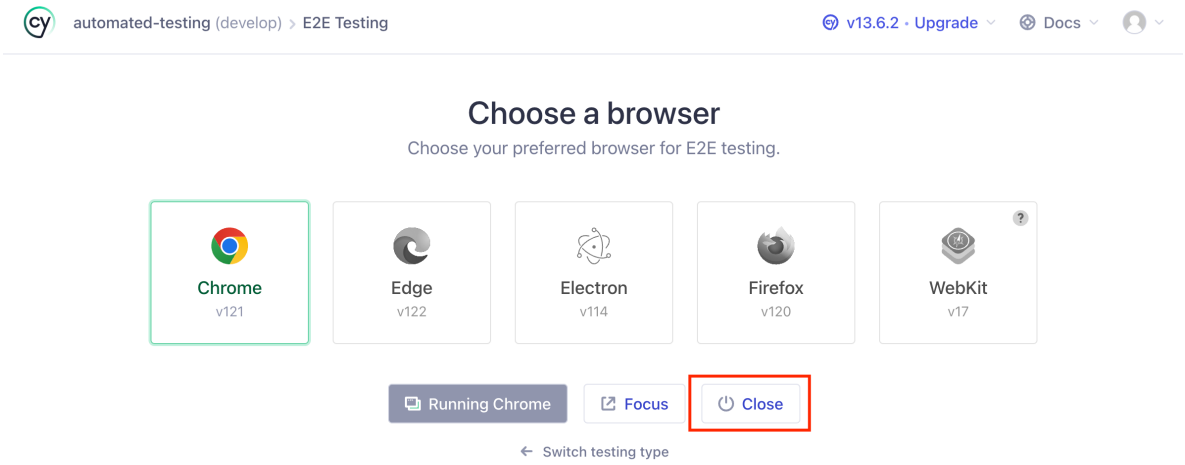
También se selecciona el navegador en el cual se ejecuta el proyecto



Y finalmente los casos de prueba a ejecutar que se encuentran en la ruta definida en el archivo **cypress.config.js** en el atributo **specPattern**, al dar clic sobre cualquiera de los test que se muestran se inicia la ejecución.



Para finalizar la ejecución delos casos de prueba se puede ingresar a la ventana de **Cypress** y dar clic en el botón **close**



Si se requiere finalizar la ejecución de cypress, en la terminal del IDE ejecutar el comando **Ctrl+C** y confirmar el fin de ejecución.

- **--e2e**: incluir esta función dentro del comando personalizado permite indicar que el tipo de test a ejecutar será **e2e**, de esta forma no será necesario seleccionar manualmente en la interfaz de cypress
- **-b:** Incluir esta función dentro del comando personalizado permite indicar qué navegador será utilizado para la ejecución de las pruebas asi:
  - **-b chrome:** Para seleccionar navegador Chrome
  - **-b firefox:** Para seleccionar navegador Chrome
  - **-b edge:** Para seleccionar navegador Chrome
  - **-b electron:** Para seleccionar navegador Chrome

- **-b webkit:** Para seleccionar navegador Safari – Se debe verificar que en el archivo **package.json** se encuentre la dependencia "playwright-webkit" así:

```
"devDependencies": {
  "playwright-webkit": "^1.40.0"
}
```

Y en el archivo **cypress.config.js** dentro del módulo de exportación agregar el siguiente atributo

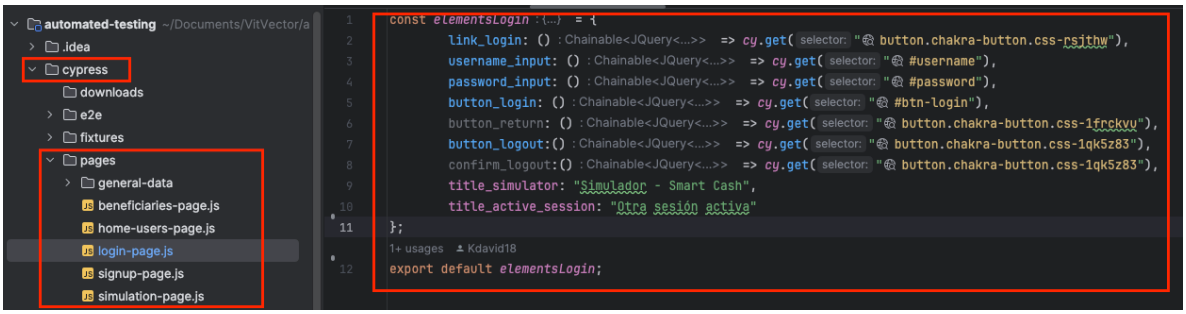
```
module.exports = defineConfig({
    e2e: {
        setupNodeEvents,
        specPattern: [
            "cypress/e2e/features/*.feature",
        ],
        experimentalWebKitSupport: true,
    },
});
```

- **cypress run**: Dentro de otro comando personalizado se puede incluir la función cypress run que se encarga de ejecutar el navegador en modo headless, esto significa que ni la interfaz de cypress ni el navegador se abrirán durante la ejecución de la prueba, el proyecto se ejecuta en back y al final se obtiene como salida en la terminal el resultado de la ejecución. Esta función también puede ser utilizada con las funciones complementarias antes descritas para indicar el navegador o los test a ejecutar.

7. Patrón de diseño

Para este proyecto se ha definido como patrón de diseño **Page Object Model**, esto indica que la estructura del proyecto va ir enfocada a la definición de páginas que contiene la web objeto de pruebas y los elementos que conforman cada una de ellas.

Para adecuar las características del proyecto a este patrón se debe crear una carpeta dentro del directorio **cypress** que puede tener como nombre **pages** y que contiene las páginas y sus elementos como en el siguiente ejemplo:



De forma opcional se puede incluir dentro del directorio **cypress/e2e** una nueva carpeta que contenga las definiciones de los pasos de prueba que a su vez serán ejecutados por la implementación contenida en el directorio **cypress/e2e/steps.**

Es válido que estas definiciones se incluyan dentro de la misma estructura de los archivos **pages**, ya depende del proyecto si se considera tener de forma independiente las definiciones y los elementos de cada página principalmente con el fin de aplicar buenas prácticas de programación como los principios SOLID.

8. Reportes de resultados

En conjunto con cypress se puede utilizar la librería de reportes especializada para Cucumber esto con el fin de obtener el detalle de la ejecución, los test aprobados, los fallidos y las evidencias de ejecución.

- Para esto se debe agregar dentro del archivo **package.json** la dependencia **cucumber-html-reporter**, asi:

```
"devDependencies": {
  "@badeball/cypress-cucumber-preprocessor": "^19.1.1",
  "@bahmutov/cypress-esbuild-preprocessor": "^2.2.0",
  "esbuild": "^0.19.7",
  "playwright-webkit": "^1.40.0",
  "multiple-cucumber-html-reporter": "^3.0.1",

}
```

- Crear un archivo dentro de la raíz del proyecto con el nombre **cucumber-html-report.js** y en este archivo definir los parámetros generales y personalizados que contiene el reporte, por ejemplo así:

```
const report = require("multiple-cucumber-html-reporter");

report.generate({
    jsonDir: "jsonreport",
    reportPath: "./reports/cucumber-htmlreport.html",
    metadata: {
        browser: {
            name: "chrome",
            version: "XX"
        },
        device: "Local test",
        platform: {
            name: "macOS",
            version: "Sonoma 14.0"
        },
    },
    customData: {
        title: 'Run Info',
        data: [
            {label: 'Project', value: 'BuddyVector'},
            {label: 'Datetime', value: new Date().toLocaleString()}
        ]
    }
})
```

9. Ejecución del proyecto

Para ejecutar el proyecto como se indica en el punto 4 una vez definidos los comandos personalizados en la terminal del IDE se ejecutan de la siguiente forma:

- **npm run + {{comando personalizado}}:** Utiliza los comandos definidos en el archivo **package.json** y las funciones asociadas a cada uno para ejecutar el proyecto.
- **npm run cypress open:** Abre directamente la interfaz de cypress para configurar la ejecución sin utilizar ningún comando personalizado
- **npm run cypress run:** No abre la interfaz de cypress y ejecuta el proyecto en modo headless
- **node cucumber-html-report.js:** Ejecuta el archive que genera los reportes de cucumber según la configuración definida.