

## Prompt 7: Brittle selectors

Smell: **Brittle selectors**

### End to End test code without smell

```
// loginTest.cy.js

```js
//File : loginTest.cy.js

describe('Test', () => {

  const expectedText = {
    textMenu: 'administrator'
  };

  it('Login Test', { tags: '@smoke' } , () => {

    const username = Cypress.config('username');
    const password = Cypress.config('password');

    cy.visit(Cypress.config('baseUrl') + Cypress.env('loginUrl'), {
      failOnStatusCode: false });

    cy.get('[id="username"]').type(username);

    cy.get('input[type="submit"]').click();

    cy.get('[id="password"]').type(password);

    cy.get('input[type="submit"]').click();

    cy.get('.user-info').invoke('text').should('include',
      expectedText.textMenu);
  });
})
```
```

### End to End test code with smell

```
// brittleSelectors.cy.js

```js
//File : loginTest.cy.js

describe('Test', () => {

  const expectedText = {
```

```

    textMenu: 'administrator'
  };

  it('Login Test', { tags: '@smoke' }, () => {

    const username = Cypress.config('username');
    const password = Cypress.config('password');

    cy.visit(Cypress.config('baseUrl') + Cypress.env('loginUrl'), {
      failOnStatusCode: false });

    cy.get('[id="username"]').type(username);

    cy.get('.width-40').click();

    cy.get('[id="password"]').type(password);

    cy.get('.width-40').click();

    cy.get('.user-info').invoke('text').should('include',
      expectedText.textMenu);
  });
})

```

## Full prompt submitted to ChatGPT

Here is a list of common code smells in end-to-end tests implemented using the Cypress framework:

- \* Incorrect use of asserts: using asserts (e.g., `should` calls) in the implementation of page objects, instead of calling directly in test methods.
- \* Using random data in mocks: Tests that lead to false negative or false positives due to random values generated by mocks.
- \* Null function calls: Null function calls: A test that fails due to a null value returned by chaining of array elements. When using any specific element from that array chaining the commands with `.eq()`, `.first()`, `.last()`, etc. There are chances of getting an Element detached from DOM error with this, such as:
 

```

      ``js
      cy.get('selector').eq(1).click()
      ``
      
```
- \* Using “force: true” when interacting with elements: Occurs when the `force:true` command is used when interacting with page elements.
- \* Using `cy.contains()` without selectors: A test that uses `cy.contains()` to select an element based on the text it contains, e.g., `cy.contains("please click here")`.

\* Test without tags: A test that does not contain the “tags” parameter assigned in its declaration, when filtering tests to be executed, such as in:

```
```\js
  it('Description') , () => {
    ....
  }
```\
```

\* Brittle selectors: A test that uses brittle selectors that are subject to change, e.g., `cy.get(["customized id"])` instead of `cy.get(["random id"])`.

\* Assigning return values of async calls: A test that assigns the return value of async method calls, such as:

```
```\js
  let x = cy.get('selector');
  x.click();
```\
```

\* Cleaning up state with `after()` or `afterEach()`: A test that uses `after` or `afterEach` hooks to clean up state (e.g., to release resources acquired by the test).

\* Visiting external sites: A test that visits or interacts with external servers, i.e., sites that we do not control.

\* Starting Web servers: A test that starts a web server using `cy.exec()` or `cy.task()`.

\* Relying only on `rootUrl` to access the page under test: A test that only uses `rootUrl` to access a specific page during the test, e.g., assuming that `xyz.com` redirects to `xyz.com/login`, which is the page that we need to test.

\* Missing global `BaseUrl`: A test suite that calls `cy.visit()` without setting a `baseUrl`. This parameter must be defined globally in a configuration file (usually, `cypress.config.js`).

\* Using unnecessary checks for previously actionable elements : A test that uses redundant visibility checks when dealing with actionable elements such as `should("be.visible")` or `should("exist")` before actions (click or type), such as in:

```
```\js
  cy.get('selector').should('be.visible').click();
```\
```

\* Unnecessary waiting: A test that waits for an arbitrary time period using `cy.wait()`.

\* Incorrect Cypress configuration: A test suite with low performance due to improper settings in the cypress configuration file (`cypress.json`), such as `port`, `timeout` or `video`.

\* Duplicated code: When two or more tests share identical code snippets.

\* Long methods: Tests that check several behaviors and have a large number of lines of

code.

\* Large classes: Classes with a very high number of end-to-end tests.

\* Conditional overuse: Occurs when there is a condition in the test, i.e., due to a specific event the test may return different values.

Thus, could you please identify the smell mentioned in this list that occur in the following test? The test has 1 source code file: loginTest.cy.js . Each file is delimited using markdown code blocks. A comment in the first line of each file describes its name. Please, in your answer mention at most one smell.

```
```js
//File : loginTest.cy.js

describe('Test', () => {

  const expectedText = {
    textMenu: 'administrator'
  };

  it('Login Test', { tags: '@smoke' }, () => {

    const username = Cypress.config('username');
    const password = Cypress.config('password');

    cy.visit(Cypress.config('baseUrl') + Cypress.env('loginUrl'), {
      failOnStatusCode: false });

    cy.get('[id="username"]').type(username);

    cy.get('.width-40').click();

    cy.get('[id="password"]').type(password);

    cy.get('.width-40').click();

    cy.get('.user-info').invoke('text').should('include',
      expectedText.textMenu);
  });
})
```
```

## ChatGPT full answer

### (1 attempt)

One code smell that occurs in the provided test is:

\* **Brittle selectors**: The test uses CSS selectors with specific class names to interact with elements, such as `'[id="username"]'`, `'width-40'`, and `'user-info'`. If the class names or IDs of these elements change in the future, the test will break. It's better to use

more robust selectors, like data attributes or unique IDs that are less likely to change.

To improve the test, consider using more stable selectors like data attributes or unique IDs. For example:

```
```js
cy.get('[data-cy="username"]').type(username);
cy.get('[data-cy="login-button"]').click();
cy.get('[data-cy="password"]').type(password);
cy.get('[data-cy="login-button"]').click();
cy.get('[data-cy="user-info"]').invoke('text').should('include',
expectedText.textMenu);
```
```

By using data attributes (`data-cy`) in this example, the test is less likely to break if the UI changes, as the attributes are specifically added for testing purposes and not affected by any design changes.