UNIVERSITY OF ANTWERP
DEPARTMENT OF COMPUTER SCIENCE

Software Testing Lab

(Summer School "Software Testing and Beyond")

Assignment 4 - GUI Testing (Cypress)

Submission Deadline: **July 3 2025, 20:00**

In this exercise we will use the `Cypress` framework to express tests that are executed against the Graphical User Interface of a web-system. Test scripts are written in JavaScript; there are a lot of java libraries available that allow for processing of HTML-files.

Important Note: For this assignment we will use `ParaBank`; a fake on-line bank system [https://parabank.parasoft.com/parabank/]

1 PREPARATIONS

Before starting the assignment you should install a series of tools and packages. Detailed installation instructions (with screen dumps) can be found on many on-line tutorials.

1. Download and install Node.js [https://nodejs.org/en/download]
2. Create a separate directory to install Cypress. The npm install will take a while.
   ```
   cd xxx
   mkdir Cypress
   cd Cypress
   npm install cypress --save-dev
   npm install
   ```
3. Open the cypress environment

`npx cypress open`

++ Click on the “Testing” button on the left to get started

++ Press "Continue" to have a series of configuration files installed

++ Choose a browser (Chrome or FireFox)

++ “Start E2E Testing in Chrome” button

4. Create the first test script (called a "spec")

   ++ Create a new spec

   ++ Enter the path

   `cypress/e2e/ParabankWebSite.cy.js`

   ++ Press the “Okay, run the spec” button. [This should open a default web-page provided by cypress.]

5. Link cypress with your favorite IDE

   (recommended is VisualStudio)

   In the web-browser that you used for E2E testing; left right there is a "Settings" button

   Go to "Device Settings" and there select the External Editor.

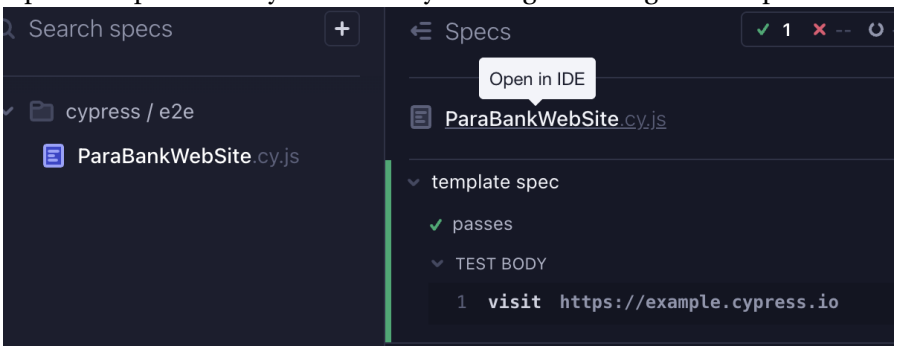
   1 Sometimes VisualStudio is not listed among the options. If that is the case go to Visual Studio Code, go to View » Command Palette and paste the following command into the field.

   `Install 'code' command in path`

   Then press Enter. You will need to close and reopen VisualStudio and Cypress.

6. Edit the code

   Open the spec within your editor by clicking on the rightmost spec-file name



   Replace the URL to point to the parabank web-site [https://parabank.parasoft.com/parabank/]

   Rename the name of the test script to “Open ParaBank spec”. You should see the code shown in Listing 1.

   Save in the IDE should automatically rerun the test script and you should see the ParaBank home page now.

7. Ask for help if you did not reach this point.

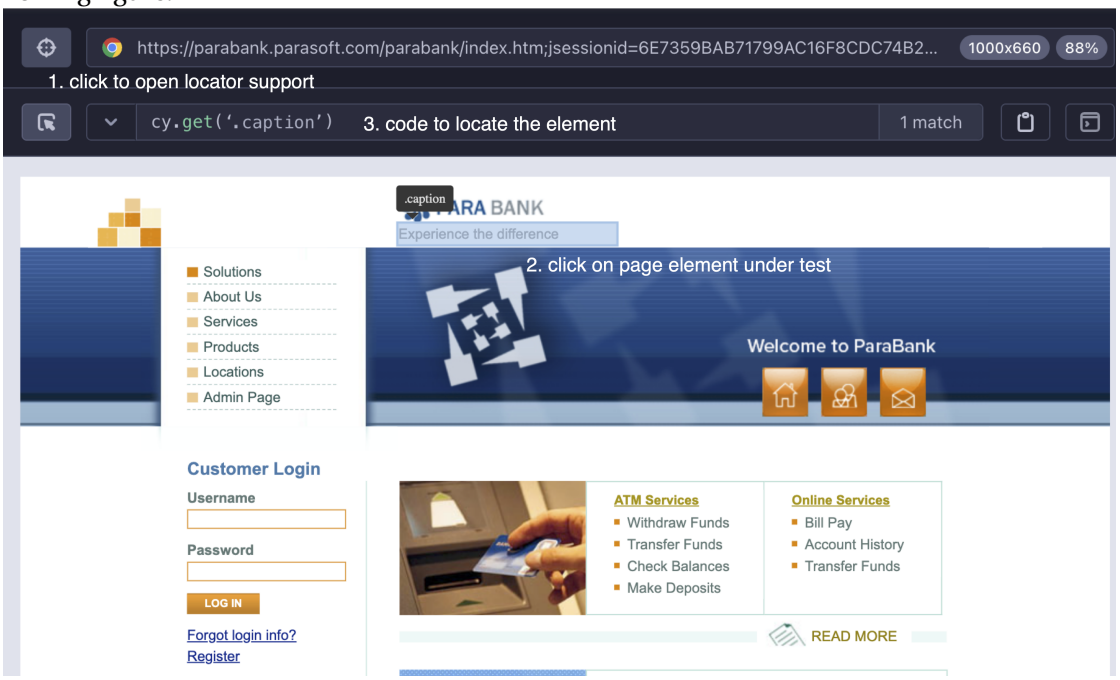
```
1 describe('Open ParaBank spec', () => {
2   it('passes', () => {
3     cy.visit('https://parabank.parasoft.com/parabank/')
```

```
    })
})
```

Listing 1: Test which opens a web-page

2 Locating Items on a Page

GUI testing for web-applications mainly involves manipulating the various elements on a given page. These can be located with a `.get()` command in the test spec. The parameter is a string which identifies the element within the representation of the web-page. The cypress editor allows to click on a page-element and immediately returns a locator as show in the following figure.



1. click besides the url to open locator support
2. click on the page element under test
3. find the code to locate the element

Finally one can verify the contents of the element using `.contains()`. This should result in the code shown in Listing 2.

```
describe('Open ParaBank spec', () => {
  it('passes', () => {
    cy.visit('https://parabank.parasoft.com/parabank/')
    cy.get('.caption').contains("Experience the difference")
  })
})
```

Listing 2: Test locating an element and verifying its contents

Input fields are located and asserted a little differently. Listing 3 shows code snippet with 2 versions for locating an input and then verifying whether it exists and is visible.

```
describe('Open ParaBank spec', () => {
  it('passes', () => {
    cy.visit('https://parabank.parasoft.com/parabank/');
    cy.get(':nth-child(2) > .input').should("exist").should("be.
        visible");
    cy.get('input[name="username"]').should("exist").should("be.
        visible");
  })
})
```

Listing 3: Test locating an input element and verifying that it exists and is visible

- **Exercise 1.** *Locating page elements.* Read the document "Cypress Locators : How to find HTML elements" [https://www.browserstack.com/guide/find-html-element-using-cypress-locators; also on BlackBoard]. Create two extra locator tests for (a) the "Password' field; (b) the "LOG IN" button and verify that they exist and are visible. Put the code in your report **(Required, 10 points)**

- **Exercise 2.** *Reflection: Coverage (RIPR Criterion).* Considering the different elements on a web-page how would you measure the coverage of a GUI test script? Write a short paragraph (150 words max) explaining how you could count how many elements have been reached by a test script. Can we also express this as a percentage? **(Required, 10 points)**

3 Multiple Tests

The test script becomes quite lengthy and verbose. Better to split up into several tests; one for verifying the right header; two for verifying the username password fields and a fourth for verifying the submit button. Listing 4 shows code snippet the first two of the four tests. Note the use of `before-each()` which avoids duplicating the visit to the log-in page.

```
Describe('Open ParaBank spec', () => {

  beforeEach(() => {
    cy.visit("https://parabank.parasoft.com/parabank/")
  })

  it('Has the right header', () => {
    cy.visit('https://parabank.parasoft.com/parabank/');
    cy.get('.caption').contains("Experience the difference");
  })

  it('Has a username field', () => {
    cy.get(':nth-child(2) > .input').
        should("exist").should("be.visible");
    cy.get('input[name="username"]').
```

```
16            should("exist").should("be.visible");
17    })
18  })
```

Listing 4: Splitting up the test and avoiding duplicate logic via `before-each()`

- **Exercise 3.** *Clean code.* Split the code for the four tests just like in Listing 4. Put the code in your report **(Required, 10 points)**

- **Exercise 4.** *Additional Tests. (ONLY WHEN LATE)* Create tests for verifying (a) the presence of the three orange buttons in the headerpane; (b) the presence of the menu items (Solutions, About Us, ...). **(Only when late, -10 to 0 points)**

4 Actions

Go to the ParaBank web-site separately and create yourself a valid username and password. Use these to log into the web-site using the extra test in Listing 5.

Special: Note the use of the `.only` after `it`: this selects which of the test cases to run; all others will be ignored. A convenient way to focus on the test under development.

```
1  it.only('Can login with a predefined username and password', () => {
2    cy.get('input[name="username"]').focus().type("      ");
3    cy.get('input[name="password"]').focus().type("     ");
4    cy.get('input[type="submit"]').click();
5  })
```

Listing 5: Action test filling in fields and clicking on buttons

- **Exercise 5.** *Screen dump.* Run the test and put a screen dump of the resulting Cypress output in your report. **(Required, 10 points)**

- **Exercise 6.** *Verify page transition (OPTIONAL).* Listing 5 is an assertion less test. It doesn't verify that we actually land in another part of the web-site with a valid username and password. It also does not verify whether we receive an error message with an invalid combination of username and password. How would you code this in cypress? **(Bonus, 20 points)**

5 Page Object

All these locators imply that the tests are very brittle. The moment something changes in the underlying web-page representation all the tests which use the page element which has changed will need to change. Read the "Page Object" patterm [https://martinfowler.com/bliki/PageObject.html; also on Blackboard] to understand how one can resolve this.

Here's how to do the Page Object in Cypress.

1. Add an extra directory `pages` at the same level of the `e2e` directory in cypress. The directory should look as follows.
   ```
   cypress
   cypress.config.js
   downloads
   e2e
   fixtures
   pages
   support
   ```
2. Create a file in the `pages` directory named `homePage.js` to contain the code for the homepage
3. Paste the following code inside `homePage.js`. (Beware the `export` statement at the end.)
   ```
   class homePage {

       header() {
           return cy.get('.caption');
       }

       usernameField() {
           return cy.get('input[name="username"]');
       }
   }

   export default new homePage();
   ```
4. Adapt the code of the `ParaBankWebSite.cy.js` test script (in the `e2e` directory.
   ```
   import homePage from "../pages/homePage";

   describe('Open ParaBank spec', () => {

       beforeEach(() => {
           cy.visit("https://parabank.parasoft.com/parabank/")
       })

       it('Has the right header', () => {
           homePage.header().contains("Experience the difference");
       })

       it.only('Has a username field', () => {
           homePage.usernameField().should("exist").should("be.visible");
       })

   })
   ```

- **Exercise 7.** *Page Object – Code.* Expand the code in `homePage.js` to have entries for the password field and submit button. Rewrite the code in Listing 5 accordingly **(Required, 10 points)**

- **Exercise 8.** *Page Object – Reflect.* Explain in a handful of paragraphs (250 words max) how this page object patterns helps with the test quality. Comment from the following perspectives (a) coverage; (b) code size; (c) readability; (d) maintainability. **(Required, 20 points)**