June 20, 2023

CS453: Automated Software Testing

Web Page Functionality Based Path Coverage

Team 6

Han Wei Guang
Zofia Marciniak
Anton Shatokhin
Brandon Ong

Spring 2023

June 20, 2023

Table of content

June 20, 2023

1. Introduction

The demand for UI/UX interaction designers has been increasing drastically[1]. Many companies have adapted the role of UI/UX as an important step of testing the usability and user-friendliness of their interfaces.
Taking as an example web interfaces, the UI/UX designers create not only the visual design of the page, but mostly the interactions by appealing to one's subconsciousness through design. The outcome should be intuitive, and suggest to the user how to navigate the page. Then, the designers improve the design through user studies of the interface.
Naturally, the next step is actual implementation of the web page. In the process of communication with the developer, many details get lost due to different needs: designers focus is on users, while developers focus is on plausibility of designer's ideas in terms of implementation. Because those needs are not transferred between designer and developer, another testing of the web interface is usually needed to catch the developer's misunderstanding of the designer's intention.

2. Problem statement

The designers make a prototype in Figma, software for prototyping UI and UX, where both visual design and interactions of buttons or interface parts is possible (click, drag, hovering, animations). It is important to preserve the interactions intended by the designer. This is currently not achieved well, because the developers have to follow the Figma design without the familiarity with the tool from the kitchen. The developers work with the Figma outcome and figure out how to program it. Through automated testing, we want to take a step back and consider the designer's intention in order to test the developer's outcome. In this project, we focus on web design.

3. Solution

To tackle this issue, we have researched different ways to export the Figma design for automated testing. We have found a community-contributed Figma plug-in[2] that helps export the design into HTML code for element presence identification.
Before that, it is necessary for the designer to prepare the Figma file for exporting.
Designers will be required to label their components in their Figma design, specifying if the component is a button or a text field as shown in *Figure 1*.

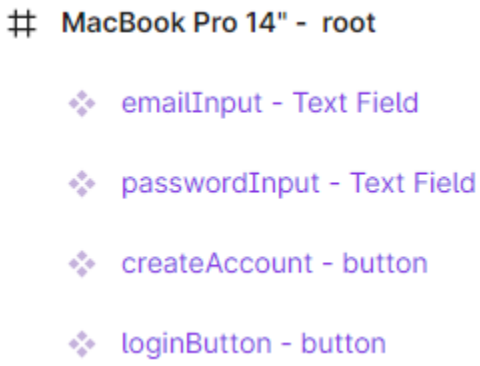

Figure 1. The labeling of Figma elements

June 20, 2023

When the labeling and design is done we can generate a HTML file with the Figma Plugin.

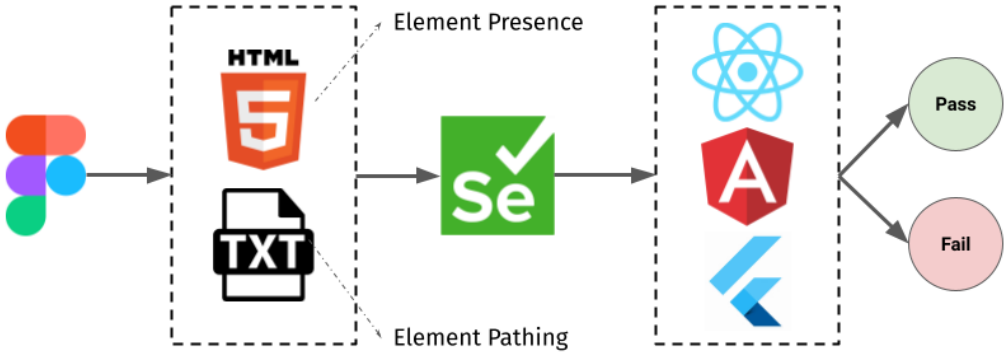

Figure 2. System diagram of the solution: Figma design is expired as HTML, with an additional TXT from a designer. Selenium is used to interact with developer's outcome to assess Pass or Fail of a test case.

The generated HTML file will consist of proper HTML elements (such as buttons and inputs) that will be parsed into our program to generate test cases.

```
<div id='macbookpro14"--homepage' class='macbookpro14"--homepage'>
    <button id='signout-button' class='signout-button'>
    Signout</button>
    <button id='home-button' class='home-button'>
    Home</button>
    <button id='mail-button' class='mail-button'>
    Mail</button>
    <button id='settings-button' class='settings-button'>
    Settings</button>
    <div id='line1' class='line1'>
    </div>
    <img id='cookielogo' class='cookielogo' />
    </img>      You, 4 days ago • feat: html files …
    <img id='cookiecolour' class='cookiecolour'/>
    </img>
    <button id='eatcookie-button' class='eatcookie-button'>
    <div id='rectangle9' class='rectangle9'>
    </div>
    <div id='eatcookie2' class='eatcookie2'>
```

HTML files should also be named <path>.html, eg. root.html, create.html, login.html, as the program paths itself based on the HTML file name.

Eg, when generating test cases for create.html, the program accesses https://domainname/create by appending “/create” on the back of the inputted URL.
These test cases will be run against the developer’s code to check if the expected elements from the HTML files are presented correctly on the deployed website.

4. Results and usage

4.1. Running the tests (example)

In order to run the code, a test web page has to be developed first (locally). After that, the following parameters have to be executed:

<-f> flag to specify the folder of the html files
<-u> flag to specify url of website to be tested (without any paths)

```
py .\tester.py -f .\figmahtmls -u http://localhost:3000
```

Test results will be shown as

```
--------Test cases in http://localhost:3000/homepage--------
5/5 tests passed
0 tests failed
--------------------------
```

If all test case on page is successful

```
--------Test cases in http://localhost:3000/homepage--------
4/5 tests passed
1 tests failed
--------Failed tests in http://localhost:3000/homepage--------
TimeoutException occured, no such element of type button with id eatcookie-button was found
--------------------------
```

If a test case failed

```
--------Test cases in http://localhost:3000/homepage--------
5/5 tests passed
0 tests failed
--------------------------
--------Test cases in http://localhost:3000/mail--------
4/4 tests passed
0 tests failed
--------------------------
--------Test cases in http://localhost:3000--------
4/4 tests passed
0 tests failed
--------------------------
--------Test cases in http://localhost:3000/settings--------
7/7 tests passed
0 tests failed
--------------------------
```

Or if you ran the test with a folder that contains HTML files.

4.2. Figma to web testing

In order to test our software, we had to generate examples of Figma designs and its corresponding web interface. Due to this method being very time consuming, and a limited number of existing examples of Figma and Web pages, we have generated only one fully working example. But we found a method to simplify the process for future automation. Using another community-contributed plugin, html.to.design[3], available also as a Chrome extension, it is possible to generate design components from html code in Figma. We have used this method to create a basic example of interactions on a shopping website[4], and the example usage is stated in the github repository of our project [5].

5. Conclusions

5.1. Obstacles

To summarize our project, we have divided the testing algorithm into two parts: element presence testing (input: html code from Figma), that checks whether all components are present on the web interface using Selenium, and element pathing, that checks whether the components route to correct pages (input: html code from Figma, path routes from TXT). For testing of our program, we have used an example developed by us Figma design and its web interface, and a real-world (simplified example). One of the obstacles of our project was designing the routing algorithm for element pathing. Finally, we have decided to settle for all paths that start at the first (index) page of the web interface.

5.2. Limitations

There are several limitations of our project that could be addressed by more experienced or if we had more time. First and most important limitation is the path routing algorithm of our tester. Since we check all the paths, the time consumed by generating and executing the paths prelongs the testing. This method is plausible for small web interfaces, but testing more

complicated ones would take too long. Another limitation is being dependent on community-contributed plugins that are limited within themselves. Labeling of the Figma elements is currently necessary. This could be addressed in the future by understanding the semantics of generated html code and making a more organized and automatized system. Lastly, the limitation is the availability of Figma to Web Interface examples, as discussed in 4.2.

5.3. Possible improvements

We have used Selenium for testing both element presence and element pathing. This makes our method a black-box testing method, and contributes to our idea of executing the user side of design. Using Selenium makes it more time consuming because of path routing. For white-box testing, a more semantic level understanding of web interface structure could speed up the process of checking both element presence and pathing, but might not be ideal to test the user view of the interface.

Reference

[1] Wang, Li. (2020). An Extensive Research on Application of Interaction Design in Products. Journal of Physics: Conference Series. 1533. 022014. 10.1088/1742-6596/1533/2/022014.
[2] https://www.figma.com/community/plugin/1128731099343788397/Figma-to-HTML-and-CSS
[3] https://chrome.google.com/webstore/detail/htmltodesign/ldnheaepmnmbjjjahokphckbpgciiaed/related
[4] https://www.asos.com/
[5] https://github.com/bojx96/CS453_Automated-Software-Testing-Project