CSE – 523

ADVANCED PROJECT – GOOGLE DOCS

FINAL REPORT

Date: 22nd March, 2020

Author - Shubhi Nigam (112672816)

PROBLEM STATEMENT:

With the rise in the number of visually impaired people in the world, our job in this project is to incorporate a set of features that will impart some of the possible visual abilities to the impaired by assisting them to have a comfortable digital experience that someone with a perfect vision possesses.

Our job at hand is to build custom-based add-on in Google Docs, highlighting the menu items, performing the exact same function as the regular menus in the taskbar do. This zoomed-in feature will help people with low-vision with easy accessibility of the document.

TASKS ASSIGNED/PERFORMED:

1. Build a sample Google Docs add-on and scrape menu metadata from the document - Added a sample add-on to Google Docs and created an html page along with associated JavaScript file to have a custom-based Google Maps displayed. Users can enter any location and the corresponding map will get displayed on the document. Scraped document data using Python libraries.

2. Downloaded complete dump of Google Docs DOM. Analyzed the HTML as well as the dynamically rendered DOM elements. Inspected the DOM elements and found that many fields were hidden initially in the page source. Copied the multi-layered data from the HTML dump. All the menu details are provided under <span aria-label>. Extracted the complete list of menus including multi-layered lists.

3. Extracted the hidden menus and other menu items using vim commands adhering to the hierarchical order of the multi-layered nested DOM list. Found a few traditional events which get triggered when a user opens a document - authMode, source, etc. Started learning Puppeteer to scrape the menu details.

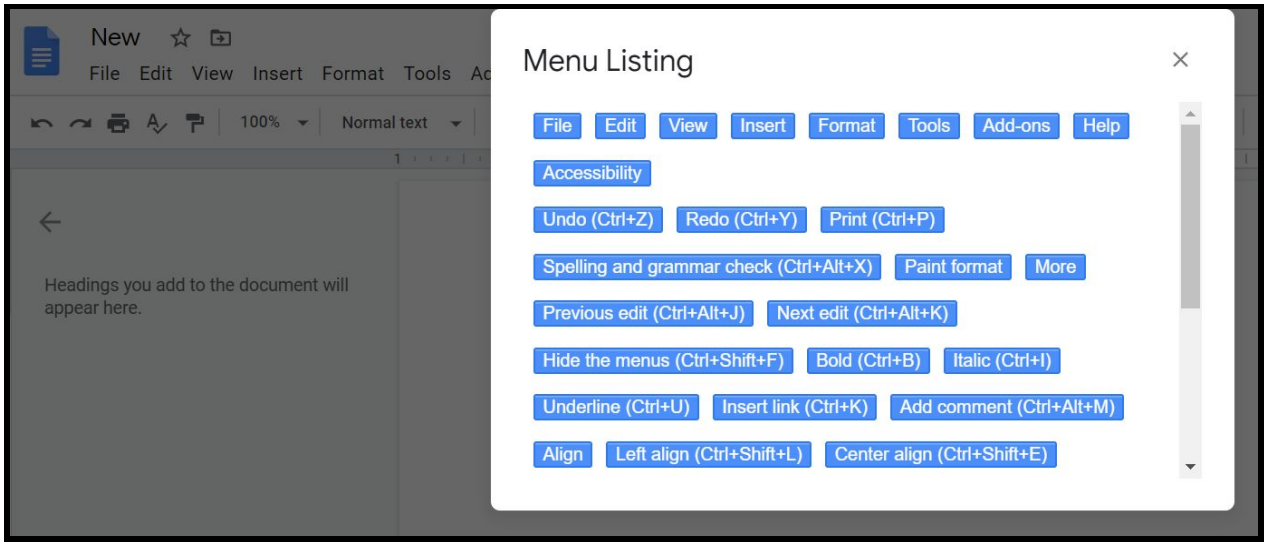
4. Transpiled code is scraped and captured from Google Docs using active listeners. Tried converting the transpiled code into human-understandable format using TypeScript software. Started exploring Angular and kx editor, the underlying architecture for Google Docs to better understand the transpiled code.

5. Explore dialog modal on Apps Script and replicate menus on the taskbar - Implemented dialog modal on Apps Script displaying multi-level hierarchy. JavaScript buttons able to render menu items dynamically on the click event. There were various issues faced - Transport Error encountered, which was dealt temporarily using browser settings. Using

this procedure, only HTML DOM was captured. Global DOM needs to be accessed from IFRAME (dialog modal).

6. Transport Error was fixed permanently and changed its severity from error to warning. Able to access global, dynamically-rendered DOM using Puppeteer. Tried Sandbox Puppeteer in headless Chrome to capture the parent DOM elements which were recorded by using the “await” functions which held the page until the entire components were loaded and then proceeded to capture the entire DOM.

7. Created JS menu buttons on dialog modal of Apps Script by accessing parent DOM using Puppeteer wherein browser-based functionalities are implemented to capture the JS rendered DOM embedded inside the main HTML page of Google Docs. Project was transformed into Google Cloud Project. A cloud function was created which used Puppeteer and Node.js to run on the cloud and access the dynamically rendered DOM elements of the parent document. Then this trigger URL is used by Apps Script to fetch resource content and passed onto the client-side function as a string to be parsed by DOM parser. Using parsed document object, class names are extracted and a menu list is created.



8. CSS styling was added to the JS buttons. Puppeteer, just like any other bot/fake browser is able to capture the public page of the Google Docs document. In order to capture in-depth functionalities of the document, Puppeteer needs to sign in the document virtually and then access the DOM which then possesses the entire document elements along with various click-events. Issues faced - Google started blocking Puppeteer when it tries to pass the user credentials by recognizing it to be a fake browser.

9. Tried accessing the Google Docs DOM by using internal GAS functions and user authentication token, however received only HTML document elements instead of dynamically loaded and rendered DOM. Switched to Puppeteer and tried passing user authentication deets to the interface. Faced timeout error of the button loading on the

page, i.e. not enough time was provided to Puppeteer to load the entire page elements and then capture the DOM.

10. Resolved the timeout issue by providing appropriate time halts to different await functions in the Puppeteer code to load all the functionalities before proceeding and clicking on the next submit button on the login page. However, Google is able to recognize Puppeteer as a fake browser/artificial bot and blocking the path further by throwing challenges like captchas, otp requests on phone numbers, etc.
    This policy was not present earlier, but now Google has created a rule for such fake browsers to avoid man-in-the-middle attacks. Researched multifarious articles on these and found that for security reasons Google has implemented this feature to block the hidden features/DOM access to the page from the fake browsers like Puppeteer, Selenium, etc. More details to be found on this implementation on the below URL:

    https://support.google.com/accounts/thread/22873505?hl=en

11. Puppeteer unable to proceed with email ids created before 2019 due to this blocking feature. So, created a new email id account - "dummydummy330@gmail.com" for testing purposes. However, due to security reasons, Google throws different challenges at different levels which cannot be incorporated in the current code.

    [Code and snapshots available on github repo]

12. Captchas are using challenge protocols and hence, appear at uncertain times (not fixed). Puppeteer is failing to capture these. Various resolvers are available online, but they are paid services which will incur additional dependency on this application. Tried Selenium and found several other challenges. Therefore, switched back to Puppeteer considering it a better platform for the project and closely related to Google-based services, however due to strict privacy policies, different challenges are thrown to Puppeteer to solve and send back the response which is impossible for Puppeteer to resolve in dynamic run-time.

USEFUL LINKS

1. Github repository - https://github.com/ShubhiNigam29/Javascript

2. Google Docs in-bound script - https://docs.google.com/document/d/1VFaU79lhqmpE4F_q-tZa3P3Qtj2FY_YigA2eywTeAdY/edit

3. Google Cloud Platform functions - https://console.cloud.google.com/functions/list?authuser=2&project=menu-1585020565561&tab=general

4. Sandbox Puppeteer - https://puppeteersandbox.com/ (currently down & unavailable)