# Design of “A Very Random Quiz” Web App

## Introduction

“The Very Random Quiz” web app is a trivia quiz. It consists of five questions varying in topic and level of difficulty. If the user answers at least four questions correctly, they are rewarded with a dad joke. If the user answers at least two incorrectly, they are punished with an inspirational quote.

## Goals

* Simple layout that is thematically consistent
* Load a random question every time a button is pressed
* Display a random inspirational quote if at least two answers are incorrect
* Display a random dad joke if at least four answers are correct
* Responsive on various devices

## Choice of APIs

This web app uses three APIs: a trivia API (<https://opentdb.com/api_config.php>), a dad joke API (<https://icanhazdadjoke.com/api>) and an inspirational quote API (<http://forismatic.com/en/api/>) to create a quiz game. The trivia API provides random questions and multiple answers. Calling this API creates a varied quiz with different questions every time. The dad joke and inspirational quote APIs obtain data that is utilised at the end of the quiz. If the user passes, a joke is generated by the dad joke API and displayed on the screen. If the user fails, a quote generated by the inspirational quote API is displayed on the screen instead.

## Design Theme

This web app was designed with the concept of a modern web page in mind—maintaining a minimalist design and a simple code (Takada). A minimalist design was achieved by utilising a consistent colour palette: a cream background, grey fonts and orange accents. The orange colour was chosen specifically because it is psychologically proven to stimulate creativity and fun (Create a Website). The background and font are subdued colours to make the orange accent pop out more. To further maintain a minimalist design, all of the text is in the same sans-serif font and only appears in three different sizes.

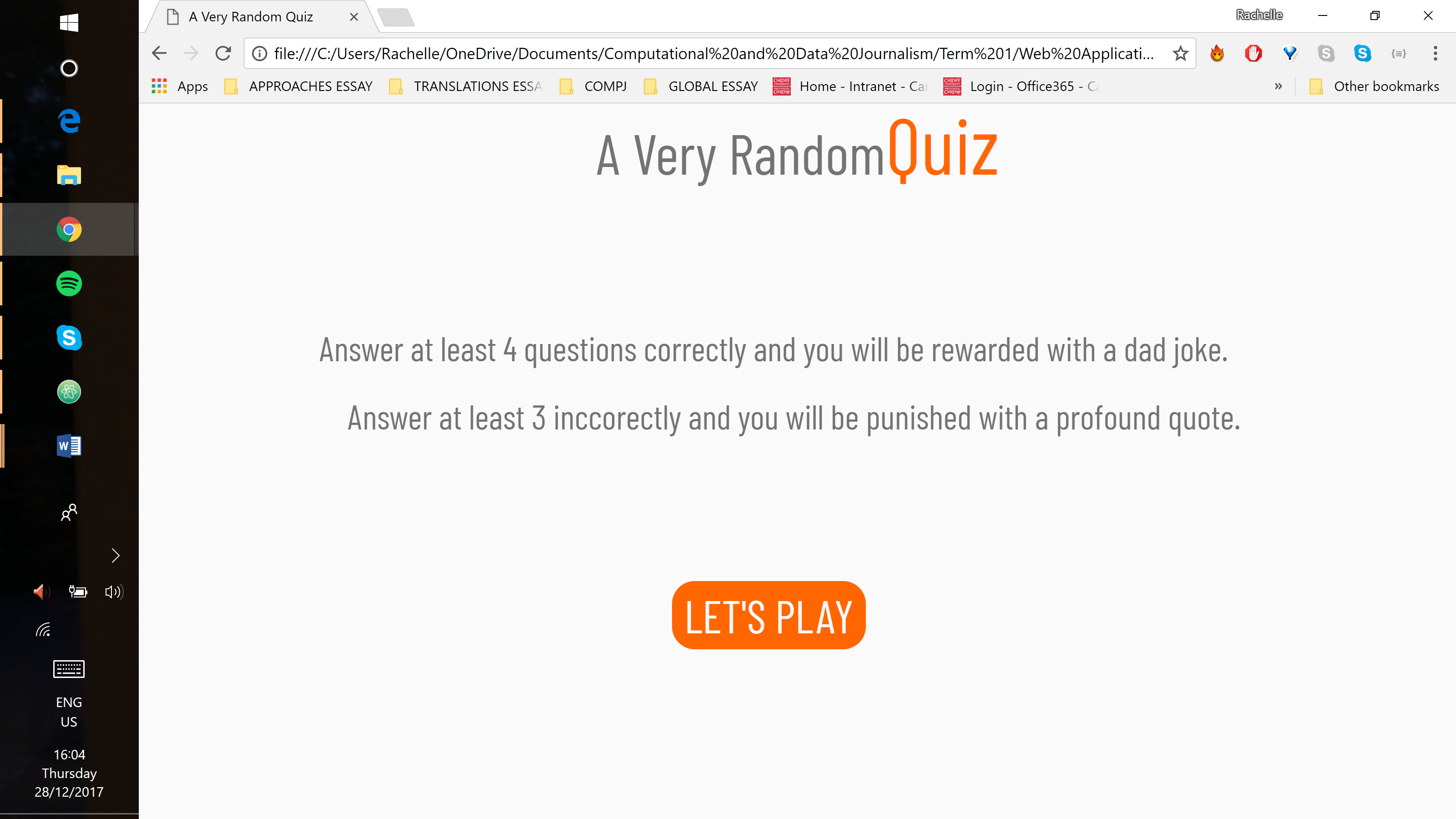
The title “A Very Random Quiz” is printed in the grey font and in a bigger size than other text. The word ‘Quiz’ is orange and in a slightly bigger font than the rest of the title. In addition, the space between the word ‘Quiz’ and the rest of the title was removed. These stylistic choices were made specifically so that the word ‘Quiz’ would be the focal point. It is also a hyperlink that leads back to the home screen and reloads the quiz. If the cursor hovers over an item that can be clicked, the cursor turns into a pointer and the item becomes orange. If that item was already orange (such as the buttons and the word ‘Quiz’) the orange becomes a more transparent shade of the orange.

The original design consisted of three HTML pages, but in order to maintain simple code, the design was restructured to just one HTML page. Doing so also eliminated the issue of losing changing data (i.e. the score) through the process of retrieving a new HTML page. There are four screens: Home Screen, Quiz Screen, Results Screen and Error Screens. The basic structure consists of one flexbox with four divs: the title div, the question div, the answers div and the button div. The title div is the only one with consistent content throughout all of the screens, so it is the only text written into the HTML. The rest of the divs are empty as their content changes with each screen and was implemented with JavaScript rather than with HTML.

### Home Screen

This is the first screen the user sees. When the page loads, all of the APIs are called and loaded. The instructions appear in the question div. There is no content in the answers div, so it is emptied and hidden to avoid a scroll bar. The instructions are followed by a button that says, “Let’s Play”. As all of this content only appears on the home screen, it was written into the JavaScript file.

APIs are invoked asynchronously, and the page is displayed before all of the data loads. The button, therefore, is hidden until the data for the questions and answers is loaded. This prevents the user from starting the quiz before it is available. Once the button is clicked, the page will switch to the quiz screen.

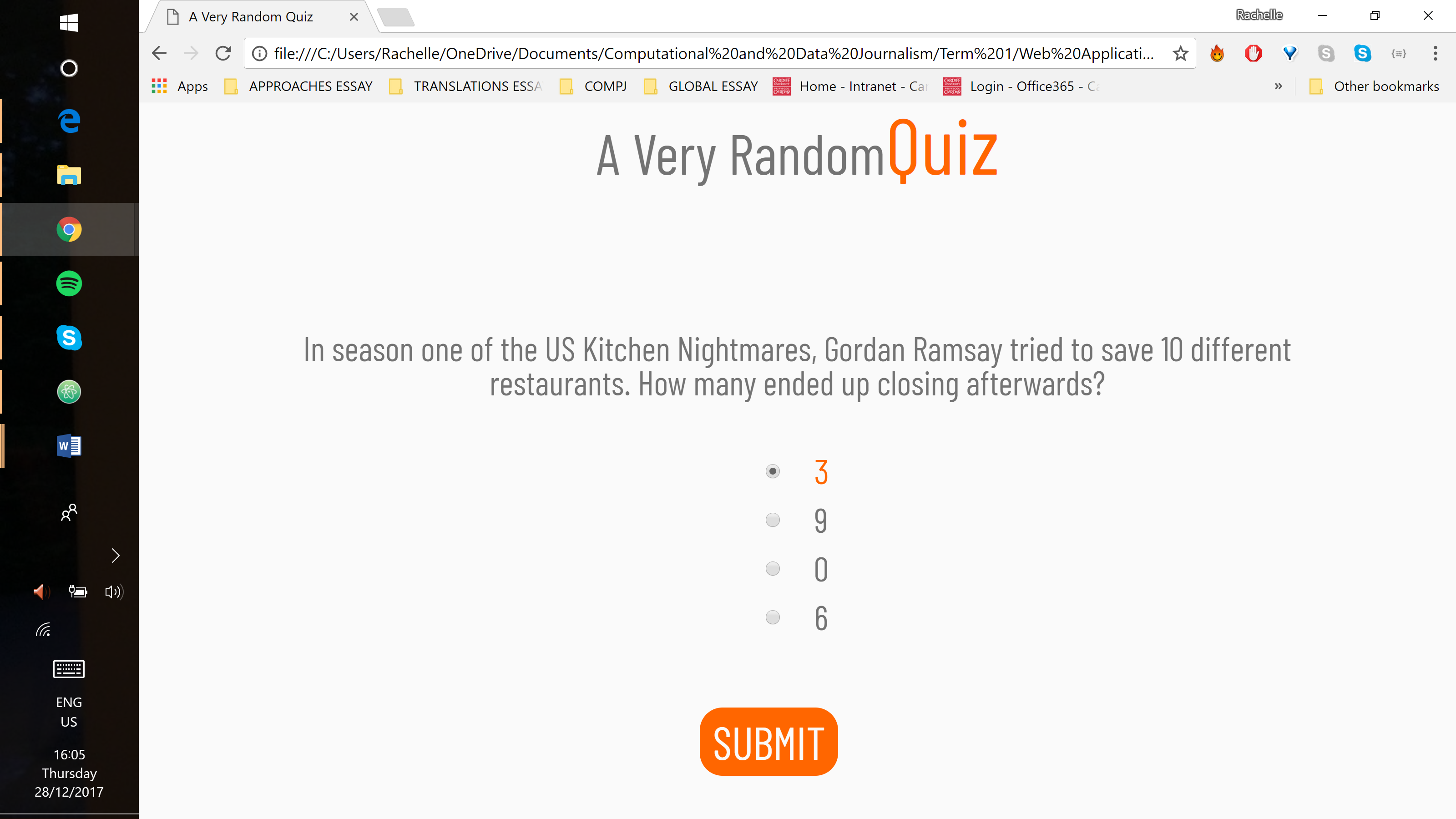


### Quiz Screen

The instructions in the question div are replaced with the question loaded from the trivia API. The API also supplies the possible answers which fill the answers div. They are structured with radio buttons and the selected answer turns orange. At the bottom, the button now says “NEXT”. This screen remains for four of the five questions.



On the last question, the button changes to “SUBMIT”. Depending on the user’s score, clicking the button will load either a joke or a quote and the page will switch to the results screen.



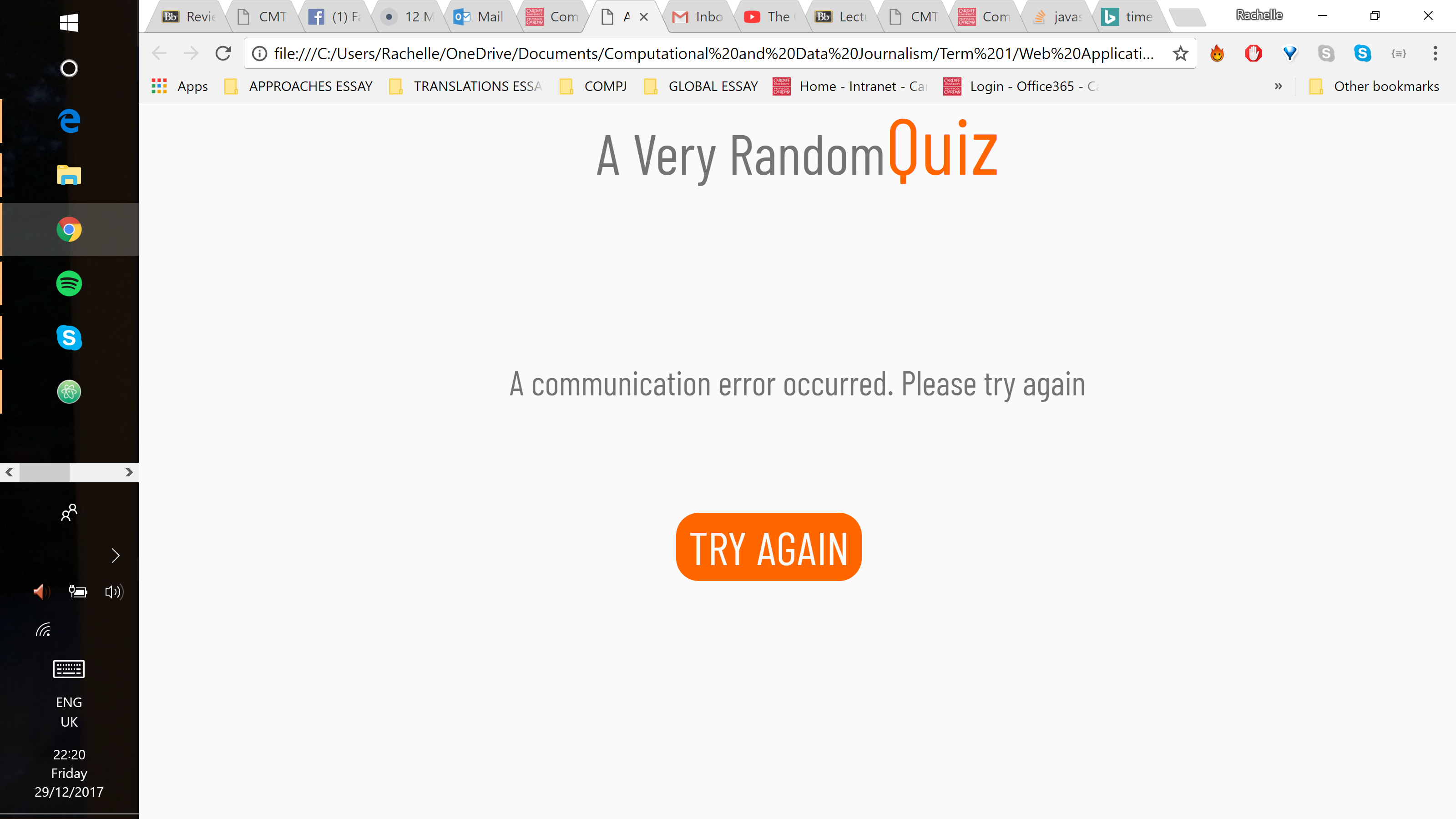
### Results Screen

If the user passed, the question is replaced with a dad joke and the button reads “Good Job! Play Again?”. If the user failed, the question is replaced with an inspirational quote and the button reads “You Suck! Play Again?”. The answers div is empty so, just like in the home screen, it is emptied and hidden, moving the button further up the page.

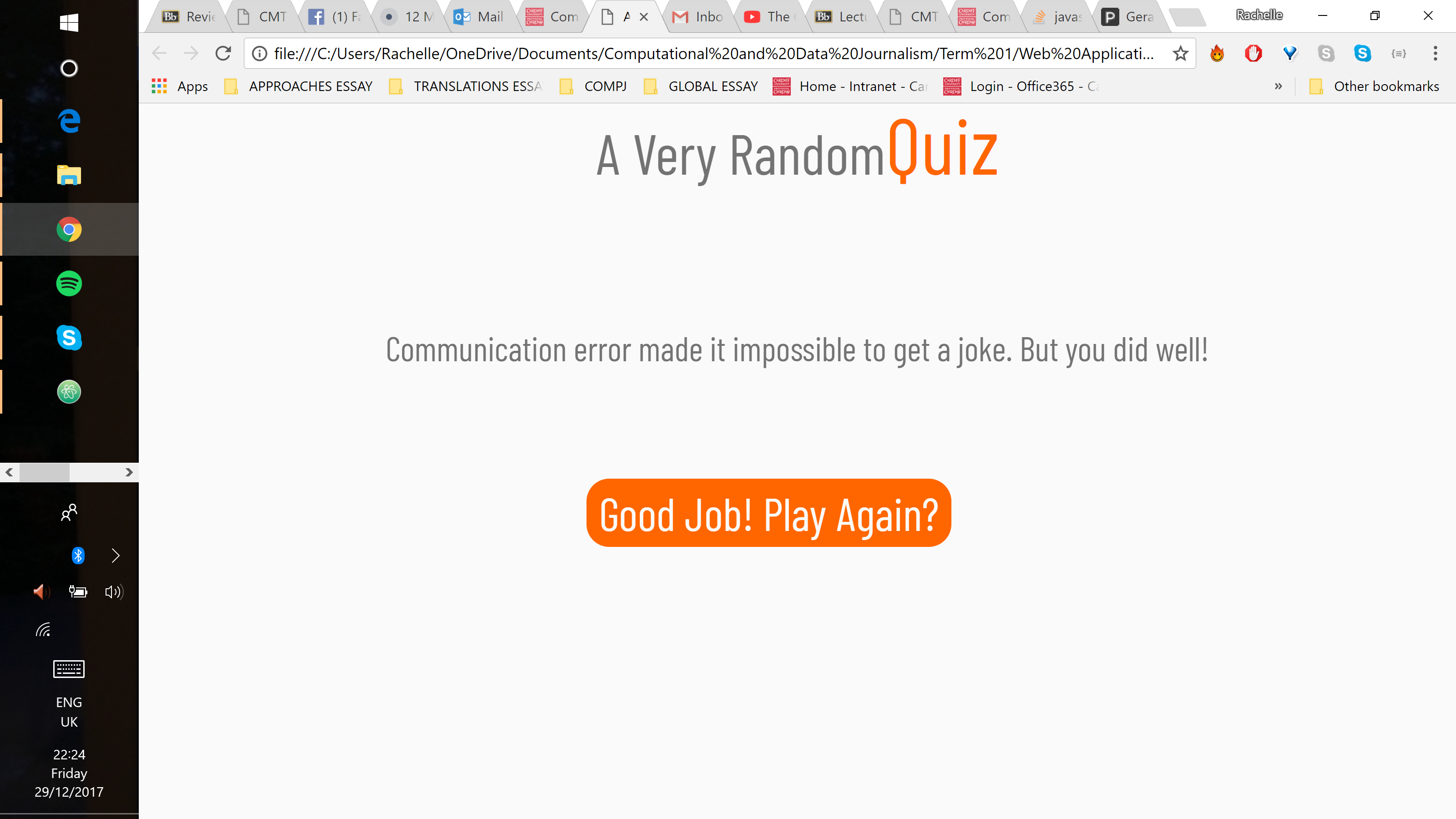
When the results are shown, the APIs are called to load new content. Just as on the home screen, the button is hidden until all of the new data loads. When the user clicks the button, they are redirected to the quiz screen rather than the home screen and automatically displays the newly-loaded questions.

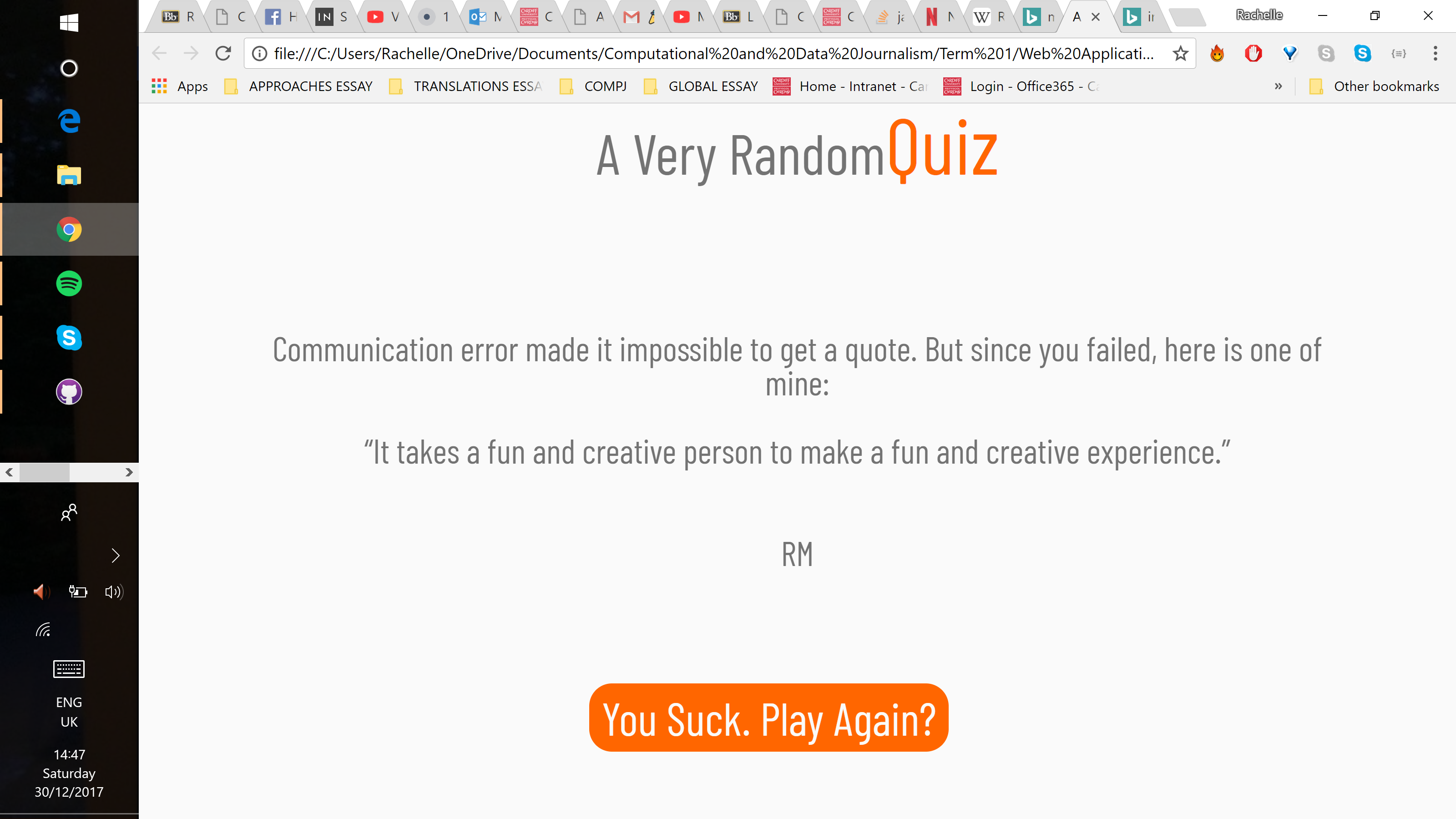
### Error Screens

All APIs are loaded when the home screen or results screen are opened. If the question API fails, the error message will display on the home screen before the user has a chance to start the quiz. The message replaces the instructions and the button reads “Try Again”. This is demonstrated in the screenshot below.



If the joke or inspirational quote APIs fail, the user is still able to finish the quiz. The error message is displayed on the results screen. It replaces what would have been the API-generated content. The button does not change. Below are screenshots for both scenarios.



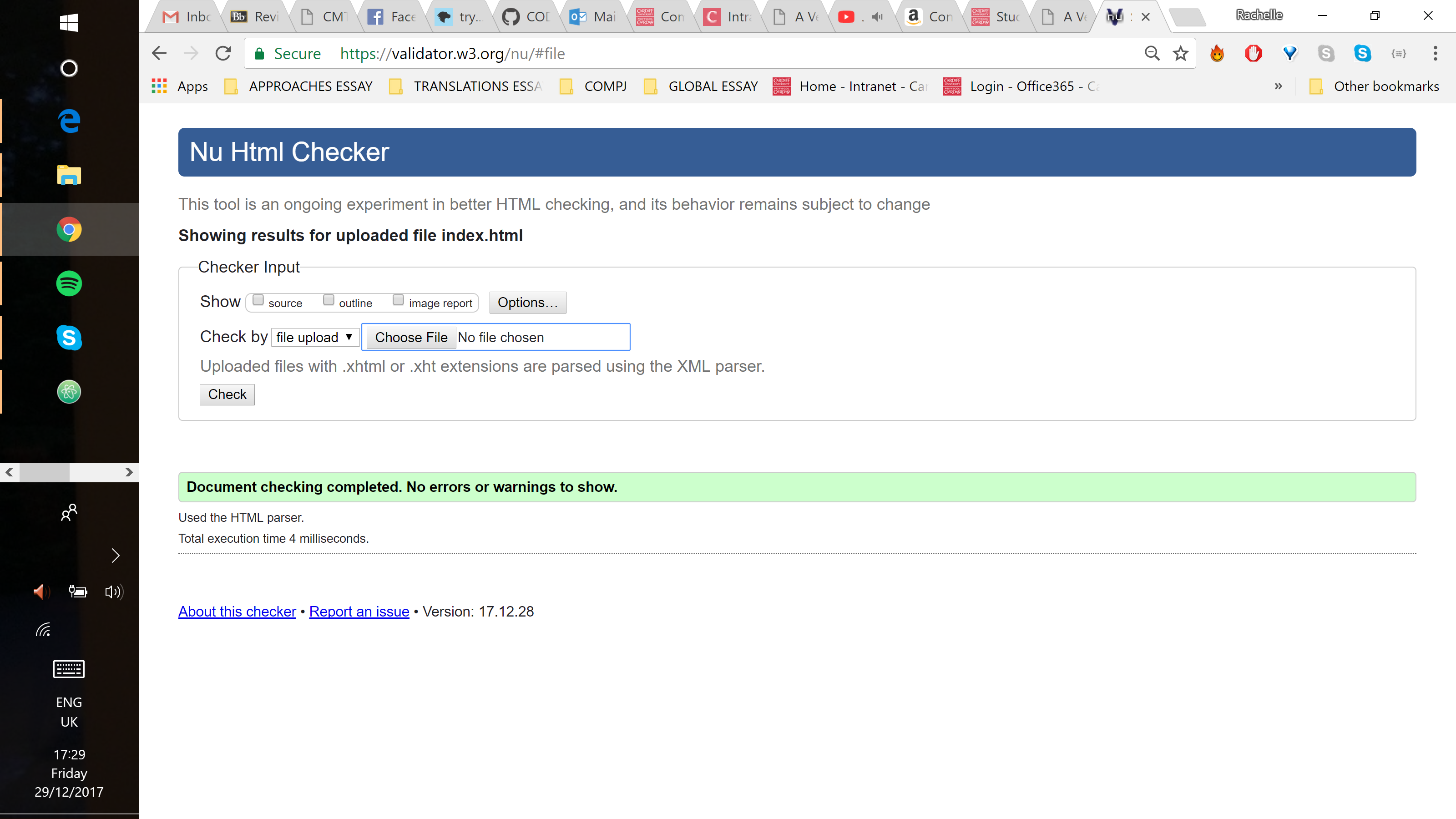


Error handling was verified by setting the URLs to invalid values.

## Validation

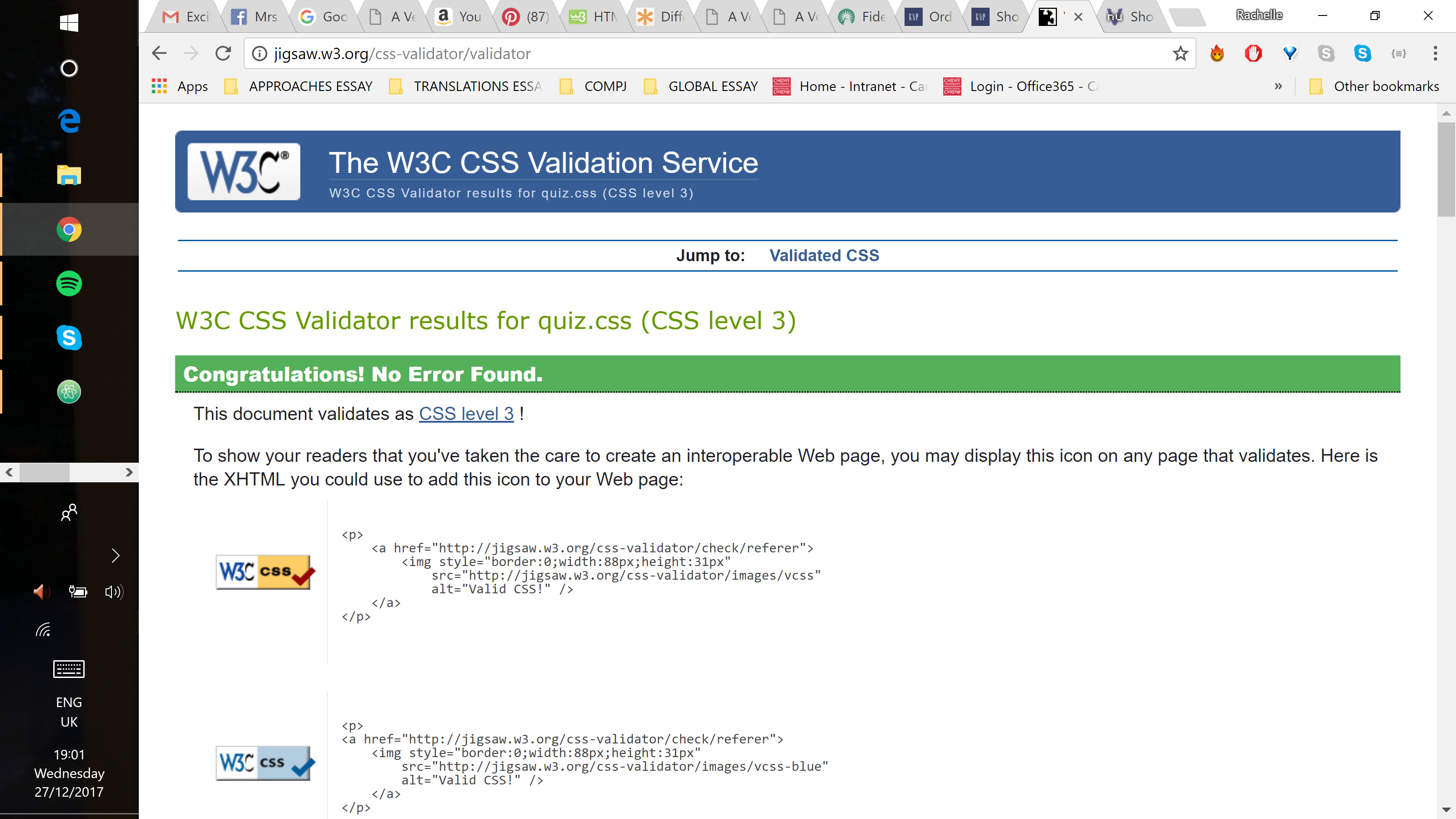
### HTML Validation

There are no mistakes in the HTML code.



### CSS Validation

There are no mistakes in the CSS code.



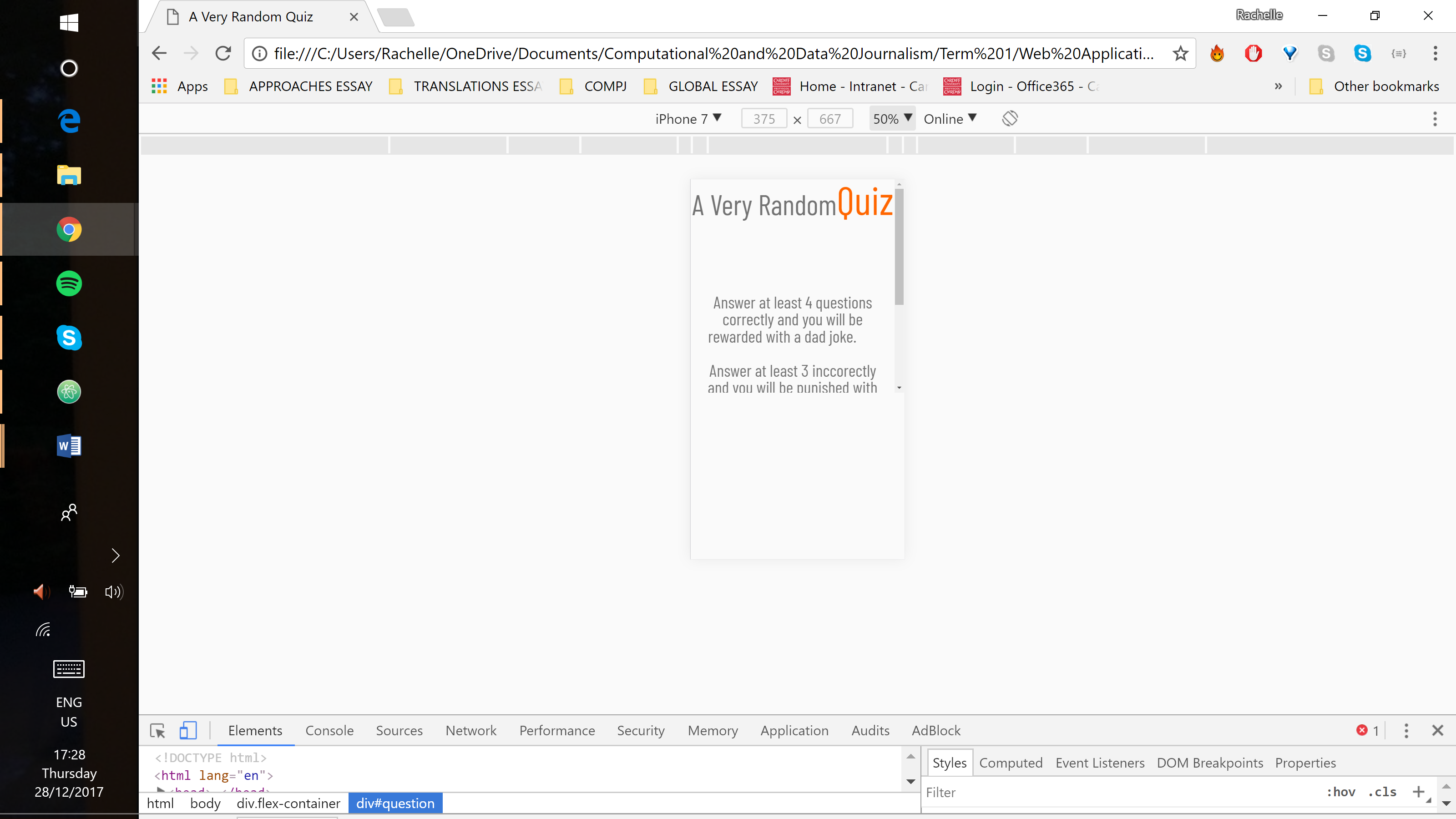
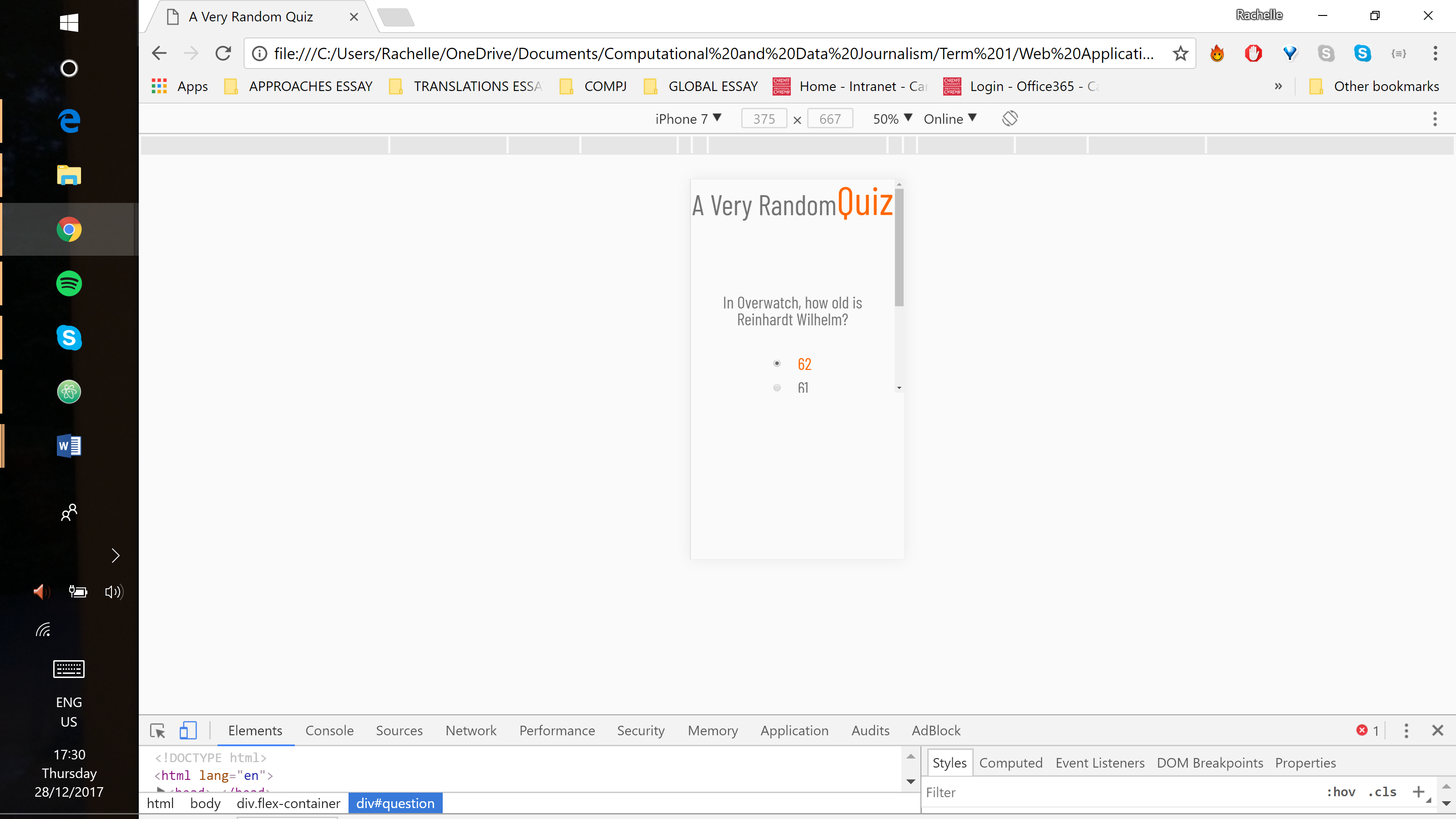
## Evaluation

The web app consists of one HTML page styled and positioned with CSS elements. JavaScript was used to call the APIs and to add interactivity. This web app demonstrates the ability to access web APIs and to retrieve, manipulate and display data. Browser debugger tools were used to better understand the performance and the execution of code in the browser. The web app also demonstrates the use of web frameworks represented by the three APIs.

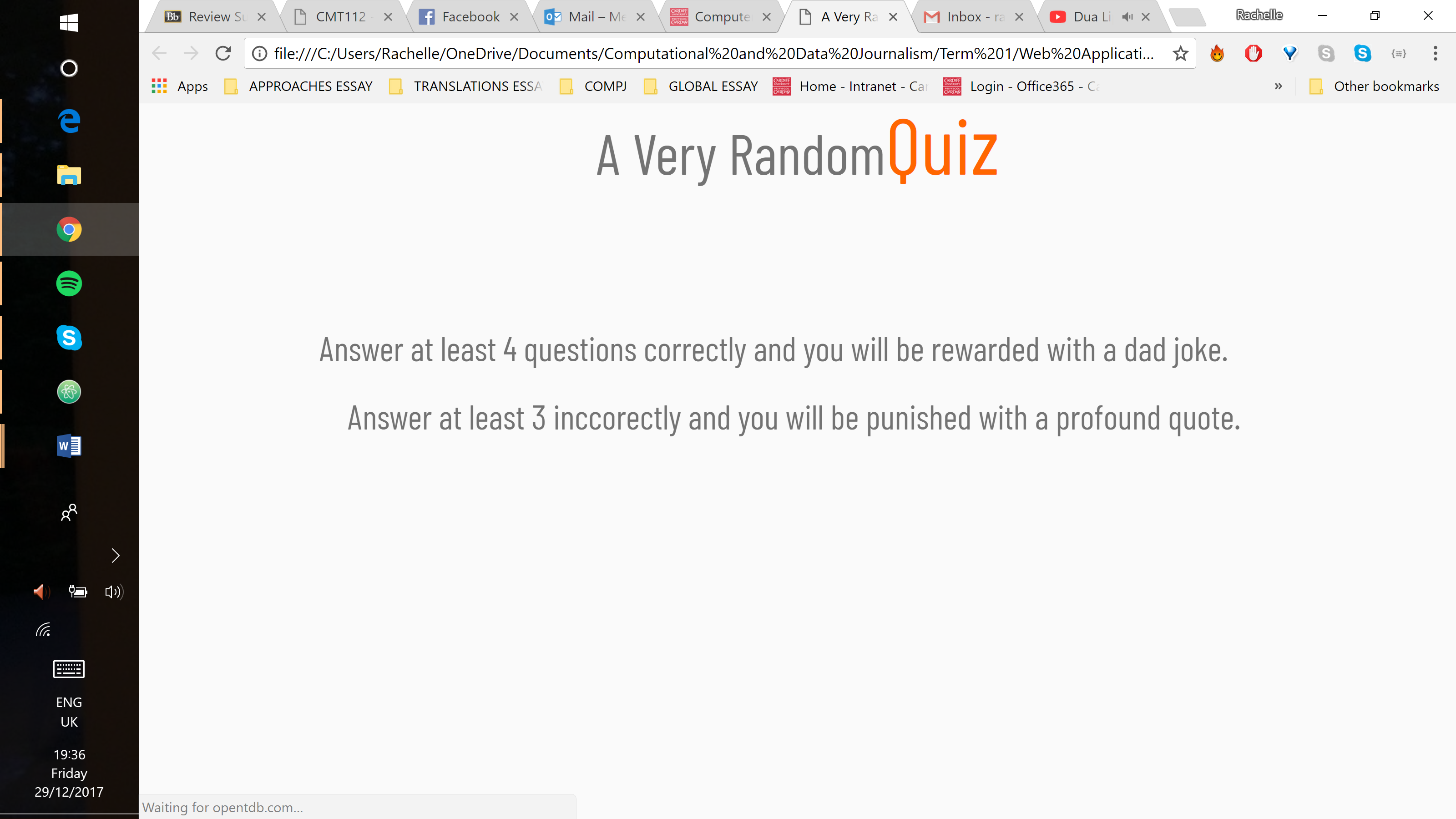
The web app displays and operates exactly as intended both on desktop and on mobile devices. Media queries were not used since the design is vertical and relatively simple. Any necessary alterations to improve the experience on mobile devices could be made in the main css without the changes impacting the desktop screen.

### Issues

Testing mobile devices in the Chrome browser proved difficult as the developer tools display blocked out half of the phone screen. Since only half of the screen was displayed, a scroll bar appeared. It is assumed that this is a browser issue and that on an actual mobile device, the full height will be visible. This issue is demonstrated in the images below.

The home screen button loads slowly. It is intentionally programmed to be hidden until all of the API data is loaded which may be the cause of the delay. Below is a screenshot before the button has appeared.



Some of the APIs require loading resources across domains. While in some situations this may be a security concern, it is not for a quiz app. Unfortunately, full resolution may require manipulation of a number of headers in the API request. Full information on the exact set and values of the headers was not found on the Internet and required experimentation to figure out.

A number of people had identical problems which helped to determine a workaround. It involved requesting a response in the JSONP format instead of JSON. This format returns JavaScript code that calls a function with the requested object. One of the parameters to the API call supplied the name of the function to call. The trick involved creating a script object in the document and setting its source to the API call URL. This loads the response from the supplied URL, which is the above-mentioned JavaScript that makes a call to a previously defined function. This method was used to obtain data from the inspirational quote API.

## Conclusion

This web app demonstrates a solid understanding of HTML and CSS as well as a growing knowledge of JavaScript. The ability to find and troubleshoot bugs in HTML and CSS is well developed. However, the ability to find and analyse bugs in JavaScript has improved, but finding a solution still poses some difficulty. Frequently, too much time was spent trying to fix a problem without the full understanding of how to do it or what to look up to understand it better. The time between understanding a problem and fixing it would be sped up with a better understanding of how JavaScript works.

References

Create A Website. *Color Psychology in Online Marketing.* Available at: <http://www.2createawebsite.com/design/color-psychology.html> [Accessed: 17 December 2017].

Design Principles. *Principles of User Interface Design.* Available at: <https://principles.design/examples/principles-of-user-interface-design> [Accessed: 19 December 2017].

Forismatic. Available at: <http://forismatic.com/en/api/> [Accessed: 17 December 2017].

GitHub. 2017. *Public APIs.* Available at: <https://github.com/toddmotto/public-apis> [Accessed: 17 December 2017].

Icanhazdadjoke. *API.* Available at: <https://icanhazdadjoke.com/api> [Accessed: 17 December 2017].

MDN web docs. 2017. *JavaScript.* Available at: <https://developer.mozilla.org/en-US/docs/Web/JavaScript> [Accessed: 19 December 2017].

Open Trivia Database*. Trivia API*. Available at: <https://opentdb.com/api_config.php> [Accessed: 17 December 2017].

# StackOverflow. 2015. [*How to make a JSONP request from Javascript without JQuery?*](https://stackoverflow.com/questions/6132796/how-to-make-a-jsonp-request-from-javascript-without-jquery)

Available at: <https://stackoverflow.com/a/28262648> [Accessed: 26 December 2017].

Takada, Mikito. *Modern web applications: an overview*. Available at: <http://singlepageappbook.com/goal.html> [Accessed: 22 December 2017].

W3 Schools. *CSS Tutorial*. Available at:  [https://www.w3schools.com/css/default.asp](%20https://www.w3schools.com/css/default.asp) [Accessed: 27 December 2017].