**Form Validation: Hands-On Project 6-1**

Rachael Herman

Colorado State University Global

ITS340: Introduction to Programming with JavaScript

Professor Alex Lazo

March 13, 2022

**Form Validation: Hands-On Project 6-1**

Form validation is an important process for web developers and programmers. While HTML and browsers can perform minimal validation, JavaScript uses client-side code to put less burden on the server during the validation process to make forms easier and more intuitive for end users (Vodnik & Gosselin, 2015). For this project, we have been tasked with creating validation functions for a form that will work in all browsers. Below is the process and challenges faced during the project.

**The Process and Challenges**

This project was different from previous weeks in that it relied more on our ability to catch and handle errors. It also involved more coding than previous assignments. After adding appropriate authorship information, a new JavaScript file was created, and the script element was added to the end of the HTML file to call on that .js file. To disable browser-based validation, novalidate = “novalidate” was added to the opening form tag in the HTML file. The first line of code for the JavaScript file was to instruct the processor to interpret the contents in strict mode with the "use strict"; line, and then create a global variable for formValidity that was set to true. I added the form elements provided in the course assignment, created the event listeners, added a validateForm() function to trigger validation of required fields upon clicking the Submit button, and added code to call the createEventListeners() function when the page finishes loading. At this point, files were saved, and the form was tested to ensure no fields could be left blank.

There were a few challenges that were relatively straightforward to correct. Multiple typos in the project code caused minor errors that were caught in the editing and review process. The biggest challenge on this first test was that the error messages were not behaving has they should; that is, none of the errors were caught. I could submit a blank form and it would return the information, displaying blank entries on screen. Using the Chrome console, I identified a syntax error in the Catch statement. It was placed inside the Try statement rather than outside. Once fixed, the form worked appropriately, as demonstrated in Figure 2 below.

The next step was to add a function to validate input elements with the number type. I came across the same problem here as with the first test in that the form was submitting and not catching errors; however, the Catch statement was appropriately placed in this part of the code. Again, I used the browser console to find errors. Several syntax errors were identified and corrected, including typos, missing elements, and extra characters. Once corrected, the form behaved as it should, with appropriate error messages shown in Figure 2 below.

**Figure 1**

JavaScript Code for Form Validation

Text

Description automatically generated

**Figure 2**

Browser Display of Error Messages

Graphical user interface, table

Description automatically generated

**Figure 3**

Browser Display of Proper Submission

Text, timeline

Description automatically generated

Overall, this project was quite challenging. After reviewing *Chapter 4: Debugging and Error Handling* in our textbook by Vodnik & Gosselin (2015), I was able to establish a solid testing environment for the code using the browser’s console. No other testing mode was necessary. The JavaScript code performs successful client-side validation to ensure fields are not left blank and the appropriate characters are used in the fields, displaying error messages when necessary.

**Reference**

Vodnik, S., & Gosselin, D. (2015). *JavaScript: The web warrior series.* (6th ed.). Cengage.