- My third artifact is from IT 270 and is an offline collection of web pages designed using HTML and CSS. It is the simplest code artifact in my portfolio, but my planned enhancements will expand upon it considerably. The website is a mock real estate site similar to real world services such as Zillow. [Launch the site]. Each page contains a few hyperlinks allowing the user to travel between pages. The home page is the only page featuring live JavaScript code, which only runs on the client. The “Chat With an Agent” dialog box at the bottom right of the home page is meant to be reminiscent of real-time customer service chats found on many modern websites. The JavaScript code for this dialog box is short and simple, allowing the user to enter text into the chat field and see it appear in the chat log as a new HTML element. Although some companies might see value in storing chat logs in the long term, the current functionality of this artifact does not permanently store the chat log since there is no dedicated site server or database, so this information disappears when the user navigates to another page.

**Structure**

- This JavaScript artifact was allowed as a last-minute touch up to the static web page developed throughout IT 270. The goal of the assignment was to experiment with JavaScript to make an HTML element interactive in some way. The assignment allowed me to take some creative liberties in implementing this code.

- This code collects some HTML elements by their CSS class name in the first three lines. The “chatInput” variable is the HTML text field allowing the user to enter text into the chat. The “chatScroll” variable is the HTML element that allows the user to scroll through messages. The “chatMessages” variable is the HTML element that contains the actual chat log, and is a child of the “chatScroll” element. The code adds an event listener to the text input element so that when the “Enter” key is pressed, the new text is added to the “chatMessages” element as an HTML “<div>” element. Additional HTML is inserted to label the sender of the message and add a separator for presentation purposes. Finally, the scroll bar is set to the maximum scroll so that the most recent chat messages are displayed when the user presses enter. Based my own goals for this interactive chat feature, the code correctly implements the design.

- Based on JavaScript guidelines from the MDN web docs, the code conforms to some standards. For example, one guideline is that JavaScript should be written with expanded syntax, meaning each statement is on its own line. Every statement in this short script is on its own line, with the longer string concatenation being split onto multiple lines for readability.

- Another coding standard is that modern JavaScript features like “let” and “const” variable declarations should be used over “var”. This script follows this rule in the variable declarations.

(6) Lastly, it is a coding standard to use strict equality operators with an extra “=” symbol to ensure that variables of different types are considered unequal. The if statement uses a strict equality and inequality symbol to test string literals against string variables.

(7) The code is consistent in style, although the long concatenated sting starting at line 7 is somewhat confusing and could probably be organized better. Instead of using strings to insert HTML elements, the code could be improved by instantiating HTML elements using the document.createElement() function.

- There is no code or procedure that is unreachable, and no leftover test routines or function stubs.

- The code does not have any simple replacement for functionality from JavaScript libraries.

(2 & 3 & 4) The HTML elements that are fetched by class name and indexed by the 0th element are repetitive and could be replaced with a function taking a string as the CSS class name.

- The storage use is efficient, although a safeguard against excess RAM usage would be to delete old messages from the HTML element list once there are more than 100 messages.

- Some string literals are used in this program, like the “Enter” literal, which is not problematic since it is only used once. Like I said before though, the large HTML string is messy and could be created using the document.createElement() function instead.

- There are no parts of this code complex enough to require being broken down into subroutines.

**Documentation**

- The code is not documented with any comments detailing its functionality. In enhancing this artifact, I should comment the code thoroughly.

**Variables**

- All the variables have meaningful and consistent names. “chatInput” for example makes it clear to the reader which HTML element is being referred to according to the CSS class name.

- All the variables are assumed to be one type and are not reused with different types, although JavaScript is a dynamically typed language.

- There are no redundant or unused variables in this script.

**Arithmetic Operations**

- The code avoids all arithmetic operations and does not do any math yet, which means there are no floating point comparisons or rounding errors, and no possibility of dividing by zero.

**Loops and Branches**

(6) The only branch in this program occurs at line 6 to check whether the key pressed is Enter, as well as whether the text input field is empty or not. An else branch is not required but could be added to handle the case where the text field is empty or a different key is pressed.

- There are no switch statements or loops to examine in this program.

**Defensive Programming**

(2 & 3 & 4) Acquiring HTML elements by CSS class name is useful, but it is important to know that there could potentially be more than one element with the given class name. In this code, I assume that there is one and only one of each element with the given class name, and index the 0th element of the returned array. This does not take into account the possibility of multiple elements, or no elements, in which case an error will occur when trying to access the 0th element. Code should be included to automatically check the size of the array before accessing elements.

- The “chatInput” field is the main input for this program. The user may enter whatever text they want into the chat log. It would be wise to add a validation stage to this input to prevent overly lengthy messages and limit the range of ascii characters allowed in messages.

- Memory allocations due to HTML insertion are not well managed and could cause excess RAM usage if not kept in check.

**Enhancements and Course Outcomes**

- In addition to fixing the above code weaknesses, limitations, and vulnerabilities, I will implement new features into this program to further enhance it. So far, this artifact only includes a client-side script. Messages in the chat log are not saved, and it is likely that a company would want to save customer chat logs for their record. To enhance this code, I will create a Node.js and SQL local server that acts as the backend of this real estate website. The user will still be able to enter messages into the chat log, but those messages will be sent to a remote SQL server using Node.js along with date, time, and IP address information for long term storage. This artifact enhancement will target the CS-499-5 course outcome by addressing existing design flaws in the code and removing memory leaks in the client chat log. This outcome will also be fulfilled in the server-side code where data will be validated to prevent SQL injections and limit security vulnerabilities.