Homework 6

ECE567: Software Engineering 1

Tanya Sharma
tds104
2 December 2023

Commands

In order to execute the both the program as well as test cases the following instructions can be followed:

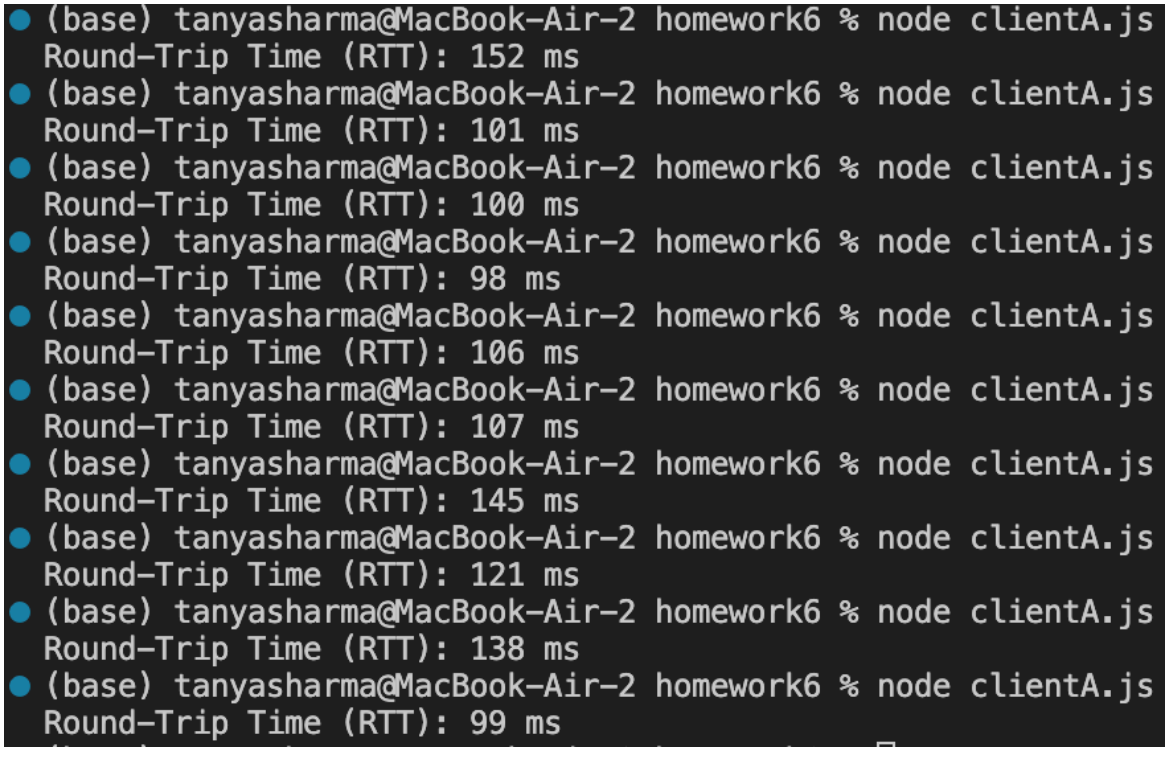
1. **npm install**
Install all necessary dependencies

2. **node clientA.js / node clientB.js**
This command will execute the main function where all our logic takes place

3. **jasmine**
This command will execute the test suite

Design - Part A

This project is a JavaScript module designed for pinging a server and measuring the Round-Trip Time (RTT) of a request. The ping function encapsulates this functionality by accepting a server URL as a parameter, validating the URL using the validateURL function, and then utilizing the Fetch API to send a request to the server. The RTT is calculated by measuring the time it takes to send the request and receive a response. The result, whether it's the RTT in milliseconds or a failure message, is wrapped in a Promise for asynchronous handling. The validateURL function ensures that the input URL adheres to a valid format using a regular expression. We demonstrate the use of this module by pinging the server at 'https://www.rutgers.edu/' and logging the RTT or a failure message based on the outcome.
Average RTT Time (after pinging the URL 10 times): **116.7 ms**



Design - Part B

This project establishes a simulated asynchronous interaction with a server using the Fetch API. It initiates by defining a server URL, along with variables for text content and a delay. Subsequently, a setTimeout function asynchronously updates the text variable after a specified delay. The code then configures a Request object for a POST request to the server, incorporating headers and the current text value. Timing measurements are taken before and after sending the request, calculating the Round-Trip Time (RTT). The Fetch API is employed to send the request to the server. Upon a successful response, the code compares the received text with the original value, logs the result, and prints the response time to the console. Any HTTP errors or exceptions during this process are also logged. In essence, this script models a scenario where data is asynchronously updated and sent to a server, and the subsequent response is analyzed, with a focus on tracking response time and ensuring data consistency. It serves as a simplified representation of asynchronous communication and error handling in JavaScript.

Observations - Part B

The following observations are drawn from testing the client and server in Part B, in different scenarios:

1. Server returns the message before client delay has finished

Client Delay	Server Delay	Text Match	RTT
100	10	TRUE	22
100	50	TRUE	77
500	100	TRUE	129
500	250	TRUE	287
500	400	TRUE	429
1000	250	TRUE	286
1000	500	TRUE	536
1000	750	TRUE	783
2000	250	TRUE	279
2000	500	TRUE	533
2000	750	TRUE	784
2000	1000	TRUE	1027

2000	1250	TRUE	1285
2000	1500	TRUE	1538
2000	1750	TRUE	1784

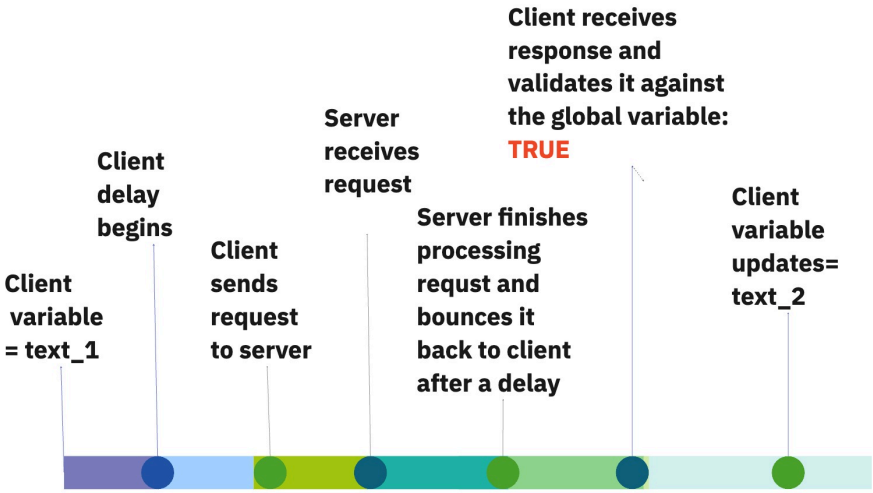
2. Server returns the message after the client delay has finished

Client Delay	Server Delay	Text Match	RTT
10	100	FALSE	126
50	100	FALSE	124
100	500	FALSE	128
250	500	FALSE	537
400	500	FALSE	540
250	1000	FALSE	1026
500	1000	FALSE	1021
750	1000	FALSE	1022
250	2000	FALSE	2042
500	2000	FALSE	2025
750	2000	FALSE	2035
1000	2000	FALSE	2026
1250	2000	FALSE	2030
1500	2000	FALSE	2030
1750	2000	FALSE	2032

Explanation

1. Server returns the message before client delay has finished

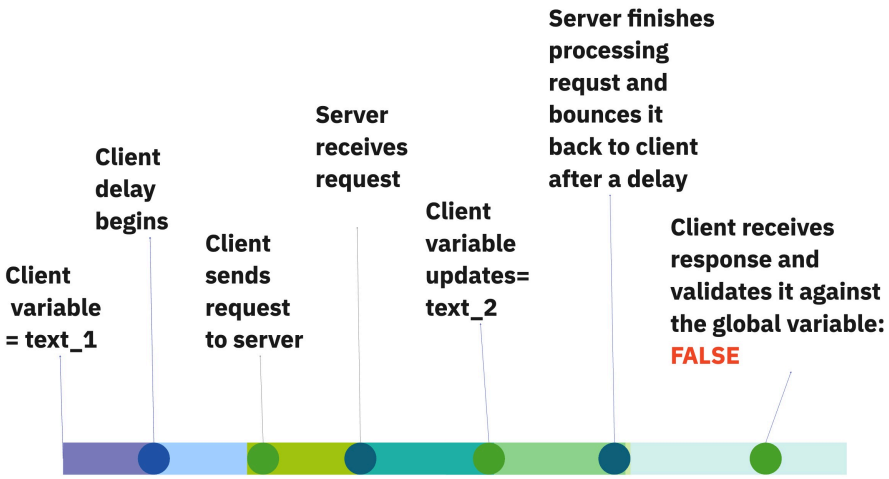
We start out with setting our global variable to “text_1” and creating an asynchronous setTimeout() function to update its value to “text_2”. In this case, the client delay is greater than delay at the server and the message is bounced back from our server before the variable has updated to its new value of “text_2”. This is why we always get a “TRUE” message because the value of the global variable has not been updated by the time we check response from the server against our variable. Both texts match in this case. The following simple diagram illustrates how this occurs:



Events:
1. Client begins delay
2. Client sends request with 'text_1'.
3. Server receives the request and starts the RTT delay.
4. RTT delay at the server completes, and the server processes the request.
5. Server sends back the response after the response delay.
6. Client receives the response.
7. Client compares the response with the original global variable ('text_1').
8.Meanwhile, at the client, the global variable is asynchronously updated with 'text_22' after the async update delay, after the response has already arrived
9. Since the delays were asynchronous, the comparison results in TRUE.

3. Server returns the message after the client delay has finished

We start out with setting our global variable to “text_1” and creating an asynchronous setTimeout() function to update its value to “text_2”. In this case, the server delay is greater than delay at the client and the message is bounced back from our server after the variable has updated to its new value of “text_2”. This is why we always get a “FALSE” message because the value of the global variable has already been updated by the time we check response from the server against our variable. Both texts do not match in this case. The following simple diagram illustrates how this occurs:



Events:
1. Client begins delay
2. Client sends request with 'text_1'.
3. Server receives the request and starts the RTT delay.
4. Meanwhile, at the client, the global variable is asynchronously updated with 'text_22' after the async update delay, before the response arrives.
5. RTT delay at the server completes, and the server processes the request.
6. Server sends back the response after the response delay.
7. Client receives the response.
8. Client compares the response with the updated global variable ('text_22').
9. Since the delays were asynchronous, the comparison results in FALSE.