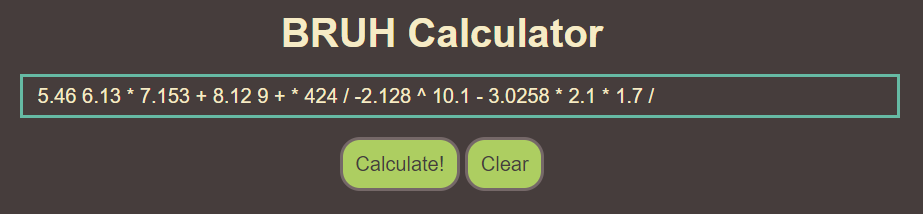
Criteria C – Computational Thinking

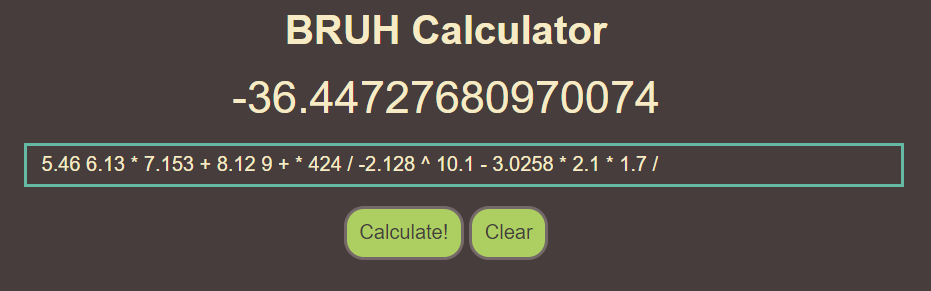
**Product Evaluation**

Audience evaluation and feedback is given in the next section.

The web application developed as a solution to the task presented was the culmination of meeting multiple success criteria throughout the planning and development phase. The product can be considered successful if it adheres to the set successful criteria.

Given a formula that is passed through the frontend input field, the Reverse Polish Notation (RPN) calculator correctly evaluates it and pushes the value to the frontend through the Vue instance. An example of this is listed below. Thus, the product fulfills Success Criteria #1 and #8, given that it has performed consistently throughout the test plan and development phases.





As complex formulas with multiple operands and operators are the key utility point of most calculators, the inclusion of these within a RPN format is essential, fulfilling Success Criteria #2 and #4. Through the feedback given by the audience, it can be deemed that the calculator performs accurately with complex RPN formulas and supports a range of multiple numerical data types, on all modern mobile and desktop browser devices. However, while the user experience on smaller screen mobile devices was not part of the success criteria, it was noted that improvements to the UI such as larger buttons optimized for a smaller screen would enhance UX. In the decision to utilize Vue, a JavaScript framework, to assist in the web application’s development, excessive and unnecessary boilerplate could be avoided for user feedback loops (hotkeys, visual feedback, etc. Success Criteria #3). As a result of this choice made in the early stages of progress, as well as favouring the switch statement instead of an if-else tree (due to the former’s leverage of jump tables), the full-stack codebase of the project is future-proof (Success Criteria #10) and performant (Success Criteria #6) while also enabling out-of-the-box support for all modern web browsers (Success Criteria #7).

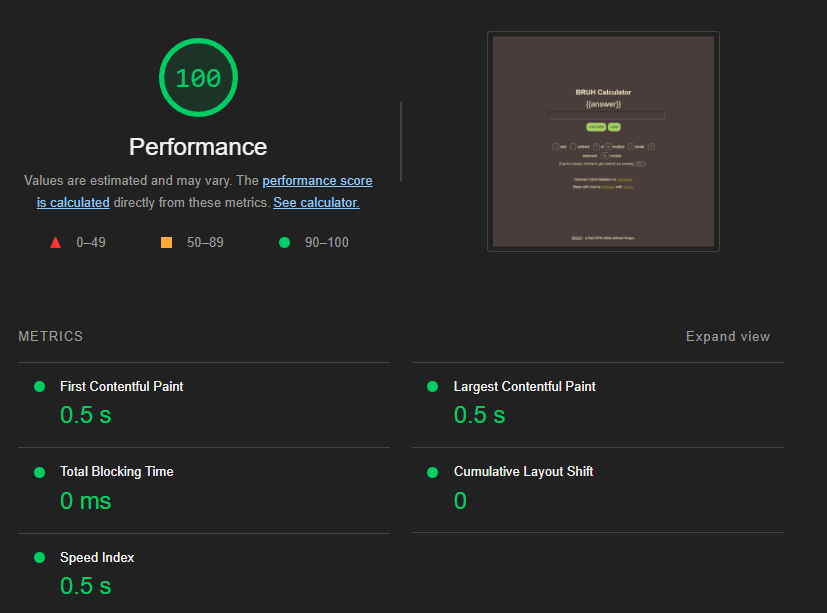
In the event of an invalid formula, the program gracefully recovers and informs the user of an error with the formula with a prompt, satisfying Success Criteria #5. This allows the user to decide whether to pursue further details, and if so, are encouraged to open the console to understand the intricacies behind the calculation and the step that resulted in the error. The current error messages were not received overwhelmingly positively by the two parties/audience during their evaluation period with the product. Common complaints from the audience include not being able to specifically distinguish between the different errors, and the addition of further filters to handle edge cases without the additional step of opening the console with a keybind they were not familiar with would have been more appreciated.

Graphical user interface, text, application, chat or text message

Description automatically generated

By planning the product’s development process beforehand, the stack data structure’s LIFO attributes and constant time complexity were identified at an early stage to be implemented in accordance with Success Criteria #6 (once more) and #9. Using Google’s Lighthouse utility to gauge and quantify the performance exhibited by the deployed version of the product hosted on Firebase, it can be observed that it performs exceptionally well on both desktop and mobile.

Desktop:



Mobile:

A screenshot of a computer

Description automatically generated with medium confidence

**Audience Evaluation**

It is critical that parties and participants that are unaffiliated with the product are able to test the solution and provide feedback for the developer to gain an outside perspective. Two parties were involved in the feedback process, family members and a university student majoring in Computer Science and Information Technology overseas who asks to remain unnamed.

|  |  |  |  |
| --- | --- | --- | --- |
| Success Criteria | Met/Partial/Not Met | Feedback (Party 1) | Feedback (Party 2) |
| Provide accurate calculations RPN. | Met | All participants in Party 1 validated that no calculation errors occurred throughout the 30 minutes they constantly attempted to break the solution. The accuracy of the calculator was confirmed through a traditional calculator app on the members’ mobile devices. | Party 2 began by attempting to break the calculator with divisions by zero. They noted that zero to the power of zero should result in undefined but said that one was also a valid (although somewhat trivial) answer. |
| Complex formulas should be supported. | Met | Participants began with inputting a formula with adjoining operators (6 7 8 + -). With most implementations, the RPN parser would calculate 6 + 7 before performing 13 - 8. However, due to the LIFO nature of the stack, the correct order of operations was achieved. | Participants in Party 2 continued to attempt to break the calculator. Exact formulas used are unknown. However, it was concluded that the calculator performed ‘well, consistently, and without errors’. |
| Allow users to enter RPN expressions in an easy-to-use interface. | Partial | Participants discovered the ‘evaluate’ hotkey quickly, this being the enter key. However, the ‘clear’ shortcut was not found at all within Party 1, being control+backspace. After being informed of this hotkey, participants agreed that there should be information regarding the user of this on the site. | Party 2 noted the somewhat Material design principles utilized throughout the interface. They found the ‘clear’ shortcut only after approximately 10 minutes into using the application. |
| Support a fundamental range of mathematical operators. | Met | All participants confirmed that all mathematical operators yielded the correct result. | Party 2’s participants noted that there should be further support for trigonometric functions, factorials, and other features normally only found on scientific calculators. |
| Provide user-friendly feedback. | Partial | While the user-readable messages prompting the participants to check the console was helpful to some, the majority who were on mobile suggested that the error should just be shown, and the extra step opening the console is an unnecessary segregation of end-user and developer experiences. | Immediate feedback on website errors stated that the console should only be for step-by-step calculations, and errors should be directly shown in the frontend. |
| The utility should be fast and responsive for users. | Met | Load times and page performance was greatly appreciated throughout the group of participants, most noting that calculations were almost instantaneous. | Also using Google Lighthouse, participants in Party 2 returned almost identical results as shown above. |
| The calculator should work on different web browsers and operating systems without any issues. | Met | Some participants were instructed to utilize Safari on mobile and on MacOS, while others utilized Chromium (Blue), Chrome, Brave, Opera, Firefox, and Microsoft Edge. No problems were reported, however, users on smaller devices reported that better mobile UI would be appreciated to improve navigation. | Party 2’s devices consisted of an Android device and Windows Personal Computer. Party 2’s participants reported no issues with the UI on desktop but suggested resizing the page for mobile devices. |
| The calculator must show its answers on the screen. | Met | All participants validated that the correct answer would appear on the webpage after calculations were finished. | Party 2’s participants suggested that while the current answer visuals were functional, aesthetic improvements such as animations could be made. |
| The RPN must utilize stacks. | Met | While only a handful of participants in Party 1 understood the backend, the decision to utilize stacks (and, by extension, the Last In First Out principle) instead of arrays was explained in simple terms. | The usage of stacks within the code was validated, as JavaScript’s compiler changes the data structure at times depending on the operations being applied to it. |
| The tool’s codebase must be future-proof and expandable. | Met | Again, the usage of Vue and switch statements instead of if-else blocks was explained in simple terms and understood by the majority of the participants in Party 1. | Members of Party 2 noted the utilization of switch statements throughout the code for performance improvements and potential for expandability. |

**Suggestions for Improvement**

As suggested by Party 2, the calculator can be further improved by adding support for advanced mathematical functions such as trigonometric functions, factorials, absolute values, rounding, and square roots. This will enable users to perform more complex calculations and expand the usefulness of the calculator for scientific and engineering applications. Adding support for these functions will require updating the calculation process, input validation, and the user interface to ensure a seamless user experience. In aiming to parallel current online scientific calculators (such as the one created by Desmos located [here](https://desmos.com/scientific)) and offer a full experience, the updating of backend to support functions and other standard utilities would be beneficial to end users.

Additionally, improving error handling mechanisms in the RPN calculator will help users better understand and correct their input errors. Also suggested by both parties, instead of directing the user to open the console to obtain detailed information about the error, the problem should be displayed on the main interface, potentially also with a UI element that illustrates which step the calculation failed at. The existing error filters should also be scrutinized in detail during the revamping and improvement process of the product, as new users looking to optimize their calculation workflow with a RPN calculator would most likely not be familiar with the intricacies, quirks, and syntax of reverse polish notation. For example, prompts should be given when a user inputs the formula 2 – 4 or 2 4 5 ++, dynamically adjusting error messages to offer tips, suggesting the user to input 2 4 - and 2 4 5 + + respectively.

The ability to parse user-defined functions and variables can help users define their own custom calculations and store frequently used values would also prove to be beneficial to exponentially improving the calculator’s functionality. This feature will enable users to create their own functions or constants and store them for future use. The calculator can store these functions and variables in a separate memory stack or use local storage to save them for future sessions. Calculation history being available in a separate panel will also increase user experience and long-term satisfaction.

While the addition of a separate parser able to evaluate infix notation would allow for the calculator to increase the prospect of potential users, this would subtract from the main goal of the solution, and implementation of this would go against the project’s philosophy.

Functional aspects of the calculator aside, aesthetic improvements such as the ability to set custom themes and more animations throughout the interface will make the calculator enjoyable to use. User guides in another panel would also be a worthy addition to the revised version of the RPN calculator to inform users about keyboard shortcuts, functions, and required syntax.