VisuSpeak

Code Quality Review

Archisha Bhattacharya
Brooklyn Coulson
Jasmeet Singh

Table of Contents

1.0 Introduction
The VisuSpeak application uses a variety of techniques that allows us to efficiently run our application. In this report we will outline the practices and principles that we used that allowed us to maintain a good quality report.

2.0 Design Principles
2.1 DRY (Don't Repeat Yourself)
Throughout our development process, we have applied the design principle DRY to eliminate redundancy and strive to reuse our code wherever possible. By adhering to the DRY design principle, we aimed to ensure that each component or logic within our system has a single representation. Our codebase reflects this methodology in our functions and components. This approach has allowed us to minimize code duplication and enhance the scalability of our application on the development side.

2.2 KISS (Keep It Simple, Stupid)
Since the beginning of our development process we have prioritized simplicity in our code structure and in its implementation, hence we decided to follow the KISS principle. We firmly believe that solutions and code should prioritize simplicity without sacrificing effectiveness or requirements. The KISS principle has not only helped us maintain a clear and concise code structure but has also fostered a plan of clarity and efficiency within our development team.

3.0 Best Practices
3.1 Code Structure and Readability
For our code structure and readability we used various different methods such as:
- Modularization: The code is well-organized into modular components and folders.
- Clear Naming Conventions: We implemented meaningful and descriptive variable names, function names, and component names. This makes it easier for the developer to understand the code structure.
- Comments and Documentation: We added comments that provide sufficient context and explain complex logic. This allows developers to understand a complex functionality in a simpler way and also fosters team collaboration.
- Separation of Concerns: We have two folders in our repository: one for testing, where changes are made and tested, and another for production, where the finalized version of the code or features are pushed. This approach ensured a minimal and highly readable code structure.

3.2 Code Quality
In terms of code quality, we prioritized clean code, and we were able to achieve this goal in the following ways:
- Coding Standards: We established coding standards within our team to prioritize readability and consistency.
- Component-Based Architecture in React: With the latest version of React, functional components are preferred over class components. This method has contributed for us as a team to maintain consistent and top-notch code quality.
- Scalability: The code architecture is designed with scalability and extensibility in mind, allowing for future modifications or new additions without the need for significant refactoring of major code components.

3.3 Error Handling
Error handling was a critical aspect of our development process which included both backend validation and user validation. We achieve it by the following:
- Back-end Validation: For backend validation, we handled errors through console logging, catching errors in the console log to monitor the behavior of our APIs and databases.

- Error Message for Users: Clear and concise error messages were implemented on the frontend to inform users when they entered incorrect information or if requirements were not met to proceed with a certain task.
- Permission Management: We integrated permission on our user interface, allowing users to grant access to features such as the camera and microphone for a seamless user experience.

3.4 Performance

In order to achieve good performance metrics we applied the following techniques:

- Minimal Library Imports: In our project structure we only imported the libraries and dependencies that were essential to run the application. By avoiding unnecessary imports, we reduced the overall size of the code which improved our application rendering and load times.
- Resources Management: On our application we compressed scripts, images, and videos used in the application while maintaining high quality. Additionally, videos were uploaded to YouTube and called through a script, eliminating the need to store them in a database. This approach ensured efficiency and prevented an increase in load times of the application .

3.5 Security

In order to achieve good security measures we applied the following techniques:

- HTTPS Deployment: The VisuSpeak website is deployed using HTTPS (Hypertext Transfer Protocol Secure), ensuring secure communication between the client and the server.
- No Local Storage of User Data: The application does not store any user data locally, end to end encryption is applied ensuring that all the data such as video capture for gesture recognition remains on the client's machine.
- Password Encryption: User passwords are securely encrypted, and they are also encrypted in the database. This means that even developers cannot view the actual passwords stored in the database.

4.0 Conclusion and Improvements

As a team , we aimed to elevate our code quality to industry standards. While we acknowledge that we did not fully implement these standards in areas due to limited expertise, we applied the principles and techniques learned throughout our academic journey. However, we recognize that there is always room for improvement. For example, refactoring the code could have been executed more efficiently from the beginning, and reducing redundant function calls would also increase the understanding of the complex logic that we have implemented. Therefore, improving on these techniques such as better coding standards, optimizing better function calls and better refactoring are in the future works of the application.