Code Testing Report

Archisha Bhattacharya
Brooklyn Coulson
Jasmeet Singh

Table of Contents

Introduction

The VisuSpeak application has three integral components: Front-end, Back-end, and AI Model. React for the front-end interface, Firebase Database for back-end data management, and MediaPipe Machine Learning framework converted to TensorFlow.js for AI Model. This report presents an analysis of the testing procedures conducted across these components to ensure the application's reliability, performance, and user satisfaction.

Front-end Testing

In developing the front-end for our application, React was the preferred framework, as it provides a wide range of libraries and its robust architecture for modern web development practices. To validate the functionality and performance of the front-end components, we used Jest, a widely-used testing library for JavaScript. Jest provided a comprehensive suite of testing utilities.

Our testing strategy was primarily based on component testing and regression testing. Component testing involved testing individual components to ensure they functioned as expected and rendered all the essential content. This approach allowed us to test each component as can be seen in Figure 1, facilitating easier debugging and maintenance. Additionally, regression testing was used to detect any unintended consequences on previously functioning components resulting from recent code modifications. By systematically retesting existing features after each code change, we aimed to maintain the stability of the front-end architecture throughout the development process.

```
describe('AboutSections component', () => {
    // Test suite to ensure all videos are visible
  test('render all video sections', () => {
    const { getAllByTitle } = render(<AboutSections />);
    const videoPlayers = getAllByTitle(/YouTube video player/);
    expect(videoPlayers.length).toBe(3);
  });
```

Figure 1. Component Testing Example

We used an Excel spreadsheet as a tracking tool (Figure 2). This spreadsheet served as a central file to document the status of test suites, tracking pass/fail outcomes, and logging the components subjected to testing. For clarity, individual test files were created to document the results of each specific test case as shown in Figure 3. This approach to test documentation allowed for better traceability and ensured that test outcomes were easily accessible and understanding.

Test Cases	Pass	Fail
AboutFeatures - render all features	X	
AboutSections - render all video sections	X	
AboutSections - render project idea description	X	
AboutSections - render project background description	X	
AboutSections - render about VisuSpeak description	X	
AboutSocial - render Learn More section	X	
AboutSocial - render Learn More description	X	
AboutSocial - render GitHub link	X	
AboutSocial - render YouTube link	X	

Figure 2. Excel Sheet Tracking Example

Test Log: Component Testing of AdminChat Module

Date: March 15, 2024

Test Case: Rendering of Child Components and Functionality

Test Description:
The objective of this test is to verify that the AdminChat module renders all essential child components correctly and exhibits expected functionality, including alert display, dragging functionality, and unmount behaviour.

Figure 3. Log Testing Example

To test the application against potential errors , we implemented error handling tests. Using React's refreshing capabilities, we intentionally triggered console error exceptions to validate the application's responsiveness and stability

in our development process. This approach helped identify and address potential vulnerabilities, thereby enhancing the overall reliability of the front-end.

AI-Model Testing

In our AI-model code testing, our primary objective was to assess the performance of the algorithm implemented within the draw_landmarks function, as we had to deploy the model on a web application. This evaluation focused specifically on measuring the execution time and resource usage, including CPU and memory, under multiple conditions. First, we measured execution time by recording the duration from before the draw_landmarks function call to after using the time.time() function. In addition, we monitored resource usage using the “psutil” library, to track the memory and cpu usage.

Our testing process began by creating a random image sized at 640x480 pixels using the generate_random_image function. Along with, random landmark points were generated to simulate input data for the draw_landmarks function. Finally, we conducted performance evaluation by using the draw_landmarks function twice. Initially, we executed the function to establish baseline performance metrics. Following this, we repeated the function call to evaluate the consistency of performance metrics under different conditions.

This testing approach provides a systematic framework for assessing the performance of the hand gesture recognition algorithm as can be seen in Figure 4. By measuring execution time and resource usage, we gained valuable insights into the efficiency and scalability of the algorithm, as we had to deploy it on a web application.

```
python performance_evaluation.py
Execution Time: 0.002064943313598633
CPU Usage Before: 0.0
Memory Usage Before: 5038530560
CPU Usage After: 0.0
Memory Usage After: 5038530560
```

Figure 4. AI Model Performance Testing Results

Back-end Testing

For the back-end development, we maintained consistency with the front-end approach by utilizing similar tools and methodologies. Firebase Database served as the backbone for data management, offering reliability and scalability crucial for handling user information effectively.

Just as in front-end testing, Jest was used as our testing library for JavaScript files. We employed it to conduct thorough testing of back-end APIs, ensuring their functionality and reliability. The testing process involved validating API endpoints, input data, error handling, and responses to guarantee efficient communication between the front-end and back-end components. Similar to the error handling tests implemented in the front-end, we used console logging to monitor API requests and responses, as well as to detect and address any potential errors or exceptions. This approach enabled us to identify and resolve issues promptly, contributing to the overall stability and performance at the back-end operations.