Hands UP

Testing Policy

updated: 28/09/2025

Contents

1 Introduction

This document outlines the **Testing Policy** of HandsUp. The purpose of this policy is to define the structure of our testing procedures to ensure that the application is thoroughly validated and works as intended.

Our testing strategy includes the following:

- Unit Testing (Backend and Frontend) with Jest
- Integration Testing with Cypress
- Performance Testing with Lighthouse
- User Testing for usability and feedback

The following sections describe the tools, procedures, and standards we applied to achieve reliable, high-quality results.

2 Unit Testing (Backend and Frontend)

2.1 Objective

The objective of unit testing is to validate individual components in isolation, ensuring that each small, independent part of the system functions correctly.

2.2 Automation

All unit tests are automated and seamlessly integrated into the **GitHub Actions pipeline**, ensuring that every change is validated before being merged into the development branch.

2.3 Framework

We use **Jest** to implement automated unit testing across both backend and frontend components.

Figure 1: Example of GitHub Actions tests passing

3 Integration Testing

3.1 Objective

Integration tests validate the interactions between different system components, ensuring they work together as expected.

3.2 Automation

All integration tests are automated and integrated into the **GitHub Actions pipeline**, guaranteeing that all parts of the system are functioning correctly before merging code.

3.3 Framework

We use **Cypress** to conduct integration tests, validating the user interface and user interactions.

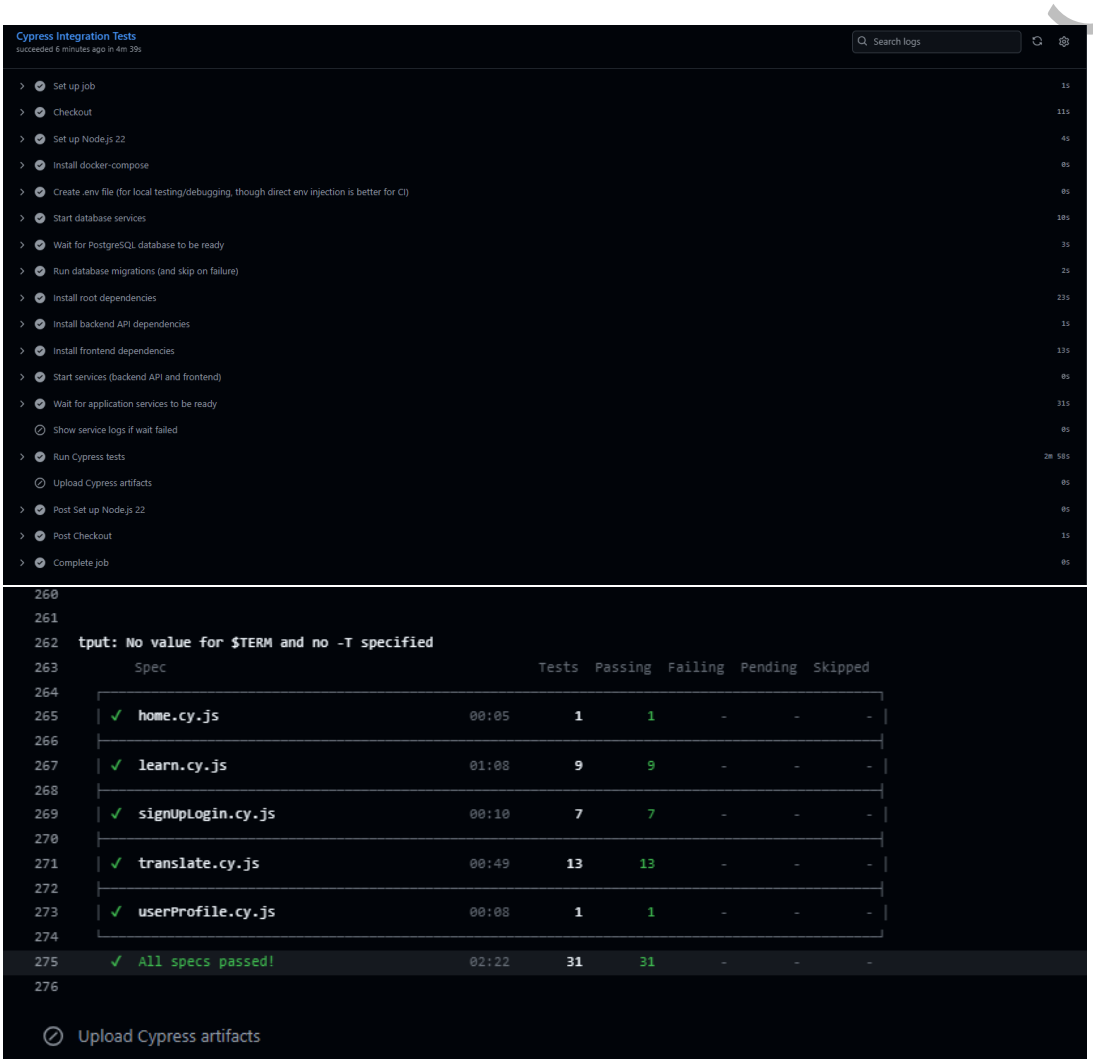

Figure 2: Example of GitHub Actions tests passing

4 Quality Assurance

For HandsUp, **quality assurance** is critical, as user experience and performance directly influence how the application is perceived. To maintain a high standard, we placed strong emphasis on **performance testing** and **usability testing**.

4.1 Performance Testing

4.1.1 Objective

Performance testing evaluates the speed, responsiveness, and stability of the application under various conditions, ensuring that it can handle expected loads efficiently.

4.1.2 Simulation Environment

To simulate real-world usage, we throttled CPU and network conditions to approximate a mid-tier device on a 3G/4G connection. This ensures results reflect typical user experiences rather than high-end developer machines.

4.1.3 Metrics

We used Lighthouse to measure the following key performance indicators:

- First Contentful Paint (FCP)
- Largest Contentful Paint (LCP)
- Speed Index (SI)
- Time to Interactive (TTI)
- Total Blocking Time (TBT)
- Cumulative Layout Shift (CLS)

4.1.4 Metric Weighting

Each metric contributes differently to the overall Lighthouse performance score:

- LCP: ∼25%
- TBT: ∼30%
- CLS: ∼15%
- FCP: ∼10%
- SI: ∼10%
- TTI: ∼10%

4.1.5 Result Scoring

Results are normalized on a scale from 0–100 using a log-normal curve. This prevents small delays from being overly penalized while still highlighting poor performance.

For HandsUp, we aimed for a performance score between **70–100**, balancing graphical features, "wow factors," and the translator page.

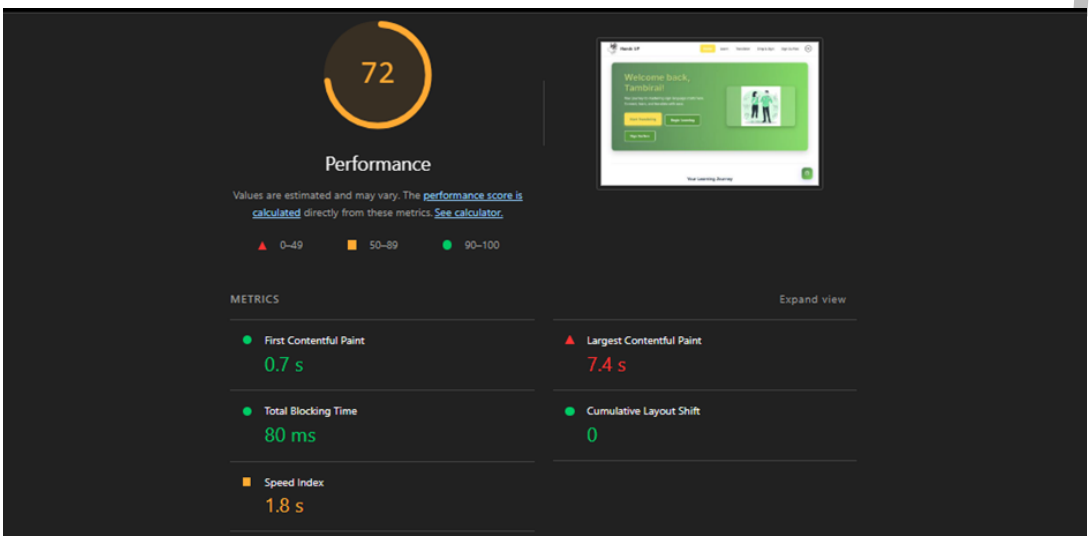

Figure 3: Lighthouse performance report example

4.2 Availability

4.2.1 Objective

Ensure that HandsUp remains up and responsive under normal and peak user load, without unexpected downtime.

4.2.2 Simulation Environment

We use **Uptime Kuma** to continuously monitor the availability of our site. The HandsUp application endpoint is added as a monitored target within Uptime Kuma. From there:

- Uptime Kuma sends requests to the application at regular intervals (configurable, typically every 60 seconds).
- If the application fails to respond within the timeout threshold, the system records a downtime event.
- Monitoring is performed externally, simulating a real user trying to reach the app.

4.2.3 Metrics

Our target uptime is **99%** during non-deployment periods. Uptime Kuma helps us achieve this by:

- Tracking response successes/failures and compiling uptime percentages.
- Recording the response time (latency) of each check, allowing us to see performance degradation before full outages occur.
- Maintaining historical logs of incidents and recovery times.

- Providing real-time notifications (e.g., via email, Discord, or Slack) when downtime is detected so issues can be addressed quickly.

This provides not only raw uptime statistics but also actionable insight into **when**, **why**, and **how long** the system may have been unavailable.

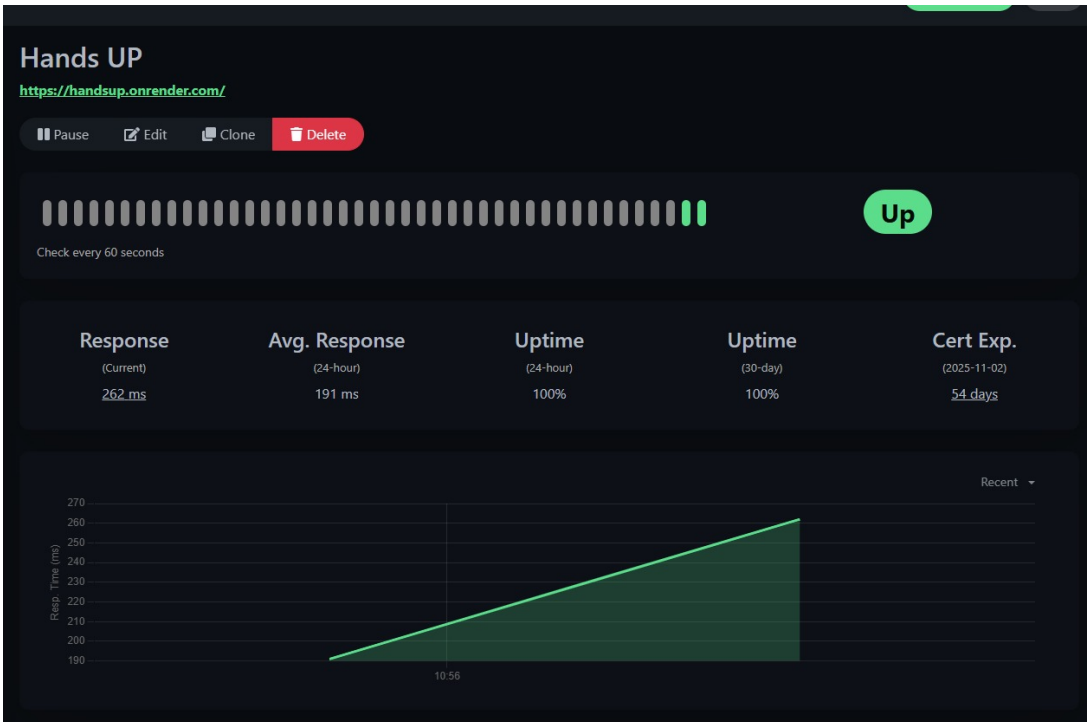

Figure 4: Example of Uptime Kuma monitoring dashboard

4.3 Usability Testing

4.3.1 Objective

Usability testing ensures that HandsUp is **intuitive, clear, and accessible** to all users. The main goal is to collect feedback from people outside the project team to guide improvements.

4.3.2 Methodology

- **Pre-Planing**: We created a Google Form for participants to provide real-time feedback while interacting with the app.
- **Implementation:** Team members engaged with different user groups to reduce bias. Participants filled out the form, and responses were reviewed collectively.
- **Follow-Up:** Based on the feedback, we refined or discussed improvements for specific features of the app.

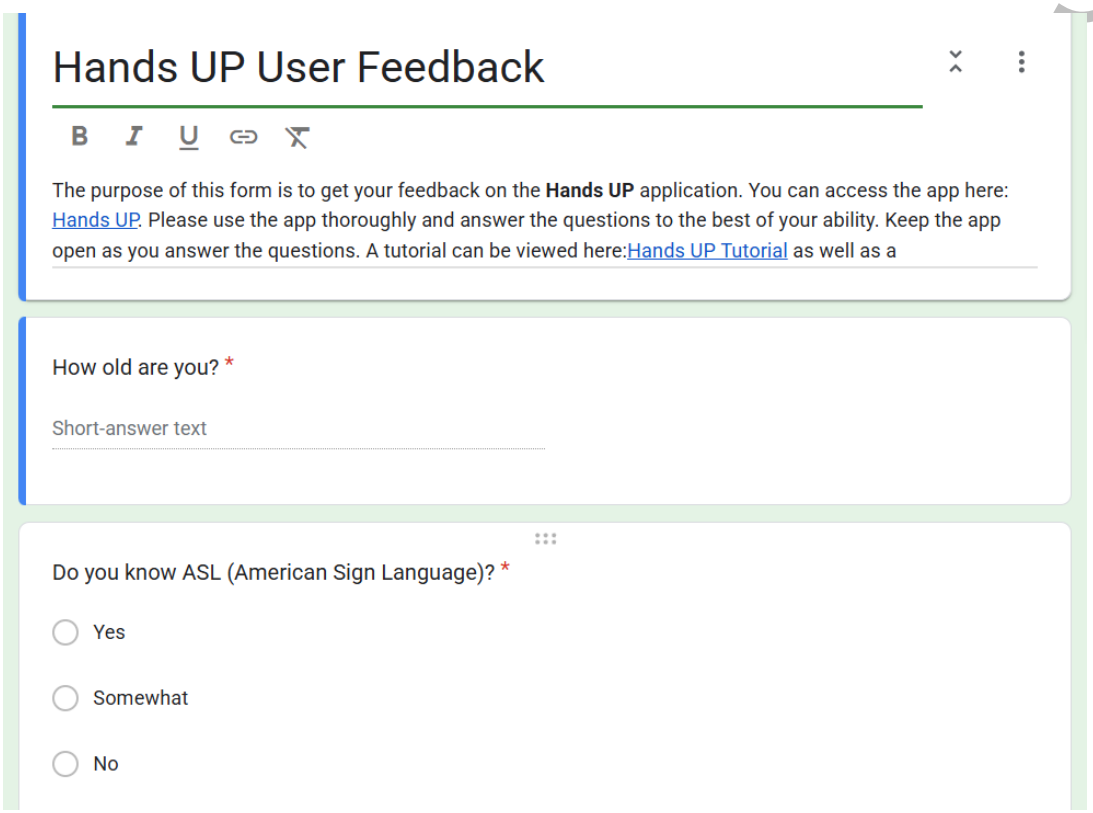

Figure 5: Example of usability testing feedback collection

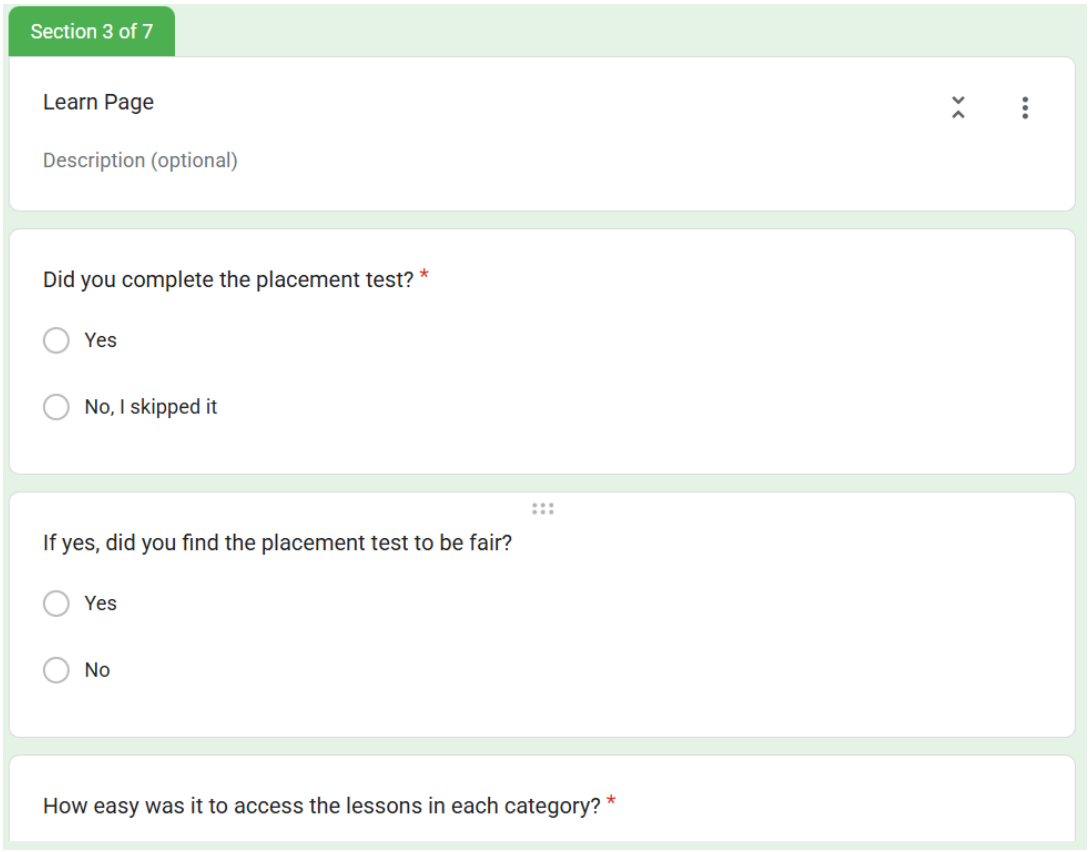

Figure 6: Example of usability testing feedback collection

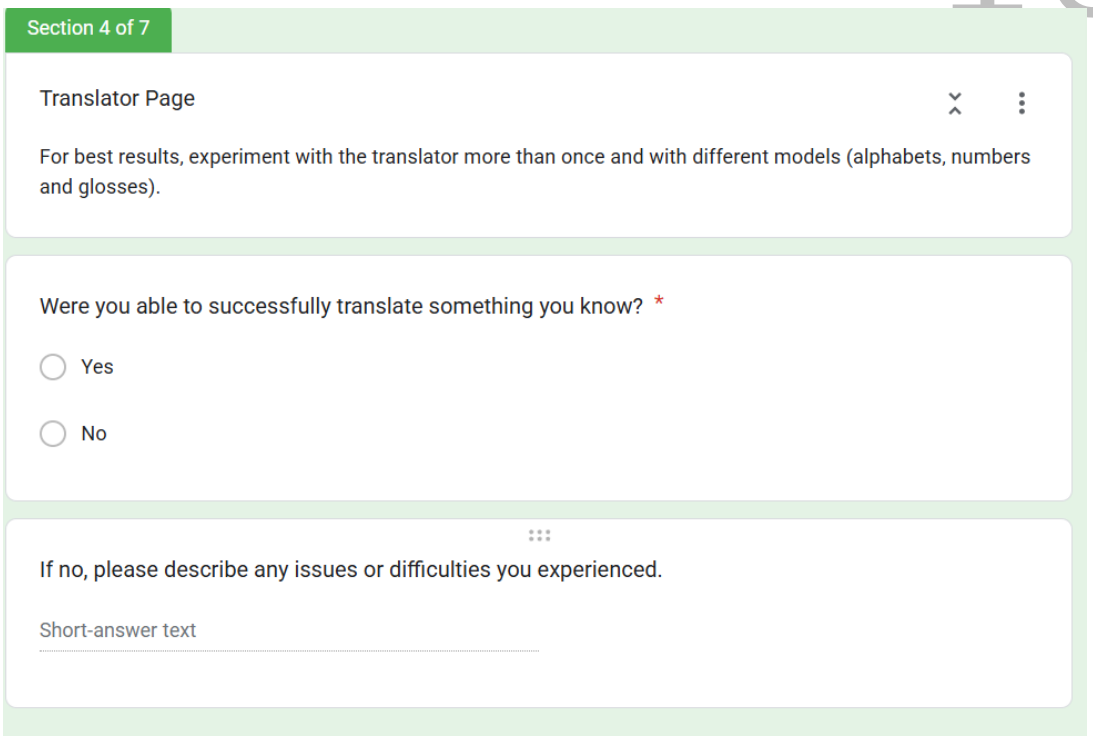

Figure 7: Example of usability testing feedback collection