ELO Learning

Security Policy

ZERO DAY

Last updated: 28 September 2025
University of Pretoria

Name	Student number
RM (Rene) Brancon	u22556771
NF (Nigel) Mofati	u22528084
TM (Tukelo) Mokwena	u22536800
S (Saskia) Steyn	u17267162
NG (Ntokozo) Tonga	u22506773

Team contact:
ZeroDay0D4y@gmail.com

Contents

1. Introduction

The purpose of this document is to define a clear framework for testing procedures, ensuring that all aspects of the application are thoroughly evaluated for functionality(usability), performance, security, and reliability for the ELO Learning platform.Considering the complexity of the system, which involves frontend and backend components, as well as the integration of third-party services and APIs, a robust testing strategy is essential to maintaining the application's quality and a seamless user experience.

The testing policy includes, but is not limited to:

- Unit testing
- Integration testing
- End-to-end testing
- Performance testing
- Usability testing
- Scalability testing

2. Unit, Integration and E2E Testing Policies

This section outlines the testing policy for our software development projects, focusing on Unit Testing, Integration Testing, and End-to-End (E2E) Testing. The goal of this policy is to ensure the quality, reliability, and stability of the codebase by enforcing rigorous testing standards across all phases of development.

We emphasize the use of automated testing to enhance efficiency and accuracy. Both unit and integration tests are integrated into our continuous integration (CI) pipeline, which is configured in GitHub Actions. Automated testing is a key component of this pipeline, preventing code from being merged into our master branch if tests fail. This approach helps to maintain high standards of code quality.Our code coverage is tracked using Codecov and the badges can be found in our GitHub repository.

3. Unit Testing Policy

Objective
The objective of unit testing is to validate the functionality of individual components within the software, ensuring that each function operates as intended. This includes testing various function logic and error handling.
Automation
All unit tests must be automated and seamlessly integrated into the GitHub Action pipeline. This integration enforces testing protocols and prevents any code from being merged into integration branches if tests fail.

Frameworks
Jest for implementing automated unit testing across our system.

4. Integration Testing Policy

Objective
The objective of integration testing is to validate the interactions between various components within a service and across multiple services, ensuring that they function together as expected.

Automation
All integration tests must be automated and seamlessly integrated into the GitHub Action pipeline.

Frameworks
Backend Integration: We utilize Jest for running integration tests within NestJS service.
Frontend Integration: Cypress is employed for conducting integration tests on the frontend, ensuring that the user interface and its interactions function correctly.

5. End-to-End (E2E) Testing Policy

Objective
The objective of end-to-end (E2E) testing is to ensure that the complete process flow of the application functions as intended from start to finish. This includes verifying the presence of components, assessing behavior during user interactions, and checking the requests and responses throughout the application.

Framework
We utilize Cypress for conducting end-to-end testing, providing robust capabilities for simulating user behavior and validating application workflows.

Automation
The end-to-end tests must be automated and integrated into the GitHub Actions pipeline as a final measure to ensure the functionality of our application.

Testing Environment
Cypress tests are executed as a part of the CI/CD pipeline running on GitHub Actions. This setup allows for comprehensive testing of the application to ensure all components work seamlessly together.

6. Testing workflow

Overview

The testing workflow outlines the systematic approach we take to ensure the quality and reliability of our software throughout the development lifecycle. This process integrates various testing types: unit, integration, and End-to-end, and leverages automation to enhance efficiency and accuracy.

Steps in the Testing Workflow

Development

Developers write code for new features or bug fixes. Alongside, they create corresponding unit tests to validate individual components.

Unit Testing

Automated unit tests are executed in the GitHub pipeline when a branch is pushed to or pulled from. The tests validate that individual functions work as expected.

Integration Testing

Once unit tests pass, integration tests are executed to validate the
interactions between components. Backend integration tests are run using Jest, while frontend integration tests utilize Cypress. These tests run in a designated environment setup within the GitHub workflow to simulate real-world scenarios.

End-to-End Testing

After integration tests are successful, end-to-end tests are conducted in the same CI/CD pipeline on Github Actions. This stage verifies that the entire application workflow functions correctly from a user perspective. E2E tests check for component presence, user interactions, and the integrity of requests and responses.

Review and Feedback

Any identified issues are documented, and feedback is shared among team members. This collaborative approach ensures that all functionality is thoroughly inspected and that necessary adjustments are made before deployment.

Deployment

Once all testing phases are complete and issues resolved, the application is deployed to the production environment by merging the code into the master branch, triggering the deployment workflow.

7. Quality Assurance Testing

Quality Assurance testing is a critical aspect of the software development lifecycle, focusing on ensuring that applications not only function correctly but also meet the highest standards of performance, load handling, and usability.

As our applications grow in complexity and user expectations rise, it becomes increasingly important to assess how well our software performs under various conditions and how user-friendly it is.

We prioritised the testing of our core quality requirements: performance, usability and scalability.

Together, these QA testing methodologies form a comprehensive approach to validating the quality of our software, ultimately leading to a more reliable, performant, and user-friendly product. This section outlines our strategies, objectives, and processes for implementing effective performance, load, and usability testing within our projects.

8. ELO Learning Application:

Reliability, Performance, and Scalability Report

Introduction

ELO Learning is a web-based platform designed to make learning mathematics more enjoyable and interactive. This report summarizes the results of system reliability, performance, and scalability testing using automated tools and synthetic load, as no real user testing was conducted. The findings are based on Uptime Robot (reliability), Google PageSpeed Insights (performance/usability), and Artillery load testing (scalability).

Reliability

Tool:	Uptime Robot
Uptime:	100% over the last 7 and 30 days, with no incidents or downtime.
Current Status:	System has been continuously up for 11 days, 1 hour, and 44 minutes.
Response Time:	Consistently low, averaging around 400-600ms.
SSL/Domain:	Valid and monitored.

Conclusion

ELO Learning demonstrates excellent reliability, with zero downtime and stable response times, ensuring users can access the platform at any time.

Performance & Usability

Tool:	Google PageSpeed Insights
Performance Score:	100/100
Accessibility:	85/100
Best Practices:	96/100
SEO:	100/100

Key Metrics

First Contentful Paint:	0.2s
Largest Contentful Paint:	0.5s
Total Blocking Time:	0ms
Cumulative Layout Shift:	0.003

Usability Observations

- The site loads extremely quickly and is visually stable.
- Accessibility is good but can be improved (e.g., color contrast, alt text).
- SEO and best practices are well implemented. The interface is intuitive and responsive, supporting a positive user experience.

Conclusion:

ELO Learning offers a fast, user-friendly experience with excellent performance and strong adherence to web standards.

Scalability

Tool: Artillery Load Testing

Methodology

Simulated 600 virtual users making requests to key API endpoints (user, singleplayer, classroom wars, multiplayer) to assess system behavior under load.

Results

User Endpoints:
600 requests completed, 0 failed.
Mean response time: 187ms.
Peak response time: 1.6s.
All requests returned valid responses (200/401/404 codes).

Singleplayer Endpoints:
600 requests completed, 0 failed.
Mean response time: 213ms.
Peak response time: 1.1s.
All requests returned valid responses.

Classroom Wars Endpoints:
600 requests completed, 0 failed.
Mean response time: 1ms (very fast).
Peak response time: 393ms.
All requests returned valid responses.

Multiplayer Endpoints:
600 requests completed, 0 failed.
Mean response time: 243ms.
Peak response time: 431ms.
All requests returned valid responses.

Conclusion:
ELO Learning scales efficiently under simulated high-traffic conditions, maintaining low response times and zero errors across all tested endpoints.

9. Summary

Strengths:
High reliability and uptime.
Excellent performance and usability scores.
Scalable architecture, handling large numbers of concurrent users with minimal latency.

Areas for Improvement:
Accessibility can be further improved to meet WCAG standards.

Continue monitoring for edge-case errors and optimize endpoints with higher peak response times.