Test Plan

FileSender

SURF
Product Manager: William van Santen
Mentor: Rogier Spoor

Windesheim University of Applied Sciences
Supervising Teacher: Rob Kaesehagen
Student: Aaron Jonk (s1170298)

Date: 26/06/2025

Version: 0.4

Version management

Version	Date	What
0.1	07/02/2025	Started the development of this file
0.2	11/02/2025	Wrote introduction, unit testing, end-to-end testing, and code coverage.
0.3	18/05/2025	Review based on feedback from Jan Meijer & Guido Aben (both part of the FileSender board), and a clean-up.
0.4	26/06/2025	Write out user test cases. Rewrite unit testing & e2e tests.

Distribution

Version	Date	Recipient
0.2	18/02/2025	Published on Codeberg
0.3	18/05/2025	Published on Codeberg
0.4	26/06/2025	Published on Codeberg

Contents

1 Introduction and Context

For the development of FileSender version 4, this project will focus on creating a robust file-sharing application with enhanced testing methodologies to ensure its stability, security, and functionality. This document outlines the various testing methods that will be applied during the development process, explaining how each will be implemented and integrated into the development pipeline. Key considerations include available development time, the importance of specific functionalities, and the overall security of the system.

As part of the chosen testing approach, specific test cases have been defined for user testing. Each test case includes detailed instructions on how the test should be conducted and records the findings from these tests. After the testing phase, the results will be compiled into a comprehensive overview, highlighting critical bugs, impactful improvements, and suggestions for optimization. This analysis will serve as the foundation for determining the improvements that can be addressed in the short and long term, allowing developers to prioritize fixes based on user feedback and system needs.

2 Unit Testing

The first category of tests outlined in this document is unit testing. Go's standard library includes built-in support for writing and running unit tests using the "go test" command. These unit tests are designed to validate individual functions and internal logic in isolation. The goal is to detect errors early and ensure that each component behaves as expected under a variety of inputs.

In FileSender, unit testing focuses on areas such as:

API Handlers
These tests verify that the different HTTP endpoints respond correctly, return appropriate status codes, and handle edge cases gracefully.

Authentication Logic
The auth internal package includes functions for a variety of authentication methods. Unit tests ensure these mechanisms work reliably.

File Handling Logic
Core functionality for creating, managing, and cleaning up file uploads is tested in isolation to verify correct behavior under different input scenarios.

In addition to testing correctness, the go test framework is used with the cover flag to generate code coverage metrics. These metrics indicate which parts of the codebase are exercised during testing. The development team aims to maintain at least 80% statement coverage, which provides a healthy baseline without overengineering the test suite.

Unit tests are executed automatically within the CI/CD pipeline. If any test fails, the pipeline will halt the pull request process, preventing untested or broken code from being merged. Writing and maintaining unit tests is included in the "definition of done" for every new feature or change.

3 End-to-End Testing

End-to-end testing ensures that the application behaves correctly from the user's perspective across various environments. For this, Playwright was selected as the testing framework due to its flexibility, cross-browser support, and ability to simulate a wide range of user interactions.

Playwright allows us to run tests across multiple browsers, including:

- Chromium (representing Chrome and Edge)
- WebKit (representing Safari)
- GeckoDriver (representing Firefox)

These tests simulate a real user interacting with the system, without requiring internal knowledge of the implementation. The primary focus of the E2E tests is to ensure that basic workflows such as:

- Uploading a file
- Downloading a file

Work reliably and without any unexpected errors. These basic flows are essential to FileSender's core functionality and must perform consistently across different browsers and platforms.

Like unit tests, E2E tests are integrated into the CI/CD pipeline and are automatically executed whenever new code is pushed. New tests are added as part of the "definition of done" to ensure that critical user paths remain stable during ongoing development.

4 User Tests

The final testing phase will involve user testing. At the end of the developmental process for the new FileSender, we will invite several users to follow detailed test scripts. These users will complete specific tasks without any prior insider knowledge, ensuring that the application is intuitive and user-friendly. Tasks will range from system administrators setting up the application to end users uploading files.

Test scripts and cases will be carefully crafted and executed methodically. By analyzing the results of these tests, we will identify areas that require improvement and give additional attention to any features that users find confusing or problematic. This feedback will be crucial for refining the application and enhancing the overall user experience.

4.1 Test Cases

Although no formal, scripted user tests were conducted during the internship, various informal user tests were continuously performed throughout development. François and Jeroen regularly ran the FileSender application on their local (and hosted) environments, and I actively tested features during development as well. These informal test cycles helped ensure basic functionality was stable across iterations.

Had formal user testing been conducted, the following test cases would have been included to verify key functionality:

Upload a file (≤2GB)
Objective: Verify that the user can upload a file.
Expected Result: The upload completes without error and the file is available for download.

Download a previously uploaded file
Objective: Confirm that users can successfully download files they or others have uploaded.
Expected Result: The file is downloaded without corruption or truncation.

Upload a large file (>2GB)
Objective: Verify that large files can be uploaded & encrypted without issue.
Expected Result: The file is uploaded successfully.

Download and validate a large file (>2GB)
Objective: Ensure that large files downloaded by the user are identical to the originals.
Validation: Use the Unix “diff” command to compare the original and downloaded file.
Expected Result: The files are identical, confirming data integrity during transfer.

Upload with browser JavaScript disabled
Objective: Test file upload functionality when JavaScript is disabled.
Expected Result: The application should handle this gracefully.