Advisory Report

FileSender

SURF
Product Manager: William van Santen
Mentor: Rogier Spoor

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

Date: 18/02/2025

Version: 1.0

Version Management

Version	Date	What
0.1	07/02/2025	Write introduction & advise on programming language
0.2	11/02/2025	Write advise on testing framework
0.3	14/02/2025	Advise on which database to use
1.0	18/02/2025	Edit based on feedback. Add PHP & Selenium to comparison table. Show SQLite performance in numbers.

Distribution

Version	Date	Recipient
0.3	14/02/2025	FileSender signal group chat
1.0	18/02/2025	Publish on Codeberg

Contents

1 Introduction and Context

This advisory report is prepared for the rewrite and implementation of new and existing functionality within FileSender version 4. During this reimplementation, recommendations will be provided, supported by the research presented in this document. These recommendations will cover various aspects, from the technologies to be used in the rewrite, such as the programming language, to the testing framework, and guidance on the future development of FileSender.

When relevant, these recommendations will be supported by a decision matrix. This matrix assigns scores to different alternatives based on how well they meet specific requirements. By summing up the scores for each option across all requirements, a total score is calculated, making it clear which option is most suitable for the project. Each option is scored per requirement on a scale from 0 to 10. The score indicates how well the option satisfies the requirement and how important that requirement is for the project.

- A score of 0 means the option does not meet the requirement at all.
- A score of 10 means the option fully meets the requirement and the requirement is highly important.
- For example, if an option meets a requirement but the requirement is not crucial, it might receive a score of 6.

2 Programming Language

The current versions of FileSender (versions 1, 2, and 3) are written in PHP. The first decision to make for the reimplementation of FileSender is selecting the appropriate programming language. The application needs to be fast, memory-safe, secure, and capable of serving static files as well as creating potential API endpoints.

The technologies being considered for comparison are PHP, Node.js (JavaScript), Python, Golang, and Rust.

	PHP	Node.js	Python	Golang	Rust
Performance (response-times)	0	3	0	7	10
Performance (memory usage)	3	0	3	10	7
Security-first design	0	0	0	10	7
Capable of serving static files	10	10	10	10	10
Good developer experience	7	7	7	5	7
Preferred by SURF	0	0	0	1	0
Total	20	20	20	43	31

Table 1

When evaluating the performance of the programming languages, Golang and Rust are closely matched[1]. As a result, PHP, Node.js and Python are no longer considered suitable candidates for the rewrite.

Another crucial factor to consider is security, which is the primary reason for reimplementing FileSender. Due to Go's simplicity, its built-in garbage collection, and its extensive standard library, Go is preferred over Rust in this regard[2].

[1] Pereira, R., Couto, M., Ribeiro, F., Rua, R., Cunha, J., Fernandes, J. P., & Saraiva, J. (2021). Ranking Programming Languages by Energy Efficiency. In GitHub. Universidade da Beira Interior. Accessed on 7 February 2025. https://haslab.github.io/SAFER/scp21.pdf Page 16

[2] Arundel, J. (2025, 3 February). Rust vs Go in 2025 — Bitfield Consulting. Bitfield Consulting. Accessed on 7 February 2025. https://bitfieldconsulting.com/posts/rust-vs-go

Lastly, it’s worth mentioning that I have no prior experience with Go, which slightly lowers its score in terms of developer experience. However, despite this, Golang stands out as the best overall option. Therefore, Golang is the recommended programming language for the FileSender rewrite.

3 Testing Framework

With the programming language chosen for FileSender version 4, it's important to establish a proper testing strategy for the new application. Go's built-in unit testing library, included in its standard library, will be utilized for application unit tests. These tests can conveniently run within the CI/CD pipeline, making them an ideal fit for our development workflow.

In addition to unit tests, end-to-end testing will be critical. These tests should simulate different browsers and configurations, such as disabling JavaScript in the browser, to ensure FileSender's robustness. To achieve this, we have evaluated three interesting libraries: Playwright, Cypress, and Puppeteer.

	Playwright	Cypress	Puppeteer	Selenium
Can be run in pipelines	1	1	1	1
Support turning JavaScript off in browser	10	0	10	10
Support for Chromium, Firefox & WebKit (Safari) testing	10	0	3	10
Total	21	1	14	21

Table 2

The first and perhaps most crucial feature is the ability to run tests within CI/CD pipelines. Automated tests should be executed with every pull request, ensuring continuous integration. Fortunately, all three options are compatible with CI/CD pipelines.

The second key requirement is the ability to disable JavaScript during testing. This is a critical feature for simulating real-world user scenarios where JavaScript may be disabled. Cypress falls short in this area, as it is heavily dependent on JavaScript for its testing framework. Although there is an open issue discussing this feature, no active development is underway to address it.

The third requirement is extensive browser support, covering Chromium, Firefox, and WebKit (Safari). Playwright and Selenium excel in this category, providing full support for all three major browsers. Puppeteer, on the other hand, offers limited compatibility, and Cypress lacks support beyond Chromium-based browsers.

Differences between Playwright and Puppeteer

Here are the differences between Playwright and Puppeteer.

Parameter	Playwright	Puppeteer
Browser Support	Supports multiple browsers including Chrome, Firefox, and WebKit.	Focuses mainly on Chrome and Chromium-based browsers, with limited support for others.

Figure 1, source

In the end, both Selenium and Playwright remain as the top choices, having received the same score in the comparison. However, Playwright stands out due to its easy learning curve and efficient parallel execution, which generally makes it faster than Selenium[3].

Given these advantages, Playwright is the most suitable option, offering robust and flexible testing capabilities for FileSender version 4.

[3] Idowu Omisola. (2023, January 11). *Playwright vs. Selenium in 2025: In-Depth Comparison*. @ZenRowsHQ; ZenRows. Accessed on 18 February 2025. https://www.zenrows.com/blog/playwright-vs-selenium#playwright-vs-selenium

4 Database

The database is a critical component of the new FileSender application. Our goal is to keep the setup, usage, and maintenance as simple as possible. Go applications offer flexibility with databases, allowing easy swaps in the future. Therefore, the current recommendation is based on present needs and is not a permanent decision. We compared MySQL, PostgreSQL, and SQLite using a decision matrix.

	MySQL	PostgreSQL	SQLite
Small library size	5	5	10
Scalable	3	3	0
Built-in within integration	0	0	3
Performance	10	7	5
Total Score	18	15	18

Table 3

MySQL and SQLite are tied in the decision matrix. However, we recommend using SQLite for the following reasons:

- SQLite has a small library size and is built into the application, eliminating the need for database configuration during setup.
- The SQLite database is created within the application itself, streamlining installation and reducing overhead.

While SQLite scored lowest in performance, this is not a major concern for our current use case. The ability to switch databases in Go means that performance issues can be addressed in the future if they arise. Additionally, our use case is not write-intensive, so SQLite's serialized write operations are unlikely to become a bottleneck.

Furthermore, benchmarks show that SQLite is already quite fast, handling around 5,000 INSERT operations per second. This level of performance is more than sufficient for our application's needs[4].

[4] Toxigon. (2025, January 22). *SQLite Performance Benchmarks: 2025 Edition Insights.* Toxigon. Accessed on 18 February 2025. https://www.toxigon.com/sqlite-performance-benchmarks-2025-edition

In conclusion, we choose SQLite for its simplicity, easy integration, and minimal maintenance. The decision is flexible, and we can revisit it if our performance or scalability needs evolve.