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: 18/05/2025

Version: 1.4

Version Management

Version	Date	What
0.1	07/02/2025	Wrote the introduction & advice on programming language
0.2	11/02/2025	Wrote the advice on testing framework
0.3	14/02/2025	Advice on which database to use
1.0	18/02/2025	Edited based on feedback. Added PHP & Selenium to comparison table. Show SQLite performance in numbers.
1.1	25/02/2025	Added additional factors to the decision matrix for the database advice.
1.2	05/04/2025	Added “Storage Approach” advise
1.3	11/04/2025	Added “Encryption” advise
1.4	18/05/2025	Review based on feedback from Jan Meijer & Guido Aben (both part of the FileSender board), and a clean-up.

Distribution

Version	Date	Recipient
0.3	14/02/2025	FileSender signal group chat
1.0	18/02/2025	Published on Codeberg
1.1	25/02/2025	Published on Codeberg
1.2	05/04/2025	Published on Codeberg
1.3	11/04/2025	Published on Codeberg
1.4	18/05/2025	Published on Codeberg

Contents

1 Introduction and Context

This advisory report outlines recommendations for the rewrite and implementation of both new and existing functionality within FileSender version 4. The recommendations provided herein are based on thorough research and analysis, covering key aspects of the reimplementation process. These include, but are not limited to, the selection of appropriate technologies (e.g., programming languages, testing frameworks), and strategic guidance for the future development of FileSender.

To ensure an objective and structured evaluation, a decision matrix has been employed where applicable. This matrix assesses various alternatives against predefined requirements, assigning weighted scores to each option based on its alignment with project priorities. Each alternative is evaluated on a scale of 0 to 10 per requirement, with the cumulative score determining the most suitable choice.

Scoring Methodology:

- 0: The option fails to meet the requirement.
- 10: The option fully satisfies the requirement, which is of high importance to the project.
- Intermediate scores (e.g., 6): Indicate partial fulfillment or cases where the requirement is of moderate significance.
- By aggregating the scores across all requirements, the decision matrix provides a clear, data-driven basis for selecting the optimal solution.

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 following technologies were evaluated:

- PHP
- Node.js (JavaScript)
- Python
- Golang (Go)
- Rust

A comperative assessment was conducted based on key criteria:

	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

Based on this evaluation, Goland and Rust significantly outperform PHP, Node.js, in performance and security[1]. While Rust demonstrates superior raw performance, Golang is favored due to its:

- Built-in garbage collection
- Extensive standard library (reducing dependency risks)
- Strong security posture (simpler memory safety guarantees)
- Enterprise-grade concurrency and scalability (ideal for long-term maintenance)

Additionally, Golang's straightforward tooling and minimal runtime dependencies make it well-suited for high-performance, maintainable applications[2]. Although Rust excels in low-level control, Golang's balance of performance, security, and developer efficiency makes it the recommended choice for FileSender 4.

[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

3 Testing Framework

A robust testing strategy is essential for FileSender 4. Golang's native unit testing framework will be used for core logic validation. For end-to-end testing, the following libraries were evaluated:

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
CI/CD pipeline compatibility	1	1	1	1
JavaScript disable support	10	0	10	10
Multi-browser support (Chromium, Firefox, WebKit)	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].

Playwright is the recommended solution due to its:

- Full browser coverage (Chromium, Firefox, WebKit)
- Support for JavaScript-disabled testing (critical for accessibility)
- Superior parallel execution vs. Selenium

[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 Storage Approach

In the initial stages of the FileSender reimplementation, the design included a database to manage data storage. However, as development progressed, it became clear that introducing a database at this point adds unnecessary complexity. We advise moving away from using a database entirely and instead adopting a filesystem-based approach for data storage.

Key Benefits:

- No database overhead: Files and metadata are stored in user-specific directories.
- Ease of backup: Standard tools (e.g., rsync) can manage data.
- Transparency: Direct file system access simplifies debugging.
- Performance: Optimized for read-heavy workloads.

Future scalability (e.g., relational data) can be addressed via Go's modular design, with SQLite as a potential upgrade path (see Section 5).

5 Database

The database is a critical component of the new FileSender application. The 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.

The evaluation compared MariaDB, PostgreSQL, and SQLite using a decision matrix.

	MariaDB	PostgreSQL	SQLite
Small library size	5	5	10
Scalable	3	5	0
Not an external application (built-in)	0	0	3
Performance	10	7	5
Favorable (open source) licensing	10	10	10
Installation process	7	6	10
Total Score	35	33	38

Table 3

In this case, SQLite is recommended due to its:

- Zero-configuration deployment
- Low maintenance (no separate server process)
- Adequate performance (5,000 INSERTs/second[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

6 Encryption

An important part of the FileSender application is the encryption and decryption of files on the client side (in the browser). The new version of FileSender will fully support, this functionality without imposing constraints on file size, allowing users to encrypt and upload files as large as 1TB.

To support large files, FileSender 4 requires streaming-encryption for files ≤1TB. Meaning that files will not be fully loaded on memory but rather read and processed in chunks.

For this decision, three solutions (the browser-native SubtleCrypto API, and two external libraries: Age and Libsodium) were assessed:

	SubtleCrypto	Age	Libsodium
Streaming Encryption	10	10	10
Streaming Decryption	10	Chunked (10)	Chunked (10)
Browser Support	10	8	10
Performance	10	7	7
Size	10	10	10
Secure	0	5	5
Total Score	50	50	52

All three options support encrypting data in either a streaming or chunked manner, which is essential for handling large files efficiently in the browser. The most significant differences arise in terms of cryptographic safety, performance consistency, and future readiness.

SubtleCrypto provides fast, native cryptographic operations with full browser support and zero dependency size. However, it is limited to legacy algorithms such as AES-GCM, AES-CBC, AES-CTR, and RSA-OAEP. While these algorithms are still widely supported, they are no longer recommended for modern security practices in 2025[5]. AES, particularly in modes like CBC and GCM, has known risks and limitations when misused, and SubtleCrypto

[5] Hcardona. (2023, December 9). *Padding Oracle Attack: Are You Vulnerable?* Winmill. https://www.winmill.com/incorrect-aes-implementation-leaves-system-vulnerable

requires developers to manually manage parameters like IVs and authentication tags. Additionally, RSA-OAEP is only suitable for asymmetric encryption and cannot be used directly for encrypting large file contents. These factors justify a score of 0 in the “Secure” category, as SubtleCrypto does not offer modern, forward-secure encryption primitives or safe-by-default APIs.

Age is a simplified encryption format that uses X25519 and ChaCha20-Poly1305 under the hood. It offers a safer abstraction than SubtleCrypto and reduces the chance of insecure usage patterns. While Age supports encryption in the browser, its current limitations include the lack of full streaming decryption pipeline and slightly reduced performance compared to native APIs.

Libsodium offers a complete cryptographic solution, including a ChaCha20-Poly1305 API, which is specifically designed for secure and authenticated streaming encryption. It is implemented in WebAssembly, providing high performance and security. Although it introduces a small initialization cost and adds external library weight, Libsodium is by far the most robust and future-ready option. It handles chunked or streamed encryption securely and correctly without burdening the developer with low-level cryptographic decisions.

Considering these criteria and the nature of the FileSender application, Libsodium is the recommended encryption library due to its balance of security, performance, and streaming capabilities.

7 Summary

File FileSender 4 reimplementation will use:

- Golang for the backend
- Playwright for E2E testing
- Filesystem storage (with SQLite as a future option)
- Libsodium for client-side encryption

This stack ensures security, performance, and maintainability while minimizing complexity.

8 References

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
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. 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
5. Hcardona. (2023, December 9). *Padding Oracle Attack: Are You Vulnerable?* Winmill. https://www.winmill.com/incorrect-aes-implementation-leaves-system-vulnerable