# Requirements

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:** 1.1

## Version Management

| Version | Date | What |
|---|---|---|
| **0.1** | 04/02/2025 | Document outlining |
| **0.2** | 05/02/2025 | Functional requirements development |
| **0.3** | 07/02/2025 | Finalized functional requirements & formatting |
| **0.4** | 12/02/2025 | Feedback incorporation & formatting updates |
| **1.0** | 18/02/2025 | Developed & finished writing nonfunctional requirements |
| **1.1** | 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** | 07/02/2025 | FileSender signal group chat |
| **1.0** | 18/02/2025 | Published on Codeberg |
| **1.1** | 18/05/2025 | Published on Codeberg |

## Contents

1 Functional Requirements ........ 5

1.1 Must Haves (MVP) ........ 5

1.1.1 Core File Management and Storage ........ 5

1.1.2 Security and Logging ........ 5

1.1.3 Compatibility and Accessibility ........ 5

1.2 Should Haves ........ 6

1.2.1 Advanced File Handling and Performance ........ 6

1.2.2 Customization and User Experience ........ 6

1.2.3 Security and Authentication ........ 6

1.2.4 Deployment and Packaging ........ 6

1.2.5 Monitoring and Logging Enhancements ........ 6

1.3 Could Haves ........ 7

1.3.1 File Upload and Download Enhancements ........ 7

1.3.2 Customization and User Experience ........ 7

1.3.3 Security and Authentication ........ 7

1.3.4 Scalability and Monitoring ........ 7

1.4 Won’t Haves ........ 7

1.4.1 Advanced Synchronization and Storage ........ 7

1.4.2 Alternative File Access Protocols ........ 7

2 Non-Functional Requirements ........ 8

2.1 Functional Suitability ........ 8

2.2 Performance Efficiency ........ 9

2.3 Compatibility ........ 9

2.4 Interaction Capability ........ 10

2.5 Reliability ........ 10

2.6 Security ........ 11

2.7 Maintainability ........ 12

**2.8 Flexibility** .......... 13
**2.9 Safety** .......... 14

# 1 Functional Requirements

The functional requirements outlined in this section define the features and capabilities that need to be implemented in the new FileSender version 4. To prioritize these requirements, the MoSCoW method has been used, categorizing them into Must-Have, Should-Have, Could-Have, and Won't-Have features. The Must-Have requirements are essential for the current MVP. It is important to note that within each category, the requirements are not listed in any particular order of priority.

Once the MVP items (Must-Haves) have been fully implemented, the focus should shift to completing the Should-Have requirements. After the Should-Haves are addressed, the Could-Have features should be prioritized for implementation, ensuring the overall project follows a structured and efficient progression.

## 1.1 Must Haves (MVP)

### 1.1.1 Core File Management and Storage

- Save files directly to the filesystem
- Support 1TB uploads via file chunking
- Serve static HTML, CSS, and JS files
- Client-side file encryption and decryption

### 1.1.2 Security and Logging

- Logging

### 1.1.3 Compatibility and Accessibility

- Support without JavaScript

## 1.2 Should Haves

### 1.2.1 Advanced File Handling and Performance

- File encryption and decryption combined with file chunking, supporting files up to 1 TB
- Parallel uploads for faster upload speed (using multiple chunks)
- Parallel downloads for faster download speed (using multiple chunks)
- Download resumption

### 1.2.2 Customization and User Experience

- Customizable served pages using templates
- Email notifications for key events (e.g. file expiration notifications, upload completion and download completion)
- Guest upload vouchers (configurable limits for non-authenticated users)
- Ability to set an email address where FileSender sends an email with the download link

### 1.2.3 Security and Authentication

- SAML and OIDC authentication support configured via an HTTP proxy server

### 1.2.4 Deployment and Packaging

- Pre-built Docker image
- Debian & Fedora packages

### 1.2.5 Monitoring and Logging Enhancements

- Extended logging to track file uploads, downloads, and other activities on FileSender (e.g., user activity logs)

## 1.3 Could Haves

### 1.3.1 File Upload and Download Enhancements

- Upload resumption (users can resume uploads later or from a different device)
- Built-in S3 bucket support

### 1.3.2 Customization and User Experience

- Customizable email notifications using templates

### 1.3.3 Security and Authentication

- Built-in SAML and OIDC authentication support
- Authentication via username and password

### 1.3.4 Scalability and Monitoring

- Usage and data metrics for monitoring
- Scalable architecture

## 1.4 Won't Haves

### 1.4.1 Advanced Synchronization and Storage

- Built-in synchronization of files and databases between multiple servers for reliability or low-latency global access

### 1.4.2 Alternative File Access Protocols

- WebDAV or Samba support

# 2 Non-Functional Requirements

For this project, non-functional requirements have been defined that specify the non-functional aspects of this project. These requirements have been established based on the ISO 25010 standard[1], with its various components tailored to meet the specific needs of this project.

To function effectively, efficiently, and reliably, the application must comply with each of the defined non-functional requirements. For this reason, no prioritization will be applied to the non-functional requirements, all will be considered equally essential.

## 2.1 Functional Suitability

- When a new feature is developed for the app, it follows the predefined wireframes. This ensures that the application displays the correct information at the expected moments while also proactively considering potential future expansions within the project.

- Whenever new API endpoints are created, corresponding API documentation will be provided. This documentation will always accurately reflect the inputs and outputs of each endpoint after implementation. This ensures clarity regarding the data requested and provided by the endpoint, allowing the app to receive and process the correct information.

- Each new feature added to the application will be automatically tested through a pipeline. This ensures that the app functions correctly across different scenarios or conditions, while consistently delivering correct information.

- At the end of the project, the application will be manually tested by members of the target audience to verify its effectiveness in helping them achieve their goals. If necessary, final adjustments will be made based on their feedback.

[1] ISO 25000. *ISO/IEC 25010.* Accessed on 18 February 2025. https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

## 2.2 Performance Efficiency

- The web application must respond to all HTTP requests within 1 second, ensuring users receive timely feedback after performing an action. This improves user experience by minimizing delays.

- The application must operate efficiently, capable of running on a machine with only 1 vCPU and 512 MB RAM. By optimizing resource usage, the cost of hosting and maintaining the system is reduced, making it more accessible and scalable.

- All (if any) images and media files must load within 1 second, to ensure that users do not experience long waiting times when accessing content. Fast loading speeds contribute to a smoother and more engaging user experience.

- File transfer speeds should be optimized based on available network conditions by utilizing techniques such as chunked uploads, parallel processing, and caching where applicable.

## 2.3 Compatibility

- When exchanging data between different applications or services, a standardized data format (JSON) must be used for both the frontend and backend. This ensures seamless interoperability and easy integration with external systems.

- The web application must be compatible with browsers and operating systems that are up to 4 years old, ensuring accessibility for users with slightly older devices. The application should function correctly and be fully usable on these platforms without requiring major modifications.

- Cross-browser compatibility must be ensured, meaning the application should work smoothly on major browsers, including Google Chrome, Mozilla Firefox, Microsoft Edge, and Safari. Regular testing should be conducted to identify and resolve any browser-specific issues.

- The application must be responsive, ensuring a consistent and user-friendly experience across desktop, tablet, and mobile devices. This includes maintaining proper layout, navigation, and functionality regardless of screen size.

## 2.4 Interaction Capability

- The UI must be designed following modern UI/UX standards, ensuring a familiar and intuitive experience for users. Elements such as button placement, text positioning, and media layout should be aligned based on other similar applications and modern industry practices. This allows users to navigate FileSender seamlessly, leveraging their existing experience with other apps.

- When performing a critical action, such as deleting a file or transfer, users must be prompted with an additional confirmation step. This helps prevent accidental actions that could result in data loss or unintended consequences.

- In the initial phase, the web application will only support English. Future versions may introduce multilingual support.

- The application must include clear visual feedback for user actions, such as loading indicators, success messages, and error notifications. This way, users will always understand the system's state and receive appropriate responses to their actions.

## 2.5 Reliability

- The FileSender application will have high reliability and fault tolerance to minimize downtime and prevent data loss during file transfers.

- The system will be designed to handle unexpected crashes or failures by implementing error handling, automatic recovery mechanisms, and logging. If an operation fails, it should be retryable where possible to ensure, minimal disruption to users.

- Transactional integrity must be maintained during file uploads and downloads. This means that if a file transfer is interrupted (e.g., due to a network failure), the system should support resumable uploads/downloads, allowing users to continue without restarting the entire process.

- The application should have a 99.9% uptime guarantee, ensuring continuous availability. This requires robust infrastructure, load balancing, and potential failover mechanisms.

- A logging and monitoring system must be implemented to detect failures in real-time for proactive issue resolution. Logs should provide detailed insights into system errors, failed transfers, and performance bottlenecks.

- In case of data corruption or transfer failures, the system should be able to detect inconsistencies through checksums (e.g., SHA-256 validation) and notify users of any integrity issues.

## 2.6 Security

- User authentication must be enforced when enabled by the admin. The application supports SAML and OIDC for secure and seamless authentication, allowing integration with enterprise identity providers and SSO solutions.

- Logging of all system changes must be implemented. Each modification must be recorded along with the user responsible, the IP address from which the change was made, and a timestamp. This ensures transparency and helps in troubleshooting or investigating security incidents.

- For all modifiable data, a "Last Modified" timestamp must be stored and displayed to users, allowing them to track when their data was last updated.

- Data encryption in transit must be enforced. All data in transit must use TLS 1.3 to protect against interception or man-in-the-middle attacks.

- Protection against brute-force attacks should be in place. Failed authentication attempts should be logged, and after a configurable number of failures, the account should be temporarily locked or require CAPTCHA verification.

- Secure file-sharing mechanisms should be enforced. File access must be token-based, with the option to set expiration times and download limits to prevent unauthorized access.

- Regular security audits and penetration testing should be conducted to identify and mitigate potential vulnerabilities in the system.

- The application must be tested against OWASP’s Top 10 vulnerabilities, including SQL injection, XSS, CSRF, and SSRF to prevent common security threats.

### 2.7 Maintainability

- Each API endpoint in the backend application must be modular and self-contained. This separation ensures that endpoints can be modified or removed independently without affecting the functionality of other endpoints to improve maintainability and reduce the risk of unintended side effects.

- The application backend should follow a clean architecture with separation between business logic, data access, and transport layers. This will make it easier to maintain, extend, and debug.

- The application should use structured logging and monitoring tools to track issues effectively and provide clear debugging information, improving long-term maintainability.

- Code reusability must be maximized by structuring the application to use reusable components. Utility functions and shared modules should be designed in a way that they can be used across different parts of the application. This approach improves efficiency and ensures consistency in the user experience.

- The frontend and backend must be well-documented, including API endpoints, authentication mechanisms, and data models. Documentation should be easily accessible to developers. For example, a markdown-based documentation in the repository.

- Automated unit tests, integration tests, and end-to-end tests must be written alongside the development process.
    - Tests should be runnable via a terminal command, allowing them to be executed efficiently in CI/CD pipelines.
    - Testing should cover both new functionalities and existing functionalities to ensure stability after changes.

- Code must be formatted and linted automatically to maintain a consistent coding style across the project. The use of Golang's built-in formatting tools should be enforced.

## 2.8 Flexibility

- The application must be designed with flexibility in mind to enable easy adaptation to future requirements, new technologies, and evolving user needs.

- The backend, should follow a modular and scalable architecture, making it easier to extend functionality, replace components, or integrate with external services without significant refactoring.

- API extensibility should be considered, ensuring that new endpoints or integrations can be added without breaking existing functionality. The API should be versioned to support backward compatibility when updates are introduced.

- The system must be designed to support multiple deployment environments (e.g., cloud-based, on-premises, or hybrid) without requiring major modifications. Configuration settings should be externalized to allow adjustments without modifying source code.

- The frontend must be responsive and adaptable to different devices' screen sizes and resolutions.
  - The web application must work on both desktop and mobile browsers while ensuring a smooth user experience across various devices.
  - It should support devices that have been updated within the last 4 years, ensuring compatibility with phones and tablets that may be up to 6-7 years old.
  - The UI should dynamically adjust to different resolutions, from smaller screens to larger screen devices.

- The system must be able to handle different file types and sizes efficiently, ensuring it remains a viable solution for various industries and use cases, such as business file sharing, media transfers, and academic research data.

## 2.9 Safety

- SQL injection and other input-based attacks must be mitigated by implementing strict input validation and sanitization for all user input fields. The application must use prepared statements and parameterized queries to prevent malicious SQL execution.

- Daily database backups must be performed automatically to ensure data recovery in case of unexpected failures, data corruption, or security incidents. The system should allow the restoration of a database to the state it was in within the last 24 hours to minimize potential data loss.

- Critical user actions, such as deleting an account or permanently removing files, must require an additional confirmation step (e.g., a confirmation dialog, email verification, or re-entering the password). This prevents accidental data loss and irreversible actions.

- SSL/TLS encryption must be enforced for all communications between the web application, backend API, and database. The system must support TLS 1.3 to ensure secure data transmission and protection against man-in-the-middle attacks.

- End-to-end encryption (E2EE) should be considered for sensitive file transfers, ensuring that only the intended recipient can access the files.

- Security patches and updates must be applied regularly to fix newly discovered vulnerabilities. Automated dependency tracking should be used to ensure the latest security patches are integrated.