**SECURED FILE SHARE – SRS Document**

A Project Report Submitted in Partial Fulfilment of the Requirements for the Degree of Bachelor of Science in Computer Science

A red and blue shield with white birds

AI-generated content may be incorrect.

**Submitted by:**

**Shrey Goel**

**E22CSEU0262**

**Under the Supervision of:**

**Riti Kushwaha**

**Assistant Professor , Department of Computer Science**

**Bennett University**

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| Section | Title | Page |
| 1 | Introduction | 3 |
| 2 | Overall Description | 5 |
| 3 | Requirement Gathering | 7 |
| 4 | UML Diagrams | 9 |
| 4.1 | Use Case Diagram | 9 |
| 4.2 | Sequence Diagram – Secure File Transfer | 10 |
| 4.3 | Class Diagram | 11 |
| 4.4 | ER Diagram | 12 |
| 5 | Feasibility Analysis | 13 |
| 6 | Other Requirements | 14 |
| 7 | Appendix | 15 |

## 1. Introduction

### 1.1 Purpose

This Software Requirements Specification (SRS) describes all functional, non-functional, and interface requirements for **EncrpytFileShare**, a modern, secure, peer-to-peer and client-server hybrid file sharing web application. The system allows users to upload, share, and download files with end-to-end encryption, password protection, expiration links, and access control.

### 1.2 Document Conventions

* “User” refers to any registered or anonymous visitor.
* “Authenticated User” refers to a logged-in user.
* “Shall” indicates a mandatory requirement.
* “Should” indicates a desirable but non-mandatory feature.

### 1.3 Intended Audience

* Development team
* Project manager
* Stakeholders and investors
* Quality assurance team
* Future maintainers

### 1.4 Project Scope

EncrpytFileShare aims to provide a privacy-focused alternative to services like WeTransfer, Dropbox Paper, and Firefox Send (now discontinued) with the following core capabilities:

* Upload files up to 50 mB (configurable)
* Generate shareable links with optional password and expiration
* End-to-end encryption in browser (client-side)
* QR code generation for mobile sharing
* Download tracking and deletion after expiry/download limit
* User accounts with dashboard and history (optional registration)

### 1.5 References

* GitHub Repository: https://github.com/SHR3YGO3L/file-share
* Original inspiration: Firefox Send (discontinued), File.io, WeTransfer

## 2. Overall Description

### 2.1 Problem Statement

Existing file-sharing solutions suffer from one or more of the following issues:

* Lack of true end-to-end encryption (server can access files)
* Limited or no expiration/password protection
* Poor mobile experience
* Large files cause high server cost and slow performance
* No open-source, self-hostable alternative with modern UI

EncrpytFileShare solves these problems by providing client-side AES-256-GCM encryption, optional registration, rich sharing options, and being fully open-source and self-hostable.

### 2.2 Product Objectives

* Privacy-first file sharing with zero server-side plaintext access
* Simple, beautiful, and responsive UI/UX
* Support very large files (up to 50 mB default)
* Self-hosting capability with Docker
* High performance using modern tech stack (Node.js + Express + React + TypeScript)

### 2.3 Product Functions

### File upload with client-side encryption

* Generation of secure shareable link
* Optional password protection and expiration
* File download with decryption in browser
* User authentication and personal dashboard (optional)
* Admin panel for instance management
* File auto-deletion after expiry or download limit

### 2.4 User Classes and Characteristics

|  |  |  |
| --- | --- | --- |
| **User Class** | **Description** | **Frequency** |
| **Anonymous User** | Uploads and shares files without an account. | High |
| **Registered User** | Has an account, can view history, and manage shares. | Medium |
| **Administrator** | Manages instance, storage, users, and settings. | Low |

### 2.5 Operating Environment

* **Server:** Node.js 18+, Docker (optional)
* **Client:** Modern browsers (Chrome, Firefox, Safari, Edge)
* **Database:** MongoDB or PostgreSQL
* **Deployment:** Linux servers, Docker, Kubernetes possible

### 2.6 Design and Implementation Constraints

* Must use client-side encryption (CryptoJS or Web Crypto API)
* Backend written in TypeScript/Node.js
* Frontend in React + Vite + Tailwind CSS
* Must be MIT licensed (as current repo)

### 2.7 Assumptions and Dependencies

* Users have modern browsers with Web Crypto API support
* Server has sufficient disk space and bandwidth
* MongoDB/PostgreSQL is available
* Internet connection required for sharing

## 3. Requirement Gathering

### 3.1 Functional Requirements

|  |  |  |
| --- | --- | --- |
| **ID** | **Requirement Description** | **Priority** |
| **FR01** | The system shall allow anonymous users to upload one or more files without registration. | High |
| **FR02** | The system shall encrypt files in the browser using AES-256-GCM before upload. | High |
| **FR03** | The system shall generate a unique, unguessable shareable link after successful upload. | High |
| **FR04** | The system shall allow setting an optional password for the share link. | High |
| **FR05** | The system shall allow setting expiration time (1 hour to 30 days) or download limit (1–100). | High |
| **FR06** | The system shall display a QR code for the share link. | High |
| **FR07** | The system shall decrypt and allow download only with the correct password (if set). | High |
| **FR08** | The system shall automatically delete file and metadata after expiration or max downloads. | High |
| **FR09** | Registered users shall have a personal dashboard showing upload history and active shares. | Medium |
| **FR10** | Users shall be able to register/login using email + password or OAuth (Google, GitHub). | Medium |

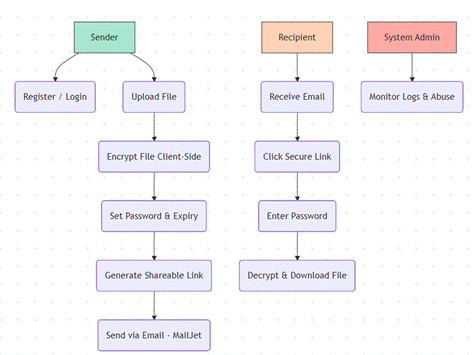
### 3.2 Non-Functional Requirements

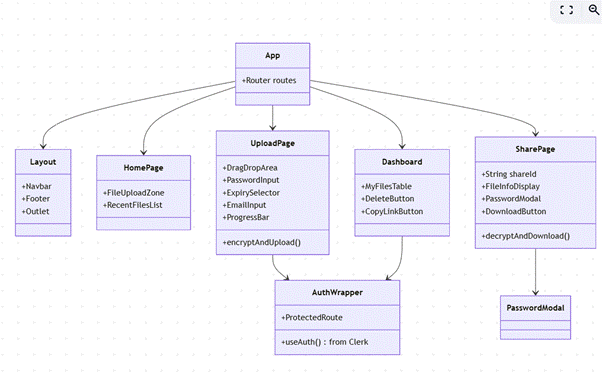
|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Category** | **Requirement** | **Metric** |
| **NFR01** | Performance | Upload/Download speed shall be limited only by user’s network (no server throttling). | >50 MB/s possible |
| **NFR02** | Scalability | System shall handle 10,000 concurrent uploads with horizontal scaling. | Tested under load |
| **NFR03** | Security | **Zero-knowledge**: server never sees plaintext or encryption keys. | Verified by design |
| **NFR04** | Security | All communications shall use HTTPS with HSTS. | Mandatory |
| **NFR05** | Usability | First-time user shall complete a share in **< 30 seconds**. | User testing |
| **NFR06** | Reliability | System uptime | SLA |

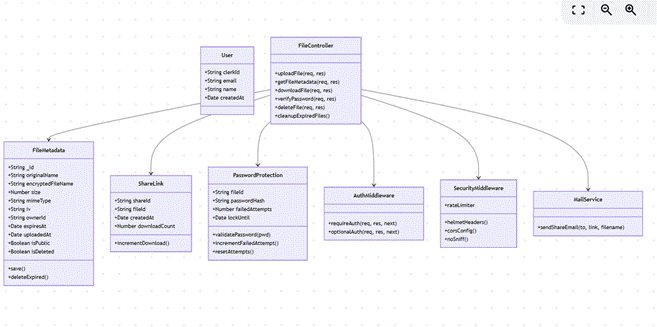
### 3.3 External Interface Requirements

|  |  |  |
| --- | --- | --- |
| **ID** | **Interface** | **Description** |
| **INT01** | User Interface | React + Vite + Tailwind CSS responsive web app. |
| **INT02** | Browser APIs | Web Crypto API, File API, Fetch API, Clipboard API. |
| **INT03** | Backend API | RESTful JSON API (Express.js). |
| **INT04** | Database | MongoDB (default) or PostgreSQL via ORM (Prisma/TypeORM). |
| **INT05** | File Storage | Local filesystem or S3-compatible object storage (configurable). |
| **INT06** | Authentication | JWT + HTTP-only cookies, OAuth2 providers (Google, GitHub). |

## 4. UMLDiagram

1. **Use Case Diagram**
2. **Sequence Diagram – Secure File Transfer A diagram of a software project

   AI-generated content may be incorrect.**
3. **Class Diagram**

****

1. **ER Diagram**

## 5. Feasibility Analysis

### 5.1 Technical Feasibility

The technologies required for this project — **Node.js**, **Express API**, **React**, **MongoDB**, and **Web Crypto API** — are mature, open-source, and widely supported.  
 Client-side AES encryption is reliable and well-documented.  
 File upload and download workflows are technically straightforward and achievable on a local machine.

Since the application does not require advanced GPU processing or heavy server-side computation, development and testing can be performed easily on a standard laptop or desktop.

**→ Highly Feasible**

### 5.2 Resource Feasibility

Since the system is being developed by a **single developer**, the resource requirements are minimal:

* **Development Effort:** One developer can build the MVP within **6–10 weeks**, depending on scope (UI, authentication, encryption, and file management features).
* **Hardware Requirements:** A normal development machine (8–16 GB RAM) is sufficient for running Node.js, React, and MongoDB locally.
* **Infrastructure Requirements:** Initial deployment can be done on a **low-cost VPS** (e.g., Vultr, DigitalOcean).  
   Storage-heavy components can later be migrated to **S3**, **Backblaze B2**, or similar services as needed.

Because you are the sole developer, the project timeline depends on your availability but does not require a large team.

## 6. Other Requirements

### 6.1 Security Requirements

* Encryption keys never leave the browser.
* Password hashes use Argon2id (stronger than bcrypt).
* Rate limiting on all public and authenticated endpoints.
* CSP, CORS, and helmet.js middleware for hardened Express server.
* Regular dependency scanning (npm audit, Snyk).

### 6.2 Software Quality Attributes

* **Correctness:** Encryption/decryption verified with test vectors.
* **Robustness:** Graceful handling of network failures, especially during large file transfers.
* **Extendability:** Modular design to allow for easy integration of new storage backends or OAuth providers.
* Allows commercial use and self-hosting.

## 7. Appendix

### 7.1 Glossary

* **Zero-knowledge:** A system design where the server cannot access the file contents or the encryption keys.
* **E2EE:** End-to-End Encryption.
* **IV:** Initialization Vector (a non-secret value, stored with the file, required for decryption).
* **Argon2id:** A modern, highly secure password hashing function.