Title: Peer-to-Peer File Sharing System with User Authentication and Transfer History

Table of Contents:

1. Introduction

1.1 Purpose The purpose of this document is to define the software requirements for a Peer-to-Peer (P2P) File Sharing System with user authentication and transfer history. The intended audience includes developers, testers, and stakeholders involved in the project.

1.2 Scope This system will enable users to:

- Authenticate using a secure login system.
- Transfer files directly using a P2P mechanism.
- View a secure log of file transfers including names, sizes, and timestamps.
- Interact through a user-friendly web interface. This system is intended for personal and small business environments.

1.3 Definitions, Acronyms, and Abbreviations

- P2P: Peer-to-Peer
- JWT: JSON Web Token
- API: Application Programming Interface
- UI: User Interface
- DB: Database
- WebRTC: Web Real-Time Communication

1.4 References

- WebRTC Documentation: https://webrtc.org/
- MongoDB Documentation: https://www.mongodb.com/docs/

1.5 Overview The document provides a structured approach to defining the system's features, functionality, and constraints. It is organized into sections detailing the product description, specific requirements, system architecture, and constraints.

---

2. Overall Description

2.1 Product Perspective The system is a decentralized file-sharing application that enables users to transfer files securely without reliance on a central server. The backend will facilitate authentication, session management, and logging of transfer history without acting as a file storage unit.

Two key advantages include:

- **Decentralized Control:** Users retain full ownership of their files without a central server.
- **Low Latency Transfers:** Direct peer-to-peer connections allow fast file exchanges compared to traditional cloud storage solutions.

2.2 Product Functions

- Secure user authentication via JWT ensures that only registered users can access the system.
- Direct file transfers using WebRTC remove dependency on a central file repository.
- Transfer history logging enables users to track files exchanged, including metadata such as timestamps and participants.

Two additional functionalities include:

- **Real-Time Notifications:** Users receive updates on transfer status to confirm delivery.
- **Session Persistence:** File transfers resume automatically in case of brief network interruptions.

2.3 User Characteristics Users are expected to have basic web navigation skills. No advanced technical expertise is required. However, users should be aware of privacy considerations when transferring sensitive files.

Two additional characteristics include:

- **Frequent Use Cases:** Users are likely to use the system for document sharing, media file transfers, or collaborative work projects.
- **Platform Independence:** The system is designed to be compatible with different operating systems and modern browsers.

2.4 Constraints

- Browser compatibility is limited to those supporting WebRTC, such as Chrome, Firefox, and Edge.
- The backend primarily serves as a signaling and logging server, meaning that files are never stored on the server.

Two additional constraints include:

- **Network Dependency:** A stable internet connection is required for seamless file transfers.

- **Limited Storage:** Users are responsible for managing storage on their own devices.

2.5 Assumptions and Dependencies

- Users have internet access to establish P2P connections.
- Supported browsers include Chrome, Firefox, and Edge.

Two additional assumptions include:

- **Authentication Services:** The system relies on third-party authentication libraries such as JWT.
- **WebRTC Library Updates:** Browser updates may introduce changes to WebRTC implementations.

---

3. Specific Requirements

3.1 External Interfaces

- User login/register page will authenticate users before file transfers.
- File transfer interface will display available peers and allow file selection.
- Transfer history page enables users to review past transactions.

Two additional interface features include:

- **Drag-and-Drop Functionality:** Users can upload files by dragging them onto the interface.
- **File Preview:** A preview option before sending allows users to confirm file details.

3.2 Functions

- Secure user authentication using hashed passwords and JWT-based session management.
- Direct peer-to-peer file transfers using WebRTC to ensure speed and efficiency.
- Transfer logs with timestamps, filenames, sender, and receiver details.

Two additional functional requirements include:

- **Searchable Transfer Logs:** Users can filter transfer history by date, file name, or recipient.

- **Download Confirmation:** Recipients confirm downloads before logs are updated.

3.3 Performance Requirements

- Login and transfer history retrieval should take no more than 2 seconds.
- Support for concurrent file transfers without noticeable delays.

Two additional performance goals include:

- **Minimal Latency:** Transfers should maintain a latency of under 500ms for real-time interactions.
- **Scalability:** The system should efficiently handle increasing numbers of active users.

3.4 Logical Database Requirements

- User Table: Contains unique user ID, username, and hashed passwords.
- Transfer Table: Stores file metadata including file name, size, timestamp, sender, and receiver.

Two additional database requirements include:

- **Indexing for Faster Queries:** Optimized database queries improve retrieval speed.
- **Encryption of Sensitive Data:** User credentials and transfer logs should be encrypted.

3.5 Design Constraints

- WebRTC will be used for real-time P2P communication, ensuring high performance.
- MongoDB will store user data and file transfer logs without saving actual files.

Two additional design constraints include:

- **Secure Storage Policies:** The system should comply with best practices for data security.
- **API Rate Limits:** Prevent excessive API calls to avoid server overload.

3.6 Software System Quality Attributes

- Security: JWT-based authentication enhances security.
- Scalability: Cloud deployment ensures system expansion.

- Usability: A responsive UI provides an intuitive experience.

Two additional quality attributes include:

- **Maintainability:** Modular code architecture simplifies updates.
- **Availability:** High uptime ensures accessibility at all times.

3.7 Object-Oriented Models

- Architecture Diagram illustrates system components.
- Class Diagram defines relationships between key entities.
- State and Collaboration Diagrams model user interactions.
- Activity Diagrams depict concurrent and distributed file-sharing actions.

Two additional modelling aspects include:

- **Error Handling Workflow:** Defines procedures for handling failed transfers.
- **Session Management Flow:** Shows token validation and expiration handling.

---

4. Appendices
    - **Security Best Practices:** Guidelines on secure authentication, encryption methods, and risk mitigation strategies.
    - **Technology Stack References:** Links to official documentation for WebRTC, MongoDB, Node.js, and React.js.
    - **Regulatory Compliance Notes:** Overview of any relevant data protection laws (e.g., GDPR, CCPA) that may impact system development.
    - **Mock-up Designs:** Wireframes and early UI/UX concepts.
5. Index
    - **Authentication:** JWT, user sessions, and credential hashing.
    - **File Transfer:** WebRTC implementation and P2P connection management.
    - **Database:** MongoDB structure, data encryption, and indexing techniques.
    - **System Performance:** Scalability considerations and performance metrics.
    - **Error Handling:** Fail-safe mechanisms, retry logic, and logging procedures.