1. Tech Stack Choices

1.1. Frontend – React

React is a popular JavaScript library developed by Facebook for building dynamic and responsive web applications. Focusing on the creation of reusable UI components, React allows developers to efficiently manage and update user interfaces. By utilizing a virtual DOM, React enhances performance, ensuring fast rendering of components when data changes occur. This approach minimizes direct interaction with the actual DOM, reducing costly operations. React's component-based architecture promotes code reusability and scalability, making it ideal for complex applications. Its unidirectional data flow and state management, often paired with tools like Redux, facilitate predictable and maintainable code. React's ecosystem supports extensive tooling and community support.

1.2. Backend – Flask

Flask is a lightweight web framework for Python, designed for simplicity and flexibility. As a micro-framework, it provides essential tools to build web applications without imposing dependencies or specific project structures. Flask emphasizes quick development with a modular design, allowing developers to choose additional components as needed. Its minimalistic core enables easy integration with databases, APIs, and other extensions. Flask uses Jinja2 for templating and Werkzeug for request handling. Ideal for small to medium projects, it offers great customization potential. Flask's active community and extensive documentation make it accessible for beginners and efficient for experienced developers aiming for rapid prototyping.

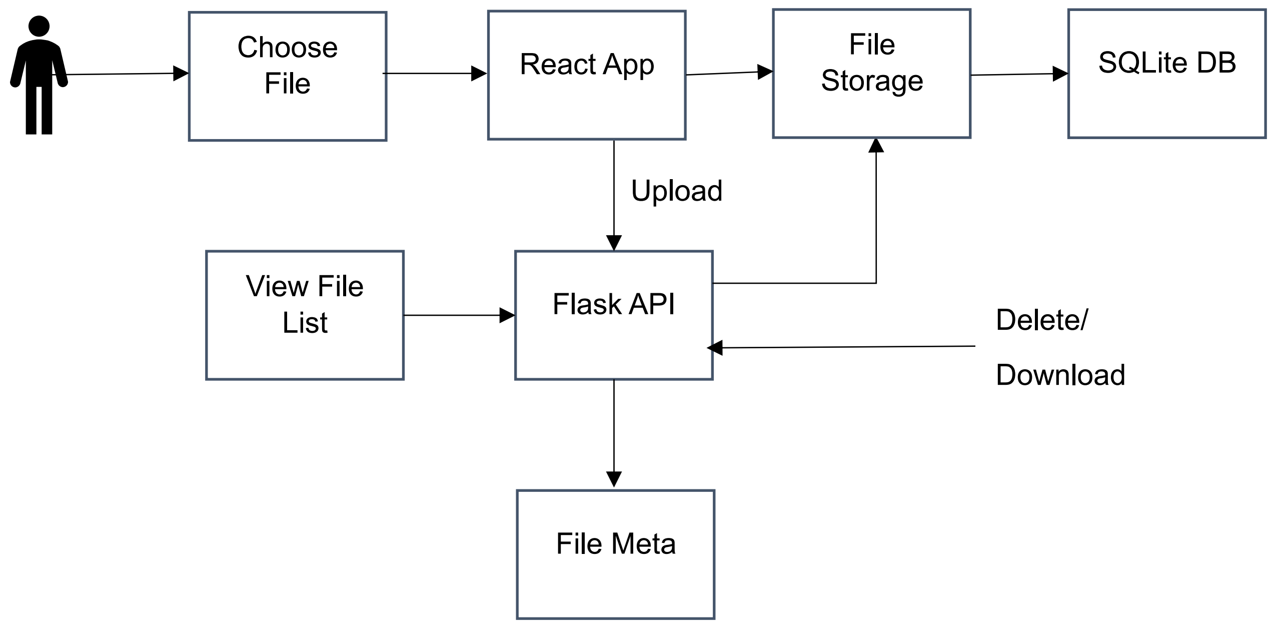
1.3. Database – SQLite3

SQLite3 is a self-contained, serverless, and zero-configuration SQL database engine, widely used for embedded database management. It's ideal for applications needing a lightweight, disk-based database without the complexity of a full DBMS. SQLite3 stores the entire database in a single file, making it highly portable and easy to manage. Despite its simplicity, it supports essential features like transactions, subqueries, triggers, and more. Its public domain license allows free usage for commercial and private purposes. Commonly used in mobile applications, desktop software, and small to medium web applications, SQLite3 balances performance and functionality with minimal overhead and setup requirements.

1.4. Scaling Considerations:

To support 1,000 users, upgrade to PostgreSQL for robust data handling and scalability. Implement caching with tools like Redis to improve performance and reduce database load. Scale the backend by deploying multiple Flask instances behind a load balancer, ensuring efficient request distribution and improved availability, while considering horizontal scaling to handle increased traffic effectively.

2. Architecture Overview



3. API Specification

Endpoint	Method	Description
/documents/upload	POST	Upload a PDF
/documents	GET	List all documents
/documents/:id	GET	Download a file
/documents/:id	DELETE	Delete a file

4. Data Flow Description

4.1. File Upload:

- User selects a PDF and submits through the frontend.
- React sends a POST request to /documents/upload with the file data.
- Flask handles the request, saves the file, and stores metadata in SQLite.
- Response is sent back to the frontend with confirmation.

4.2. File Download:

- User clicks download link/button.
- React sends a GET request to /documents/:id.

- Flask retrieves the file and sends it as a response.
- The file is downloaded to the user’s device.

5. Assumptions

- Maximum file size is 10MB.
- No user authentication implemented.
- Adequate server storage for PDF files.
- Concurrency limited by Flask’s built-in server, suitable for development purposes only.