Digital Document Permission Web Application

MAIN FEATURE:

upload, request, and grant access to digital documents securely

1. User Authentication and Role Management: Admin, Requestor, Login/Signup functionality
2. Document Management: Upload documents in different formats like PDF, images etc.
3. Permission Workflow: Request for document, Approval/Rejection, Notifications
4. Digital Signatures: Enable signing documents using APIs like DocuSign or Aadhaar eSign.
5. Access Control and Sharing: Provide restricted access using access links and QR
6. Audit Trails: Maintain logs of document uploads, access requests, approvals, and downloads.
7. Security Features: Data encryption and secure file transfer
8. Dashboard: Provide a dashboard for users to manage their documents and permissions.

------ TECH STACK------

Frontend:

1. HTML, CSS, JavaScript
2. Frontend Framework: React.js, Angular.js or Vue.js
3. UI Framework: Bootstrap or Material-UI

Backend:

1. Programming Language: Node.js, Python, PHP
2. Frameworks: Express.js (if using Node.js).
3. Database: Relational: MySQL, PostgreSQL (for structured data like user roles, document metadata). NoSQL: MongoDB (if you need flexibility for handling diverse document data).
4. Authentication: JWT (JSON Web Tokens) , OAuth2.0
5. File Storage:

   Cloud Storage: AWS S3, Google Cloud Storage, or Azure Blob Storage.

   Local Storage: For development/testing purposes.
6. Digital Signature API: DocuSign, Adobe Sign, or India-specific Aadhaar eSign API.
7. Search and Metadata Management: ElasticSearch for fast document searches.

8. DevOps & Hosting: AWS, Google Cloud, Git/Github (for version control)

---- BUILDING THE APPLICATION----

1. Set Up the Development Environment:

- Install Node.js, Python, or PHP depending on the backend.
- Use a package manager (npm, pip, or Composer) to install dependencies.

2. Develop the Backend:

- Create APIs for:
  - User authentication and role management.
  - Document upload and metadata storage.
  - Permission request workflow.
- Use RESTful or GraphQL APIs.

3. Develop the Frontend:

- Create user interfaces for:
  - Login/Signup.
  - Document upload and management.
  - Access request and approval workflows.
- Use state management (e.g., Redux for React).

4. Integrate File Storage:

- Use cloud storage APIs (e.g., AWS SDK for S3).
- Encrypt documents before upload (use libraries like CryptoJS).

5. Implement Notification System:

- Use WebSockets for real-time updates or third-party services like Twilio for SMS/email.

6. Add Security Features:

- Implement HTTPS using SSL certificates.
- Validate user inputs to prevent SQL Injection and XSS attacks.
- Use hashing algorithms (e.g., bcrypt) for password storage.

7. Test the Application:

- Perform unit testing for APIs using tools like Postman.
- Use Selenium or Cypress for end-to-end frontend testing.

8. Deploy the Application:

- Use Docker for containerization.
- Deploy the application on a cloud platform.

-----TOOLS AND LIBRARIES---------

Frontend Tools:

- React/Angular CLI for scaffolding.
- Axios or Fetch API for HTTP requests.

Backend Tools:

- ORM: Sequelize (for Node.js) or SQLAlchemy (for Python).
- Nodemailer (Node.js) for sending emails.

File Storage Tools:

- AWS SDK for S3.
- Google Cloud Storage client libraries.

Testing Tools:

- Jest (for JavaScript testing).
- Postman for API testing.

➔ Points to consider:

Scalability: Use cloud services and optimize queries for large-scale applications.

Compliance: Ensure compliance with data privacy laws like GDPR or India’s IT Act.

Documentation: Write proper documentation for developers and users.