

To implement the functionality for your document-sharing platform using **AES and post-quantum cryptography (PQC)** with **Next.js for the frontend** and **Django for the backend**, here's how each component will come together:

## 1. Frontend (Next.js)

- **File Upload Interface:** You will create a form where users can upload documents (e.g., exam papers). This form will be integrated into Next.js pages, allowing users to drag and drop or browse for files to upload.
- **User Authentication:** Before uploading, users will authenticate themselves. This can be implemented using JWT (JSON Web Tokens) or a session-based system, with authentication handled by Django.
- **Secure Key Exchange:** Once the document is uploaded, the frontend will interact with the backend API to initiate the encryption process using **PQC** for secure key exchange.
- **Document Sharing Interface:** Provide an easy-to-use interface where users can share documents securely with others. This will include selecting recipients and possibly adding expiration dates or view/download restrictions.


## 2. Backend (Django)

- **AES Encryption:** After the document is uploaded, the backend will first generate a random symmetric key, which will be used to encrypt the document using **AES**. AES is ideal for this task as it's fast and efficient for encrypting large files.

```
python Copy code  
  
from Crypto.Cipher import AES  
from Crypto.Random import get_random_bytes  
  
def encrypt_file(file_data):  
    key = get_random_bytes(32) # AES-256 key  
    cipher = AES.new(key, AES.MODE_GCM)  
    ciphertext, tag = cipher.encrypt_and_digest(file_data)  
    return key, ciphertext, cipher.nonce, tag
```

**Post-Quantum Cryptography:** After AES encrypts the document, the randomly generated AES key itself will be encrypted using a **post-quantum algorithm**, like **Kyber** (a lattice-based cryptographic algorithm).

python

 Copy code

```
from kyber import Kyber512

def encrypt_aes_key(aes_key, recipient_public_key):
    kyber = Kyber512()
    encrypted_key = kyber.encrypt(aes_key, recipient_public_key)
    return encrypted_key
```

- This ensures that the AES key, which protects the document, remains secure even against future quantum attacks.
- **Document Storage:** The encrypted document and its associated metadata (e.g., encrypted AES key, recipient info) are then stored securely in the backend, using a database like PostgreSQL or a file storage service.
- **Decryption Process:** When a user requests to access a shared document, they will use their private key to decrypt the AES key first (PQC decryption). After retrieving the AES key, they can decrypt the document.

### 3. Encryption Workflow

- **Encryption on Upload:**
  - User uploads a file via Next.js frontend.
  - Django backend generates a symmetric AES key and encrypts the file with it.
  - The AES key is then encrypted using a recipient's public post-quantum key (e.g., Kyber).
  - The encrypted file, AES key, and necessary metadata (such as nonce, tag, recipient info) are stored securely in the database.
- **Decryption on Download:**
  - When a user requests the file, the AES key is decrypted using their private post-quantum key.
  - The document is then decrypted with AES using the decrypted symmetric key.

### 4. Integration Between Next.js and Django

- **APIs:** The Next.js frontend will interact with Django through RESTful APIs or GraphQL endpoints. Some important API routes include:
  - POST /api/upload/: To upload a document for encryption.
  - GET /api/download/:doc\_id/: To download and decrypt a document.
- **File Encryption Status:** Implement real-time feedback (e.g., using WebSockets) on the encryption progress and successful sharing of documents.

### 5. User Experience and Security

- **Frontend UI/UX:** The Next.js frontend will handle interactions smoothly, showing the encryption progress and ensuring a user-friendly flow.
- **Security Features:**
  - **Private and Public Key Generation:** Each user will have a pair of post-quantum private and public keys generated upon account creation.
  - **Document Access Control:** Only authorized users (with the correct keys) will be able to decrypt and access shared documents.