1. Test Cases for All Endpoints

POST /ops/login

Test Cases:

- Valid credentials → 200 OK + access token
- Invalid password → 401 Unauthorized
- Non-existent user → 401 Unauthorized

POST /ops/upload

Test Cases:

- Valid JWT (ops user) + valid file (.docx, .pptx, .xlsx) → 200 OK
- Invalid JWT / No token → 401 Unauthorized
- Client user → 403 Forbidden
- Upload invalid file type (.exe, .pdf) → 400 Bad Request

POST /client/signup

Test Cases:

- New user → 200 OK + verification URL
- Duplicate email → 400 Bad Request
- Invalid email format → 422 Validation Error

GET /client/verify

Test Cases:

- Valid token → 200 OK → sets user as verified
- Invalid/expired token → 400/401
- Repeated use of same token → 400 or 409

POST /client/login

Test Cases:

- Valid email + password (verified user) → 200 OK + access token
- Unverified email → 403 Forbidden
- Invalid credentials → 401 Unauthorized

GET /client/files

Test Cases:

- Valid client token → 200 OK + file list
- Invalid token → 401 Unauthorized
- Ops user token → 403 Forbidden

GET /client/download-file/{file_id}

Test Cases:

- Valid client token + existing file → 200 OK with download URL
- Invalid file ID → 404 Not Found
- Non-client user → 403 Forbidden

GET /client/download/{token}

Test Cases:

- Valid client token + matching token → file download
- Invalid token → 401 Unauthorized
- Token from another user → 403 Forbidden
- Expired token → 401 Unauthorized
- File not found → 404

GET / (Read Root)

Test Cases:

- Always accessible → 200 OK
- Response: { "message": "API is working" }

GET /health

Test Cases:

- Always accessible → 200 OK
- Response: { "status": "healthy" }

2. Plan on deploying this to the production environment

1. Containerize the project using Docker

- Write a Dockerfile to run
  the app with uvicorn (optionally with Gunicorn for multi-worker support).
- Docker ensures portability and easy deployment.

2. Move secrets to environment variables

- Store sensitive values (SECRET_KEY, DB credentials, email credentials) in a .env file.
- Use python-dotenv to load them, or configure
  them as environment variables in Docker/cloud.

3. Deploy to a cloud hosting service

- Beginner-friendly: Render, Railway,
  Fly.io (easy to connect GitHub and auto-deploy).
- More control: AWS EC2, DigitalOcean, or Heroku.
- Set up auto-deploy from your GitHub repo.

4. Store uploaded files on a cloud bucket (e.g., S3)

- For production, use Amazon S3 or similar for file storage.
- Generate pre-signed URLs for secure downloads.
- For now, local storage is used, which is fine for development.

5. Add NGINX as a reverse proxy

- Use NGINX to:
- Handle HTTPS (SSL)
- Forward requests to your FastAPI app
- Optionally add caching, rate limiting, etc.

6. Security practices already implemented

- File type checks for uploads
- Download URLs are secure, token-based, and expire after use
- JWT required for all protected routes
- Passwords hashed with bcrypt (passlib)
- Only verified clients can download files

7. Basic monitoring is in place

- / route shows API is running
- /health route for uptime monitoring

8. Next phase improvements

- Add CI/CD with GitHub Actions for automated testing and deployment
- Use PostgreSQL (or managed MySQL) for production
- Add centralized logging and error tracking (e.g., Sentry)
- Move file storage fully to S3 or another cloud storage provider