✅ Problem Statement

Build a secure file-sharing system with role-based access (client/ops), allowing clients to upload files, and ops to access/download them securely via verification links — with email-based user verification and security features.

🧠 Core Requirements

Feature	Description
🔐 User Registration	With email verification
👨‍💼 Roles	`client` and `ops` users
📤 File Upload	Only by clients
📥 File Access	Only by ops, with secure download links
📧 Email Integration	For verification and file access links
💄 Frontend (Optional)	Clean, modern UI if time permits

🧩 Tech Stack Used

- **Backend:** FastAPI
- **Database:** PostgreSQL (via SQLAlchemy ORM)
- **Auth:** JWT tokens (access + refresh)
- **Security:** Password hashing, role-based access, time-limited download links
- **Mail:** FastAPI Email or SMTP for email sending
- **Frontend (if any):** Basic Tailwind/React optional
- **Deployment (future):** Docker/Render/Vercel optional

🚀 Step-by-Step Implementation

1. Project Setup

- Set up FastAPI project structure with:
  - `main.py`, `routers/`, `models/`, `schemas/`, `auth/`, `utils/`, `database/`
- Installed dependencies: `fastapi`, `sqlalchemy`, `passlib`, `python-jose`, `email-validator`, `python-multipart`, `alembic` (for migrations), etc.

---

2. Database Models

Used SQLAlchemy to define models:

`User`

```
python
CopyEdit
class User(Base):
    id = Column(Integer, primary_key=True)
    email = Column(String, unique=True, index=True)
    hashed_password = Column(String)
    role = Column(Enum("client", "ops", name="user_roles"))
    is_verified = Column(Boolean, default=False)
    created_at = Column(DateTime, default=datetime.utcnow)
```

`File`

```
python
CopyEdit
class File(Base):
    id = Column(Integer, primary_key=True)
    filename = Column(String)
    file_path = Column(String)
    uploaded_by = Column(Integer, ForeignKey("users.id"))
    uploaded_at = Column(DateTime, default=datetime.utcnow)
```

---

3. User Signup & Verification

Signup Flow:

- Client registers → password is hashed
- Email verification token (JWT or random) is generated
- Email sent with link:
  `http://yourapp.com/verify-email?token=abc123...`

Email Token:

```
python
CopyEdit
token_data = {
    "sub": user.email,
    "exp": datetime.utcnow() + timedelta(hours=24)
}
token = jwt.encode(token_data, SECRET_KEY, algorithm="HS256")
```

`/verify-email` Endpoint:

- Verifies token, activates `user.is_verified`

---

4. Login and JWT Authentication

- User logs in with email & password

- On success, returns:
  - `access_token` (short-lived)
  - `refresh_token` (longer-lived)
- Auth protected routes use dependency `get_current_user`

---

5. Role-Based Access Control (RBAC)

Used decorators like:

```
python
CopyEdit
def require_role(required_role: str):
    def wrapper(current_user: User = Depends(get_current_user)):
        if current_user.role != required_role:
            raise HTTPException(status_code=403, detail="Access denied")
        return current_user
    return wrapper
```

Used as:

```
python
CopyEdit
@router.post("/upload")
def upload_file(..., current_user: User = Depends(require_role("client"))):
```

---

6. Secure File Upload (Client only)

- Clients can upload `.pdf`, `.zip`, etc.
- Files stored in `/uploads/` folder
- Filename + uploader ID is stored in DB

```
python
CopyEdit
with open(file_path, "wb") as f:
    f.write(file.file.read())
```

---

7. Ops Download with Secure Link

Flow:

- Ops can request download link for any file
- A signed token is generated (expires in X minutes)
- A download link is returned:
  `http://yourapp.com/download?token=abc...`

Token example:

```
python
CopyEdit
```

```
token_data = {
    "file_id": file.id,
    "exp": datetime.utcnow() + timedelta(minutes=10)
}
token = jwt.encode(token_data, SECRET_KEY, algorithm="HS256")
```

Download Endpoint:

```
python
CopyEdit
@router.get("/download")
def download_file(token: str):
    payload = jwt.decode(token, SECRET_KEY, algorithms=["HS256"])
    file_id = payload.get("file_id")
    # fetch file by ID, return with FileResponse
```

---

8. Email Notification (Optional)

- When client uploads a file, ops get notified via email with download link
- Email integration via `smtplib` or `FastAPI-Mail`

---

9. Frontend UI (Optional)

- A minimal Tailwind or React frontend to:
  - Register/Login
  - Upload Files (client)
  - View & Download Files (ops)
  - Show email verification pending screen

🔐 Security Features

Feature	Explanation
Password Hashing	`passlib.bcrypt` to hash + verify passwords
Email Verification	Prevents fake signups
JWT Tokens	Access control for protected routes
Time-limited Download Links	Stops replay/download abuse
Role Checks	Clients can't access ops features

📁 Project Structure (Simplified)

```
pgsql
CopyEdit
secure_file_sharing/
|
```

```
├── main.py
├── routers/
│   ├── auth.py
│   ├── file.py
│   └── verify.py
├── models/
│   └── user.py, file.py
├── schemas/
│   └── user.py, file.py
├── utils/
│   ├── auth.py (token funcs)
│   └── email.py (email sender)
├── database.py
└── uploads/   ← saved files
```

📌 Conclusion: What We Built

We delivered a **secure, email-verified, role-based file sharing system** that:

- Uses FastAPI with best practices
- Separates concerns clearly
- Ensures security at every level
- Easily extendable (e.g., frontend, notifications)
- Demonstrates real-world backend skills