File Sharing system:-

Ques – 1) Write Test Cases for the our project File Sharing System?

### Ans:- Test Case Categories

1. **User Signup and Login**
2. **File Upload (for Ops User)**
3. **File Listing**
4. **File Download**

**Test Cases**

**1. User Signup and Login**

**Signup (POST /signup)**

* **Valid Input**:
  + Input: {"email": "testuser@example.com", "password": "password123"}
  + Expected Output: 201 Created, {"message": "User created"}
* **Missing Email or Password**:
  + Input: {"email": "testuser@example.com"}
  + Expected Output: 400 Bad Request, {"message": "Missing email or password"}

**Login (POST /login)**

* **Valid Credentials**:
  + Input: {"email": "testuser@example.com", "password": "password123"}
  + Expected Output: 200 OK, {"token": "<JWT\_TOKEN>"}
* **Invalid Credentials**:
  + Input: {"email": "testuser@example.com", "password": "wrongpassword"}
  + Expected Output: 401 Unauthorized, {"message": "Invalid credentials"}
* **Non-Existent User**:
  + Input: {"email": "nonexistent@example.com", "password": "password123"}
  + Expected Output: 401 Unauthorized, {"message": "Invalid credentials"}

**2. File Upload (for Ops User)**

**Upload File (POST /upload-file)**

* **Valid Ops User and File Type**:
  + Headers: Authorization: Bearer <JWT\_TOKEN>
  + Input: A valid file (.pptx, .docx, .xlsx)
  + Expected Output: 201 Created, {"message": "File uploaded"}
* **Non-Ops User**:
  + Headers: Authorization: Bearer <JWT\_TOKEN>
  + Input: A valid file (.pptx, .docx, .xlsx)
  + Expected Output: 403 Forbidden, {"message": "Operation not permitted"}
* **Invalid File Type**:
  + Headers: Authorization: Bearer <JWT\_TOKEN>
  + Input: An invalid file type (e.g., .txt)
  + Expected Output: 400 Bad Request, {"message": "Invalid file type"}
* **Missing Token**:
  + Input: A valid file (.pptx, .docx, .xlsx)
  + Expected Output: 401 Unauthorized, {"message": "Token is missing!"}

**3. File Listing**

**List Files (GET /files)**

* **Valid Request**:
  + Headers: Authorization: Bearer <JWT\_TOKEN>
  + Expected Output: 200 OK, List of files in JSON format
* **Missing Token**:
  + Expected Output: 401 Unauthorized, {"message": "Token is missing!"}

**4. File Download**

**Download File (GET /download-file/int:file\_id)**

* **Valid Request and File Exists**:
  + Headers: Authorization: Bearer <JWT\_TOKEN>
  + Expected Output: 200 OK, {"download\_link": "<ENCRYPTED\_URL>"}
* **File Does Not Exist**:
  + Headers: Authorization: Bearer <JWT\_TOKEN>
  + Expected Output: 404 Not Found, {"message": "File not found"}
* **Missing Token**:
  + Expected Output: 401 Unauthorized, {"message": "Token is missing!"}

Ques 2) How do you plan on deploying this to the production Environment?

### Ans:- Steps to Deploy Flask Application to Production:-

**1. Prepare Your Application**

* **Configuration**: Ensure your Flask application (app.py) is configured to work in a production environment. This includes setting DEBUG = False and app.secret\_key for session management.
* **Dependencies**: Prepare a requirements.txt file listing all Python dependencies your application requires. You can generate this file using pip freeze > requirements.txt.
* **Environment Variables**: Ensure sensitive information like database credentials and secret keys are managed using environment variables, not hard-coded in the application.

**2. Choose a Hosting Platform**

Select a hosting platform suitable for your application requirements and budget:

* **Platform as a Service (PaaS)**: Platforms like Heroku, Google App Engine, or AWS Elastic Beanstalk provide managed environments where you can deploy your Flask application easily.
* **Infrastructure as a Service (IaaS)**: Services like AWS EC2, DigitalOcean Droplets, or Google Compute Engine offer more control over the server environment but require more setup.

**3. Setup Database**

* **Database Service**: Use a managed database service like Amazon RDS, Google Cloud SQL, or self-hosted MySQL/PostgreSQL servers. Ensure your application connects securely to the database.
* **Database Schema**: Ensure your database schema is properly defined and any necessary migrations are scripted and ready to run.

**4. Configure Web Server**

* **Web Server**: Use a production-ready web server like Nginx or Apache to serve your Flask application. Configure the web server to proxy requests to your Flask application.
* **WSGI Server**: Deploy your Flask application using a WSGI server like Gunicorn or uWSGI. These servers handle concurrent requests and manage the Flask application lifecycle.

**5. Security Considerations**

* **SSL/TLS**: Enable HTTPS to encrypt data transmitted between clients and your server. Obtain an SSL certificate (e.g., from Let's Encrypt) and configure your web server to use it.
* **Firewall and Access Control**: Configure firewall rules to restrict access to your server and database. Use SSH keys for secure access to server instances.

**6. Logging and Monitoring**

* **Logging**: Implement logging within your Flask application to record important events and errors. Configure logs to be stored centrally or in a dedicated logging service.
* **Monitoring**: Set up monitoring for your application and infrastructure using tools like Prometheus, Grafana, or application-specific monitoring services (e.g., AWS CloudWatch).

**7. Continuous Integration/Continuous Deployment (CI/CD)**

* **Automated Deployment**: Implement CI/CD pipelines using tools like Jenkins, GitLab CI/CD, or GitHub Actions. Automate testing, building, and deploying your Flask application to ensure reliability and efficiency.

**8. Backup and Disaster Recovery**

* **Backup Strategy**: Implement regular database backups and server snapshots to prevent data loss in case of failures.
* **Disaster Recovery Plan**: Define a plan for restoring services quickly in case of server or application failures. Test the plan regularly.

**9. Load Testing and Optimization**

* **Load Testing**: Conduct load testing to simulate high traffic conditions and identify performance bottlenecks. Optimize database queries, caching strategies, and application code as needed.

**10. Documentation and Maintenance**

* **Documentation**: Document your deployment process, configuration settings, and any troubleshooting steps. Maintain up-to-date documentation for future reference.
* **Maintenance**: Regularly update dependencies, apply security patches, and review application performance. Monitor server logs and user feedback to improve the application.