INTERNSHIP REPORT AT FUTURE INTERNS

Task 3 : Secure File Sharing System

Report Prepared By : **ISSA HASSAN YOUSSOUF**

Internship period : 17/07/2025 to 17/08/2025
Company : Future Interns

INTRODUCTION

As part of my cybersecurity internship offered by Future Interns, I carried out a project called "Secure File Sharing System", whose main objective is to design and deploy a web application allowing file sharing in a secure way.

This project is part of a DevSecOps approach, where security is integrated from the development phase. The application was developed in Python using the Flask micro-framework, and implements robust cryptographic mechanisms to ensure the confidentiality, integrity, and access control of the files exchanged.

The central idea is that when a user deposits a file on the server, it is immediately encrypted using the Advanced Encryption Standard (AES) algorithm and then stored securely. Upon download, the file is decrypted on the fly and returned to the authorized user. An authentication token system is set up to protect the various operations (upload, download, deletion).

The project covers several key aspects of cybersecurity:

- Data privacy with AES-256 symmetric encryption (GCM mode);
- Secure the encryption key via environment variables;
- Authentication of actions by a secure token;
- Event logging (upload, download, delete logs);
- Protection against unauthorized access (file validation, secure names).

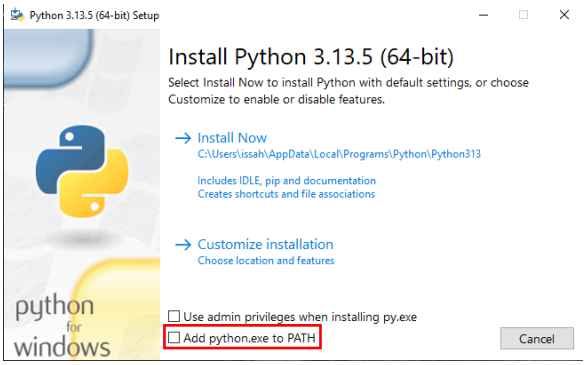
This application is a concrete demonstration of how security principles can be integrated into a web application, and has allowed me to strengthen my skills in secure development, applied cryptography and defensive software architecture.

TOOLS USED

- Python Flask: API Server
- PyCryptodome: Cryptographic Library
- Postman: To test endpoints
- Visual Studio Code: For HTML code structure

INSTALLER PYTHON

- Go to the search bar: https://www.python.org/downloads/
- Download and install and choose (Add Python to PATH).



CREATE THE PROJECT FOLDER AND VIRTUAL ENVIRONMENT

Open PowerShell in the folder:

- Create the Future folder

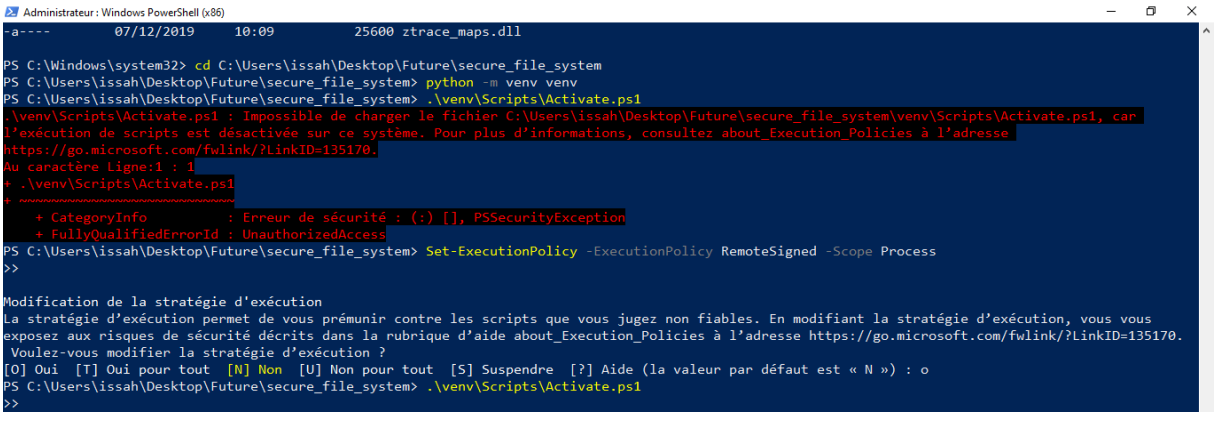
```
mkdir C:\Users\issah\Desktop\Future\secure_file_system
```

```
cd C:\Users\issah\Desktop\Future \secure_file_system
```

- Create and activate venv

```
python -m venv venv
```

```
.\venv\Scripts\Activate.ps1
```



BASIC FILES AND FOLDERS

- Create the tree

```
mkdir uploads, decrypted, keys, templates
```

```
(venv) PS C:\Users\issah\Desktop\Future\secure_file_system> mkdir uploads, decrypted, keys, templates


    Répertoire : C:\Users\issah\Desktop\Future\secure_file_system


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        10/08/2025     20:16                uploads
d-----        10/08/2025     20:16                decrypted
d-----        10/08/2025     20:16                keys
d-----        10/08/2025     20:16                templates
```

- Create empty files

```
ni app.py, templates\index.html, requirements.txt, run.ps1, .gitignore -Force
```

```
(venv) PS C:\Users\issah\Desktop\Future\secure_file_system> ni app.py, templates\index.html, requirements.txt, run.ps1, .gitignore -Force


    Répertoire : C:\Users\issah\Desktop\Future\secure_file_system


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        10/08/2025     20:16              0 app.py


    Répertoire : C:\Users\issah\Desktop\Future\secure_file_system\templates


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        10/08/2025     20:16              0 index.html


    Répertoire : C:\Users\issah\Desktop\Future\secure_file_system


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        10/08/2025     20:16              0 requirements.txt
-a----        10/08/2025     20:16              0 run.ps1
-a----        10/08/2025     20:16              0 .gitignore
```

INSTALL OUTBUILDINGS

Fill in `requirements.txt` then:

```
pip install -r requirements.txt
```

```
(venv) PS C:\Users\issah\Desktop\Future\secure_file_system> pip install -r requirements.txt
>>

[notice] A new release of pip is available: 25.1.1 -> 25.2
[notice] To update, run: python.exe -m pip install --upgrade pip
```

GENERATE AND STORE SECRETS (master key + admin token)

- Generate Master Key (base64)

In PowerShell (during venv enabled): `python -c "import base64,os; print(base64.b64encode(os.urandom(32)).decode())"`

```
(venv) PS C:\Users\issah\Desktop\Future\secure_file_system> python -c "import base64,os;print(base64.b64encode(os.urandom(32)).decode())"
>>
i5q4rGvQoHWYzzu7eFJpWiVS3osvHtDhdeYaAzg38t0=
```

- Create an admin token

```
python -c "import secrets; print(secrets.token_urlsafe(32))"
```

```
(venv) PS C:\Users\issah\Desktop\Future\secure_file_system> python -c "import secrets;print(secrets.token_urlsafe(32))"
>>
7yCzH_rqLCtNmugBQFV-CaEcaRRfGV6q6nTLYTofPIs
```

- Persistent Storage (setx)

Replace `<BASE64_KEY>` and `<ADMIN_TOKEN>` with the generated values:

```
setx FILE_VAULT_MASTER_KEY "<i5q4rGvQoHWYzzu7eFJpWiVS3osvHtDhdeYaAzg38t0=>"
```

```
setx FILE_VAULT_ADMIN_TOKEN "<7yCzH_rqLCtNmugBQFV-CaEcaRRfGV6q6nTLYTofPIs>"
```

```
(venv) PS C:\Users\issah\Desktop\Future\secure_file_system> setx FILE_VAULT_MASTER_KEY "<i5q4rGvQoHWYzzu7eFJpWiVS3osvHtDhdeYaAzg38t0=>"

RÉUSSITE : la valeur spécifiée a été enregistrée.
(venv) PS C:\Users\issah\Desktop\Future\secure_file_system> setx FILE_VAULT_ADMIN_TOKEN "<7yCzH_rqLCtNmugBQFV-CaEcaRRfGV6q6nTLYTofPIs>"

RÉUSSITE : la valeur spécifiée a été enregistrée.
```

Note: `setx` creates the variable for future sessions.

- requirements.txt

Paste this in `requirements.txt`

Then:

```
pip install -r requirements.txt
```

```
(venv) PS C:\Users\issah\Desktop\Future\secure_file_system> pip install -r requirements.txt
>>
Collecting Flask>=2.0 (from -r requirements.txt (line 1))
  Downloading flask-3.1.1-py3-none-any.whl.metadata (3.0 kB)
Collecting pycryptodome>=3.10 (from -r requirements.txt (line 2))
```

```
Using cached blinker-1.9.0-py3-none-any.whl (8.5 kB)
Using cached click-8.2.1-py3-none-any.whl (102 kB)
Using cached itsdangerous-2.2.0-py3-none-any.whl (16 kB)
Using cached jinja2-3.1.6-py3-none-any.whl (134 kB)
Using cached MarkupSafe-3.0.2-cp313-cp313-win_amd64.whl (15 kB)
Downloading werkzeug-3.1.3-py3-none-any.whl (224 kB)
Using cached colorama-0.4.6-py2.py3-none-any.whl (25 kB)
Installing collected packages: waitress, pycryptodome, markupsafe, itsdangerous, colorama, blinker, werkzeug, jinja2, click, Flask
Successfully installed Flask-3.1.1 blinker-1.9.0 click-8.2.1 colorama-0.4.6 itsdangerous-2.2.0 jinja2-3.1.6 markupsafe-3.0.2 pycryptodome-3.23.0 wait
ress-3.0.2 werkzeug-3.1.3

[notice] A new release of pip is available: 25.1.1 -> 25.2
[notice] To update, run: python.exe -m pip install --upgrade pip
```

COMPLETE AND SECURE CODE — `app.py`

This file in `app.py` is the recommended version for a secure POC: AES-GCM, single nonce per file, metadata, token protection, secure filenames, logging.

- Screenshots:

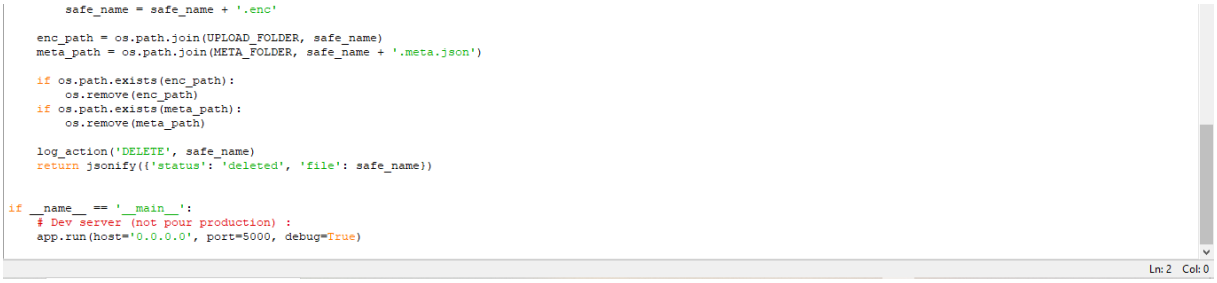
app.py - C:\Users\issah\Desktop\Future\secure_file_system\app.py (3.13.5)

File Edit Format Run Options Window Help

```python
# app.py
import os
import base64
import json
import io
import logging
from datetime import datetime
from flask import Flask, request, send_file, render_template, jsonify, abort
from werkzeug.utils import secure_filename
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes

# --- Config ---
UPLOAD_FOLDER = os.path.join(os.getcwd(), 'uploads')
META_FOLDER = os.path.join(os.getcwd(), 'keys')       # métadonnées par fichier
DECRYPTED_FOLDER = os.path.join(os.getcwd(), 'decrypted')
ALLOWED_EXT = None  # None = autoriser tout ; tu peux restreindre par extension si souhaité

for d in (UPLOAD_FOLDER, META_FOLDER, DECRYPTED_FOLDER):
    os.makedirs(d, exist_ok=True)

# Chargement de la clé maître (base64 dans variable d'environnement)
MASTER_KEY_B64 = os.environ.get('FILE_VAULT_MASTER_KEY')
ADMIN_TOKEN = os.environ.get('FILE_VAULT_ADMIN_TOKEN')

if not MASTER_KEY_B64:
    print("⚠ WARNING: FILE_VAULT_MASTER_KEY not set. Using ephemeral key (NOT for production).")
    MASTER_KEY = get_random_bytes(32)
else:
    try:
        MASTER_KEY = base64.b64decode(MASTER_KEY_B64)
        if len(MASTER_KEY) != 32:
            raise ValueError("Master key must decode to 32 bytes (AES-256).")
    except Exception as e:
        print("Invalid FILE_VAULT_MASTER_KEY:", e)
        raise

if not ADMIN_TOKEN:
    print("⚠ WARNING: FILE_VAULT_ADMIN_TOKEN not set. Set a token in env for protection.")
```

```python
# Logging
logging.basicConfig(filename='access.log', level=logging.INFO, format='%(asctime)s %(message)s')

app = Flask(__name__)


def require_token():
    """Vérifie le token envoyé (header X-Auth-Token ou champ form token)."""
    if not ADMIN_TOKEN:
        return False
    token = request.headers.get('X-Auth-Token') or request.form.get('token')
    return token == ADMIN_TOKEN


def log_action(action, filename):
    ip = request.remote_addr or "local"
    logging.info(f"{action} file={filename} ip={ip}")


# --- Crypto helpers (AES-GCM) ---
# Format stocké : nonce (12 bytes) | tag (16 bytes) | ciphertext
def encrypt_bytes(plaintext: bytes) -> bytes:
    nonce = get_random_bytes(12)
    cipher = AES.new(MASTER_KEY, AES.MODE_GCM, nonce=nonce)
    ciphertext, tag = cipher.encrypt_and_digest(plaintext)
    return nonce + tag + ciphertext


def decrypt_bytes(blob: bytes) -> bytes:
    if len(blob) < (12 + 16):
        raise ValueError("Blob too small")
    nonce = blob[:12]
    tag = blob[12:28]
    ciphertext = blob[28:]
    cipher = AES.new(MASTER_KEY, AES.MODE_GCM, nonce=nonce)
    return cipher.decrypt_and_verify(ciphertext, tag)
```

```python
# --- Routes ---
@app.route('/')
def index():
    return render_template('index.html', token_required=bool(ADMIN_TOKEN))


@app.route('/upload', methods=['POST'])
def upload():
    if ADMIN_TOKEN and not require_token():
        abort(401, description="Missing or invalid X-Auth-Token")

    if 'file' not in request.files:
        return jsonify({'error': 'no file part'}), 400

    file = request.files['file']
    if file.filename == '':
        return jsonify({'error': 'empty filename'}), 400

    # sanitize filename
    original_name = secure_filename(file.filename)
    if original_name == '':
        return jsonify({'error': 'invalid filename after sanitization'}), 400

    data = file.read()
    encrypted_blob = encrypt_bytes(data)

    # saved filename (add .enc to avoid confusion)
    enc_name = original_name + '.enc'
    enc_path = os.path.join(UPLOAD_FOLDER, enc_name)
    with open(enc_path, 'wb') as f:
        f.write(encrypted_blob)

    # metadata
    meta = {
        'original_name': original_name,
        'stored_name': enc_name,
        'uploaded_at': datetime.utcnow().isoformat() + 'Z',
        'uploader_ip': request.remote_addr
    }

    meta_path = os.path.join(META_FOLDER, enc_name + '.meta.json')
    with open(meta_path, 'w', encoding='utf-8') as mf:
        json.dump(meta, mf, ensure_ascii=False, indent=2)

    log_action('UPLOAD', enc_name)
    return jsonify({'status': 'ok', 'stored_name': enc_name, 'original_name': original_name})


@app.route('/files', methods=['GET'])
def list_files():
    if ADMIN_TOKEN and not require_token():
        abort(401, description="Missing or invalid X-Auth-Token")

    files = []
    for fname in os.listdir(UPLOAD_FOLDER):
        if not fname.endswith('.enc'):
            continue
        meta_file = os.path.join(META_FOLDER, fname + '.meta.json')
        meta = {}
        if os.path.exists(meta_file):
            try:
                with open(meta_file, 'r', encoding='utf-8') as mf:
                    meta = json.load(mf)
            except Exception:
                pass
        files.append({'stored_name': fname, 'meta': meta})
    return jsonify(files)


@app.route('/download/<path:stored_name>', methods=['GET'])
def download(stored_name):
    if ADMIN_TOKEN and not require_token():
        abort(401, description="Missing or invalid X-Auth-Token")

    safe_name = secure_filename(stored_name)
    if not safe_name.endswith('.enc'):
        safe_name = safe_name + '.enc'

    enc_path = os.path.join(UPLOAD_FOLDER, safe_name)
    if not os.path.exists(enc_path):

        return jsonify({'error': 'file not found'}), 404

    # load and decrypt
    with open(enc_path, 'rb') as ef:
        blob = ef.read()
    try:
        plaintext = decrypt_bytes(blob)
    except Exception as e:
        return jsonify({'error': 'decryption failed', 'msg': str(e)}), 500

    # obtain original filename from metadata if present
    meta_path = os.path.join(META_FOLDER, safe_name + '.meta.json')
    original_name = None
    if os.path.exists(meta_path):
        try:
            with open(meta_path, 'r', encoding='utf-8') as mf:
                meta = json.load(mf)
                original_name = meta.get('original_name')
        except Exception:
            original_name = None

    if not original_name:
        # fallback
        original_name = safe_name[:-4]

    # stream back without writing to disk
    bio = io.BytesIO(plaintext)
    bio.seek(0)
    log_action('DOWNLOAD', safe_name)
    # Flask 2.0+: use download_name param, older versions use attachment_filename
    return send_file(bio, as_attachment=True, download_name=original_name)


@app.route('/delete/<path:stored_name>', methods=['DELETE'])
def delete(stored_name):
    if ADMIN_TOKEN and not require_token():
        abort(401, description="Missing or invalid X-Auth-Token")

    safe_name = secure_filename(stored_name)
    if not safe_name.endswith('.enc'):
```



```
        safe_name = safe_name + '.enc'

    enc_path = os.path.join(UPLOAD_FOLDER, safe_name)
    meta_path = os.path.join(META_FOLDER, safe_name + '.meta.json')

    if os.path.exists(enc_path):
        os.remove(enc_path)
    if os.path.exists(meta_path):
        os.remove(meta_path)

    log_action('DELETE', safe_name)
    return jsonify({'status': 'deleted', 'file': safe_name})


if __name__ == '__main__':
    # Dev server (not pour production) :
    app.run(host='0.0.0.0', port=5000, debug=True)
```

Ln: 2 Col: 0

Lines of code:

```python
# app.py
import os
import base64
import json
import io
import logging
from datetime import datetime
from flask import Flask, request, send_file, render_template, jsonify, abort
from werkzeug.utils import secure_filename
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes

# --- Config ---
UPLOAD_FOLDER = os.path.join(os.getcwd(), 'uploads')
META_FOLDER = os.path.join(os.getcwd(), 'keys') # metadata per file
DECRYPTED_FOLDER = os.path.join(os.getcwd(), 'decrypted')
ALLOWED_EXT = None # None = allow all; you can restrict by extension if desired

for d in (UPLOAD_FOLDER, META_FOLDER, DECRYPTED_FOLDER):
    os.makedirs(d, exist_ok=True)

# Loading the master key (base64 in environment variable)
MASTER_KEY_B64 = os.environ.get('FILE_VAULT_MASTER_KEY')
ADMIN_TOKEN = os.environ.get('FILE_VAULT_ADMIN_TOKEN')

if not MASTER_KEY_B64:
    print("⚠ WARNING: FILE_VAULT_MASTER_KEY not set. Using ephemeral key (NOT for production).")
    MASTER_KEY = get_random_bytes(32)
else:
    try:
        MASTER_KEY = base64.b64decode(MASTER_KEY_B64)
        if len(MASTER_KEY) != 32:
            raise ValueError("Master key must decode to 32 bytes (AES-256).")
    except Exception as e:
        print("Invalid FILE_VAULT_MASTER_KEY:", e)
        raise
```

```
if not ADMIN_TOKEN:
    print("⚠ WARNING: FILE_VAULT_ADMIN_TOKEN not set. Set a token in
env for protection.")

# Logging
logging.basicConfig(filename='access.log', level=logging.INFO,
format='%(asctime)s %(message)s')

app = Flask(__name__)

def require_token():
    """Checks the sent token (X-Auth-Token header or form token
field)."""
    if not ADMIN_TOKEN:
        return False
    token = request.headers.get('X-Auth-Token') or
request.form.get('token')
    return token == ADMIN_TOKEN

def log_action(action, filename):
    ip = request.remote_addr or "local"
 logging.info(f"{action} file={filename} ip={ip}")

# --- Crypto helpers (AES-GCM) ---
# Format stocké : nonce (12 bytes) | tag (16 bytes) | ciphertext
def encrypt_bytes(plaintext: bytes) -> bytes:
    nonce = get_random_bytes(12)
    cipher = AES.new(MASTER_KEY, AES. MODE_GCM, nonce=nonce)
    ciphertext, tag = cipher.encrypt_and_digest(plaintext)
    return nonce + tag + ciphertext

def decrypt_bytes (blub: bytes) -> bytes:
    if len(blob) < (12 + 16):
        raise ValueError("Blob too small")
 nonce = blob[:12]
    tag = blob[12:28]
    ciphertext = blob[28:]
    cipher = AES.new(MASTER_KEY, AES. MODE_GCM, nonce=nonce)
    return cipher.decrypt_and_verify(ciphertext, tag)

# --- Routes ---
@app.route('/')
def index():
    return render_template('index.html',
token_required=bool(ADMIN_TOKEN))

@app.route('/upload', methods=['POST'])
def upload():
```

```
    if ADMIN_TOKEN and not require_token():
        abort(401, description="Missing or invalid X-Auth-Token")

    if 'file' not in request.files:
        return jsonify({'error': 'no file part'}), 400

    file = request.files['file']
    if file.filename == '':
        return jsonify({'error': 'empty filename'}), 400

    # sanitize filename
 original_name = secure_filename(file.filename)
    if original_name == '':
        return jsonify({'error': 'invalid filename after
sanitization'}), 400

    data = file.read()
    encrypted_blob = encrypt_bytes(data)

    # saved filename (add .enc to avoid confusion)
 enc_name = original_name + '.enc'
    enc_path = os.path.join(UPLOAD_FOLDER, enc_name)
    with open(enc_path, 'wb') as f:
f.write(encrypted_blob)

    # metadata
 meta = {
        'original_name': original_name,
        'stored_name': enc_name,
        'uploaded_at': datetime.utcnow().isoformat() + 'Z',
        'uploader_ip': request.remote_addr
}
    meta_path = os.path.join(META_FOLDER, enc_name + '.meta.json')
    with open(meta_path, 'w', encoding='utf-8') as mf:
        json.dump(meta, mf, ensure_ascii=False, indent=2)

 log_action('UPLOAD', enc_name)
    return on the({'status': 'ok', 'stored_name': enc_name,
'original_name': original_name})

@app.route('/files', methods=['GET'])
def list_files():
    if ADMIN_TOKEN and not require_token():
        abort(401, description="Missing or invalid X-Auth-Token")

    files = []
    for fname in os.listdir(UPLOAD_FOLDER):
        if not fname.endswith('.enc'):
            continue
```

```
  meta_file = os.path.joint(META_FOLDER, fname + '.meta.json')
  meta = {}
        if os.path.exists(meta_file):
            try:
                with open(meta_file, 'r', encoding='utf-8') as mf:
  meta = json.load(mf)
            except Exception:
                pass
  files.append({'stored_name': fname, 'meta': meta})
    return jsonify(files)

@app.route('/download/<path:stored_name>', methods=['GET'])
def download(stored_name):
    if ADMIN_TOKEN and not require_token():
        abort(401, description="Missing or invalid X-Auth-Token")

  safe_name = secure_filename(stored_name)
    if not safe_name.endswith('.enc'):
  safe_name = safe_name + '.enc'

    enc_path = os.path.join(UPLOAD_FOLDER, safe_name)
    if not os.path.exists(enc_path):
        return jsonify({'error': 'file not found'}), 404

    # load and decrypt
    with open(enc_path, 'rb') as ef:
  blob = ef.read()
    try:
        plaintext = decrypt_bytes(blob)
    except Exception as e:
        return jsonify({'error': 'decryption failed', 'msg': str(e)}),
500

    # obtain original filename from metadata if present
    meta_path = os.path.join(META_FOLDER, safe_name + '.meta.json')
  original_name = None
    if os.path.exists(meta_path):
        try:
            with open(meta_path, 'r', encoding='utf-8') as mf:
  meta = json.load(mf)
  original_name = meta.get('original_name')
        except Exception:
  original_name = None

    if not original_name:
        # fallback
  original_name = safe_name[:-4]

    # stream back without writing to disk
```

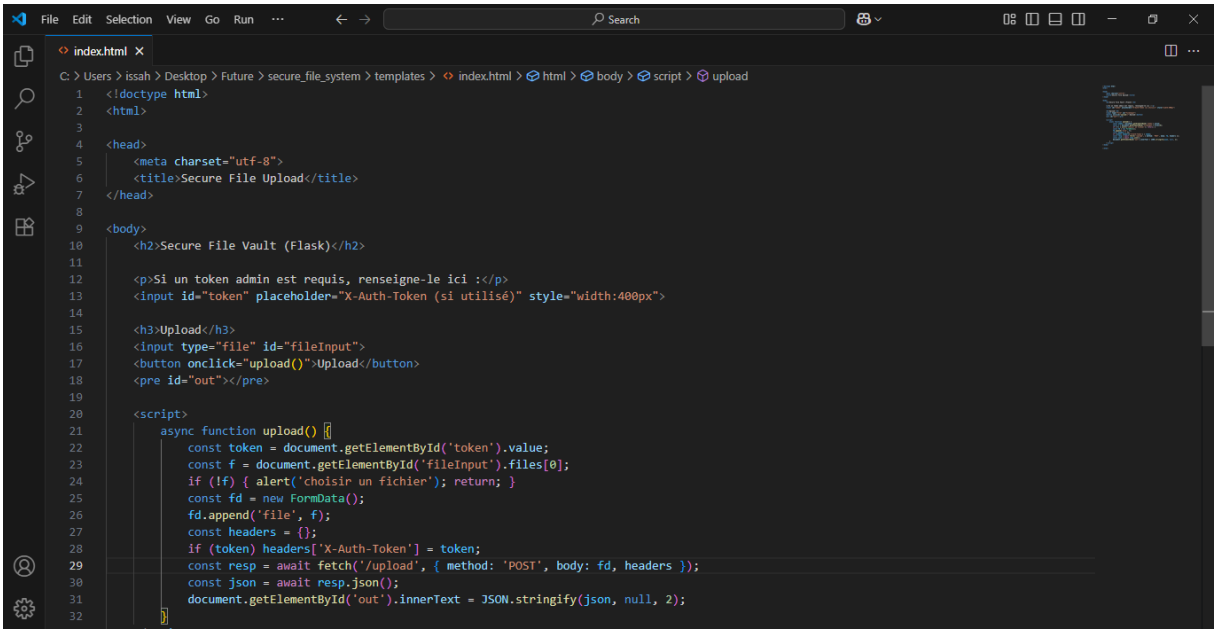
```
    bio = io. BytesIO(plaintext)
 bio.seek(0)
 log_action('DOWNLOAD', safe_name)
    # Flask 2.0+: use download_name param, older versions use
attachment_filename
    return send_file(bio, as_attachment=True,
download_name=original_name)

@app.route('/delete/<path:stored_name>', methods=['DELETE'])
def delete (stored_name):
    if ADMIN_TOKEN and not require_token():
        abort(401, description="Missing or invalid X-Auth-Token")

 safe_name = secure_filename(stored_name)
    if not safe_name.endswith('.enc'):
 safe_name = safe_name + '.enc'

    enc_path = os.path.join(UPLOAD_FOLDER, safe_name)
    meta_path = os.path.join(META_FOLDER, safe_name + '.meta.json')

    if os.path.exists(enc_path):
os.remove(enc_path)
    if os.path.exists(meta_path):
os.remove(meta_path)

 log_action('DELETTE', safe_name)
    return jsonify({'status': 'deleted', 'file': safe_name})

if __name__ == '__main__':
    # Dev server (not for production):
 app.run(host='0.0.0.0', port=5000, debug=True)
```

Code Notes:

- We use **AES-GCM** (random nonce 12 bytes); the registered blob = `nonce | tag | ciphertext`.
- The **master key** is read from the environment variable `FILE_VAULT_MASTER_KEY` in **base64 format**.
- **Endpoints protected** by `X-Auth-Token` if `FILE_VAULT_ADMIN_TOKEN` is set.
- Files are returned from memory (stream) without writing a decrypted file to disk because this is more secure.
- `files` returns the metadata for inventory.
- `access.log` contains a simple upload/download/delete history.

`templates/index.html` (simple interface web)

- Screenshots:

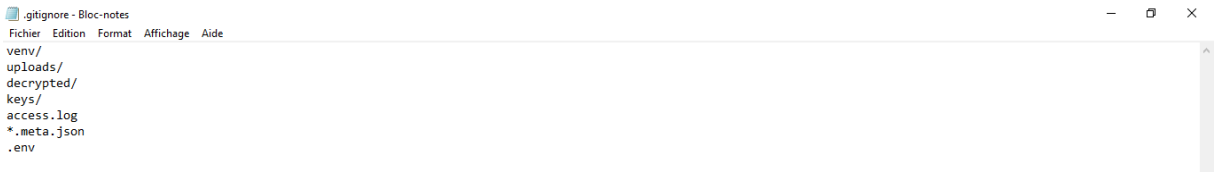


- Lines of code:

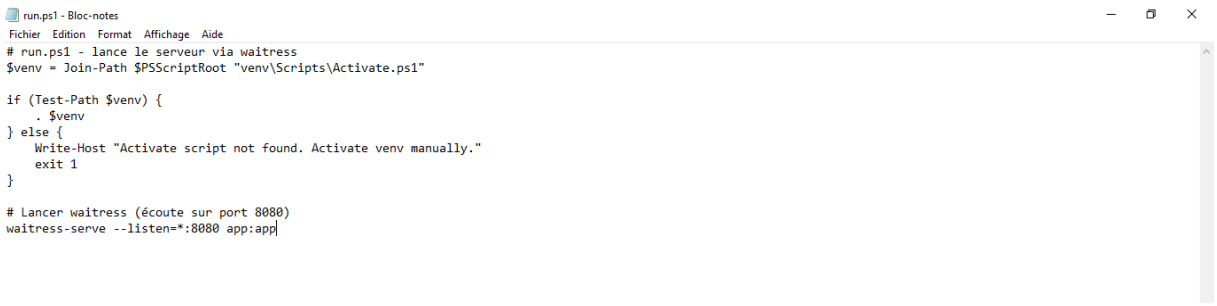
```
<!doctype html>
<html>

<head>
    <meta charset="utf-8">
    <title>Secure File Upload</title>
</head>

<body>
    <h2>Secure File Vault (Flask)</h2>

    <p>If an admin token is required, fill it in here:</p>
    <input id="token" placeholder="X-Auth-Token (si utilisé)"
style="width:400px">

    <h3>Upload</h3>
    <input type="file" id="fileInput">
    <button onclick="upload()">Upload</button>
    <pre id="out"></pre>

    <script>
        async function upload() {
            const token = document.getElementById('token').value;
            const f = document.getElementById('fileInput').files[0];
            if (!f) { alert('choose a file'); return; }
            const fd = new FormData();
```

```
            fd.append('file', f);
            const headers = {};
            IF (token) haders['s-auth-token'] = token;
            const resp = await fetch('/upload', { method: 'POST', body:
fd, headers });
            const json = await resp.json();
            document.getElementById('out').innerText =
JSON.stringify(json, null, 2);
}
    </script>
</body>

</html>
```

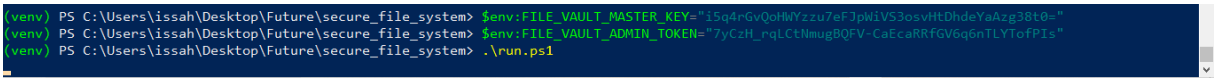
Create `.gitignore` :



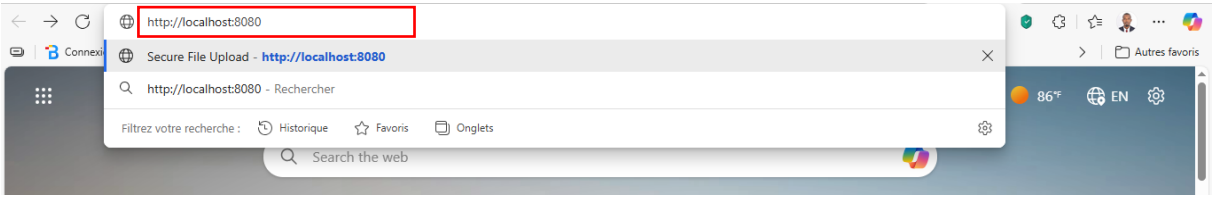
Create `run.ps1` to activate the environment and then launch the app in production mode with **waitress**

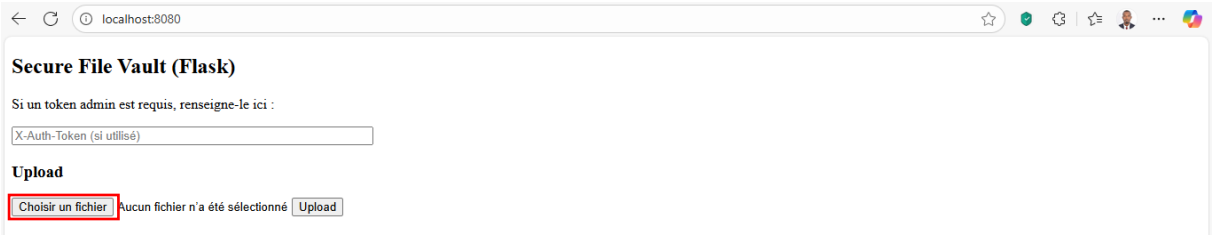


Running on PowerShell :

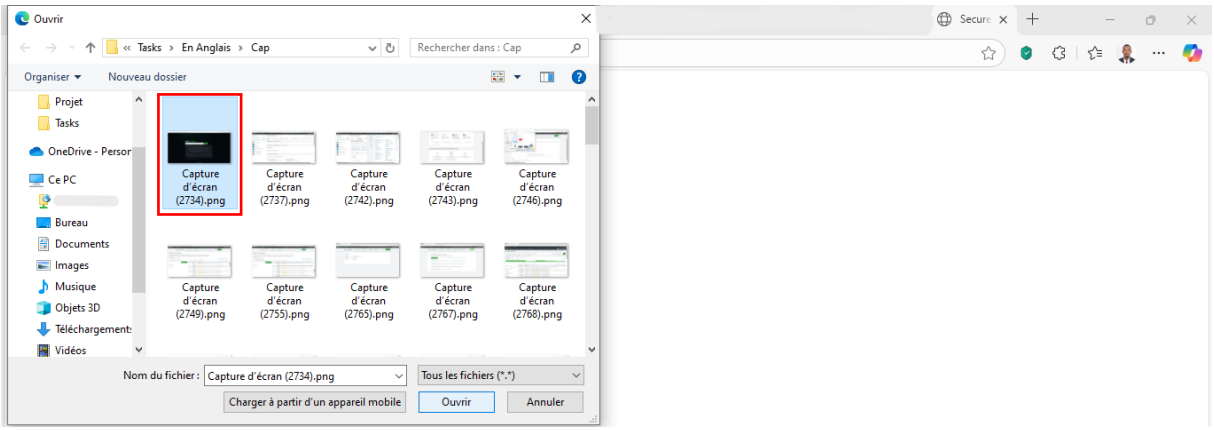


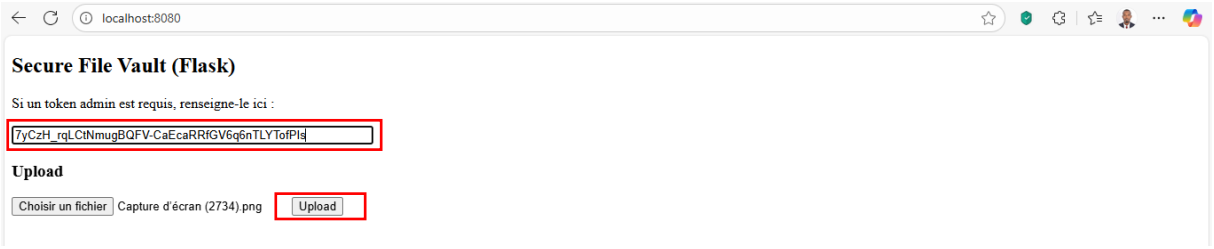
Test via browser: http://localhost:8080



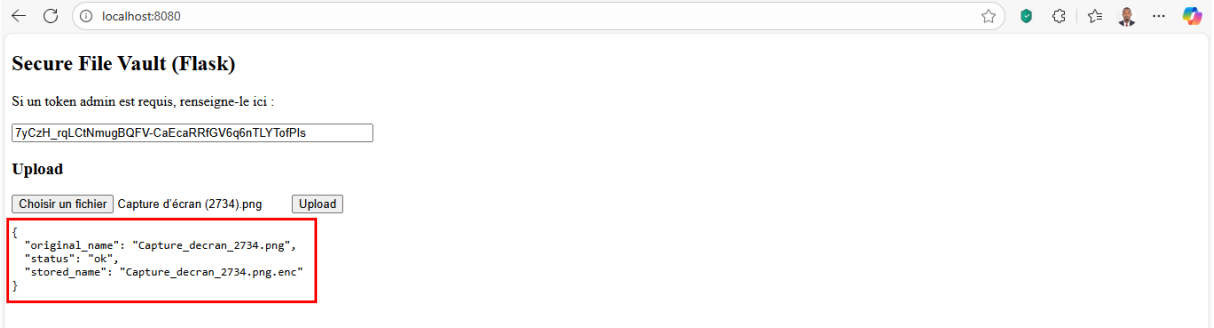


Click on choose a file:





After selecting the file, we set the admin token to upload the file.



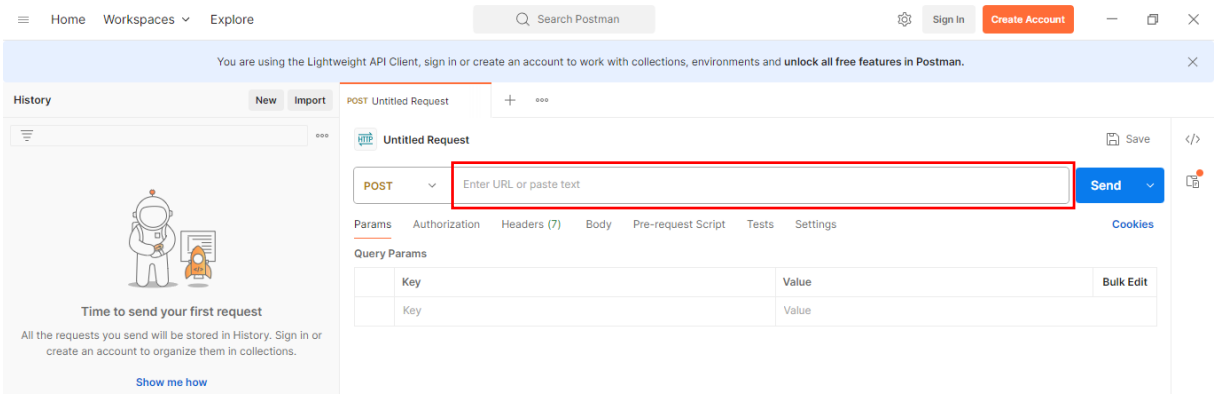
The screenshot shows that the Upload is successful, we go to test on Postman.

DOWNLOAD AND INSTALL POSTMAN

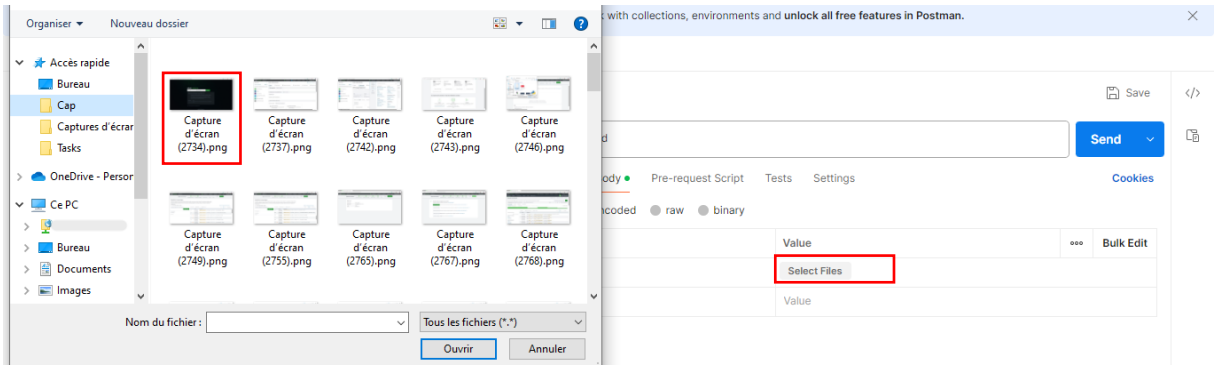
- Aller sur https://www.postman.com/downloads/
- Click on the Windows button
- Download the installation file
- Launch the installer and follow the instructions

Perform an upload test (POST)

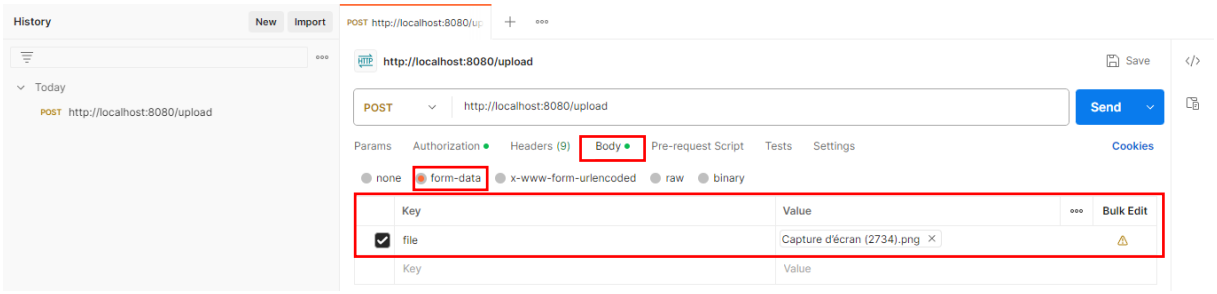
- In the URL bar at the top: http://localhost:8080/upload
- Change the HTTP method to **POST** (left of the URL bar)



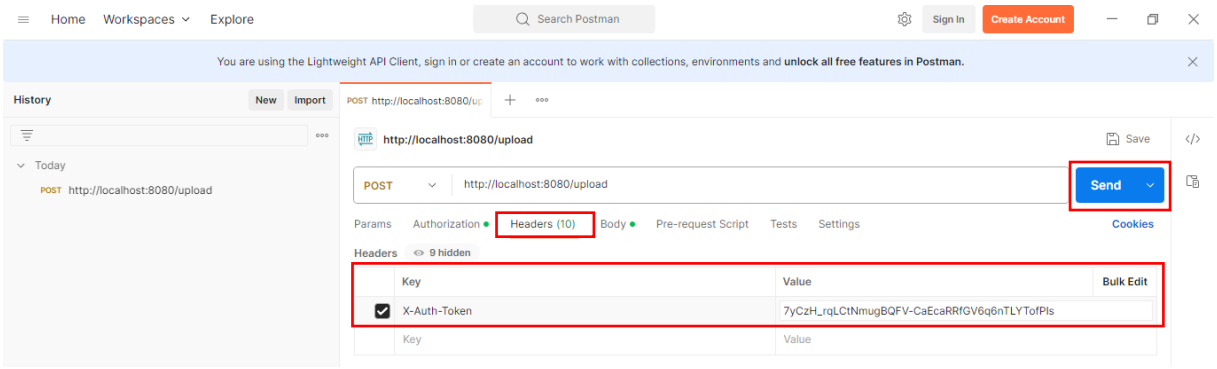
- Adds a new key:
  - **Key** : `file`
  - On the right, change the type from "Text" to **File**
  - Click on **Select Files** and choose a file from your PC to upload



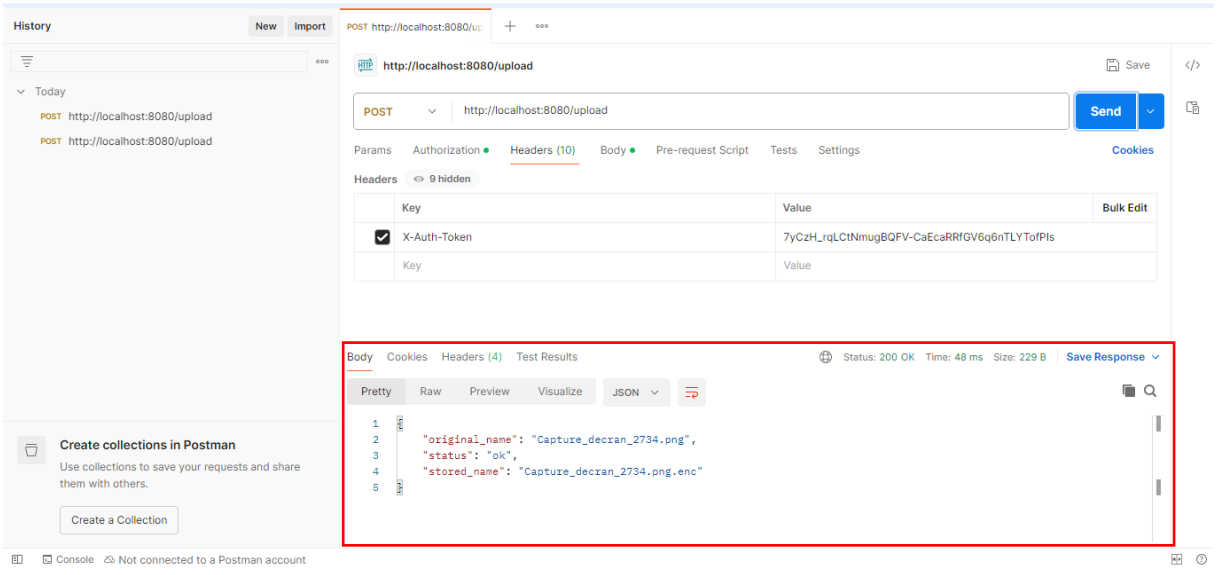
- Click on the **Body** tab → select **form-data**



- Click the **Headers tab**
- Add a header:
  - **Key** : `X-Auth-Token`
  - **Value** : `7yCzH_rqLCtNmugBQFV-CaEcaRRfGV6q6nTLYTofPIs`
- Click Send

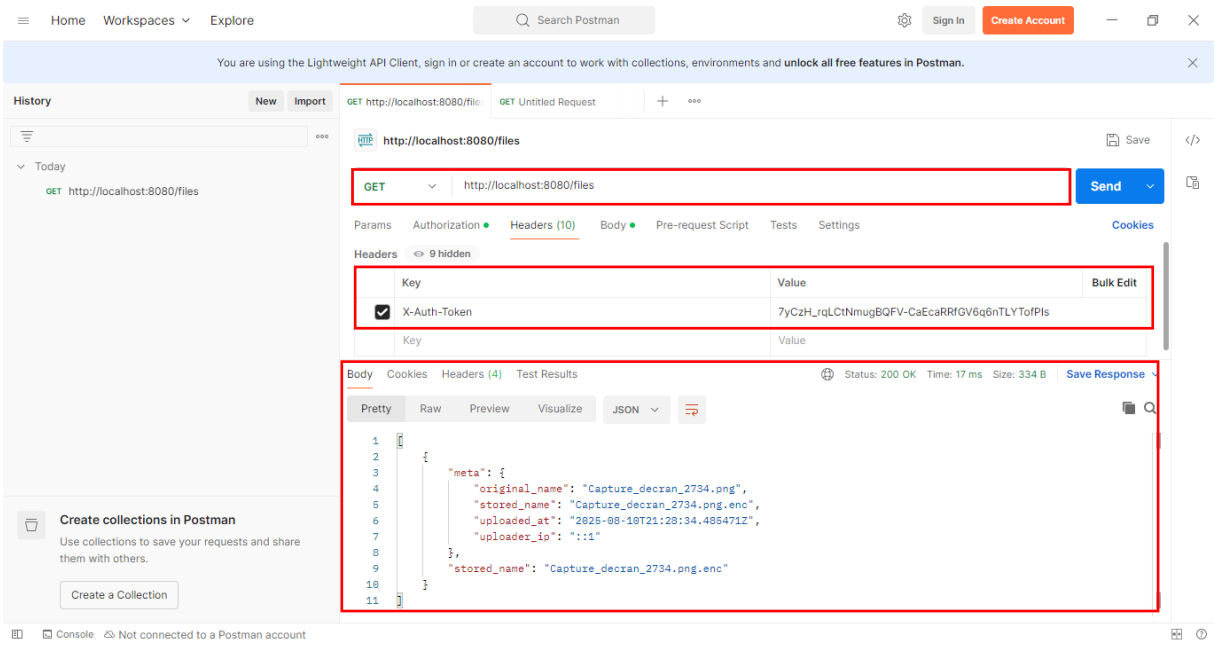


Test result:



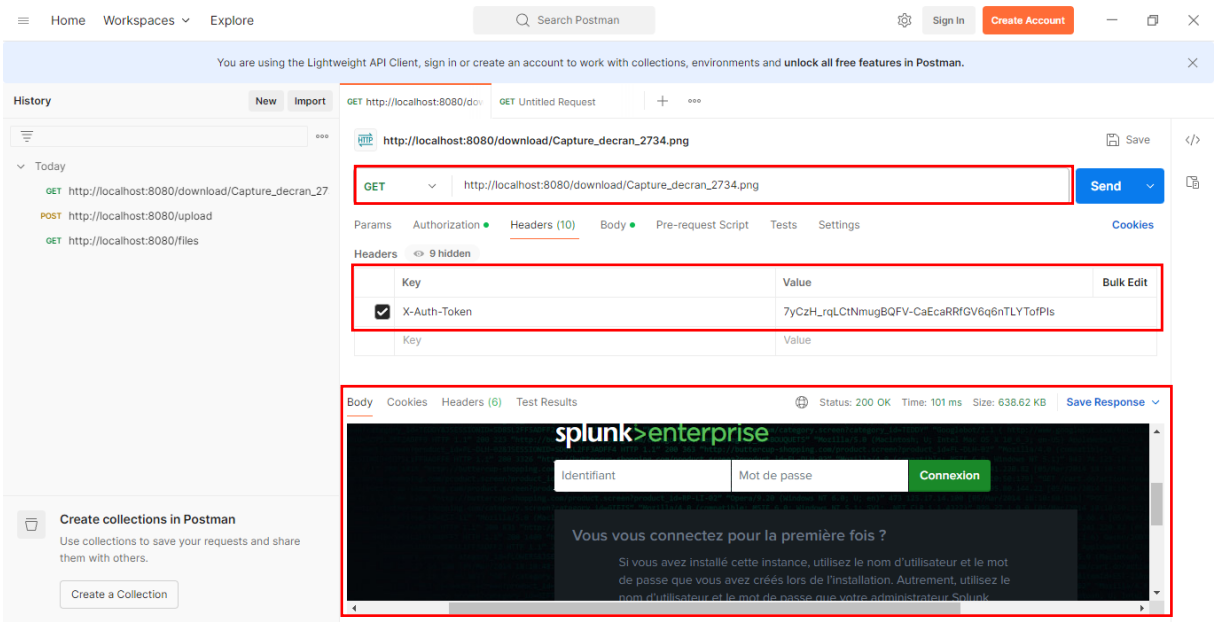
Test the file list (GET)

- Opens a new tab
- Set the method to **GET**
- URL : http://localhost:8080/files
- Add the same **X-Auth-Token** to Headers
- Click Send
- We will see the list of uploaded files



Upload a file (GET)

- New tab
- GET **method**
- URL : http://localhost:8080/download/Capture_decran_2734.png
- Add the `X-Auth-Token header`
- Click Send
- Postman will display the


Home
Workspaces
Explore
Search Postman
Sign In
Create Account
You are using the Lightweight API Client, sign in or create an account to work with collections, environments and unlock all free features in Postman.
History
New
Import
GET http://localhost:8080/do
GET Untitled Request
Today
GET http://localhost:8080/download/Capture_decran_27
POST http://localhost:8080/upload
GET http://localhost:8080/files
http://localhost:8080/download/Capture_decran_2734.png
Save
GET
http://localhost:8080/download/Capture_decran_2734.png
Send
Params
Authorization
Headers (10)
Body
Pre-request Script
Tests
Settings
Cookies
Headers
9 hidden
Key
Value
Bulk Edit
X-Auth-Token
7yCzH_rqLCtNmugBQFV-CaEcaRRfGV6q6nTLYTofPIs
Key
Value
Body
Cookies
Headers (6)
Test Results
Status: 200 OK
Time: 101 ms
Size: 638.62 KB
Save Response
splunk>enterprise
Identifiant
Mot de passe
Connexion
Vous vous connectez pour la première fois ?
Si vous avez installé cette instance, utilisez le nom d'utilisateur et le mot de passe que vous avez créés lors de l'installation. Autrement, utilisez le nom d'utilisateur et le mot de passe que votre administrateur Splunk
Create collections in Postman
Use collections to save your requests and share them with others.
Create a Collection

CONCLUSION :

The creation of this secure file sharing application allowed me to concretely apply several fundamental notions of cybersecurity, ranging from data encryption to securing access. By combining web development best practices with IT security principles, I was able to design a system that could effectively protect sensitive files from the risk of leakage, modification, or unauthorized access.

Through the use of the AES-256 algorithm, files are not only robustly encrypted, but also authenticated, ensuring their integrity. The secure storage of the master key via an environment variable, as well as the use of an authentication token to limit access, increases the level of protection of the system.

This project also made me aware of the importance of error handling, event logging, and action traceability in any secure application. It represents a concrete application of the skills acquired in application security, cryptography and access management.

Finally, this task allowed me to understand the challenges related to the development of secure tools in a real environment, and constitutes a solid basis to go further in the integration of cybersecurity within web and cloud projects.