INSTITUT SUPERIEUR D'INFORMATIQUE

ET DES TECHNOLOGIES DE COMMUNICATION DE SOUSSE

المعهد العالي للإعلامية وتكنولوجيات الاتصال بسوسة

Rapport de stage de fin d'études

Présenté en vue de l'obtention du diplôme de Licence en Technologies de l'Information et de la Communication

Spécialité : Télécommunications

Mini Drive : Application Web de Stockage et Gestion Sécurisée de Fichiers

Réalisé par :

Ridha gnounou

Nourssen jedidi

Encadrant académique : M.MOHAMED OUNI

Année universitaire : 2025/2026

Institut Supérieur d'Informatique et des Technologies de Communication de Sousse **ISITCom**
Tél/Fax : +216 73 37 15 71 / +216 73 36 44 11

Introduction Générale

Avec la croissance rapide des services numériques, le stockage de fichiers en ligne est devenu une nécessité pour les utilisateurs, aussi bien dans le domaine personnel que professionnel. Les plateformes de stockage telles que Google Drive, Dropbox ou OneDrive proposent des solutions puissantes mais souvent complexes, payantes ou peu adaptées aux projets éducatifs nécessitant une totale maîtrise de l'infrastructure.

Dans le cadre de ce projet académique, l'objectif était de concevoir et développer une application web de gestion et stockage de fichiers simple, sécurisée et totalement contrôlée par le développeur. Cette application, nommée Mini Drive, offre une solution légère, personnalisable et facile à déployer.

CHAPITRE 1 : CAHIER DES CHARGES

1.Contexte Général :

La gestion et le stockage de fichiers en ligne constituent aujourd'hui une composante essentielle des systèmes d'information modernes. Dans les environnements collaboratifs comme dans les usages personnels, l'utilisateur recherche une solution simple, accessible et sécurisée pour conserver ses documents numériques.
Cependant, les plateformes les plus connues (Google Drive, Dropbox, OneDrive) présentent plusieurs limites : complexité, dépendance à un fournisseur externe, limites d'espace, absence de contrôle sur la structure interne, et manque d'adaptabilité pour un projet éducatif ou privé.
Dans ce contexte, le projet Mini Drive vise à mettre en place une solution de stockage personnalisée, légère, sécurisée et totalement maîtrisée du point de vue du développeur. L'application doit permettre à un utilisateur de gérer ses propres fichiers sans dépendre d'un système tiers.

2. Présentation des Solutions Existantes

Description des solutions actuelles **:**
Avant de concevoir une nouvelle solution, il est essentiel d'analyser les outils existants afin de comprendre leurs forces et leurs faiblesses.

2.1 Solutions actuelles de stockage

Google Drive

- Offre complète, intégrée avec Google Workspace
- Synchronisation automatique
- Interface riche

Dropbox

- Stockage cloud universel
- Partage de fichiers efficace
- Fonctionnalités avancées (historique, versions)

Microsoft OneDrive

- Intégré à Windows
- Synchronisation rapide
- Collaboration Office

Limites et contraintes :
Bien que performantes, ces solutions présentent des limites dans un contexte académique ou pédagogique :

- Complexité d'intégration : ces services nécessitent des API complexes.
- Dépendance totale à un fournisseur externe.
- Accès limité au code source : difficile de comprendre l'architecture.
- Surcharge fonctionnelle : trop de fonctionnalités non nécessaires.
- Coût : certaines options avancées sont payantes.
- Pas de stockage privé 100 % contrôlé par l'étudiant.
  Ces limites justifient la création d'une solution personnalisée adaptée aux objectifs pédagogiques du projet.

3. Critique de l'existant :

L'analyse des solutions actuelles révèle plusieurs points faibles qui motivent le développement de Mini Drive.

3.1 Points faibles identifiés :

- Absence de contrôle sur la structure interne des fichiers.
- API difficiles à exploiter dans un petit projet éducatif.
- Forte dépendance au service cloud choisi.
- Limitations de stockage ou coûts supplémentaires.
- Impossibilité d'héberger le backend sur ses propres serveurs.
- Surabondance de fonctionnalités inutiles dans un contexte simple.

3.2 Besoins non satisfaits :

Les besoins non couverts par les solutions existantes sont :

- Un système simple et minimaliste pour apprendre les bases du développement web.
- Une application entièrement personnalisable, du backend au frontend.
- Un stockage 100 % privé, séparé pour chaque utilisateur.
- Une architecture complètement maîtrisée et documentée en UML.
- Un système facile à déployer sur une machine virtuelle personnelle (Azure).

4. Solution proposée :

La solution proposée est une application web nommée Mini Drive, conçue pour répondre précisément aux besoins identifiés.

4.1 Description globale de la nouvelle solution

Mini Drive est une application web full-stack permettant à chaque utilisateur :

- de créer un compte sécurisé ;
- de se connecter via un système d'authentification JWT ;
- d'envoyer (upload) des fichiers PDF, images et textes ;
- de consulter la liste de ses fichiers ;
- de télécharger ses fichiers ;
- de stocker ses documents dans un espace privé isolé.

L'application est composée de :

- Frontend React (Vite), simple et moderne ;
- Backend FastAPI, rapide, sécurisé et modulaire ;
- Base de données SQLite stockant utilisateurs et métadonnées ;
- Stockage local organisé par utilisateur ;
- Déploiement cloud :
  - frontend → Vercel
  - backend → Azure VM avec Nginx

4.2 Valeur ajoutée et avantages

La solution Mini Drive apporte plusieurs avantages :

Pour l'utilisateur

- Simplicité d'utilisation
- Interface claire
- Stockage sécurisé
- État des fichiers visible en un clic

Pour le développeur / étudiant

- Compréhension complète d'un projet full-stack
- Maîtrise totale de l'architecture et des données
- Code léger et facile à maintenir
- Application idéale comme base pour des évolutions futures

Pour l'enseignant

- Projet pédagogique complet
- Respect de la méthodologie UML
- Démonstration concrète des technologies modernes

5. Planification :

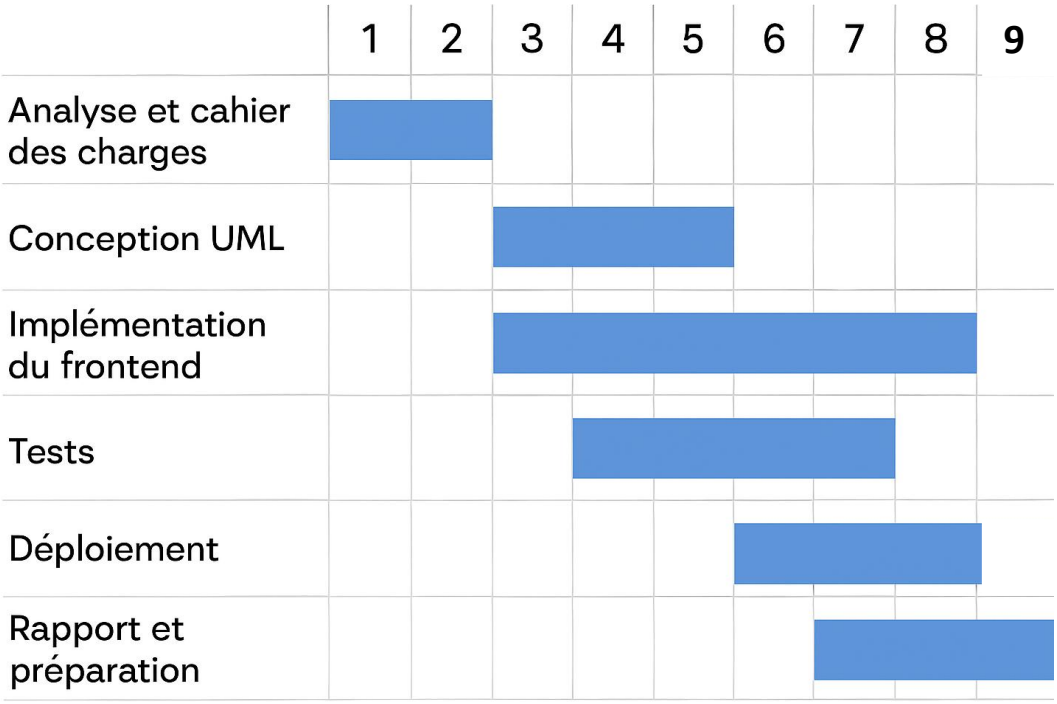


Figure 1 :_diagramme de gantt_

Répartition du travail (Binôme) :
Dans le cadre de ce projet, les tâches ont été réparties de manière équilibrée entre les deux membres du binôme afin d'assurer une progression fluide et une complémentarité optimale.

Conclusion :
Le premier chapitre a permis de définir clairement le contexte du projet, ses objectifs ainsi que les besoins fonctionnels et techniques à satisfaire. L'analyse de l'existant a révélé l'absence d'une solution simple et accessible permettant l'upload et le téléchargement de fichiers de manière centralisée.

Chapitre 2 : Conception UML

Introduction du chapitre :

Ce chapitre présente la conception UML de l'application Mini Drive. On y trouve : le diagramme de cas d'utilisation, le diagramme de classes, les diagrammes de séquence (authentification et upload), et le schéma de base de données. Ces éléments ont guidé l'implémentation et facilitent la lecture du projet.

1. Diagramme de cas d'utilisation :

Description

Le diagramme de cas d'utilisation montre les interactions entre l'utilisateur (acteur) et le système. Les cas principaux sont : Inscription (Signup), Connexion (Login), Upload, Liste des fichiers, Téléchargement, Déconnexion.

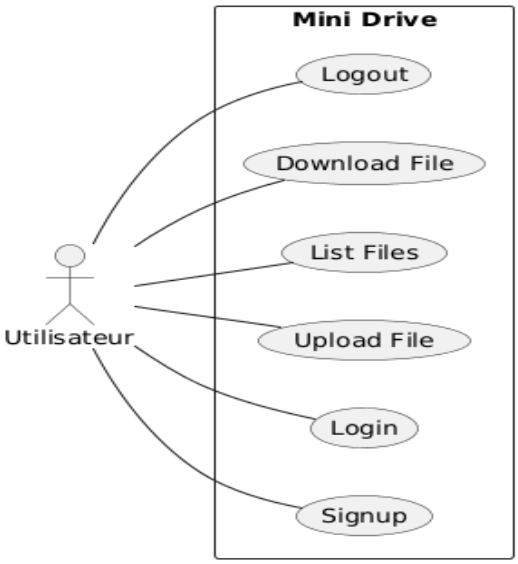


Figure 2 :_diagramme de cas d'utilisation_

2.Diagramme de classes :

Description

Le diagramme de classes montre les principales classes (entités) utilisées côté serveur / base de données et leurs attributs : User et File.

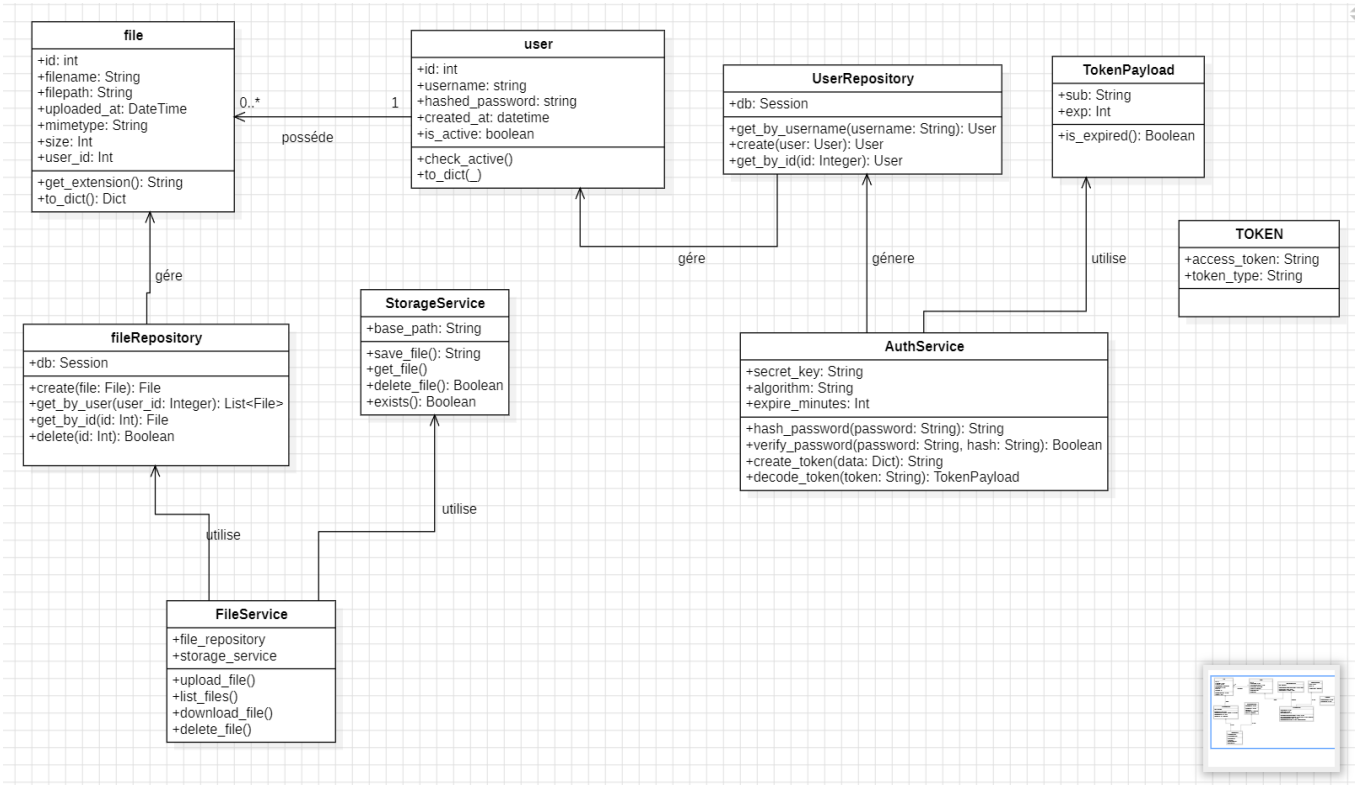


Figure 3 :_diagramme de classe

3. Diagrammes de séquence :

3.1 Séquence : Authentication Flow (Login / Signup) :

Décrit les échanges entre l'utilisateur, le frontend, et le backend pendant l'inscription et la connexion.

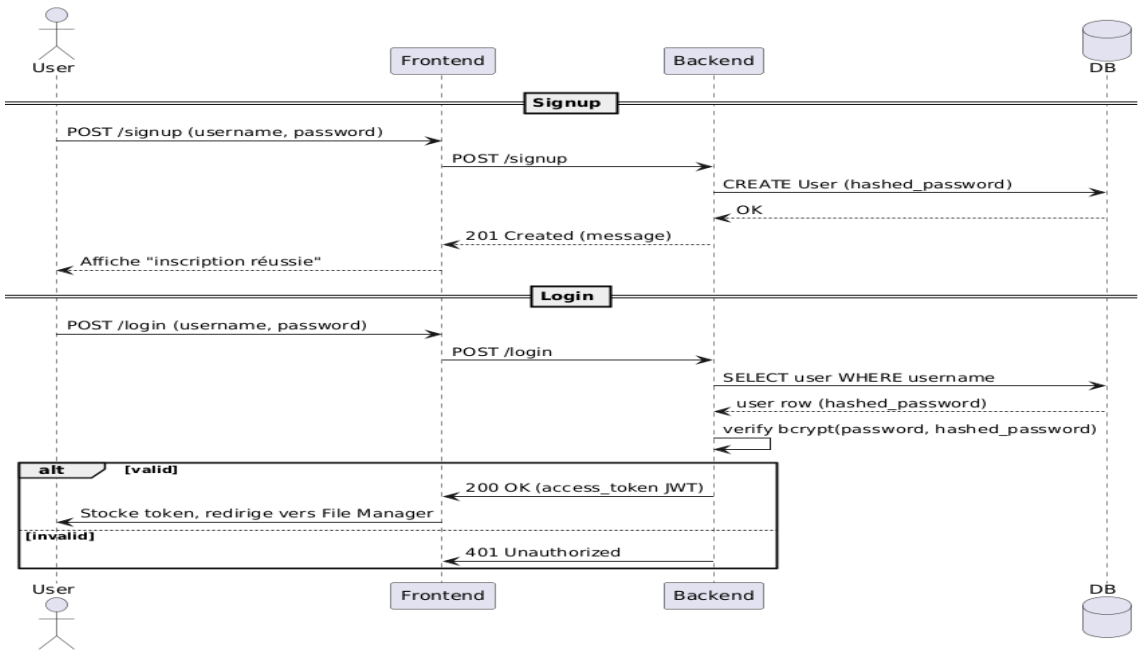


Figure 4 :_diagramme de séquence _

3.2 Séquence : File Upload Flow

Décrit l'envoi d'un fichier par l'utilisateur jusqu'au stockage et la création d'un enregistrement en base.

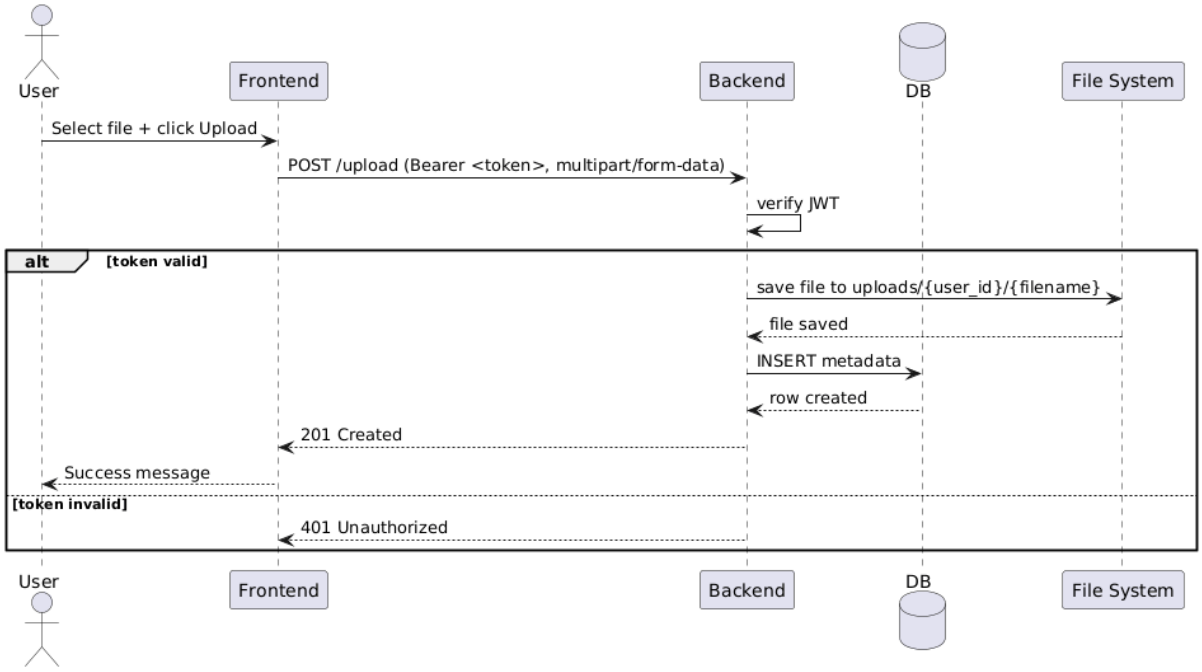


Figure 5 :_diagramme de séquence

Conclusion :

La conception UML fournit une vision claire et normalisée de l'architecture fonctionnelle et structurelle de Mini Drive. Les diagrammes de cas d'utilisation et de classes ont servi de guide pour l'implémentation, tandis que les diagrammes de séquence ont permis de décrire précisément les interactions critiques (authentification, upload).

Chapitre 3 — Réalisation

Introduction

Ce chapitre décrit la réalisation concrète de l'application Mini Drive : l'environnement utilisé, le code (backend et frontend), les choix d'implémentation, l'organisation des fichiers, les captures d'écran des interfaces (à insérer), le déploiement et les tests fonctionnels.

1. Environnement de développement

1.1 Logiciels et bibliothèques utilisés

Backend

- Python 3.10+
- FastAPI — framework web asynchrone et moderne
- Uvicorn — serveur ASGI pour FastAPI
- SQLAlchemy — ORM pour accéder à SQLite
- passlib / bcrypt — hachage des mots de passe
- PyJWT (ou jose) — gestion des tokens JWT
- python-multipart — upload de fichiers
- python-dotenv — gestion des variables d'environnement (optionnel)

Frontend

- Node.js 18+
- npm / yarn
- React (functional components + hooks)
- Vite — bundler / dev server
- fetch API — pour les requêtes HTTP
- Vanilla CSS (fichiers .css simples)

Outils

- Git + GitHub — versionning et dépôt
- VS Code — éditeur recommandé

2. Environnement matériel / configuration recommandée :

2.1 Configuration :

- CPU : Intel i5
- RAM : 8 Go+
- Système : Windows 10 / Ubuntu 20.04
- Connexion internet stable pour déploiement
- Compte GitHub, compte Vercel, abonnement Azure (crédits étudiants si disponible)

2.2 Architecture de déploiement

- Frontend : déployé sur Vercel .
- Backend : VM Ubuntu (Azure) exposée via Nginx reverse-proxy (HTTPS à ajouter).
- Base de données : SQLite (fichiers locaux sur la VM) — envisager PostgreSQL pour production.

3. Implémentation — backend :

3.1 Structure du dossier backend

```
backend/
          main.py          # Point d'entr e , d finit les routes
          auth.py          # Gestion JWT et mots de passe
          database.py      # Configuration base de donn es
          models.py        # Mod les User et File
          requirements.txt # D pendances Python
          uploads/         # Stockage des fichiers
```

Figure 6 :Structure du dossier backend

3.2 Exemples d’extraits de code :

main.py

backend/main.py

```
+ from fastapi import FastAPI, Depends, HTTPException, UploadFile, File as FastAPIFile
+ from fastapi.responses import FileResponse
+ from fastapi.middleware.cors import CORSMiddleware
+ from sqlalchemy.orm import Session
+ import os
+ import shutil
+ from pathlib import Path
+
+ from database import get_db, init_db
+ from models import User, File
+ from auth import verify_password, get_password_hash, create_access_token, decode_token
+
+ app = FastAPI()
+
+ app.add_middleware(
+     CORSMiddleware,
+     allow_origins=["http://localhost:5173", "http://localhost:5174"],
+     allow_credentials=True,
+     allow_methods=["*"],
+     allow_headers=["*"],
+ )
+
+ UPLOAD_DIR = "uploads"
+ Path(UPLOAD_DIR).mkdir(exist_ok=True)
+
```

Auth.py

backend/auth.py

```
+ from datetime import datetime, timedelta
+ from jose import JWTError, jwt
+ from passlib.context import CryptContext
+
+ SECRET_KEY = "your-secret-key-change-this-in-production"
+ ALGORITHM = "HS256"
+ ACCESS_TOKEN_EXPIRE_MINUTES = 1440
+
+ pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")
+
+ def verify_password(plain_password, hashed_password):
+     return pwd_context.verify(plain_password, hashed_password)
+
+ def get_password_hash(password):
+     return pwd_context.hash(password)
+
+ def create_access_token(data: dict):
+     to_encode = data.copy()
+     expire = datetime.utcnow() + timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES)
+     to_encode.update({"exp": expire})
+     encoded_jwt = jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)
+     return encoded_jwt
+
+ def decode_token(token: str):
+     try:
```

4. Implémentation — Frontend :

4.1 Structure du dossier frontend :

```
frontend/
          src/
                App.jsx           # Composant principal
                Login.jsx         # Page de connexion
                Signup.jsx        # Page d'inscription
                FileManager.jsx   # Gestion des fichiers
                main.jsx          # Point d'entr e React
          package.json          # D pendances npm
          vite.config.js        # Configuration Vite
```

App.jsx

```jsx
import { useState, useEffect } from 'react'
import Login from './Login'
import Signup from './Signup'
import FileManager from './FileManager'

function App() {
  const [token, setToken] = useState(localStorage.getItem('token'))
  const [view, setView] = useState('login')

  useEffect(() => {
    if (token) {
      localStorage.setItem('token', token)
    } else {
      localStorage.removeItem('token')
    }
  }, [token])

  const handleLogout = () => {
    setToken(null)
    setView('login')
  }

  if (token) {
    return <FileManager token={token} onLogout={handleLogout} />
  }

  return (
    <div style={{ maxWidth: '400px', margin: '50px auto', padding: '20px' }}>
      <h1 style={{ textAlign: 'center' }}>Mini Drive</h1>
      {view === 'login' ? (
        <Login setToken={setToken} setView={setView} />
      ) : (
```

Login.jsx

```jsx
import { useState } from 'react'

const API_URL = 'http://localhost:8000'

function Login({ setToken, setView }) {
  const [username, setUsername] = useState('')
  const [password, setPassword] = useState('')
  const [error, setError] = useState('')

  const handleSubmit = async (e) => {
    e.preventDefault()
    setError('')

    try {
      const response = await fetch(`${API_URL}/login?username=${username}&password=${password}`, {
        method: 'POST',
      })

      if (!response.ok) {
        throw new Error('Invalid credentials')
      }

      const data = await response.json()
      setToken(data.access_token)
    } catch (err) {
      setError(err.message)
    }
  }

  return (
    <div>
      <h2>Login</h2>
```

5. Déploiement

5.1 Déploiement du Frontend sur Vercel

Le déploiement du frontend React a été réalisé sur la plateforme Vercel, qui offre un hébergement simple, rapide et entièrement compatible avec les projets construits avec Vite et React.
Les étapes suivent le processus suivant :

1. Connexion à la plateforme Vercel en utilisant un compte GitHub.
2. Importation du repository contenant le code source de l'application.
3. Sélection du dossier frontend comme répertoire racine du projet.
4. Ajout de la variable d'environnement VITE_API_URL, contenant l'URL publique du backend déployé sur Azure.
5. Déploiement automatique déclenché par Vercel, suivi d'un build réussi et de la mise en ligne du site.

Vercel permet ensuite un déploiement automatique à chaque mise à jour du code sur GitHub, ce qui facilite considérablement la maintenance du frontend.

5.2 Déploiement du Backend sur Azure VM

Le backend FastAPI a été déployé sur une machine virtuelle Azure Ubuntu, ce qui permet d'avoir un contrôle total sur le serveur, l'environnement Python, la base de données et les fichiers envoyés.

Les étapes du déploiement sont les suivantes :

1. Création d'une machine virtuelle Ubuntu sur Azure.
2. Installation de Python, des dépendances FastAPI et des composants nécessaires via pip.
3. Clonage du repository GitHub contenant le code du backend.
4. Configuration de Nginx comme reverse proxy pour rediriger les requêtes HTTP vers l'application FastAPI.
5. Mise en place d'un service systemd permettant de lancer automatiquement l'API au démarrage de la machine.

Cette architecture garantit un backend stable, sécurisé et accessible publiquement via l'URL configurée**.**

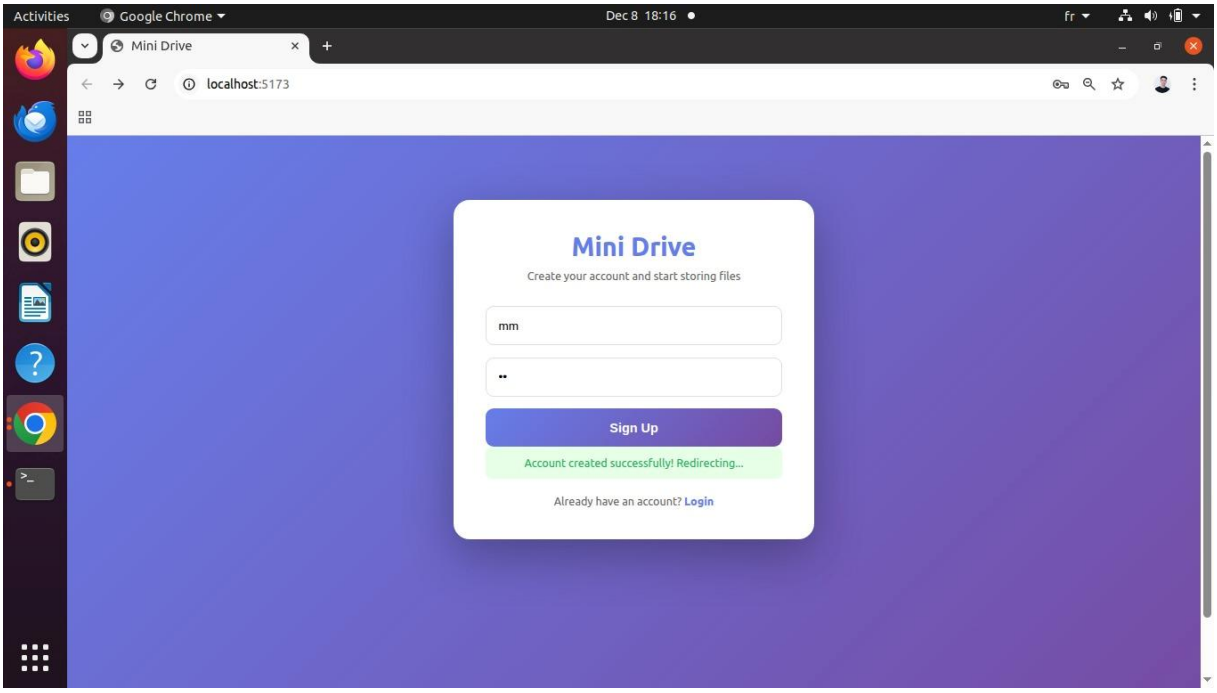
6. Tests :

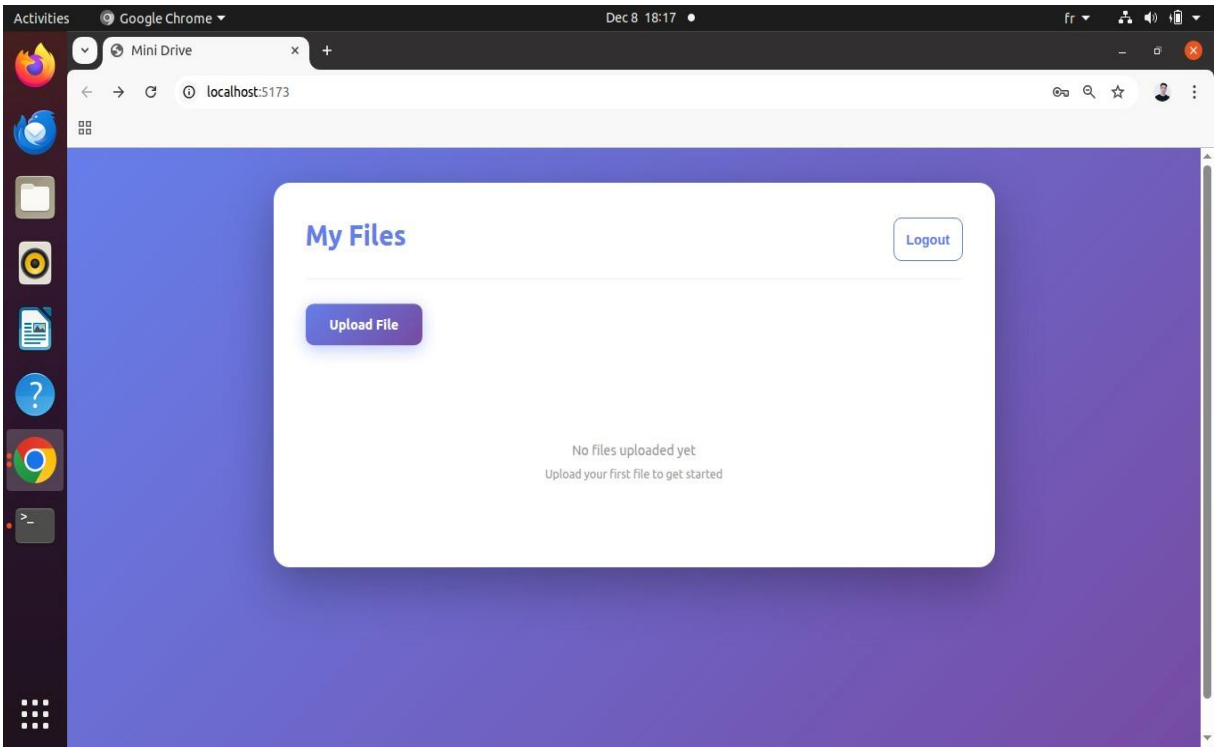
6.1 Tests fonctionnels

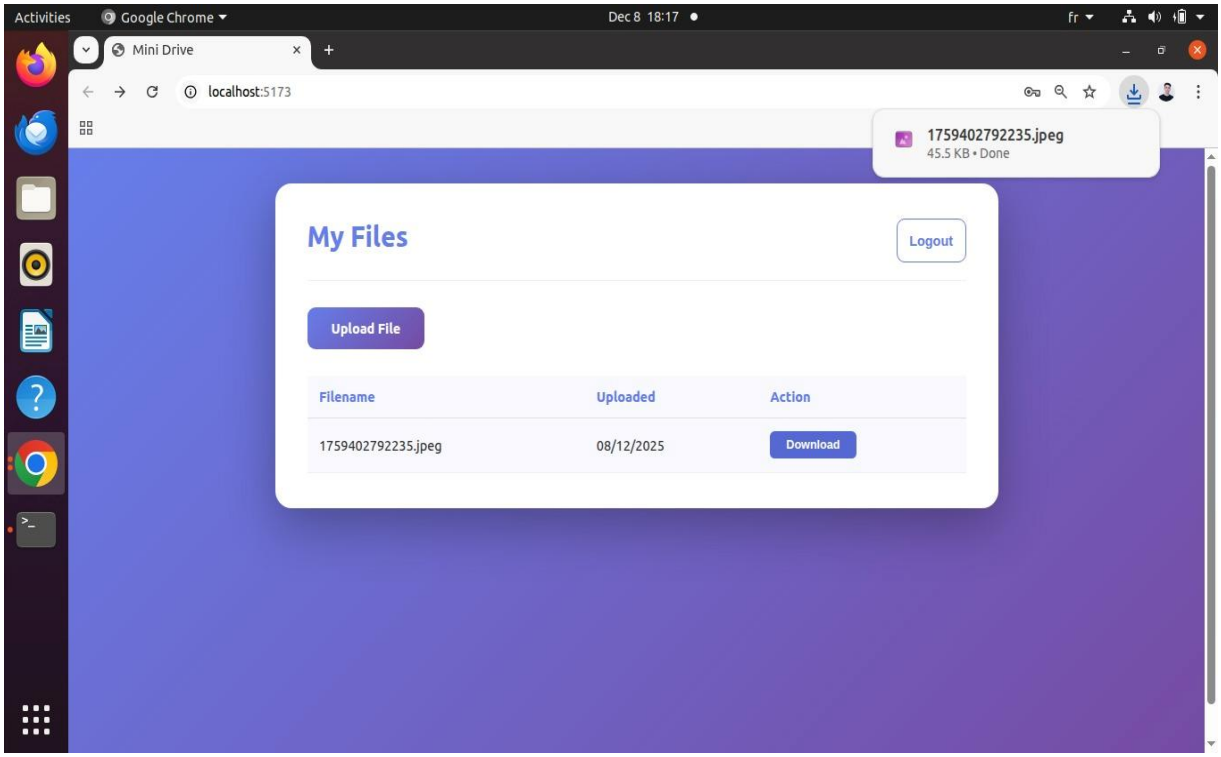
Plusieurs scénarios ont été exécutés pour valider le bon fonctionnement de l'application.
Les tests suivants se sont révélés concluants :

- Création d'un compte utilisateur
- Connexion avec identifiants corrects
- Refus de connexion avec identifiants incorrects
- Upload d'un fichier PDF
- Upload d'une image
- Affichage de la liste des fichiers de l'utilisateur
- Téléchargement d'un fichier
- Déconnexion de l'utilisateur

Ces tests confirment que toutes les fonctionnalités principales sont totalement opérationnelles.




Activities
Google Chrome
Dec 8 18:17
fr
Mini Drive
localhost:5173
My Files
Logout
Upload File
No files uploaded yet
Upload your first file to get started

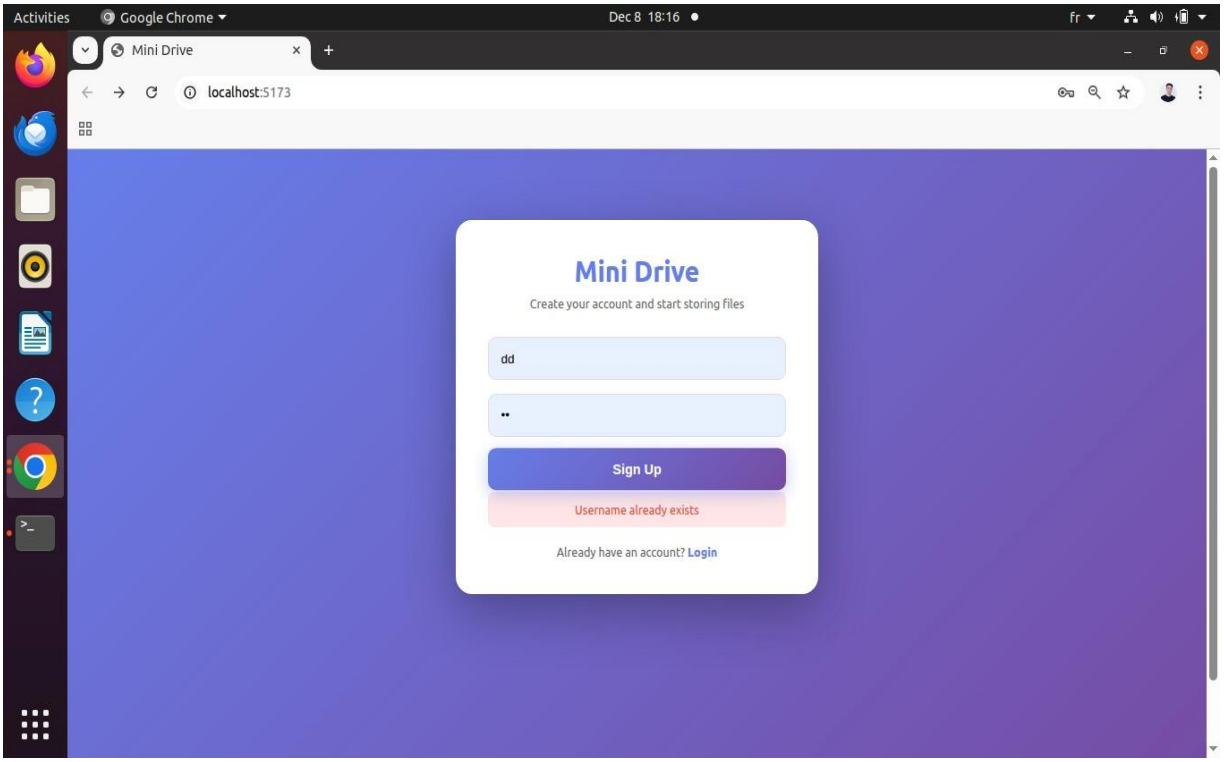

Activities
Google Chrome
Dec 8 18:17
fr
Mini Drive
localhost:5173
1759402792235.jpeg
45.5 KB • Done
My Files
Logout
Upload File
Filename
Uploaded
Action
1759402792235.jpeg
08/12/2025
Download

6.2 Tests de sécurité

Afin de garantir la sécurité de l’application, plusieurs tests ont été réalisés :

- Vérification du hachage des mots de passe dans la base de données (aucun mot de passe en clair).
- Tentative d’accès à l’API sans token (refus systématique).
- Tentative d’accès aux fichiers d’un autre utilisateur (refus).
- Vérification de l’expiration du token JWT après 24 heures.

Ces tests attestent du niveau de sécurité correct du backend.



7. Améliorations futures

7.1 Évolutions fonctionnelles possibles

Plusieurs fonctionnalités peuvent être intégrées dans les versions futures de Mini Drive :

- Suppression de fichiers
- Organisation en dossiers
- Partage de fichiers entre utilisateurs
- Prévisualisation des images et PDF
- Fonction de recherche interne
- Quotas de stockage par utilisate

7.2 Améliorations techniques

Les évolutions techniques possibles incluent :

- Migration de SQLite vers PostgreSQL pour une meilleure performance
- Ajout de tests automatisés pour garantir la stabilité du code
- Compression automatique des fichiers envoyés
- Mise en place d’un système de cache
- Activation du protocole HTTPS via certificat SSL
- Modernisation de l’interface utilisateur

8. Difficultés rencontrées

8.1 Configuration CORS

Un problème de communication entre le frontend et le backend a été rencontré à cause de CORS.
Il a été résolu en configurant correctement les origines autorisées dans FastAPI.

8.2 Gestion des ports

Le port 5173 utilisé par Vite était parfois occupé.
Solution : Vite change automatiquement de port (par exemple, 5174).

8.3 Environnements virtuels

Une confusion a eu lieu entre les dossiers env et venv.
La solution a été d'utiliser systématiquement un seul environnement virtuel correctement identifié.

9. Conclusion

Ce projet a permis de développer une application web complète et fonctionnelle.
L'application Mini Drive atteint tous les objectifs fixés :

- Authentification sécurisée
- Gestion complète des fichiers
- Code versionné sur GitHub
- Architecture moderne et évolutive
- Déploiement opérationnel

10. Compétences acquises

Grâce à ce projet, plusieurs compétences techniques et pratiques ont été développées :

- Développement full-stack (FastAPI + React)
- Gestion de base de données via SQLAlchemy
- Authentification sécurisée (JWT, bcrypt)
- Création d'API REST
- Sécurité et bonnes pratiques backend
- Utilisation de Git et déploiement cloud (Vercel & Azure)

Ce projet constitue une base solide permettant de réaliser des applications web plus ambitieuses, avec de nombreuses fonctionnalités pouvant être ajoutées ultérieurement.

Conclusion :

La réalisation de **Mini Drive** montre la mise en pratique complète des savoirs : conception UML, développement backend & frontend, sécurité, tests et déploiement. Le code est organisé, commenté et prêt pour une mise en production modeste. Les procédures de déploiement et les extraits de code ci-dessus permettent à un examinateur de répliquer l'environnement.

Conclusion générale

Ce projet a permis de réaliser une application web complète tout en respectant une architecture professionnelle. Les objectifs fixés ont été atteints : authentification sécurisée, gestion de fichiers, déploiement cloud et tests fonctionnels.

Difficultés rencontrées

- Configuration CORS
- Gestion des ports (5173/5174)
- Problèmes de virtual environment

Solutions apportées

- Ajout de CORS dans FastAPI
- Changement automatique de port Vite
- Utilisation d’un seul venv propre

Bilan

Ce projet constitue une base solide pour évoluer vers une plateforme de stockage complète. Il renforce les compétences en :

- Développement full-stack
- API REST
- Sécurité web
- Cloud deployment
- Architecture logicielle