📦 Exercice technique – File Downloader (orienté Frontend / Full Stack)

🎯 Objectif

Développer une petite application web permettant de **lister et télécharger des fichiers** depuis un dossier côté serveur, avec une priorité donnée à la **qualité de l'interface utilisateur (UI)** et à une **expérience utilisateur (UX) fluide**.

L'accent est mis sur la partie **frontend**, mais un **backend simple** est également attendu.

🧰 Fonctionnalités attendues

1. Interface Web (Frontend)

L'interface doit être développée **obligatoirement en React**.

L'utilisation de **Material UI (MUI)** est **fortement appréciée**.

La page principale doit :

- Afficher la **liste des fichiers disponibles**
- Montrer pour chaque fichier :
  - Nom
  - Taille (si fourni par l'API)
  - Date de dernière modification (si fournie)
- Proposer un bouton / icône / action claire pour **télécharger** le fichier
- Gérer les états :
  - **Chargement** de la liste
  - **Erreur** de chargement
  - **Aucun fichier disponible**

👉 **Bonus UX/UI appréciés** :

- Design responsive (mobile inclus)

- Barre de recherche
- Tri (nom / taille / date)
- Feedback visuel sur un téléchargement (loading, succès, erreur)
- Toasts / Snackbars MUI pour les notifications
- Skeleton loaders
- Gestion propre des erreurs (serveur, réseau, vide)
- Dark mode optionnel

---

2. API / Backend

Le backend doit être écrit en **Python**, avec **préférence pour Flask**.

Il n'a pas vocation à être complexe, mais doit être :

- Simple
- Documenté
- Lisible

Endpoints attendus :

Méthode	URL	Description
GET	`/api/files`	Renvoie la liste des fichiers en JSON
GET	`/download/<nom>`	Télécharge le fichier demandé

Exemple de réponse JSON :

```
[
  {
    "name": "document.pdf",
    "size": 123456,
    "last_modified": "2025-01-15T10:32:00Z"
  },
  {
    "name": "image.png",
    "size": 45678,
    "last_modified": "2025-01-10T09:12:00Z"
  }
```

```
]
```

Le backend peut lire les fichiers depuis un simple dossier local `files/`.

❌ **Aucun usage de Docker n'est requis**.

🧪 Tests attendus

Au minimum **un test par endpoint ou comportement clé** :

Backend (pytest recommandé)

- `GET /api/files` retourne bien la liste
- `GET /download/<nom>` renvoie bien un fichier ou une erreur 404

Frontend (React Testing Library, Jest, Vitest, Playwright, etc.)

- Test d'un composant clé (liste, état loading, message d'erreur...)
- OU test d'intégration simple (navigation, affichage des fichiers...)

L'objectif : **montrer que tu sais tester ton code**, même légèrement.

🛠️ Stack attendue

Frontend (obligatoire)

- **React**
- **Material UI (fortement recommandé)**
- Architecture propre, composants bien séparés

Backend (obligatoire)

- **Python**
- **Flask (fortement recommandé)**
- Code simple, structuré et intelligible

📁 Livrables

Un repository Git **public** contenant :

- Le frontend (React)
- Le backend (Python)
- Les tests
- Un fichier **README.md**

📜 Contenu attendu du README

1. **Présentation rapide du projet**
2. **Instructions d'installation**
    - Pré-requis Node / Python
    - Installation des dépendances
3. **Commandes pour lancer :**
    - Le backend
    - Le frontend
4. **Exemples d'appels API**
5. **Procédure pour exécuter les tests**
6. **(Bonus)** Quelques explications sur les choix techniques & UX

💡 Prises d'initiative bienvenues

Toute amélioration UX, technique ou design est valorisée :

- Recherche + tri avancé
- Prévisualisation des fichiers (images, PDF...)
- Thème personnalisé ou Dark Mode
- Accessibilité (ARIA, navigation clavier...)
- Internationalisation
- Organisation en monorepo (NX, Turborepo, etc.)
- Gestion avancée des erreurs / retries
- Animation ou transitions (MUI, Framer Motion)