Résumé définitif et étapes de développement

Pour conclure, voici le plan de développement complet et final qui inclut toutes les fonctionnalités dont nous avons discuté :

1. **Interface utilisateur (Kivy) :**
    - Crée une interface intuitive pour la navigation dans les fichiers, l'affichage des disques et la liste des appareils.
    - Ajoute une interface pour la gestion des licences, capable d'afficher des boîtes de dialogue pour les messages et les demandes d'autorisation.
2. **Module de gestion de licences (Python et serveur web) :**
    - Développe une API web avec **Flask** ou **Django** pour la gestion sécurisée des licences.
    - Intègre le code Python pour envoyer des requêtes à cette API afin de valider la clé de licence de l'utilisateur.
3. **Gestion du système de fichiers (Python) :**
    - Utilise les modules Python os et psutil pour scanner les disques et parcourir les dossiers, afin d'afficher leur contenu dans l'interface Kivy.
4. **Module de communication réseau (Python et C++) :**
    - Implémente la détection des appareils via **sockets UDP**.
    - Développe les fonctions de **transfert de fichiers** via **sockets TCP**, optimisées pour les fichiers lourds grâce à la lecture par **morceaux**.
    - Crée un **module en C++** (lié via **Pybind**) pour gérer la création de **réseaux ad-hoc**. Ce module sera appelé par le script Python si aucun réseau existant n'est détecté.
5. **Logique d'interaction et de contrôle (Python) :**
    - Connecte les éléments de l'interface utilisateur Kivy aux fonctions Python pour un flux de travail fluide.
    - Mets en place une logique pour vérifier l'état du réseau au démarrage. Si aucun pair n'est trouvé, l'application demandera à l'utilisateur de créer un hotspot via le module C++.
6. **Déploiement :**
    - Utilise **PyInstaller** pour compiler l'application en un exécutable autonome, en t'assurant que le module C++ est correctement intégré au package final.