# Affichage et crud des fichiers

## 1. app/files/page.tsx

#### Rôle :

Page principale de gestion des fichiers d'un projet.  
Informations utilisées :

* selectedProject depuis useProjectsStore pour obtenir le projectId
* utilise useSesssion de lib/auth/auth-client pour obtenir l’id de l’user connecté

#### Props envoyées :

À FileFilters : états de filtrage et leurs setters

À FileTree : files, onRefresh, projectId

À FileList : files, view, onRefresh, projectId

## 2. components/models/files/FileList.tsx

#### Rôle :

Affichage en grille/liste les fichiers

avec actions CRUD d’un item. Et ene bas le boutton ajouter

#### Props reçues :

files, view, onRefresh, projectId

Informations utilisées :

Données des fichiers depuis le parent

#### Props envoyées :

À FileCard : file, view, onEdit, onDelete

À FileForm : initialValues, onSubmit, loading

## 3. components/models/files/FileTree.tsx

#### Rôle :

Affichage hiérarchique des fichiers en arbre

#### Props reçues :

files, onRefresh, projectId

#### Informations utilisées :

Hiérarchie via parentFileId

#### Props envoyées :

À FileTreeNode : file, children, childrenMap, isExpanded, onToggle, onRefresh, level

## 4. components/models/files/FileTreeNode.tsx

Rôle :

Nœud individuel de l'arbre avec gestion de la hiérarchie

Props reçues :

file, children, childrenMap, isExpanded, onToggle, onRefresh, level

## 5. components/models/files/FileFilters.tsx

Rôle : Composant de filtrage par type, statut et recherche

Props reçues : États de filtrage et leurs setters depuis la page parent

## 6. components/models/files/FileCard.tsx

Rôle : Carte d'affichage d'un fichier individuel

Props reçues : file, view, onEdit, onDelete

## 7. components/models/files/FileForm.tsx

Rôle : Formulaire de création/édition de fichiers

Props reçues : initialValues, onSubmit, loading

## //

## // app/files/[fileId]/page.tsx

En analysant votre schéma Prisma fourni et le script de la page app/files/[fileId]/page.tsx, voici ce que fait cette page dynamique :

**Fonctionnalité principale**

Cette page affiche les **détails complets d'un fichier spécifique** selon votre schéma Prisma. Elle récupère un fichier par son fileId et présente toutes ses informations de manière structurée.

**Ce que fait la page en détail**

**1. Authentification et sécurité**

* Vérifie que l'utilisateur est connecté via getServerSession()
* S'assure que l'utilisateur a accès au projet contenant le fichier
* Utilise la relation many-to-many Projects ↔ User pour vérifier les permissions

**2. Récupération des données selon votre schéma**

La fonction getFileDetails() récupère le fichier avec toutes ses relations :

typescript

*// Relations principales selon votre schéma*

project: { id, name } *// Relation Files → Projects*

uploader: { id, name, image } *// Relation Files → User (uploaderId)*

parentFile: { id, name, type } *// Auto-référence Files → Files (parentFileId)*

childFiles: [...] *// Relation inverse Files ← Files*

roadMaps: [...] *// Relation optionnelle Files ↔ RoadMap*

**3. Affichage structuré en deux colonnes**

**Colonne principale (2/3 de l'écran) :**

* **Informations générales** : Nom, URL, extension, version, ordre d'affichage/développement, dates
* **Dépendances** : Champs import, export, useby de votre schéma
* **Script/Contenu** : Affichage du champ script avec coloration syntaxique
* **Fichiers enfants** : Liste des fichiers ayant ce fichier comme parent

**Colonne latérale (1/3 de l'écran) :**

* **Créateur** : Informations sur l'uploader du fichier
* **Fichier parent** : Lien vers le fichier parent s'il existe
* **RoadMaps liées** : RoadMaps associées à ce fichier
* **Actions rapides** : Boutons pour modifier, télécharger, supprimer

**4. Navigation et actions**

* **Breadcrumb** : Retour vers /files avec le nom du projet
* **Actions** : Télécharger, partager, modifier le fichier
* **Liens contextuels** : Navigation vers fichiers parents/enfants et roadmaps

**5. Métadonnées dynamiques**

* Génère automatiquement le titre et la description de la page
* SEO optimisé avec le nom du fichier et du projet

**Utilisation selon votre architecture**

Cette page s'intègre parfaitement dans votre système de gestion de projets :

1. **Depuis /files** : Clic sur un fichier → Redirection vers /files/[fileId]
2. **Navigation hiérarchique** : Liens entre fichiers parents/enfants
3. **Intégration RoadMap** : Liens vers les roadmaps liées
4. **Gestion des permissions** : Respect de la sécurité via les relations utilisateur-projet

**Conformité à votre schéma**

La page respecte parfaitement votre modèle Files :

* Affiche tous les champs : name, url, extension, type, description, fonctionnalities, etc.
* Utilise les relations : project, uploader, parentFile, childFiles, roadMaps
* Respecte les enums : FileType, Status
* Gère la hiérarchie via parentFileId

Cette page offre donc une **vue complète et interactive** d'un fichier dans votre système de gestion de projets, avec toutes ses métadonnées et relations selon votre schéma Prisma.

## // app/api/files/route.ts

**Rôle principal du fichier**

Ce fichier est une **Route Handler Next.js** qui expose l'endpoint /api/files pour gérer tous les fichiers de votre application selon votre schéma Prisma.

**Fonctionnalités selon votre schéma Files**

**GET /api/files**

* Récupère tous les fichiers accessibles à l'utilisateur connecté
* Filtre par projet, type, statut, recherche textuelle
* Pagination avec limit et offset
* Inclut toutes les relations : project, uploader, parentFile, childFiles, roadMaps

**POST /api/files**

* Crée un nouveau fichier selon votre modèle exact
* Valide tous les champs : name, url, type, description, fonctionnalities, etc.
* Gère la hiérarchie avec parentFileId
* Respecte les ordres order (affichage) et devorder (développement)

**Conformité à votre schéma**

Le fichier respecte parfaitement votre modèle Files :

**Sécurité et permissions**

* Vérifie l'authentification via getServerSession()
* Filtre automatiquement par les projets accessibles à l'utilisateur
* Respecte la relation many-to-many Projects ↔ User
* Logs d'activité dans ActivityLogs

**Utilisation dans votre architecture**

Ce fichier sert d'**API centrale** pour :

1. **Pages de recherche globale** : Rechercher des fichiers dans tous les projets
2. **Dashboards** : Afficher les fichiers récents ou favoris
3. **Intégrations** : APIs pour applications tierces
4. **Rapports** : Statistiques sur les fichiers

## // app/api/files/[fileId]/route.ts

**Rôle principal du fichier**

Ce fichier est une **Route Handler Next.js** qui gère les opérations sur un fichier spécifique via l'endpoint /api/files/[fileId]. Il permet de **modifier et supprimer** un fichier individuel selon votre modèle Files.

**Fonctionnalités détaillées**

**PUT /api/files/[fileId]**

**Rôle** : Met à jour un fichier existant

**Ce qu'il fait** :

1. **Authentification** : Vérifie que l'utilisateur est connecté
2. **Validation de l'ID** : S'assure que fileId est un UUID valide
3. **Vérification des permissions** : Utilise hasFileAccess() pour vérifier que l'utilisateur peut modifier ce fichier
4. **Validation des données** : Valide le body avec le schéma fileUpdateSchema
5. **Mise à jour** : Modifie le fichier dans la base de données avec les nouvelles données
6. **Relations incluses** : Retourne le fichier avec uploader, parentFile, childFiles
7. **Log d'activité** : Enregistre l'action dans ActivityLogs

**DELETE /api/files/[fileId]**

**Rôle** : Supprime un fichier existant

**Ce qu'il fait** :

1. **Authentification** : Vérifie que l'utilisateur est connecté
2. **Validation de l'ID** : S'assure que fileId est un UUID valide
3. **Vérification des permissions** : Utilise hasFileAccess() pour vérifier que l'utilisateur peut supprimer ce fichier
4. **Récupération du nom** : Obtient le nom du fichier pour le log
5. **Suppression** : Supprime le fichier de la base de données
6. **Log d'activité** : Enregistre la suppression dans ActivityLogs

**Système de permissions selon votre schéma**

La fonction hasFileAccess() vérifie que l'utilisateur a le droit d'accéder au fichier selon deux critères :

typescript

*// Accès autorisé si :*

file.uploaderId === userId || *// L'utilisateur est le créateur du fichier*

file.project.users.some((u) => u.id === userId) *// OU l'utilisateur est membre du projet*

Cela respecte parfaitement votre schéma où :

* Files appartient à un Projects (relation obligatoire)
* Files a un uploader (User qui l'a créé)
* Projects a une relation many-to-many avec User

**Champs modifiables selon votre schéma**

Le fichier permet de modifier tous les champs optionnels de votre modèle Files :

* **Informations de base** : name, extension, url, type, description
* **Métadonnées** : fonctionnalities, import, export, useby, script, version
* **Organisation** : order (affichage), devorder (ordre de développement), status
* **Hiérarchie** : parentFileId (pour la relation "FileHierarchy")

**Intégration dans votre architecture**

Ce fichier s'intègre parfaitement dans votre système :

1. **Depuis la page de détail** /files/[fileId] : Boutons "Modifier" et "Supprimer"
2. **Depuis la liste des fichiers** : Actions rapides sur chaque fichier
3. **Depuis l'arbre hiérarchique** : Gestion des fichiers dans FileTree
4. **Logs d'activité** : Traçabilité complète dans ActivityLogs selon votre schéma

**Sécurité et conformité**

* **Authentification** : Obligatoire via getServerSession()
* **Autorisation** : Vérification des permissions projet/uploader
* **Validation** : Schémas Zod stricts pour tous les inputs
* **Audit** : Logs automatiques de toutes les modifications
* **Relations** : Respect des contraintes de votre schéma Prisma

Ce fichier est donc l'**API de gestion individuelle des fichiers** qui complète parfaitement les routes de création (/api/projects/[projectId]/files) et de consultation globale (/api/files).

**Architecture simple et claire pour la gestion des fichiers**

**En me basant uniquement sur votre schéma Prisma fourni, voici une architecture épurée :**

**Structure recommandée**

**text**

**app/**

**├── files/**

**│ ├── page.tsx # Page principale des fichiers**

**│ └── [fileId]/**

**│ └── page.tsx # Détail d'un fichier**

**│**

**components/models/files/**

**├── FileList.tsx # Affichage liste/grille**

**├── FileCard.tsx # Carte individuelle**

**├── FileForm.tsx # Formulaire création/édition**

**├── FileFilters.tsx # Filtres (type, statut, recherche)**

**└── FileModal.tsx # Modale pour formulaire**