BTS SIO — Option SLAM

Documentation Technique de Réalisation

Plateforme de gestion des médias

Application web Next.js — medias-e2cvar

Candidat	Cyrille COUR
Établissement	Aristée
Organisation	École de la 2e Chance du Var (E2C Var)
Stack technique	Next.js 16 / React 19 / TypeScript / MySQL 8 (Prisma 7)
Date	Octobre 2026

1. Contexte et objectif

Application interne pour l'E2C Var (École de la 2e Chance) permettant de mettre en ligne, consulter, télécharger, organiser et supprimer des médias (images, vidéos, audio, PDF). Usage interne, nombre d'utilisateurs limité.

Le projet est parti d'un document de cadrage technique proposant une stack Next.js + MySQL, avec un choix initial de stocker les fichiers en base (BLOB). Au démarrage, une base de données existante a été découverte sur l'environnement local (Laragon) : elle contenait déjà une application média avec de vraies données.

2. La décision structurante : reprendre l'existant

Point de bascule du projet. Deux directions étaient possibles :

	Document de cadrage initial	Base de données réelle existante
Stockage des fichiers	Fichiers en base (LONGBLOB)	Fichiers sur le disque, la base ne garde que le chemin (files.path)
Schéma	Nouvelle table media	Tables existantes users / files / user_file_links
Données	Vierge	10 comptes + 96 fichiers déjà référencés

Décision retenue : reprendre l'existant → stockage sur disque, réutilisation du schéma en place, aucune donnée écrasée. Ce choix a été explicitement validé plutôt que de suivre le document d'origine. Conséquence : le schéma a été introspecté depuis la base (prisma db pull) pour coller exactement au réel, au lieu d'être réécrit à la main.

Les 96 fichiers physiques (initialement absents en local) ont ensuite été récupérés depuis la prod via WinSCP et déposés dans backend/files/.

3. Stack technique finale

Couche	Techno	Rôle
Frontend	Next.js 16 (App Router) · React 19 · TypeScript	Interface, galerie, formulaires
Styles	Tailwind CSS v4	Design, couleur principale #843F86 (violet/rose)
Backend / API	Next.js API Routes (runtime Node.js)	Upload, lecture streamée, CRUD, dossiers
Authentification	NextAuth v5 (Auth.js) — Credentials + bcryptjs	Connexion contre les comptes existants
Base de données	MySQL 8 (Laragon en local)	Métadonnées, utilisateurs, dossiers
ORM	Prisma 7 + adapter @prisma/adapter-mariadb	Requêtes typées, introspection
Stockage média	Système de fichiers (backend/files/)	Fichiers physiques, servis en streaming

Particularités marquantes de versions récentes

- **Prisma 7** utilise le nouveau générateur `prisma-client` (moteur « client », sans binaire Rust) qui exige un driver adapter → `@prisma/adapter-mariadb` (le driver MariaDB parle le protocole MySQL, compatible MySQL 8). Le client est généré dans `src/generated/prisma/` (non versionné).
- **Next.js 16** rend le paramètre `params` des routes asynchrone (`await params`).
- **NextAuth v5** : configuration différente de la v4 (server actions, `handlers`, `auth()`).

4. Modèle de données

Schéma introspecté depuis la base existante, puis étendu (ajout des dossiers).

- **users** : user_id, email (unique, collation insensible à la casse), password (bcrypt), username, user_role — enum('new','editor','admin').
- **files** : file_id, user_id (FK → users), folder_id (FK → folders, ajouté), filename (nom d'affichage), path (chemin disque, ex. /backend/files/xxx.png), size (BigInt), mimetype, created_at, updated_at.
- **folders** (ajoutée) : folder_id, name, user_id (créateur), created_at. Relation files.folder_id en ON DELETE SET NULL : supprimer un dossier ne supprime pas les médias, ils redeviennent « non classés ».
- **user_file_links** : table héritée de l'ancienne app, conservée mais non utilisée.

Évolution du schéma appliquée via `prisma db push` (non destructif : ajout de la table folders et de la colonne files.folder_id, les 96 fichiers restant intacts).

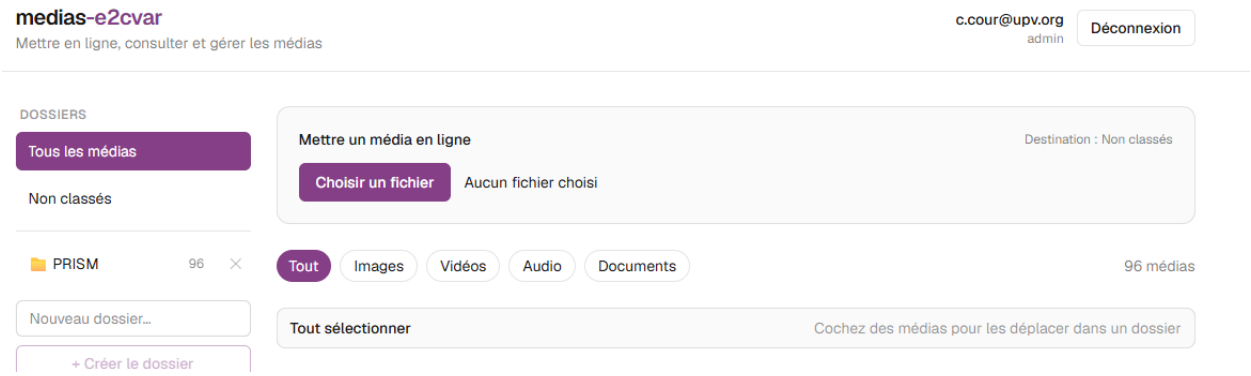
5. Fonctionnalités implémentées

5.1 Authentification

- Page de connexion /login (email + mot de passe).
- Vérification via bcrypt contre la table users existante (10 comptes réels).
- Toute l'application est protégée : redirection vers /login si non connecté, API en 401.
- En-tête avec nom d'utilisateur, rôle, et bouton de déconnexion (server action).
- Session JWT (NextAuth), portant id et role de l'utilisateur.

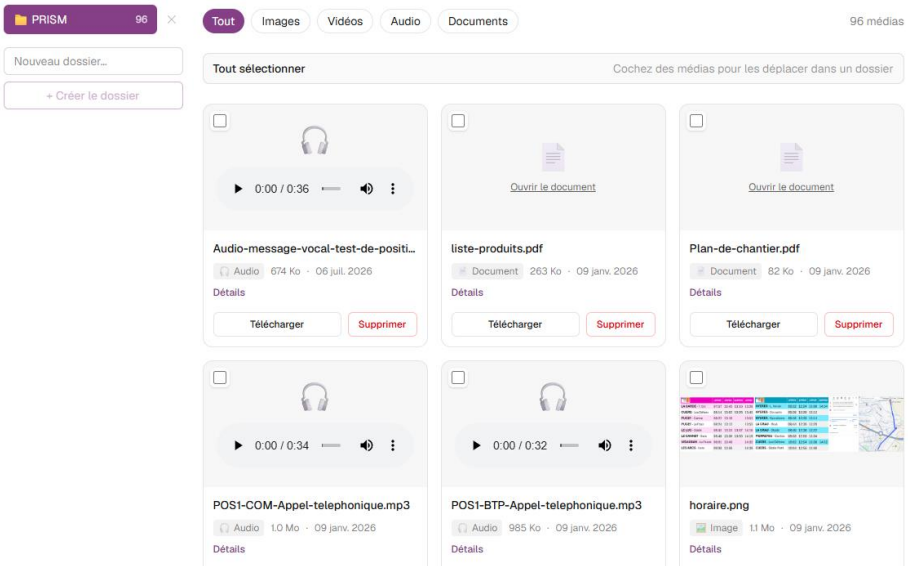
5.2 Gestion des médias

- Upload avec aperçu (image), renommage du fichier avant envoi, validation taille et type MIME côté client et serveur (image / audio / vidéo / PDF).
- Consultation : aperçu image, lecteur audio/vidéo avec support des requêtes HTTP Range (seek), ouverture des PDF.
- Téléchargement (?download → Content-Disposition: attachment).
- Suppression (fichier disque + enregistrement base).
- Panneau de détails par média : qui l'a envoyé, dates d'ajout / modification, type MIME, taille.
- Rattachement automatique de chaque upload à l'utilisateur connecté.



5.3 Organisation par dossiers (projets)

- Création / suppression de dossiers depuis un menu latéral.
- Filtrage de la galerie par dossier (+ « Tous les médias » et « Non classés »), compteurs par dossier.
- Upload directement dans le dossier ouvert.
- **Déplacement en lot** : cases à cocher sur les cartes, « Tout sélectionner », puis « Déplacer vers … » un dossier (route `PATCH /api/media`, `updateMany`). Répond au besoin de ranger d'un coup de nombreux fichiers dans un projet.



5.4 Design

- Couleur principale #843F86 intégrée comme token de thème Tailwind (brand), avec variantes hover et adaptation au mode sombre. Appliquée aux boutons, filtres, sélections, liens.
- Interface responsive (galerie en grille, menu latéral repliable sur mobile), en français.

6. Architecture / structure du code

```
src/
├─ app/
│  ├─ api/
│  │  ├─ media/route.ts          # GET liste (filtre dossier), POST upload, PATCH
déplacement
│  │  ├─ media/[id]/route.ts     # GET streaming (+Range, +download), DELETE
│  │  ├─ folders/route.ts        # GET liste (+compteurs), POST création
│  │  ├─ folders/[id]/route.ts   # DELETE
│  │  └─ auth/[...nextauth]/route.ts
│  ├─ login/                     # page + server action de connexion
│  ├─ page.tsx                   # accueil protégé (header + galerie)
│  └─ layout.tsx
├─ components/                   # Gallery, UploadForm, MediaCard, LoginForm
├─ lib/                          # prisma, storage, media, format, types
├─ auth.ts                       # configuration NextAuth (Credentials + bcrypt)
└─ generated/prisma/             # client Prisma généré (non versionné)

prisma/schema.prisma             # schéma (introspecté + étendu)
backend/files/                   # fichiers média sur disque (non versionnés)
```

Principes appliqués

- **Séparation des responsabilités** : helpers dédiés (`storage.ts` pour le disque, `media.ts` pour la sérialisation JSON, `types.ts` pour les types partagés client/serveur sans dépendance Node).
- **Sérialisation sûre** : conversion du BigInt (size) en number pour le JSON, exposition d'une URL de lecture et des infos d'uploader.
- **Singleton Prisma** pour éviter d'ouvrir une connexion à chaque hot-reload.

7. Points techniques notables

Problèmes rencontrés et résolus au fil du développement :

- **Prisma 7 sans adapter** → erreur `Using engine type "client" requires either "adapter" or "accelerateUrl"`. Résolu en branchant `@prisma/adapter-mariadb` sur le PrismaClient.
- **prisma db push ne régénère pas toujours le client** → `prisma.folders` undefined au runtime. Résolu en lançant explicitement `npx prisma generate` après chaque changement de schéma.
- **Streaming disque + Range** : lecture via `fs.createReadStream` convertie en flux web (`Readable.toWeb`), gestion des statuts 200 / 206 Partial Content / 416.
- **Sécurité des chemins** : les uploads sont écrits sous un nom UUID, et la résolution du chemin disque n'utilise que le basename du chemin stocké (anti path-traversal). Effet de bord utile : on peut changer le dossier de stockage sans casser les chemins existants.
- **Renommage avant upload** : si l'utilisateur retire l'extension, elle est réattachée automatiquement à partir du fichier d'origine (pour un téléchargement correct).
- **Fichiers absents** : quand un enregistrement pointe vers un fichier non présent, la lecture renvoie 410 (avec `Cache-Control: no-store` pour qu'un fichier ajouté ensuite s'affiche aussitôt). L'UI affiche alors une carte « Fichier absent du stockage ».
- **Nom de dossier avec tirets** (`medias-e2c-var`) géré dans l'URL de connexion Prisma.
- **Casse des noms de fichiers** : sensible sous Linux (prod) vs insensible sous Windows (local) — documenté pour le déploiement.

8. Sécurité

- Toutes les routes API et pages exigent une session authentifiée.
- Mots de passe bcrypt (jamais stockés ni comparés en clair).
- Validation taille + type MIME côté serveur (on ne se fie pas à l'extension).
- Anti path-traversal sur la résolution des fichiers disque.
- Secrets hors dépôt : `.env` et `Prod.md` sont git-ignorés.
- `Cache-Control: immutable` sur les fichiers servis (perf), `no-store` sur les erreurs.

9. Vérifications effectuées

Tests manuels de bout en bout réalisés sur l'ensemble des fonctionnalités :

- Gating auth : API 401 sans session, / redirige (307) vers /login, /login accessible.
- Connexion complète (flux NextAuth Credentials) → session avec id/role.
- Upload → écriture disque + enregistrement base ; lecture 200 + Range 206.
- Renommage vérifié (« Mon super visuel » → « Mon super visuel.png »).
- Dossiers : création, filtrage, compteurs, suppression (médias conservés).
- Déplacement en lot de plusieurs médias vers un dossier (moved: N).
- Détails d'un média (uploader, dates) corrects.
- Cas d'erreur : type refusé 415, fichier absent 410, non authentifié 401.
- Cohérence base ↔ disque : les 96 fichiers de la base tous présents sur le disque.
- Nettoyage complet des données de test après vérification.

10. Déploiement

Un guide de mise en production dédié (Prod.md, git-ignoré) détaille tout ce qui change entre local et prod (Evolix) : variables d'environnement (AUTH_SECRET, AUTH_URL, DATABASE_URL, MEDIA_STORAGE_DIR), création base MySQL + utilisateur dédié, application du schéma, dossier de stockage pérenne hors dépôt, build (npm ci → prisma generate → build), service systemd/PM2, reverse proxy Nginx/Apache (taille max d'upload, HTTPS, cookies), sauvegardes, procédure de mise à jour.