
README Pense-Bête : Projet Gestion Usagers - PASQ

Ce document résume les principales procédures, fonctionnements, et points d'attention pour le projet "Gestion Usagers - PASQ", en se basant sur les étapes de développement et de résolution de problèmes rencontrées.

1. Architecture Générale et Technologies Clés

- **Framework Principal :** Next.js (App Router)
- **Langages :** TypeScript, React (pour le frontend)
- **Styling :** TailwindCSS
- **Base de Données & ORM :** PostgreSQL avec Prisma
- **Authentification :** NextAuth.js
- **Structure des Dossiers Principaux :**
    - src/app/ : Pages et routes de l'application (logique serveur pour les Route Handlers, composants serveur/client pour l'UI).
    - src/components/ : Composants React réutilisables (principalement des composants clients "use client";).
    - src/lib/ : Configurations et utilitaires (ex: authOptions.ts, prisma.ts).
    - src/types/ : Définitions de types TypeScript (ex: user.ts).
    - src/utils/ : Fonctions utilitaires (ex: secteurUtils.ts, formatDateForInput.ts).
    - prisma/ : Schéma (schema.prisma) et migrations.

2. Gestion des Données avec Prisma

- **Schéma (prisma/schema.prisma) :**
    - Définit les modèles de données (User, Gestionnaire, Adresse, Problematique, ActionSuivi, Document).
    - **Point d'Attention (Relations) :** Les relations entre modèles (ex: User et Gestionnaire, User et Adresse) sont définies avec des clés étrangères (ex: gestionnaireId, adresseId) et des champs de relation (@relation).
    - **Unicité :** Le champ email sur Gestionnaire est @unique. Le champ email sur User (usagers) n'est PAS unique et est optionnel.
- **Migrations :**
    - Après toute modification du schema.prisma, exécuter :
        1. npx prisma migrate dev --name "nom_descriptif_de_la_migration" (crée et applique la migration en développement).
        2. npx prisma generate (met à jour le client Prisma et ses types TypeScript).
    - **Point d'Attention (Données Existantes) :** Lors de migrations qui modifient/suppriment des colonnes, Prisma avertit si des données sont présentes. Évaluer l'impact et prévoir une stratégie de migration de données si nécessaire AVANT d'appliquer la migration destructive.
- **Opérations CRUD avec Prisma Client :**
    - **Création/Mise à Jour avec Relations :**

        - Lors de la création (prisma.user.create) ou de la mise à jour (prisma.user.update) d'un enregistrement qui a des relations, utiliser la syntaxe connect pour lier à des enregistrements existants.
            - Exemple : data: { nom: "...", adresse: { connect: { id: adresseId } }, gestionnaire: { connect: { id: gestionnaireId } } }
            - **NE PAS** passer les clés étrangères (adresseId, gestionnaireId) comme des champs de premier niveau dans data si on utilise connect pour la même relation.
        - Pour désassigner une relation optionnelle lors d'une mise à jour : data: { gestionnaire: { disconnect: true } } ou data: { gestionnaireId: null }.
    - **Recherche (findUnique, findFirst, findMany) :**
        - findUnique : Ne peut être utilisé que sur des champs @id ou @unique. Une erreur surviendra si on l'utilise sur un champ non unique (ex: User.email). Utiliser findFirst dans ce cas.

3. Authentification et Autorisation (NextAuth.js)

- **Configuration :** src/lib/authOptions.ts.
- **Provider Principal :** CredentialsProvider basé sur email et mot de passe pour les **Gestionnaires**.
- **Session :** Utiliser useSession (côté client) et getServerSession (côté serveur/Route Handlers) pour accéder aux informations de l'utilisateur connecté.
- **Contenu de la Session :** S'assurer que le callback session dans authOptions.ts peuple correctement l'objet session.user avec les informations nécessaires (comme id, nom, prenom, email, role du **Gestionnaire**).
- **Protection des Routes/API :** Vérifier la session et le rôle de l'utilisateur dans les Route Handlers et pour l'affichage conditionnel de l'UI.

4. Style et Design (TailwindCSS)

- **Fichiers Clés :**
    - tailwind.config.js : Pour définir la palette de couleurs personnalisée, les polices, les espacements, etc.
    - src/app/globals.css : Pour les styles globaux de base, l'importation des polices, et les directives @tailwind.
    - src/app/layout.tsx : Applique la structure de page globale et peut contenir des classes de fond pour <body>.
- **Cohérence Visuelle :**
    - **Header Unifié :** Un composant global (ex: src/components/GlobalHeader.tsx) utilisé dans layout.tsx assure la présence du logo, du titre de l'application, de la navigation principale et des actions utilisateur sur toutes les pages.
    - **Titres de Page :** Appliquer un style cohérent (ex: grand, gras, couleur bleue principale) aux titres de chaque page (dans les fichiers src/app/**/page.tsx).
    - **Conteneurs de Contenu :** Utiliser des conteneurs div avec fond blanc, coins arrondis, padding et ombre (bg-white p-6 rounded-lg shadow-md) pour le contenu principal de chaque page afin de le faire ressortir sur le fond gris clair.
- **Points d'Attention (Visibilité du Texte) :**
    - **Problème fréquent :** Texte invisible si sa couleur est identique ou trop proche de la couleur de fond de son conteneur.

    - **Labels de formulaire (<label>) :** Doivent avoir une classe de couleur de texte explicite et contrastante (ex: text-gray-700, text-slate-800).
    - **Inputs (<input>, <textarea>, <select>) :**
        - Texte saisi : Doit avoir une classe de couleur de texte explicite (ex: text-gray-900).
        - Placeholder : Doit avoir une classe de couleur distincte (ex: placeholder-gray-400).
    - **Boutons :** Le texte des boutons doit contraster avec leur fond. Attention aux boutons à fond clair.
    - **Texte général :** S'assurer que les paragraphes et autres éléments textuels ont une couleur de texte lisible.

5. Fonctionnalités Spécifiques et Points de Débogage Rencontrés

- **Importation Excel (src/app/api/users/import/route.ts) :**
    - **Dépendance aux Noms de Colonnes :** Le script est sensible aux noms exacts des colonnes dans le fichier Excel. Toute modification des en-têtes Excel nécessite une mise à jour du mapping dans le script.
    - **Gestion des Relations (Adresse, Gestionnaire) :** Crucial d'utiliser la syntaxe connect: { id: ... } dans prisma.user.create() pour lier les relations, et non d'assigner les clés étrangères directement au premier niveau de l'objet data de create.
    - **Recherche de Gestionnaire :** Si l'Excel contient le prénom du gestionnaire, utiliser prisma.gestionnaire.findFirst({ where: { prenom: { equals: ..., mode: 'insensitive' } } }). Gérer le cas où aucun gestionnaire n'est trouvé ou si plusieurs ont le même prénom. L'utilisation de l'email (unique) du gestionnaire dans l'Excel est plus robuste.
    - **Parsing des Dates :** Utiliser cellDates: true avec XLSX.read et une fonction parseExcelDate robuste pour gérer différents formats.
- **Formulaires (UserForm.tsx, GestionnaireSettings.tsx) :**
    - **Dates dans les Inputs :** Les champs <input type="date"> attendent une valeur string au format YYYY-MM-DD. Utiliser une fonction utilitaire formatDateForInput(date: string | Date | null | undefined): string pour convertir les objets Date ou les chaînes ISO avant de les passer à l'attribut value.
    - **Déclenchement de Soumission par Parent (useImperativeHandle) :** Pour des scénarios comme "sauvegarder avant de générer RGPD", UserForm.tsx peut exposer une méthode submitForm() via useImperativeHandle et forwardRef pour que le composant parent puisse la déclencher. Une méthode isDirty() peut aussi être exposée pour vérifier les modifications non sauvegardées.
- **Génération PDF (RGPD) :**
    - **Côté client avec jsPDF :**
        - La fonction initiateRgpdAttestationGeneration construit le PDF.
        - La mise en page (polices, tailles, positionnement yPos, sauts de page) demande des ajustements précis. Une fonction helper addProcessedText peut aider.
        - Après génération, un fetch PATCH vers une API met à jour User.rgpdAttestationGeneratedAt.
        - Le composant page doit ensuite rafraîchir ses données (ex: refetch userData) pour que l'UI (texte du bouton RGPD) se mette à jour.

    - **Erreurs de rendu @react-pdf/renderer :** Si cette bibliothèque est utilisée ailleurs, attention à l'erreur Invalid ' ' string child outside <Text> component. Tout texte doit être dans un composant <Text>. Une telle erreur peut bloquer le rendu de parties non liées de l'application.

6. Build Docker et Déploiement

- **Dockerfile :**
    - **npx prisma generate :** Cette commande DOIT être exécutée dans le Dockerfile APRÈS la copie du schema.prisma et l'installation des node_modules, et AVANT npm run build. C'est essentiel pour que le client Prisma soit à jour avec le schéma.
    - **Variables d'Environnement :** DATABASE_URL doit être disponible au moment du build (si Next.js y accède) et absolument à l'exécution.
- **Processus avant docker build :**
    1. S'assurer que npx prisma migrate deploy (ou dev si c'est pour un environnement de test) a été exécuté et que la base de données est à jour avec le schéma.
    2. Exécuter npm run build **localement** pour attraper les erreurs TypeScript ou ESLint avant de lancer le build Docker.
- **Erreurs de Build Courantes :**
    - Erreurs TypeScript si le code n'est pas valide ou si les types Prisma ne sont pas à jour.
    - Problèmes de configuration Next.js.

Ce document est un point de départ. N'hésitez pas à l'étoffer avec plus de détails spécifiques à mesure que votre projet évolue. J'espère qu'il vous sera utile !