🚀Transition Option 3 — Hybride progressif vers une vraie app web

Voici une feuille de route de prompts pour **Cursor** qui te permet de transformer progressivement ton site statique (HTML/CSS/JS + localStorage) en une **application web sécurisée avec backend**. On avance par **sprints** : chaque sprint déplace une fonctionnalité clé du localStorage vers une **API Node/Express + Postgres**.

Sprint 1 — Backend Auth & DB (base du système)

Prompt à donner à Cursor :

> **Contexte** : On passe d'un MVP localStorage à une app web avec backend sécurisé. Tâche : Créer un dossier `server/` avec un backend **Node.js + Express**. Ajoute : - DB : PostgreSQL (via `pg` ou `prisma`). - Tables : `users(id, name, email unique, role, password_hash, created_at)`. - Routes : `POST /auth/signup`, `POST /auth/login`, `GET /auth/me`. - Authentification : hash des mots de passe avec **bcrypt**, génération de **JWT** pour les sessions. - Middleware : `authRequired` qui vérifie le token. - Fichier `.env` avec `DATABASE_URL`, `JWT_SECRET`, `PORT`. - README expliquant comment lancer `npm install`, `npm run dev`, migrations DB, et tester avec curl ou Postman. Critères d'acceptation : on peut créer un user, se connecter, obtenir un JWT, et récupérer `/auth/me` avec ce token.

Sprint 2 — Brancher le frontend (auth côté client)

Prompt à donner à Cursor :

> **Contexte** : Le front utilisait localStorage pour auth. On passe sur l'API. Tâche : Dans `js/auth.js`, remplacer la logique localStorage par des appels fetch à l'API : - `signup-form` → POST `/auth/signup`. - `login-form` → POST `/auth/login`. - Sauvegarder le **JWT** dans `localStorage` (clé `token`). - Pour récupérer l'utilisateur courant → GET `/auth/me` avec header `Authorization: Bearer TOKEN`. - Gérer erreurs (email déjà utilisé, mauvais mot de passe) avec messages clairs. Critères d'acceptation : après login, `dashboard.html` charge le profil utilisateur via API et non plus via store.js.

Sprint 3 — Projets & rôles dans le projet

Prompt à donner à Cursor :

> **Contexte** : On déplace la gestion de projets/membres du localStorage vers l'API. Tâche : - Backend : - Tables :

`projects(id, name, code, owner_id, status, created_at)`. - Table pivot : `project_members(project_id, user_id, role_in_project, joined_at)`. - Routes : `POST /projects`, `GET /projects`, `GET /projects/:id`, `POST /projects/:id/members`. - Middleware : accès restreint aux membres du projet. - Frontend : - `projects.html` → charger via GET `/projects`. - Création projet → POST `/projects`. - Ouverture projet → GET `/projects/:id`. Critères d'acceptation : les projets sont persistés en DB et visibles par leurs membres.

Sprint 4 — Invitations & e-mails

Prompt à donner à Cursor :

Contexte : On veut inviter BCT/BET/Entreprises par e-mail ou lien. Tâche : - Backend : - Table `invites(id, project_id, email, role_proposed, token, status, expires_at)`. - Routes : `POST /projects/:id/invites` (créer invitation), `POST /invites/:token/accept`, `POST /invites/:id/cancel`. - Envoi e-mail : utiliser nodemailer (SMTP, variables .env : `SMTP_HOST`, `SMTP_USER`, `SMTP_PASS`). - Frontend : - Dans `project.html` onglet **Membres**, le bouton « Inviter » appelle l'API. - Afficher bannière « Vous êtes invité en tant que [rôle] » si `invite=TOKEN` dans l'URL. Critères d'acceptation : un utilisateur invité peut rejoindre un projet, ou refuser.

Sprint 5 — Documents & visas (workflow)

Prompt à donner à Cursor :

Contexte : On déplace la soumission/visa des docs côté API. Tâche : - Backend : - Tables : `docs(id, project_id, title, type, author_id, state, version, url, created_at)` + `doc_history(doc_id, at, by, action)`. - Routes : `POST /projects/:id/docs`, `GET /projects/:id/docs`, `POST /docs/:id/transition` (changer état). - Stockage fichiers : pour l'instant URL externe ; prévoir intégration S3 plus tard. - Frontend : - `project.html` onglet Documents → charger docs via API. - Bouton Soumettre → POST. - Boutons Visa/Réviser → POST transition. Critères d'acceptation : workflow complet persistant côté serveur.

Sprint 6 — Journal & notifications (backend)

Prompt à donner à Cursor :

Contexte : Centraliser la traçabilité et les notifs. Tâche : - Backend : - Table `notifications(id, user_id, text, at, read)`. - Routes : `GET /notifications`, `POST /notifications/mark-read`. - Ajouter insertion de notif lors de : création projet, invite, soumission doc, visa, etc. - Frontend : - Dashboard → charger `/notifications` filtrées par user_id. - Badges de compteur non lus dans le header. Critères d'acceptation : les notifs viennent bien de la DB et non du localStorage.

Sprint 7 — Profil & Paramètres (connexion avec backend)

Prompt à donner à Cursor :

> **Contexte** : La page `profile.html` existe déjà (fiche profil). On connecte au backend. Tâche : - Backend : route `PUT /users/me` pour mettre à jour profil (nom, poste, téléphone, avatar base64 ou URL). - Frontend : `profile.html` → charger données via `/auth/me`, mise à jour via PUT `/users/me`. - Sécurité : seul l'utilisateur courant peut éditer son profil. Critères d'acceptation : les infos modifiées persistent en DB, avatar visible dans le header.

Stratégie de déploiement

- **Frontend** : Vercel/Netlify (pages statiques).
- **Backend** : Railway/Render/Fly.io avec Node/Express.
- **DB** : Postgres managé (Railway, Supabase, Neon).
- **Stockage fichiers** (plus tard) : S3 ou équivalent.
- **Mail** : SMTP (service tiers : SendGrid, Mailgun, etc.).

Étape actuelle (32 revisitée)

J'ai reformulé l'**étape 32** pour qu'elle corresponde à ce que tu veux : la page **Profil utilisateur** doit être visible partout (pas seulement dans un projet) et ce n'est pas un simple formulaire, mais une vraie page profil avec sections bien présentées et un design correct.

Étape 32 — Page « Profil utilisateur » (globale, avec design dédié)

Prompt à donner à Cursor :

> **Contexte** : L'utilisateur doit pouvoir gérer son identité et ses informations de profil depuis **n'importe où dans l'application**, et pas seulement dans un projet. Tâche : Créer une nouvelle page `profile.html` accessible depuis le header (menu utilisateur ou avatar). Cette page n'est pas un simple formulaire : elle doit afficher les informations de l'utilisateur dans un **layout de type fiche profil** avec sections claires (Informations personnelles, Contact, Entreprise, Sécurité, Préférences, Avatar). Chaque section est modifiable soit par édition en ligne (inline editing), soit via une petite modale légère. Critères d'acceptation : - `profile.html` est accessible depuis toutes les pages de l'app via le header. - Les données utilisateur (Nom complet, Poste, Téléphone, Email, Avatar) s'affichent dans un design clair, lisible, avec hiérarchie visuelle. - Les champs sont éditables avec un bouton **Modifier/Enregistrer** ou via modale. - L'avatar peut être mis à jour (upload → base64 → aperçu direct). - Responsive et harmonisé avec le reste du site (pas un formulaire brut, mais une fiche profil bien designée).