Dossier d'Architecture Technique

Projet : PediGuide - Formulaire de pré-diagnostic pédiatrique - **Date :** 29 Janvier 2026

---

1. Synthèse du besoin et enjeux

L'objectif est de fournir une interface de pré-diagnostic qui soit accessible aux parents (souvent en situation de stress) et un backend sécurisé capable de traiter ces données de manière sécurisée.

Les contraintes techniques majeures identifiées sont :

1. **Expérience Utilisateur (UX) :** L'application doit être instantanée, interactive, intuitive et accessible pour les parents en enfants en pédiatrie.
2. **Performance Serverless :** L'hébergement étant "à la demande" (Serverless), le temps de démarrage du backend doit être minime.
3. **Intégrité des données :** Typage strict nécessaire de bout en bout pour éviter les erreurs médicales techniques.

---

2. Choix de la Stack Technique

A. Frontend : Vue.js 3 + Vite + TypeScript

Pourquoi ce choix ?
Nous avons opté pour une **SPA (Single Page Application)** pour le patient pour garantir une fluidité totale sans rechargement de page.

- **Problème résolu :** La complexité de gestion d'un formulaire interactif.
- **Justification technique :**
  - **Vue.js :** Permet une meilleure organisation logique du code qu'une approche classique, facilitant la maintenance des composants

complexes.

- **Vite :** Offre un temps de compilation quasi-instantané et des bundles optimisés pour les connexions mobiles 4G/5G des parents.
- **TypeScript :** Sécurise le développement en empêchant les erreurs de typage sur les objets "Symptômes".

B. Backend : Express.js en mode Serverless

Pourquoi ce choix ?
Au lieu d'un serveur VPS classique tournant 24h/24, nous utilisons une architecture **Serverless** nous permettant d'avoir un serveur lancé uniquement quand nous en avons besoin.

- **Problème résolu :** Coût d'infrastructure et gestion de la montée en charge (pics d'affluence aux urgences/cabinets).
- **Justification technique :**
  - L'architecture Serverless permet de ne consommer des ressources que lorsqu'un formulaire est envoyé.
  - **Express.js** est utilisé pour sa robustesse et sa capacité à être facilement "wrappé" en fonction serverless via Vercel, tout en restant compatible avec un hébergement classique si besoin de migrer plus tard.

C. Base de Données & ORM : Supabase (PostgreSQL) + Drizzle

Pourquoi ce choix ?
C'est le choix le plus critique de notre architecture. Nous avons écarté *Prisma* (souvent standard) au profit de *Drizzle*.

- **Problème technique spécifique : Le "Cold Start".**
  - Sur une architecture Serverless, le backend s'éteint après inactivité. Au réveil, les ORM classiques comme Prisma sont lourds à charger, causant une latence de 1 à 3 secondes.
- **La Solution Drizzle ORM :**
  - **Légèreté :** Drizzle n'a pas de moteur d'exécution (runtime) lourd. Il génère du SQL pur.
  - **Performance :** Le temps de réponse est quasi-instantané, même après une mise en veille du serveur.

- **Sécurité :** Il permet d'exploiter la puissance de PostgreSQL via Supabase tout en gardant un typage TypeScript strict synchronisé avec le code.

D. Infrastructure & CI/CD : Vercel + GitHub Actions

Pourquoi ce choix ?
L'automatisation du déploiement est vitale pour garantir la qualité sans perdre de temps.

- **Problème résolu :** Risque d'erreur humaine lors des mises en production et hétérogénéité des environnements.
- **Justification technique :**
  - **GitHub Actions :** Orchestre les tests de qualité avant chaque modification, et automatise le déploiement lors de la création d'un tag sur la branche `main` .
  - **Vercel :** Gère automatiquement les certificats SSL (HTTPS), le CDN (distribution mondiale du contenu) et le déploiement atomique (si le déploiement échoue, l'ancien site reste en ligne).

---

3. Schéma d'Architecture de Données

Le flux de données suit une logique unidirectionnelle sécurisée :

1. **Client (Vue) :** Saisie et Validation locale.
2. **API Gateway (Vercel/Express) :** Validation serveur et authentification.
3. **ORM (Drizzle) :** Construction de la requête SQL optimisée.
4. **Database (Supabase) :** Persistance des données.

---

4. Accessibilité

L'accessibilité numérique est un pilier fondamental de ce projet, en cohérence avec le contexte médical et les enjeux d'inclusion. L'architecture technique a été conçue pour intégrer les bonnes pratiques d'accessibilité dès la conception, sans complexifier inutilement l'implémentation.

A. Choix techniques pour l'accessibilité

Pourquoi ces choix ?
Dans un contexte de santé pédiatrique, les utilisateurs peuvent se trouver en situation de stress, de fatigue ou présenter des handicaps (visuels, moteurs, cognitifs). L'accessibilité n'est pas une contrainte, mais un levier de qualité.

- **Problème résolu :** Garantir que tous les parents/enfants puissent utiliser l'application, quels que soient leurs capacités, leur matériel ou leur contexte d'utilisation.
- **Structure sémantique HTML**
  - **Justification technique :**
    - Utilisation systématique d'éléments HTML sémantiques (`<form>`, `<label>`, `<input>`, `<button>`, `<fieldset>`) plutôt que des `<div>` génériques.
    - Hiérarchie de titres cohérente pour faciliter la navigation au lecteur d'écran.
    - Repères de navigation (landmarks) pour identifier les zones principales.

    Impact : Les technologies d'assistance comprennent naturellement la structure, sans nécessiter de surcouche ARIA complexe.
- **Formulaire accessible**
  - **Justification technique :**
    - Labels visibles et explicitement associés à chaque champ (pas de placeholders seuls).
    - Messages d'erreur contextuels, annoncés dynamiquement aux lecteurs d'écran.
    - Indication claire des champs obligatoires (texte, pas uniquement couleur).

    Impact : Réduction du taux d'erreur et d'abandon, amélioration de l'expérience pour tous les utilisateurs.
- **Navigation au clavier**
  - **Justification technique :**
    - Tous les éléments interactifs sont accessibles au clavier.
    - Focus visible et contrasté (≥ 3:1) sur tous les éléments.

        - Ordre de tabulation logique suivant le flux naturel du formulaire.

      Impact : Utilisable par les personnes à mobilité réduite, les utilisateurs de lecteurs d'écran et les utilisateurs experts préférant le clavier.

- **Utilisation mesurée d'ARIA**
    - **Justification technique :**
        - ARIA utilisé uniquement lorsque le HTML standard ne suffit pas (ex : indicateur d'étapes, zones dynamiques).
        - Respect de la règle "No ARIA is better than Bad ARIA".

      Impact : Évite les conflits avec les technologies d'assistance tout en enrichissant l'expérience lorsque nécessaire.

B. Accessibilité cognitive et lisibilité

Dans un contexte de pré-diagnostic pédiatrique, la charge cognitive doit être minimisée :

- **Langage simple :** Questions claires, sans jargon médical inutile.
- **Parcours progressif :** Formulaire découpé en étapes courtes pour rassurer et éviter la surcharge.
- **Feedback explicite :** Messages de validation et d'erreur compréhensibles par tous.

Justification technique :

- Découpage du formulaire en composants Vue distincts pour chaque étape.
- Système de validation progressif côté client pour un feedback immédiat.

C. Contrastes et lisibilité visuelle

Standards respectés :

- Contrastes texte/fond ≥ 4,5:1 (texte normal) et ≥ 3:1 (texte large).
- Taille de texte minimale : 16px.
- Zones cliquables : 44×44px minimum.
- Informations jamais transmises uniquement par la couleur.

Justification technique :

- Variables CSS centralisées pour la gestion des couleurs et contrastes.

- Système de design tokens intégré dans Vue/Vite pour garantir la cohérence.

D. Conformité et référentiel

L'application vise une conformité aux **critères WCAG 2.1 niveau AA**, qui constitue le standard international et la base du RGAA (référentiel français). Cette conformité s'inscrit naturellement dans la démarche de qualité du projet, sans surcoût technique significatif grâce aux choix d'architecture réalisés en amont.

---

5. Sécurité et RGPD

Bien que le prototype utilise des services cloud standards, l'architecture a été pensée pour la conformité :

- **Minimisation des données :** Seules les données strictement nécessaires au pré-diagnostic sont collectées.
- **Séparation :** L'architecture permet techniquement de séparer la base de données du serveur d'application.
- **Chiffrement :** Toutes les communications sont chiffrées en transit (TLS 1.3) et la base de données est chiffrée au repos.

---

6. Conclusion

L'architecture technique de **PediGuide** ne répond pas seulement à un besoin fonctionnel, elle adresse des contraintes de performance et de scalabilité modernes.

L'utilisation du couple **Drizzle + Serverless** démontre une volonté d'optimiser les ressources et l'expérience utilisateur, garantissant que la technologie s'efface au profit du service médical rendu.