Analyse Préliminaire de la Codebase Monka.care

Rapport de Recommandations Techniques & Stratégiques

The Vibe Company

Stan Girard & Antoine Dewez

Février 2026

Destinataires : Étienne, Maël, Raphaël — Monka.care

Classification : Confidentiel

Version : 1.1 — 23 février 2026

Ce document présente les résultats de l'analyse automatisée de 14 repositories, croisée avec le cahier des charges Klésia, le call de qualification, et les réponses techniques de l'équipe Monka.

Table des matières

Chapitre 1

Résumé exécutif

Contexte de la mission

Monka.care accompagne les aidants familiaux depuis quatre ans avec une plateforme numérique de soutien, d'orientation et de suivi. L'entreprise entre dans une phase d'accélération produit autour de deux projets structurants :

— **Projet Klésia** : mise en production d'un parcours complet conforme au cahier des charges validé avec la mutuelle. C'est la priorité immédiate.
— **Expérimentation Article 51** : candidature à une expérimentation avec le ministère de la Santé, avec des exigences fortes en traçabilité, structuration des données et conformité réglementaire.

Diagnostic technique

The Vibe Company a analysé les **14 repositories** de Monka.care. Le constat est net :

— 4 repositories actifs contiennent du code exploitable (API principale, back-office, application mobile, infrastructure).
— 10 repositories sont obsolètes ou vides. Ils n'apportent rien au produit et compliquent la gestion du code.

Sur le plan fonctionnel, l'écart avec le cahier des charges Klésia est important. Aucune des fonctionnalités attendues n'est implémentée :

— Les parcours Diagnostic, Mise en relation et Hospitalisation (D/M/H) n'existent pas.
— Le rôle de Care Manager, central dans le dispositif d'accompagnement, n'est pas modélisé.
— Le Plan Personnalisé de Prévention (PPP) n'a pas de structure exploitable.
— Le scoring situationnel et fonctionnel (S/F), nécessaire à l'orientation des aidants, est absent.

La base technique existe, mais le produit attendu reste entièrement à construire.

Recommandation stratégique

Une réécriture totale serait trop risquée. Un empilement sur l'existant serait trop fragile pour les exigences à venir. Nous recommandons une approche hybride :

Brownfield sur le **backend et le CRM** : consolider l'API existante, restructurer le modèle de données, étendre les fonctionnalités métier sur des fondations déjà en place.

Greenfield sur les **frontends** : reconstruire les interfaces utilisateur (application aidant, espace professionnel, back-office) sur une stack moderne, alignée avec les parcours du cahier des charges.

Cette stratégie s'appuie aussi sur le développement assisté par IA pour accélérer la production de code, la génération de tests et la documentation technique.

The Vibe Company s'engage à fournir l'accompagnement technique nécessaire pour passer d'un existant fragmenté à une plateforme conforme aux attentes de Klésia et prête pour l'expérimentation Article 51. Les chapitres suivants détaillent le diagnostic, l'architecture cible et les conditions de la mission.

Chapitre 2

Besoins client et cahier des charges

2.1 Vision et ambition

Monka x Prev&Care vise à construire un système hybride de gestion de la vulnérabilité des aidants familiaux. Le principe : combiner l'autonomie du digital (*self-care*) avec l'expertise humaine (IDEC, Care Managers). L'enjeu est concret — sortir les aidants de l'isolement en leur proposant un accompagnement calibré sur leur situation et leur aisance avec le numérique.

La stratégie repose sur trois axes :

1. **Le questionnaire d'évaluation** — Les ressentis subjectifs deviennent des scores de vulnérabilité (score S pour l'aidant, score F pour l'aidé). Ces scores déterminent le parcours (Digital, Mixte ou Humain) et alimentent un Plan d'action Personnalisé (PPP). Le questionnaire passe de 30 à 150 questions, avec une navigation gamifiée.
2. **Le CRM LifeLine** — Un cockpit centralisé pour les accompagnateurs (IDEC et Care Managers). Ils s'y répartissent les utilisateurs selon leurs affinités métiers, pilotent les parcours et assurent le suivi des binômes Aidant/Aidé.
3. **Les analytics d'usage et d'efficience** — Des données objectivables pour prouver la valeur du modèle hybride : scalabilité sur de grandes populations, réduction des coûts d'intervention, comparaison avec le parcours 100% humain que propose aujourd'hui Prev&Care.

La plateforme couvre quatre applicatifs : l'application mobile MyMonka (aidants), le CRM LifeLine (Care Managers et équipes internes), les systèmes de mailing et notifications, et l'export de KPIs consolidés.

2.2 Les 5 epics du cahier des charges

Le backlog fonctionnel est structuré en cinq epics. Elles couvrent toute la chaîne, de l'acquisition de l'aidant au pilotage stratégique.

2.2.1 Epic 1 — Onboarding (jusqu'au questionnaire)

L'objectif est de maximiser l'autonomie digitale (parcours D/M) pour limiter les coûts opérationnels, tout en gardant une prise en charge humaine fluide (parcours H) via le Care Manager.

- Un pré-questionnaire web responsive : collecte des données Aidant, consentement RGPD, branding paramétrable par client, choix binaire (téléchargement de l'app en autonomie ou mise en contact avec un interlocuteur).
- Un système d'activation simplifié par code B2B, lien email et QR code. L'objectif : supprimer les frictions techniques (ressaisie, code d'activation).
- Un mécanisme de relance automatisé en trois paliers (J+2, J+5, J+7) avec escalade vers phoning CM, puis suppression RGPD des données après délai.
- Un onboarding assisté par le CSM (création de compte, orientation D ou H) et un remplissage proxy du questionnaire par le CM lors d'un entretien téléphonique, avec génération PDF du diagnostic.

2.2.2 Epic 2 — Scoring et attribution des parcours

Cette epic transforme les réponses au questionnaire en indicateurs de vulnérabilité. Elle définit le parcours, désigne le référent et génère le PPP.

- Un moteur de scoring qui applique des seuils de vulnérabilité (définis par le comité médical) pour attribuer le statut "Non vulnérable" (parcours D) ou "Vulnérable" (parcours M). Le parcours H est attribué manuellement pour les aidants non aptes au numérique.
- Une segmentation et attribution automatique du référent : le CM est référent par défaut ; l'IDEC est désignée si la vulnérabilité est exclusivement "Santé" (indicateurs S1–S4). La réversibilité manuelle est prévue.
- Un routage du Plan d'action (PPP) : distribution automatique des tâches selon l'expertise métier (santé vers IDEC, administratif/social vers CM), typage des actions (autonome vs accompagnée), validation par le référent avant visibilité aidant.

2.2.3 Epic 3 — Interface CRM LifeLine

Cette epic fournit aux accompagnateurs leur interface de pilotage quotidienne.

- Un dashboard avec vue d'ensemble de l'activité du jour, pool des aidants non attribués (parcours H) et auto-assignation possible par les CM.
- Des fiches profils détaillées (Aidant et Aidé) avec vues partagées IDEC/CM, cloisonnement des accès selon le cercle de soin, historique des transmissions pour éviter les doublons.
- Une messagerie bidirectionnelle : le CM échange avec l'aidant via MyMonka, au même titre que l'IDEC.
- Un module de gestion des calendriers et RDV avec synchronisation automatique, notifications de confirmation et sélecteur de types de RDV (bilan initial, suivi, urgence).

2.2.4 Epic 4 — Cycle de vie et coordination

Cette epic assure la continuité de l'accompagnement dans le temps et formalise les transitions.

- Un bilan automatique à 3 mois : notification push/mail pour le parcours D, tâche "Appel de bilan trimestriel" pour les parcours M/H, mise à jour des scores avec visualisation de la progression.
- Des passerelles de bascule entre parcours (D vers M/H en cas de fragilité apparue, M/H vers D en cas de stabilisation), avec changement de référent possible. Traçabilité obligatoire : le motif doit être documenté.
- Un processus de clôture conjointe : validation requise des deux accompagnateurs (IDEC et CM), envoi du PPP finalisé en PDF à l'aidant, collecte du motif de clôture.
- Une collecte des motifs d'attrition (mini-sondage en cas d'abandon) pour alimenter le reporting et repérer les points de friction produit.

2.2.5 Epic 5 — Pilotage, analytics et reporting

Cette epic centralise les données pour mesurer la valeur du service et améliorer les processus.

- Reporting stratégique Klésia (vue externe) : indicateurs d'usage (nombre d'aidants, répartition par parcours, diagnostics délivrés, RDV réalisés), indicateurs de performance (NPS, taux de clôture, SLA), indicateurs médico-économiques (coût moyen par aidant, comparaison avec modèles traditionnels).
- Pilotage opérationnel (vue interne Monka + Prev&Care) : funnel d'onboarding (taux d'activation, taux de complétion questionnaire, efficacité des relances), qualité opérationnelle (SLA, charge ops, satisfaction post-bilans), production PPP, santé produit (incidents, support, rétention), suivi des transitions entre parcours.

2.3 Priorités fonctionnelles

Le client a clairement établi les priorités. Le tableau 2.1 distingue l'indispensable du souhaitable.

Fonctionnalité	**Priorité**
Refonte complète du parcours utilisateur MyMonka	Must-have
Questionnaire 150 questions avec navigation gamifiée	Must-have
Moteur de règles complexes (scoring, attribution)	Must-have
Plan d'action Personnalisé (PPP) — Key feature	Must-have
Logique de niveaux d'accès payants (upgrade / downgrade / freemium)	Must-have
Système in-house de prise de RDV (type Calendly)	Nice-to-have

Table 2.1 – Priorisation des fonctionnalités

2.4 Contraintes techniques et réglementaires

2.4.1 Environnement réglementaire

Le projet touche au secteur de la santé. Le cadre réglementaire est exigeant.

— **Hébergement de Données de Santé (HDS)** — L'infrastructure est chez AZNetwork, hébergeur certifié HDS. L'accès aux environnements de production passe par VPN (Tunnelblick). Un besoin identifié : mettre en place un accès sécurisé pour un chercheur basé à Singapour (modèle ML Python sur une vue de la base de données).
— **RGPD** — Consentement explicite requis dès le pré-questionnaire, suppression des données de formulaire après un délai défini, cloisonnement des accès selon le rôle (CM, IDEC, cercle de soin). L'exercice RGPD sur le périmètre des vues partagées CM/IDEC reste à valider.
— **Interopérabilité santé** — À moyen terme, intégration avec les référentiels nationaux : RPPS (Répertoire Partagé des Professionnels de Santé), Pro Santé Connect (authentification des professionnels), Mon Espace Santé, DPI hospitaliers. Ces sujets sont identifiés mais restent à prioriser dans le temps.

2.4.2 Contraintes d'équipe et organisationnelles

— **Équipe réduite** — Deux développeurs internes : Raphaël (Tech Lead) et Yann (expert métier). Tous deux souhaitent monter en compétence sur les pratiques IA/vibe coding.
— **Stack existante** — Backend NestJS 11 / TypeScript 5.9 / PostgreSQL 10.6, frontend React + Vite, mobile React Native 0.81.5 + Expo 54, déploiement Docker via Portainer.
— **CI/CD** — GitHub Actions en place (backend, mobile, backoffice) mais avec des versions anciennes (v1/v2) à mettre à jour. Tests unitaires et e2e sur le backend, e2e en cours sur LifeLine, pas de tests e2e sur MyMonka.
— **Décision architecturale pendante** — Greenfield ou refactoring de l'existant (MyMonka + LifeLine) ? C'est un choix structurant à trancher vite, conditionné par l'audit de code.

Le tableau 2.2 récapitule les composants techniques.

Composant	**Stack**	**Node.js**
Backend (monka-api)	NestJS 11, TypeScript 5.9, PostgreSQL 10.6, TypeORM	v22
Frontend Pro / Backoffice	React, Vite, Cypress (e2e)	v18–20
Mobile (MyMonka)	React Native 0.81.5, Expo 54, EAS Build	v20.19.4
Auth	JWT + Passport + Auth0	–
Monitoring	Sentry (@sentry/nestjs)	–
Hébergement	AZNetwork (HDS), Docker Hub, Portainer	–

TABLE 2.2 – Synthèse de la stack technique existante

Chapitre 3

État des Lieux Technique

L'audit technique a porté sur l'ensemble des 14 repositories de l'écosystème Monka.care. Ce chapitre dresse la cartographie complète, puis détaille les quatre composants actifs et pertinents pour le projet Klésia.

3.1 Cartographie de l'écosystème

Le patrimoine logiciel de Monka.care compte 14 repositories Git. Seuls 4 sont activement maintenus et pertinents pour les livrables Klésia.

Repository	Rôle	Dernière activité	État	Pertinence
`monka-api`	Backend API NestJS, coeur métier	2026-02-19	Actif	**Haute**
`monka-backoffice`	CRM LifeLine (IDEC)	2026-02-10	Actif	**Haute**
`mobileAppRN`	App mobile aidants (MyMonka)	2026-02-19	Actif	**Haute**
`monka-frontend`	Frontend web aidants	2026-02-19	Actif	**Haute**
`monka-ece`	Portail abonnement Stripe	2025-10-06	Inactif	Faible
`rpps-api`	Annuaire RPPS/FINESS	2025-03-14	Inactif	Faible
`monka-admin`	Backoffice Forest Admin	2024-07-19	Inactif	Faible
`monka-pro`	Portail intervenants terrain	2024-10-24	Inactif	Nulle
`monka_pwa`	Ancienne plateforme V1 (mono-repo)	2023-02-14	Inactif	Nulle
`monka`	Méta-repo git submodules	2023-05-30	Inactif	Nulle
`monka-ia`	Placeholder IA/ML	2023-06-05	Vide	Nulle
`monka-infra`	Placeholder infrastructure	—	Vide	Nulle
`frontend-proxy`	Reverse proxy NGINX (ancien)	2023-02-13	Inactif	Nulle
`pro-proxy`	Reverse proxy NGINX (ancien)	2023-02-24	Inactif	Nulle

Table 3.1 – Cartographie des 14 repositories Monka.care

Sur 14 repositories, 4 sont actifs et pertinents pour Klésia, 3 sont marginalement utiles et 7 sont

obsolètes ou vides.

3.2 Backend API (`monka-api`)

3.2.1 Stack technique

— **Framework** : NestJS 11 + TypeScript 5.9 + Node 22 LTS
— **Base de données** : PostgreSQL 10.6 (via Docker) + TypeORM 0.3.16
— **Authentification** : Auth0 (JWKS/RS256) pour le mobile, JWT symétrique pour le CRM et les professionnels
— **Intégrations** : Stripe, Mailjet, Calendly, Sentry, MinIO

3.2.2 Forces

— **Architecture modulaire propre** : 54 modules NestJS, 65 controllers, 57 services, 37 entités — séparation correcte des responsabilités avec contrôleurs distincts par contexte (app, CRM, pro).
— **Authentification dual** : Auth0 RS256 pour le mobile (standard sécurisé), JWT local pour le CRM avec un guard `DenyByDefault` qui bloque tout par défaut.
— **Validation des entrées** : 89 classes DTO avec plus de 1 000 décorateurs `class-validator`, `ValidationPipe` global en mode strict (`whitelist`, `forbidNonWhitelisted`).
— **Historique de données conséquent** : 250 migrations couvrant 7 ans d'évolution du schéma, exécutions up/down symétriques.
— **Dépendances principales à jour** : NestJS 11, TypeScript 5.9 et Node 22 sont sur leurs versions stables courantes. Stripe 18 a deux versions majeures de retard (v20 disponible).

3.2.3 Faiblesses

— **PostgreSQL 10.6 en fin de vie** (EOL novembre 2022) : plus de correctifs de sécurité depuis 3 ans, 8 versions majeures de retard. La CI teste sur PostgreSQL 16+ tandis que la production tourne sur 10.6 — divergence critique.
— **Module `survey` monolithique** : 19 500 LOC (42 % du code total), dont le `SurveyService` de 2 228 lignes (God Object qui mêle scoring, PDF, Excel, CRUD). Duplication massive : 3 fichiers d'adapters quasi identiques (5 277 LOC), 4 fichiers de définitions dupliqués (9 777 LOC).
— **Couverture de tests insuffisante** : 35 fichiers de tests unitaires et 8 e2e pour 342 fichiers source (∼13 %). Le test du service le plus critique (`survey.service.spec.ts`) vérifie uniquement que `true === true` — test fantôme.
— **CORS wildcard en production** : `ALLOWED_HOST=*` dans `.env.production`.
— **Secrets committés** : clés Stripe LIVE, JWT_SECRET, clés de chiffrement, Mailjet et Calendly présents dans les fichiers `.env.staging` et `.env.production` versionnés.
— **Chiffrement PII désactivé** : les transformateurs `EncryptedString` sont commentés dans `IdentityEntity`.

— **Anti-patterns récurrents** : 113 usages de `any`, 68 `console.log` en production, 6 occurrences de `forEach(async ...)` (erreurs silencieuses).

3.2.4 Notation détaillée

Critère	Note / 10	Commentaire
Stack technique	7	NestJS 11, TS 5.9, Node 22 à jour ; Stripe 18 en retard (v20)
Architecture	7	Modulaire, mais `SurveyService` = God Object
Tests	3	∼13 %, test fantôme sur le module critique
CI/CD	4	Tests + Docker, mais pas de lint, pas de type-check, tag `latest` uniquement
Sécurité	2	Secrets committés, CORS wildcard, PII non chiffrées
Dette technique	4	19 K LOC de duplication dans `survey/`, dépendances abandonnées
Moyenne	**4,5**	

TABLE 3.2 – Notation `monka-api`

Verdict. L'architecture tient : le framework est moderne, la modularisation NestJS fonctionne, et 46 000 LOC de logique métier tournent en production. La dette est concentrée dans le module `survey/` (44 % du code) et dans les pratiques de sécurité. Recommandation : brownfield avec assainissement ciblé du module `survey` et rotation immédiate des secrets.

3.3 Application Mobile (`mobileAppRN`)

3.3.1 Stack technique

— **Framework** : React Native 0.81 + Expo SDK 54 + React 19 + TypeScript 5.9
— **Navigation** : React Navigation v6 (stack + bottom tabs)
— **État serveur** : React Query v3 (obsolète, package renommé `@tanstack/react-query`)
— **UI** : Gluestack UI + NativeWind/Tailwind CSS
— **Build** : EAS Build avec 4 profils (development, preview, production, iOS simulator)

3.3.2 Forces

— **Stack récente** : Expo 54 est la dernière version stable. React Native 0.81 a quelques versions mineures de retard (0.84 disponible), mais reste dans le cycle de support actif.
— **Architecture modulaire** : 20 modules métier dans `modules/` avec un hook React Query par domaine, séparation correcte API/hooks/types.
— **Fonctionnalités opérationnelles** : Auth0, questionnaire multi-blocs, recommandations, messagerie, abonnements Stripe, notifications push, analytics Matomo, monitoring Sentry.
— **Deep linking configuré** avec intégration push notifications.

3.3.3 Faiblesses

— **Sécurité critique** : secrets Auth0 de production exposés en clair dans `auth0-configuration.js` et `app.json`. L'API Management Auth0 est appelée directement depuis le mobile (signup, delete user, update password) — toute personne décompilant l'APK peut créer et supprimer des comptes.
— **Tests quasi inexistants** : 2 fichiers de test pour 311 fichiers source. Jest est configuré mais le `react-test-renderer` installé (v18) est incompatible avec React 19.
— **Fichier monstre** : `usePostalCode.tsx` contient 34 320 lignes de codes postaux embarqués en dur (+377 Ko de bundle parsé au démarrage).
— **`App.tsx` monolithique** : 1 048 lignes qui regroupent providers, 58 écrans, navigation conditionnelle — toute modification de routing est risquée.
— **136 usages de `any`**, 156 `eslint-disable`, 72 `console.log` en production.
— **`eval()` dans le questionnaire** (`SurveyScreen.tsx` L44) : exécution de code arbitraire pour évaluer les conditions de visibilité SurveyJS.
— **React Query v3 obsolète** : 2 versions majeures de retard, package renommé.
— **Accessibilité désactivée** : `allowFontScaling={false}` utilisé massivement — anti-pattern majeur pour une app destinée aux aidants.

3.3.4 Notation détaillée

Critère	Note / 10	Commentaire
Stack technique	7	RN 0.81, Expo 54 à jour ; React Query v3 obsolète
Architecture	5	Modules bien séparés, navigation monolithique
Tests	1	2 fichiers, architecture non testable
CI/CD	2	Lint + typecheck uniquement, zéro CD
Sécurité	1	Secrets production dans le repo, Management API depuis mobile
Dette technique	3	34 K lignes dans un fichier, 120 `any`, `eval()`
Moyenne	**3,2**	

TABLE 3.3 – Notation `mobileAppRN`

Verdict. Les failles de sécurité sont critiques (secrets exposés, Management API côté client) et la dette technique est profonde. La stack sous-jacente est moderne (Expo 54), mais 80 % des écrans doivent être refaits pour le nouveau parcours UX, et le nettoyage brownfield coûte autant qu'un greenfield. Recommandation : greenfield avec récupération des modules métier existants (API calls, types, interfaces — ∼30 % du code réutilisable).

3.4 CRM LifeLine (`monka-backoffice`)

3.4.1 Stack technique

— **Framework** : React 18 + Vite 4 + TypeScript (strict) + React Router v6

— **État** : Zustand 4 (auth, breadcrumbs) + React Query v3 (server state)
— **UI** : Tailwind CSS 3 + composants internes
— **Lazy loading** : 29 pages chargées à la demande

3.4.2 Forces

— **Architecture la plus propre des 4 projets** : séparation claire types/API/composants/pages/stores, ∼24 500 LOC bien structurées.
— **Système de rôles extensible** : `ProtectedRoute` + composant `<Roles>` supportent N rôles par design — ajouter le rôle Care Manager est trivial.
— **Lazy loading systématique** : toutes les pages passent par `React.lazy`, code splitting effectif.
— **State management adapté** : Zustand pour l'auth, React Query pour le server state, pas de Redux superflu.
— **Auth bien faite** : cookie-based avec refresh automatique, singleton anti-concurrence sur le refresh, DOMPurify systématique contre le XSS.

3.4.3 Faiblesses

— **0 test unitaire**, 4 tests Cypress dont 2 cassés (routes et sélecteurs inexistants).
— **CI lint-only** : pas de build check, pas de type-check, pas de tests en pipeline.
— **112 usages de `any`** dans 36 fichiers, 72 messages d'erreur identiques `"Login Error"` qui rendent le debugging impossible.
— **React Query v3** (2 majeures de retard), Vite 4 (3 majeures de retard), `@typescript-eslint` v5 (3 majeures de retard).
— **Dépendance morte TinyMCE** : le package `@tinymce/tinymce-react` est installé mais jamais importé dans le code source. Seul Quill (∼150 Ko) est utilisé pour le texte riche.
— **Image de 48 Mo** (`src/assets/img.png`) committée dans le repo — alourdit chaque clone.
— **Dockerfile naïf** : `vite preview` utilisé comme serveur de production (non prévu pour cet usage), pas de multi-stage build.

3.4.4 Notation détaillée

Critère	Note / 10	Commentaire
Stack technique	6	React 18 + Vite OK, mais React Query v3 et Vite 4 en retard
Architecture	7	La plus propre, extensible, lazy loading
Tests	1	0 test unitaire, 2/4 tests Cypress cassés
CI/CD	3	Lint seul, pas de build ni type-check en CI
Sécurité	5	Auth cookie correcte, DOMPurify ; secrets `.env` committés
Dette technique	5	112 `any`, TinyMCE mort, mais codebase contenue (24 K LOC)
Moyenne	**4,5**	

TABLE 3.4 – Notation `monka-backoffice`

Verdict. Le CRM est le composant le plus sain de l'écosystème. La stack tient, l'architecture est extensible, et les évolutions Klésia (rôle Care Manager, pool non-attribués, bascules de parcours) s'intègrent dans la structure existante sans refactoring majeur. Recommandation : brownfield avec nettoyage de la dette (`any`, messages d'erreur, montées de versions) et extension pour le CDG Klésia.

3.5 Frontend Web (`monka-frontend`)

3.5.1 Stack technique

— **Framework** : React 18 + Vite 5 / CRA (hybride) + TypeScript 5.3
— **Questionnaire** : SurveyJS (versions incompatibles : `survey-react` v1.9 + `survey-core` v2.5)
— **Auth** : Auth0 React SDK 2.x avec tokens en `localStorage`
— **Paiement** : Stripe — SDK serveur (14.9) installé côté frontend en plus du SDK client
— **État** : React Query v3 + React Context

3.5.2 Forces

— **SurveyJS intégré avec sauvegarde progressive** : chaque réponse est persistée en temps réel via `onValueChanged`, reprise automatique à la bonne page — fonctionnalité critique pour un questionnaire de 150 questions.
— **27 fichiers de tests** avec utilitaires centralisés (`renderWithQueryClient`, `APINock`) — meilleure couverture que les autres frontends.
— **Composants de gamification embryonnaires** : pages d'accueil, introduction et transitions entre blocs déjà codées.

3.5.3 Faiblesses

— **4 systèmes de style concurrents** : styled-components (87 fichiers), Ant Design (14 fichiers), Material-UI v4 (4 fichiers) et inline styles (75 occurrences). Sept systèmes de style

au total en comptant les CSS bruts et les CSS-in-JS des librairies.
— **2 build systems** : CRA (`yarn start`, `yarn test`) et Vite (`yarn serve` en Docker). Migration à moitié faite — le comportement dev/prod diverge.
— **Versions SurveyJS incompatibles** : `survey-react` v1.9 et `survey-core` v2.5 mélangent deux générations d'API. Ca fonctionne par chance, mais ce n'est pas pérenne.
— **axios 0.21.1** : CVEs connues (SSRF, regex DoS), version EOL depuis 2 ans.
— **Storybook 6** installé avec 8 stories non maintenues (2 majeures de retard).
— **Dépendances mortes** : `react-hook-form` (installé, jamais importé), `stripe` SDK serveur, `@nestjs/testing` dans un frontend React.
— **Dockerfile dev en production** : `vite preview` n'est pas un serveur de production. Base Debian 11 (EOL).

3.5.4 Notation détaillée

Critère	Note / 10	Commentaire
Stack technique	4	2 build systems, 4 libs UI, axios 0.21 avec CVEs
Architecture	5	Modules bien séparés, mais routage CRA/Vite hybride
Tests	3	27 fichiers (8 % couverture), rien sur SurveyJS ni Stripe
CI/CD	5	Tests + lint + typecheck en CI, mais cache cassé
Sécurité	4	Auth0 OK, mais tokens en localStorage, `.env` committés
Dette technique	3	7 systèmes de style, dépendances mortes, versions incompatibles
Moyenne	**4,0**	

TABLE 3.5 – Notation `monka-frontend`

Verdict. Le frontend web est dans un état hybride instable : migration Vite inachevée, versions SurveyJS incompatibles, 4 librairies UI concurrentes. Stabiliser le build, retirer Material-UI et CRA, migrer SurveyJS v1 vers v2, monter React Query v5 — l'effort de nettoyage brownfield consomme autant de temps qu'un greenfield, sans la base propre qui va avec. Recommandation : greenfield avec Vite 7, Tailwind, SurveyJS v2 et récupération de la logique de sauvegarde progressive.

3.6 Constats transverses

Quatre problèmes systémiques traversent l'ensemble de l'écosystème.

3.6.1 Sécurité : secrets committés dans 5+ repositories

Des secrets de production sont versionnés dans les repositories suivants :

Repository	Secrets exposés
`monka-api`	Clé Stripe LIVE, JWT_SECRET, clés de chiffrement, Mailjet, Calendly
`mobileAppRN`	Secrets Auth0 production (client ID + API secret), config Firebase, DSN Sentry
`monka-frontend`	Clés Stripe test, Auth0 staging, identifiants divers
`monka-backoffice`	Secret d'extraction, clé Cypress Cloud
`monka-admin`	Injection SQL identifiée (routes/patients.ts)

TABLE 3.6 – Secrets de production exposés dans les repositories

Toute personne ayant accès aux repositories (y compris l'historique Git) dispose des clés de production. La rotation immédiate de tous les secrets est la priorité numéro un, indépendamment de toute décision greenfield/brownfield.

3.6.2 Tests : couverture quasi nulle

Repository	Fichiers test	Fichiers source	Couverture estimée
`monka-api`	43	324	∼15 %
`mobileAppRN`	2	311	∼0,6 %
`monka-backoffice`	0	157	0 %
`monka-frontend`	27	332	∼8 %
Total	72	1 124	∼6 %

TABLE 3.7 – Couverture de tests par repository

Le module le plus critique de la plateforme — le scoring des questionnaires dans `monka-api` — a un unique test qui vérifie `expect(true).toBeTruthy()`. Les tests Stripe du backend sont désactivés (`describe.skip`). Deux des quatre tests Cypress du backoffice pointent vers des routes inexistantes. En l'état, toute modification de code est un pari sans filet.

3.6.3 CI/CD : pipelines basiques, pas de quality gates

Repository	Lint	Typecheck	Tests	Build check	Security scan
`monka-api`	—	—	Oui	—	—
`mobileAppRN`	Oui	Oui	—	—	—
`monka-backoffice`	Oui	—	—	—	—
`monka-frontend`	Oui	Oui	Oui	—	—

TABLE 3.8 – Checks CI par repository (— = absent)

Aucun pipeline n'inclut de vérification de sécurité (audit de dépendances, scan de secrets). Aucun ne vérifie que le build aboutit avant un déploiement. Tous les déploiements Docker utilisent le tag `latest` : pas de rollback possible.

3.6.4 Infrastructure : pas d'IaC, pas de versioning

— **Pas d'Infrastructure as Code** : la configuration des conteneurs est gérée manuellement via Portainer, sans versionnement. Si l'accès Portainer est perdu, la configuration de production est perdue.
— **Images Docker non versionnées** : tous les workflows poussent uniquement le tag `latest` — pas de SHA, pas de tag sémantique, pas de rollback possible.
— **PostgreSQL 10.6 en fin de vie** : EOL depuis novembre 2022, soit plus de 3 ans sans correctifs de sécurité. Upgrade vers PostgreSQL 17+ nécessaire.
— **Dockerfiles naïfs** : pas de multi-stage build, pas de `.dockerignore`, pas de `USER` non-root, `vite preview` utilisé comme serveur de production sur deux frontends.

3.6.5 Synthèse des notations

Repository	Stack	Archi	Tests	CI/CD	Sécu	Dette	Moy.
`monka-api`	7	7	3	4	2	4	**4,5**
`mobileAppRN`	7	5	1	2	1	3	**3,2**
`monka-backoffice`	6	7	1	3	5	5	**4,5**
`monka-frontend`	4	5	3	5	4	3	**4,0**

TABLE 3.9 – Synthèse des notations par critère (/10)

Les stacks sous-jacentes sont globalement modernes (NestJS 11, React Native 0.81, Expo 54), mais les pratiques de sécurité, de test et de déploiement sont très en deçà des standards attendus pour une plateforme de santé. Le chapitre suivant détaille la stratégie technique qui en découle.

Chapitre 4

Recommandations stratégiques

4.1 Approche hybride recommandée

L'audit des 14 repositories montre une réalité contrastée : certains composants sont des bases solides, d'autres sont un passif technique qu'il coûte plus cher de remettre à niveau que de reconstruire. On ne recommande ni un greenfield intégral ni un brownfield intégral, mais une approche hybride : garder ce qui marche, jeter ce qui freine.

Le tableau 4.1 résume la décision pour chaque composant applicatif.

Table 4.1 – Stratégie recommandée par composant

Composant	**Stratégie**	**Justification synthétique**
Backend (monka-api)	Brownfield	80 % du code sain, 44 500 LOC de logique métier
CRM LifeLine (monka-backoffice)	Brownfield	Repo le plus propre, stack qu'on choisirait en greenfield
App mobile (mobileAppRN)	Greenfield	Sécurité critique, UX à refondre intégralement
Frontend web (monka-frontend)	Greenfield	4 systèmes de style, build hybride, SurveyJS cassé

4.1.1 Backend (monka-api) – garder et assainir

Le backend porte la logique métier de la plateforme. On le garde, pour quatre raisons factuelles :

1. **Volume de logique métier irremplaçable.** 46 000 lignes de code fonctionnel encodent quatre ans de règles métier : scoring par vulnérabilité, pondérations des questionnaires, logique de recommandation, intégrations Stripe/Apple/Google Play, notifications push. Reconstruire cette logique à partir de zéro est irréaliste.

2. **Architecture NestJS correcte.** 54 modules, 65 contrôleurs, 57 services et 37 entités TypeORM suivent les conventions du framework. Le pattern contrôleur/service/entité est respecté. Les DTOs utilisent `class-validator` avec 92 classes et 942 décorateurs de validation.
3. **Stack à jour.** NestJS 11, TypeScript 5.9, Node 22 LTS. Pas d'obsolescence de framework. Les montées de version restantes sont des évolutions, pas des refontes.
4. **Schéma de données mature.** 250 migrations sur sept ans d'historique. Le modèle relationnel est complet, les entités et leurs relations correctement implémentées. Ce capital est directement réutilisable.

La dette est concentrée et identifiée : le module `survey/` représente 44 % du code source (19 581 LOC) avec de la duplication massive et un *God Object* de 2 228 lignes non testé. Cette dette est isolable : on peut ajouter les nouvelles fonctionnalités (parcours D/M/H, rôle Care Manager, moteur de règles enrichi) sans toucher au code existant du module survey, et le refactoring peut être planifié dans un second temps.

4.1.2 CRM LifeLine (monka-backoffice) – garder et étendre

Le CRM est le repository le plus propre de l'écosystème. On le garde :

1. **Stack alignée avec nos choix greenfield.** React 18, Vite, TypeScript strict, Zustand, React Query, Tailwind CSS : c'est la stack qu'on choisirait pour un nouveau projet. Les montées de version (React Query v3 vers v5, Vite 4 vers 7) sont incrémentales, pas structurelles.
2. **Architecture extensible pour le rôle Care Manager.** Le système de rôles repose sur des chaînes de caractères injectées dans un `ProtectedRoute` et un composant `<Roles>`. Ajouter le rôle CM est trivial : une ligne dans l'enum, des ajustements dans la Sidebar et le routeur. Pas de changement d'architecture.
3. **24 500 LOC fonctionnelles en production.** Dashboard, fiches aidants et proches, messagerie, agenda, alertes, export PDF/Excel : tout est opérationnel. Un greenfield demanderait un effort disproportionné pour atteindre la parité fonctionnelle.

La dette du CRM est bornée : 112 usages de `any`, 72 messages d'erreur génériques (`throw Error("Login Error")`), deux éditeurs de texte riche concurrents et quelques dépendances inutilisées. Ces irritants peuvent être nettoyés en deux à trois jours.

4.1.3 Application mobile (mobileAppRN) – repartir de zéro

L'application mobile est le composant le plus compromis de l'écosystème. On la reconstruit :

1. **Failles de sécurité structurelles.** Les secrets Auth0 de production sont en clair dans le repository (`auth0-configuration.js`, `app.json`). L'API Management Auth0 est appelée directement depuis le client mobile : quiconque décompile l'application peut créer, supprimer ou modifier des comptes utilisateurs. La fonction `eval()` est utilisée dans

le moteur de questionnaire. Corriger ces failles impose de réécrire intégralement le flux d'authentification, le signup et la gestion des mots de passe.

2. **UX à refondre entièrement.** Le client a qualifié l'application de « pas bonne » et le parcours est incompris par les utilisateurs. Le questionnaire doit passer de cinq blocs basiques à 150 questions gamifiées. Si 80 % des écrans doivent être refaits, garder l'architecture existante n'apporte rien.
3. **Effort brownfield équivalent au greenfield.** L'effort brownfield et greenfield sont comparables. Le brownfield consomme son avantage en migrations (React Query v3 vers v5, React Navigation v6 vers v7), en nettoyage (`App.tsx` de 1 048 lignes, `usePostalCode.tsx` de 34 320 lignes) et en corrections de sécurité. Pour un effort comparable, le greenfield produit un résultat propre, sécurisé et testable.
4. **Code récupérable à 30–40 %.** Les interfaces TypeScript (26 fichiers), les appels API (`modules/*/xxx.api.ts`), les types métier et les assets sont directement réutilisables. On ne repart pas d'une feuille blanche : on repart d'une feuille propre.

4.1.4 Frontend web (monka-frontend) – repartir de zéro

Le frontend web a des problèmes structurels qui rendent le brownfield plus coûteux que le greenfield :

1. **Quatre systèmes de style concurrents.** Styled-components (87 fichiers), Ant Design (14 fichiers), Material-UI v4 (4 fichiers) et styles inline (75 occurrences). Cette cohabitation produit un bundle surdimensionné et rend toute évolution visuelle imprévisible.
2. **SurveyJS dans un état incohérent.** Le renderer (`survey-react`) est en version 1.9, le core (`survey-core`) en version 2.5. Ces versions ne sont pas compatibles. La migration vers SurveyJS v2 est obligatoire dans tous les cas.
3. **Build hybride CRA/Vite.** Le développement local et les tests passent par Create React App (abandonné), la production par Vite. Deux systèmes de build en parallèle, deux configurations à maintenir, des comportements dev/prod divergents.
4. **Effort comparable.** L'effort brownfield et greenfield sont comparables. Le brownfield n'est pas plus rapide parce que la dette technique absorbe tout le gain du code existant.

4.2 Sécurité : actions immédiates

L'audit a identifié des vulnérabilités à corriger **avant** tout développement fonctionnel. Ces actions sont les mêmes quelle que soit la stratégie greenfield ou brownfield : c'est de la mise en conformité, pas de l'évolution produit.

4.2.1 Rotation de tous les secrets

Des clés de production sont commitées en clair dans l'historique Git de cinq repositories :

— **monka-api** : clé Stripe LIVE, Mailjet, Calendly, `JWT_SECRET`, clés de chiffrement (`ENCRYPTION_KEY` et `ENCRYPTION_IV` identiques entre staging et production).

— **mobileAppRN** : `apiClientId` et `apiSecret` Auth0 de production, clé Firebase, DSN Sentry.
— **monka-frontend** : clés Stripe test/staging, credentials Auth0.
— **monka-backoffice** : secret d'extraction Excel.
— **monka-admin** : URI PostgreSQL de production avec mot de passe.

Tous ces secrets doivent être régénérés, migrés vers des gestionnaires de secrets (GitHub Secrets, EAS Secrets) et retirés des fichiers `.env.*` versionnés.

4.2.2 Retirer les fichiers `.env` des repositories

Le `.gitignore` de monka-api ne protège que `.env` sans suffixe. Les fichiers `.env.staging`, `.env.production`, `.env.prod`, `.env.rc` et `.env.docker.*` sont tous versionnés. Il faut les supprimer du suivi Git (`git rm -cached`), les ajouter au `.gitignore` et les remplacer par des fichiers `.env.example` sans valeurs sensibles.

4.2.3 Corriger l'injection SQL dans monka-admin

Le fichier `routes/patients.ts` (lignes 111–122) de monka-admin concatène des entrées utilisateur directement dans une requête SQL. À corriger immédiatement, indépendamment de la décision d'archiver ou non le repository.

4.2.4 Migrer l'API Management Auth0 côté serveur

L'application mobile appelle directement l'API Management Auth0 depuis le client (`signup`, `deleteUser`, `updatePassword`) avec un token `client_credentials` obtenu depuis les secrets embarqués dans le bundle. Ces opérations doivent passer côté backend, seul habilité à détenir les credentials de l'API Management.

4.2.5 Configurer CORS correctement

Le fichier `.env.production` de monka-api contient `ALLOWED_HOST=*`. Même si le mécanisme d'`includes()` ne produit pas un wildcard classique, la permissivité excessive des requêtes sans header `Origin` pose un risque. Les domaines autorisés doivent être listés explicitement.

4.3 Stack technique recommandée

4.3.1 Composants greenfield

Application mobile

TABLE 4.2 – Stack technique recommandée – application mobile

Couche	Choix et justification
Framework	**Expo SDK 54 + Expo Router v6.** Routing fichier, deep linking automatique, typage natif. Plus maintenable que React Navigation configurable.
State serveur	**@tanstack/react-query v5.** Standard de l'industrie, devtools intégrés, mutations, cache intelligent. Remplace react-query v3 obsolète.
State client	**Zustand.** Léger, sans boilerplate, adapté aux rares cas de state client pur (préférences, UI). Évite Redux (présent mais jamais utilisé dans l'existant).
UI Kit	**Gluestack UI + NativeWind 4.** Déjà maîtrisé par l'équipe existante, composants accessibles, styling Tailwind cohérent.
Formulaires	**react-hook-form + zod.** Pour le questionnaire 150 questions et les formulaires de profil. Validation type-safe.
Animations	**react-native-reanimated v4.** Nécessaire pour la gamification du questionnaire (transitions, progression visuelle).

Frontend web

TABLE 4.3 – Stack technique recommandée – frontend web

Couche	Choix et justification
Bundler	**Vite 7 + React 19.** SPA pure derrière Auth0, pas de SSR nécessaire. Élimine le build hybride CRA/Vite actuel.
Styling	**Tailwind CSS 4 + shadcn/ui.** Un seul système de style. shadcn/ui fournit des composants accessibles et personnalisables sans vendor lock-in. Élimine Material-UI, Ant Design et styled-components.
Questionnaire	**SurveyJS v2** (`survey-core` + `survey-react-ui`). Migration obligatoire depuis les versions incompatibles actuelles. Theme builder, CSS variables, custom renderers.
State	**TanStack Query v5 + Zustand.** Même architecture que le mobile pour la cohérence de l'équipe.
Dates	**dayjs** uniquement. Élimine `moment` (300 Ko) et `date-fns` (duplication avec dayjs).
Tests	**Vitest + Testing Library.** Compatible Vite nativement, même API que Jest.

4.3.2 Composants brownfield

Backend (monka-api)

Les interventions sur le backend relèvent de l'assainissement ciblé, pas de la refonte :

— **PostgreSQL 10.6 vers 17.** La version actuelle est en fin de vie depuis novembre 2022, soit plus de trois ans sans correctifs de sécurité. La CI teste sur PostgreSQL 16+ (image `latest`) tandis que la production tourne sur 10.6 : une migration qui utilise une fonctionnalité post-10 passerait la CI et casserait la production. L'upgrade est impérative. PostgreSQL 17 est la cible recommandée (stable et éprouvé ; PostgreSQL 18 est disponible mais trop récent pour une migration critique).

— **Éclater le *God Object* `SurveyService`.** Ce service de 2 228 lignes mélange scoring, génération PDF, export Excel, CRUD, diagnostics anonymes et adaptateurs mobile. Il faut le découper en cinq à six services spécialisés. Ce refactoring peut être planifié après les échéances.

— **Ajouter la validation stricte.** Le `ValidationPipe` global est correctement configuré, mais certains contrôleurs acceptent des `@Body()` sans DTO typé (notamment les routes anonymes du questionnaire). Les endpoints exposés sans guard doivent être audités.

— **Remplacer les dépendances abandonnées.** `cls-hooked` (abandonné depuis 2019, APIs `async_hooks` dépréciées) par `AsyncLocalStorage` natif. `@nestjsx/crud` (archivé) par une configuration NestJS standard. `jwt-decode` (déprécié) par `jose`.

CRM LifeLine (monka-backoffice)

— **React Query v3 vers @tanstack/react-query v5.** Migration de l'API (signatures de `useQuery`, imports, gestion d'erreurs). Trois jours estimés.
— **Nettoyer les 112 usages de any.** Concentrés dans `form.ts` (8), `transmissionReport.tsx` (14) et les modules API. Le typage strict détectera les régressions lors des extensions CDG.
— **Ajouter le rôle Care Manager.** Une ligne dans l'enum `AccountRole`, des ajustements dans `main.tsx` (routes), `Sidebar.tsx` (navigation), `RootRedirect` (redirection par défaut) et les composants `<Roles>` concernés.
— **Retirer la dépendance morte TinyMCE.** Le package `@tinymce/tinymce-react` (500 Ko) est installé mais jamais importé dans le code source. Seul Quill est utilisé. Supprimer TinyMCE des dépendances.

4.4 Méthodologie de développement

4.4.1 CI/CD renforcée dès le départ

Chaque repository actif aura un pipeline complet, sans exception :

— **Lint** (ESLint strict, pas de `eslint-disable` non justifié)
— **Type-check** (`tsc -noEmit`, zéro erreur tolérée)
— **Tests unitaires et d'intégration** (seuil de couverture minimum sur les nouveaux modules)
— **Build** (vérification que l'application compile avant tout merge)
— **Scan de sécurité** (`npm audit`, analyse des secrets, scan des dépendances)

Aujourd'hui, aucun repository n'a ces checks au complet : monka-api n'exécute ni le lint ni le type-check en CI, monka-backoffice ne vérifie pas le build, mobileAppRN n'exécute pas les tests. C'est une des premières actions à mener.

4.4.2 Revue automatique de code

En complément des revues manuelles, un outil de revue automatique de pull requests tel que **Greptile** sera intégré dans les pipelines CI. Chaque PR déclenchera une analyse automatique qui vérifie la cohérence architecturale, détecte les anti-patterns, signale les régressions potentielles et produit un résumé des changements. L'objectif : accélérer le cycle de revue et garantir un niveau de qualité constant, y compris sur le code généré par IA.

4.4.3 Livraisons et communication

— **Livraisons incrémentales** : le développement produit des incréments déployables en staging. Les fonctionnalités sont livrées en *feature flags* si nécessaire.
— **Communication quotidienne** : point de synchronisation avec l'équipe Monka. Les blocages sont remontés dans la journée.