Cahier des charges du Projet « Axora » : Spécifications Techniques et Fonctionnelles pour l'Architecture Sidecar Officinale

1. Synthèse Exécutive et Cadrage Stratégique

1.1. Introduction : La Nécessité d'une Rupture Technologique au Comptoir

L'écosystème numérique de la pharmacie d'officine française traverse une période de mutation technologique profonde et sans précédent. Ce secteur, historiquement structuré autour de logiciels de gestion (LGO) monolithiques, se trouve aujourd'hui à la croisée des chemins, tiraillé entre la stabilité nécessaire des processus transactionnels réglementaires (télétransmission SESAM-Vitale, gestion des stocks) et **l'impératif urgent d'agilité imposé par les nouvelles missions du pharmacien** (Bilan de Prévention, Vaccination, Tests de dépistages, TROD Angine, TROD Cystite, Entretiens Bilan Partagé de Médication, Entretiens Asthme, Entretiens AVK et AOD, etc...)

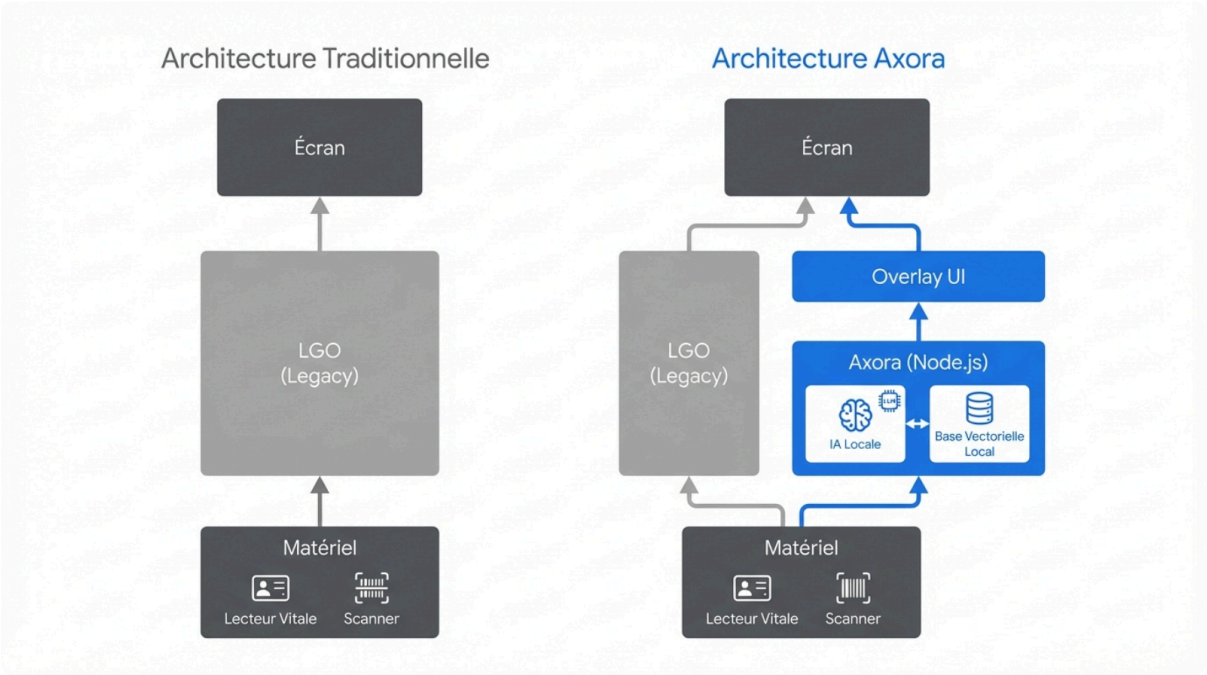
Les architectures fermées des éditeurs historiques, développées sur plusieurs décennies, accusent une dette technique qui freine considérablement l'innovation rapide, notamment en matière d'expérience utilisateur (UX), d'interopérabilité fluide avec les périphériques modernes, et d'accès instantané à des données pharmacologiques enrichies.

Le projet **Axora** naît de ce constat critique : l'innovation au comptoir ne peut plus attendre les cycles de refonte quinquennaux des éditeurs historiques ni se satisfaire de mises à jour cosmétiques. Axora propose une approche radicalement différente basée sur **le paradigme architectural du "Sidecar"**. Plutôt que de tenter une refonte intégrale et risquée du LGO existant —une entreprise industrielle titanesque soumise à des certifications lourdes— Axora se positionne comme une **couche d'intelligence satellite**.

Ce module compagnon, totalement autonome, **fonctionne en parallèle du logiciel principal sans le modifier ni perturber son cœur transactionnel**. En exploitant les capacités multitâches des systèmes d'exploitation modernes (Windows 10/11) et la puissance de l'écosystème open source, Axora **intercepte les flux périphériques** (cartes à puce, scanners) et **contextuels** pour offrir au pharmacien un **"cockpit augmenté"**.

L'objectif est de redonner au pharmacien la maîtrise de son outil de travail, transformant le poste de délivrance en un hub clinique intelligent, sécurisé et souverain.

Architecture Comparative : Monolithe vs Modèle Sidecar Axora



Le modèle Sidecar d'Axora (à droite) intercepte les événements matériels (Carte Vitale, Scanner) directement via l'OS, traitant l'information en parallèle du LGO (Legacy) pour projeter une interface augmentée (Overlay) au pharmacien.

1.2. Checklist Méthodologique de Structuration

Pour garantir la rigueur et l'exhaustivité de ce document de référence, nous suivrons scrupuleusement les étapes suivantes, validées avant toute rédaction technique approfondie :

1. **Définition du Socle Architectural Transversal :** Établir des fondations logicielles robustes (basées sur le couple Electron et Node.js), définir des stratégies de sécurité strictes pour protéger le système (cloisonnement via l'Isolation de Contexte, gestion sécurisée des Modules Natifs) et industrialiser le déploiement (automatisation CI/CD, certification par Code Signing) afin de garantir la stabilité opérationnelle de l'application

2. **Spécification Détaillée des Interfaces Matérielles (HAL) :** Détailler les protocoles de communication de bas niveau essentiels au dialogue avec les lecteurs de cartes (via le standard **PC/SC** et les commandes **APDU**) et à l'interception des scanners de codes-barres (profil clavier **HID**, utilisation de **Global Hooks**), en apportant une solution technique robuste aux contraintes de **concurrence d'accès** ("device sharing") imposées par Windows.

3. **Conception des Modules d'Intelligence Artificielle (Stratégie Hybride & Prototypage) :** Adopter une approche de développement pragmatique visant à accélérer le "Time-to-Market" en découplant la logique métier de l'infrastructure d'exécution.
    - **Priorité à l'Intégration API (Cloud)** : Pour la phase de développement et de validation fonctionnelle, l'intelligence du système reposera sur les API managées de Mistral AI. Cela permet de déployer immédiatement des capacités avancées (complétion de chat, agents conversationnels, et analyse multimodale pour l'OCR) sans subir les contraintes techniques de l'inférence locale.
    - **Abstraction Architecturale :** Concevoir une couche d'abstraction logicielle (Interface/Adapter Pattern) qui isole les appels IA. Concrètement, le logiciel appellera une fonction générique `askAI()`, qui sera connectée aux API Mistral dans un premier temps.

- **Transition vers le "Local-First"** : Cette structure modulaire permettra, dans un second temps et de manière transparente, de "débrancher" l'API Cloud pour la remplacer par des modèles locaux quantifiés (BioMistral) lors du passage en production, garantissant in fine la souveraineté des données sans retarder le développement initial.

4. **Définition Définition de l'Expérience Utilisateur (UX) : L'Approche "Dual-Mode" (Volet Compagnon & Hub Immersif) :** Afin de garantir une accessibilité optimale et de ne jamais surcharger l'espace visuel du pharmacien, l'interface abandonne la superposition transparente au profit d'une stratégie d'affichage à deux niveaux, distincte et contextuelle :
    - **Le Mode "Compagnon" (L'Onglet Discret) :** En utilisation courante (lorsque le pharmacien travaille dans son LGO), Axora se matérialise sous la forme d'un volet latéral rétractable (Side-Drawer) ou d'une pastille flottante ancrée sur le bord de l'écran ("Docked Widget").
        - **Zéro Distraction :** Par défaut, cet onglet est replié (taille minimale : ~50px de large) pour ne masquer aucune information critique du logiciel métier.
        - **Interaction à la Demande :** Au survol ou au clic, le volet se déplie ("Slide-out") pour afficher les notifications contextuelles rapides (ex: "Carte Vitale lue", "Alerte péremption"). Cela crée un espace visuel propre, avec un fond opaque garantissant un contraste de lecture parfait, sans conflit avec les couleurs du LGO en arrière-plan.
    - **Le Mode "Hub/Dashboard Axora" (L'Espace de Travail Immersif) :** Lorsque l'utilisateur sollicite explicitement l'application (via un raccourci clavier ou un clic sur l'onglet pour accéder aux modules complexes comme l'IA Clinique), Axora bascule en mode "Focus".
        - **Dashboard Central :** L'interface s'étend alors pour occuper une partie significative ou la totalité de l'écran (overlay modal), offrant une ergonomie riche pour la visualisation de tableaux de bord, la comparaison de prix ou la lecture de réponses IA détaillées.
        - **Gestion du Focus :** Ce mode capture temporairement les entrées clavier/souris pour permettre une interaction fluide, avant de redonner la main au LGO dès la fermeture.

2. Architecture Technique Transversale

2.1. Socle Applicatif : La Puissance d'Electron et Node.js

Le choix du framework **Electron** s'impose comme le standard industriel incontournable pour le développement d'applications de bureau cross-platform intégrant des technologies web modernes. Pour Axora, Electron ne se limite pas à un rôle de conteneur web ; il agit comme le véritable chef d'orchestre de processus natifs complexes, comblant le fossé entre le web et le matériel.

2.1.1. Séparation Stricte des Processus

L'architecture d'Electron repose sur une distinction fondamentale entre le **Processus Principal (Main Process)** et les **Processus de Rendu (Renderer Processes)**. Cette séparation est vitale pour la stabilité et la sécurité d'Axora.

- **Le Main Process (Orchestrateur) :** Ce processus unique, exécuté dans un environnement Node.js complet, est responsable de la gestion du cycle de vie de l'application, de la création des fenêtres et, surtout, de l'orchestration des modules natifs. C'est au sein du Main Process que résideront les instances critiques de node-pcsclite pour la communication avec la carte Vitale et les hooks globaux du clavier pour l'interception des scanners. Ce processus doit être impérativement découplé du rendu graphique pour éviter tout gel de l'interface (UI freeze) lors de calculs intensifs ou d'attentes d'E/S matérielles.

- **Le Renderer Process (Interface) :** Basé sur Chromium, ce processus est chargé de l'affichage de l'interface utilisateur "Overlay". Pour Axora, nous utiliserons **React** comme bibliothèque de vue, bénéficiant de son écosystème riche et de sa gestion performante du DOM virtuel. La communication avec le Main Process se fera exclusivement via le mécanisme **Context Isolation** et les canaux **IPC (Inter-Process Communication)** sécurisés (ipcRenderer.invoke / ipcMain.handle). Cette architecture prévient les vulnérabilités de type RCE (Remote Code Execution) en interdisant l'accès direct aux API Node.js depuis le contexte de la page web.

2.1.2. Gestion de la Performance et de l'Empreinte Mémoire

Les postes de travail en officine sont souvent des machines aux ressources contraintes, exécutant déjà un LGO gourmand, des navigateurs web et divers utilitaires de sécurité. Axora doit donc se montrer exemplaire en matière de consommation de ressources.

- **Cible Mémoire :** L'application en veille ("Idle state") doit viser une empreinte mémoire vive (RAM) raisonnable.

- **Stratégie pour l'IA :** L'activation des modules d'IA générative (comme BioMistral) représente un pic de charge significatif. Pour ne pas déstabiliser le processus principal, l'inférence IA sera déléguée à un **processus enfant détaché (spawned child process)** ou un worker thread dédié. Cette isolation permet non seulement de protéger le Main Process d'un crash potentiel du moteur d'IA, mais facilite également un nettoyage agressif de la mémoire (kill process) une fois la tâche terminée, libérant instantanément les 4 à 6 Go de RAM utilisés par le modèle.

2.2. Gestion des Modules Natifs et Compilation

La nature "Sidecar" d'Axora impose une interaction profonde avec le système d'exploitation, nécessitant l'usage de modules natifs (C++ Addons via Node-API) pour PC/SC, les hooks clavier globaux et la base de données vectorielle SQLite. Cette dépendance introduit une complexité significative dans la chaîne de compilation

2.2.1. La Problématique de l'ABI Electron

Electron utilise une version de l'ABI (Application Binary Interface) V8 différente de celle du Node.js standard installé sur le système. Les modules natifs compilés pour Node.js ne fonctionneront pas directement dans Electron et provoqueront des erreurs de type NODE_MODULE_VERSION mismatch.

- **Solution :** L'intégration de l'outil **@electron/rebuild** (successeur d'electron-rebuild) est obligatoire dans les scripts de post-installation (npm run rebuild). Cet outil télécharge les en-têtes C++ spécifiques à la version d'Electron cible et recompile les

sources des modules natifs (node-pcsclite, sqlite-vec, node-global-key-listener) pour garantir une compatibilité binaire parfaite

2.2.2. Ciblage Architectural

Standardisation x64 et AVX2 : Compte tenu de l'objectif final d'exécuter des modèles d'IA souverains (BioMistral), Axora ciblera nativement l'architecture **x64** (et potentiellement **ARM64** pour les futurs PC Windows Copilot+). L'utilisation de jeux d'instructions CPU modernes (AVX2) est pré-requise pour garantir une inférence locale fluide. Par conséquent, le support pour les architectures 32-bits (x86), incompatibles avec les librairies d'IA performantes et techniquement obsolètes, est officiellement abandonné.

Flexibilité via API Mistral (Mode Hybride) : Toutefois, pour ne pas lier la performance de l'application à la seule puissance brute du poste client, l'architecture intègre une **capacité de délestage vers le Cloud**. Le système est conçu pour basculer de manière transparente vers les **API de Mistral AI** (mode connecté). Cette solution permet d'assurer une qualité de service constante même sur des postes x64 moins puissants en phase de déploiement, ou d'accéder à des modèles plus larges (Mistral Large) pour des tâches complexes, offrant ainsi une résilience architecturale face à l'hétérogénéité du parc matériel officinal.

3. Module 1 : Lecture Contextuelle Vitale

Ce module constitue la première interaction physique et numérique du parcours patient. Il a pour mission de détecter, identifier et lire la Carte Vitale plus rapidement que le LGO ne peut le faire, permettant un accueil personnalisé immédiat ("Bonjour M. Dupont") et une vérification proactive des droits (si techniquement possible).

3.1. Objectifs Fonctionnels et Performance

- **Réactivité Extrême :** Le système doit détecter l'insertion d'une carte et afficher les informations rapidement. Cette vitesse est cruciale pour créer une perception de fluidité supérieure à celle du logiciel métier historique.

- **Lecture "Zone Claire" :** L'objectif est d'extraire les données administratives publiques (Nom, Prénom, NIR, Rang gémellaire, Droits AMO) accessibles sans code PIN porteur, conformément aux spécifications SESAM-Vitale.

- **Notification Non-Intrusive :** Les informations doivent être présentées via un "Toast" discret en haut de l'écran, avec un code couleur intuitif pour le statut des droits (Vert = À jour, Orange = Bientôt expiré, Rouge = Invalide) (si techniquement réaliste).

3.2. Spécifications Techniques Détaillées

3.2.1. Stack Technologique : PC/SC sur Node.js

Pour interagir avec le matériel, nous utiliserons la bibliothèque **@pokusew/pcsclite** (ou son ancêtre node-pcsclite), qui constitue le binding standard de l'industrie pour accéder à l'API système PC/SC (winscard.dll sous Windows, pcsc-lite sous Linux/macOS).

- **Justification du Choix :** Contrairement aux wrappers Python (pyscard) ou aux appels en ligne de commande, cette librairie offre une architecture **100% événementielle** native à Node.js. Elle expose des événements comme reader.on('status',...) qui permettent de réagir aux changements d'état du lecteur sans bloquer la boucle d'événements (Event Loop) d'Electron, garantissant que l'interface reste fluide même pendant les opérations d'E/S.

3.2.2. Gestion de la Concurrence d'Accès (Le Défi Windows)

Le défi technique majeur réside dans la **gestion de l'accès concurrent au lecteur de carte**. Sous Windows, le gestionnaire de ressources de cartes à puce (Smart Card Resource Manager) arbitre l'accès. Si le LGO est mal conçu et verrouille le lecteur en mode exclusif (SCARD_SHARE_EXCLUSIVE) dès qu'il est lancé, Axora ne pourra pas y accéder.

- **Stratégie de Connexion "Shared" :** Axora doit systématiquement initier la connexion au lecteur en mode partagé : SCARD_SHARE_SHARED. Cela signale au système d'exploitation que d'autres applications peuvent également avoir besoin d'accéder à la carte.

- **Transaction Atomique Rapide :** Pour minimiser les conflits, la lecture doit être une opération "éclair" et atomique. Le workflow est le suivant :
  1. Détection de l'événement SCARD_STATE_PRESENT.
  2. SCardConnect (en mode Shared).
  3. SCardBeginTransaction (Verrouillage temporaire du lecteur pour empêcher le LGO d'interrompre la séquence APDU).
  4. Envoi en rafale (burst) des commandes APDU (Select + Read).
  5. SCardEndTransaction (Libération du verrou).
  6. SCardDisconnect (Déconnexion immédiate pour laisser la main au LGO).

- **Gestion des Erreurs et Backoff :** Si le LGO détient déjà un verrou exclusif, la tentative de connexion d'Axora échouera. Le module doit gérer cet échec silencieusement ou tenter une unique ré-essai avec un délai très court (backoff de 100ms) avant d'abandonner pour ne pas geler le matériel.

3.2.3. Séquencement APDU et Parsing des Données

La communication avec la puce de la Carte Vitale suit la norme ISO 7816-4 et utilise des commandes **APDU** (Application Protocol Data Unit). Le module "Accueil Éclair" devra implémenter la séquence précise suivante :

1. **Lecture de l'ATR (Answer To Reset) :**
    - Dès la mise sous tension de la carte, le lecteur reçoit l'ATR.
    - Axora doit comparer cet ATR avec une base de connaissances embarquée (issue du fichier smartcard_list.txt de pcsc-tools).[1]
    - *Règle d'Identification :* Si l'ATR commence par la séquence 3B 75 13..., il s'agit très probablement d'une Carte Vitale. Si la séquence est 3B AC..., c'est une Carte de Professionnel de Santé (CPS). Cette étape de filtrage permet d'ignorer les cartes non pertinentes (bancaires, fidélité) avant même de tenter une lecture.

2. **Sélection de l'Application (SELECT FILE) :**
    - Commande APDU : 00 A4 04 00 (Instruction Select File par nom DF) + Longueur de l'AID + AID Vitale (Identifiant d'Application). L'AID spécifique de l'application Vitale (souvent A0 00 00 01 18 45 43) doit être confirmé via les spécifications techniques du GIE SESAM-Vitale ou par rétro-ingénierie sur des cartes de test.

3. **Lecture des Données (READ BINARY) :**
    - Une fois l'application sélectionnée, il faut naviguer vers les Fichiers Élémentaires (EF) contenant les données d'identité.
    - Commande APDU : 00 B0 (Read Binary) + Offset P1/P2 + Longueur Le.

4. **Décodage TLV (Tag-Length-Value) :**
    - Les données renvoyées par la carte sont des flux d'octets bruts structurés en format BER-TLV.
    - L'utilisation d'une librairie de parsing robuste (comme ber-tlv ou une implémentation interne optimisée) est nécessaire pour décoder ces flux.
    - *Mapping des Tags (à valider) :* Le parser doit extraire les valeurs associées aux

tags spécifiques, par exemple Tag 80 pour le Nom, Tag 81 pour le Prénom, et Tag 82 pour le NIR (Numéro d'Inscription au Répertoire).

3.3. Algorithme de Flux (User Flow)

L'interaction se décompose en un flux logique précis :

1. **État Veille :** Le service PC/SC d'Axora écoute passivement les événements matériels en arrière-plan.
2. **Événement Matériel :** Le système détecte un changement d'état (status change) signalant qu'une carte est présente (card inserted).
3. **Filtrage Contextuel :** Analyse immédiate de l'ATR. Si la carte n'est pas reconnue comme une Vitale, le processus s'arrête là.
4. **Extraction Atomique :** Tentative de connexion partagée suivie de la séquence APDU rapide.
5. **Parsing et Normalisation :** Décodage des buffers hexadécimaux en chaînes de caractères UTF-8 lisibles.
6. **Action UI (Overlay) :** Envoi des données JSON via IPC au processus de rendu pour l'affichage de la notification "Bonjour M. DUPONT".
7. **Libération :** Déconnexion immédiate du lecteur pour permettre au LGO d'effectuer ses propres opérations de lecture sécurisée pour la facturation.

4. Module 2 : "Filet de Sécurité FMD" - Sérialisation Intelligente et Traçabilité

La directive européenne sur les médicaments falsifiés (FMD) impose la vérification systématique de l'authenticité des boîtes via la lecture d'un code Datamatrix 2D. En pratique, les scanners de codes-barres agissent comme des périphériques d'entrée clavier (HID - Human Interface Device). Cette méthode, bien que universelle, pose un problème majeur : l'interprétation erronée des caractères spéciaux (notamment le séparateur FNC1) par Windows ou le LGO, conduisant à des erreurs de saisie et à l'absence de contrôle de péremption *avant* la validation de la délivrance.

4.1. Objectifs Fonctionnels : Zéro Erreur, Zéro Délai

- **Interception Globale :** Le module doit être capable de "capturer" le flux de données provenant du scanner, quelle que soit l'application qui possède le focus (même si le pharmacien navigue dans le LGO ou une autre fenêtre).
- **Sanitisation des Données :** Détecter et corriger à la volée les erreurs de transmission du caractère séparateur FNC1, souvent transformé par les pilotes de scanner ou l'OS en caractère GS (ASCII 29), en séquence Ctrl+], ou purement et simplement ignoré.[1]
- **Parsing GS1 Robuste :** Découper la chaîne brute pour extraire les quatre informations critiques : GTIN (Code Produit), Numéro de Lot, Date d'Expiration, et Numéro de Série (SN) unique.
- **Logique Métier Locale (Business Logic) :** Effectuer des contrôles immédiats : comparer la date d'expiration extraite avec la date du jour (Date.now()) et vérifier le numéro de lot contre une base de données locale de rappels de lots.
- **Blocage Visuel Préventif :** En cas d'anomalie (produit périmé ou rappelé), afficher instantanément une alerte rouge bloquante par-dessus le LGO, *avant* que le pharmacien ne puisse valider la délivrance, agissant comme un véritable "airbag" de sécurité.

4.2. Spécifications Techniques Détaillées

4.2.1. Capture Clavier Bas Niveau (Global Hook)

Les écouteurs d'événements clavier standard du navigateur (DOM keydown, keypress) sont inopérants lorsque la fenêtre d'Axora n'a pas le focus. Il est impératif d'utiliser un hook système au niveau de l'OS pour intercepter tous les événements clavier.

- **Choix de la Bibliothèque :** Nous évaluerons **node-global-key-listener** et **iohook** (ou son fork maintenu @tkomde/iohook).
    - *Analyse Comparative :* iohook utilise des hooks natifs C++ puissants mais

nécessite une compilation complexe et peut parfois être instable lors des mises à jour d'Electron.[18] node-global-key-listener est une solution plus légère, souvent préférée récemment pour sa simplicité d'intégration et sa moindre propension à déclencher les faux positifs des antivirus, un risque connu avec les technologies de type keylogger.[19] Pour Axora, nous privilégierons node-global-key-listener dans un premier temps, avec une possibilité de bascule vers une solution uiohook-napi si des besoins plus bas niveau émergent.

- **Algorithme de Distinction Humain vs Machine :**
  - Les scanners envoient les caractères à une vitesse "surhumaine", typiquement avec un intervalle de 10 à 20 ms entre chaque frappe.
  - *Logique de Détection :* Le module implémentera un buffer glissant. Si une séquence de plus de 10 caractères est reçue avec un intervalle moyen inférieur à 50 ms, elle est identifiée comme provenant d'un scanner. Si la frappe est plus lente et irrégulière, elle est considérée comme une saisie manuelle humaine et doit être ignorée pour respecter la confidentialité des saisies du pharmacien.

4.2.2. Parsing GS1 et Gestion Critique du FNC1

Le standard GS1 pour les codes Datamatrix utilise le caractère de fonction 1 (FNC1) comme séparateur pour marquer la fin des champs à longueur variable (comme le numéro de lot ou le numéro de série).

- **Le Problème FNC1 :** En mode émulation clavier, le FNC1 n'a pas de représentation standard. Il est souvent transmis sous forme de caractère non imprimable (ASCII 29 GS) ou via des séquences d'échappement.
- **Librairie de Parsing :** L'utilisation de la bibliothèque **interpretGS1scan** (projet maintenu par GS1 ou un équivalent communautaire fiable) est impérative.[21] Cette librairie contient la logique complexe nécessaire pour interpréter correctement les Identifiants d'Application (AI) comme 01 (GTIN), 17 (Expiration), 10 (Lot) et 21 (Série), et gérer les séparateurs manquants ou altérés.
- **Structure de Données Normalisée :** Le parser doit retourner un objet JSON standardisé pour le traitement :

```
JSON
{
  "gtin": "034009xxx",
  "batch": "H5522",
  "expiry": "251231", // Format YYMMDD
  "serial": "SN98765",
  "raw": "...chaine brute..."
}
```

4.2.3. Moteur de Règles (Business Rules Engine)

Une fois les données parsées et structurées, un moteur de règles léger, exécuté directement dans le Main Process pour une latence minimale, évalue la conformité du produit.

- **Règle de Péremption :** Le moteur parse la date d'expiration (format YYMMDD).
    - Si Date(Expiry) < Date(Today), une alerte **ROUGE** (Critique) est déclenchée.
    - Si Date(Expiry) < Date(Today + 3 mois), une alerte **ORANGE** (Avertissement : Péremption proche) est levée.
- **Règle de Rappel de Lot :** Au démarrage (et via mise à jour silencieuse quotidienne), Axora charge une liste noire locale (fichier JSON chiffré) contenant les identifiants des lots rappelés, issue des données de l'ANSM. Le moteur vérifie si le couple GTIN + BatchID scanné est présent dans cette liste de rappel.

4.3. Interface Overlay "Alerte Bloquante"

La réaction de l'interface utilisateur doit être immédiate et impossible à ignorer en cas de danger.

1. Si une anomalie critique est détectée, le Main Process demande au Renderer d'activer une fenêtre dédiée.
2. Cette fenêtre est configurée avec les propriétés alwaysOnTop: true, fullscreen: true, et un fond rouge semi-transparent. Elle affiche en caractères géants le motif du blocage ("PRODUIT PÉRIMÉ" ou "LOT RAPPELÉ").
3. **Focus Stealing :** La fenêtre prend immédiatement le focus clavier/souris. Cela a pour effet secondaire bénéfique d'interrompre potentiellement la séquence de validation "Entrée" que le pharmacien pourrait avoir l'habitude de faire par réflexe dans le LGO, offrant une barrière de sécurité physique.

5. Module 3 : "Assistant Clinique RAG-Pharma" - IA Locale

Ce module représente le cœur cognitif d'Axora. Il vise à introduire l'intelligence artificielle générative directement au comptoir pour assister le pharmacien face aux questions complexes ("Quelle est la posologie de l'amoxicilline pour un nourrisson de 6kg?", "Y a-t-il une interaction entre ce traitement et le jus de pamplemousse?"). La contrainte majeure est de fournir ces réponses sans dépendance à une connexion internet constante et en garantissant une confidentialité absolue des requêtes.

5.1. Architecture RAG (Retrieval-Augmented Generation) Locale

Pour un usage médical, l'utilisation brute d'un LLM est proscrite en raison des risques d'hallucinations. L'approche RAG (Génération Augmentée par la Récupération) est obligatoire. Le modèle ne doit pas inventer une réponse, mais synthétiser une réponse en se basant *uniquement* sur des documents médicaux de référence fiables et vérifiés (Résumés des Caractéristiques du Produit - RCP, Base Claude Bernard, Vidal, Thésaurus de l'ANSM) stockés localement.

5.1.1. Base de Connaissance Vectorielle : SQLite-vec

Pour le stockage et l'interrogation sémantique des documents médicaux, nous utiliserons **SQLite** enrichi de l'extension moderne **sqlite-vec**.[1]

- **Justification du Choix :** Contrairement aux bases de données vectorielles distantes (Pinecone, Weaviate) ou aux serveurs lourds nécessitant Docker (ChromaDB), sqlite-vec est une solution "in-process", ultra-légère et sans dépendance complexe. Elle permet de stocker les embeddings (représentations vectorielles du texte) directement dans un fichier .db local classique. C'est la solution idéale pour une installation sur un poste officinal standard, minimisant l'empreinte système et la latence.
- **Pipeline d'Ingestion des Données :** Un script d'ingestion (exécuté lors des mises à jour mensuelles de la base de données) transforme les documents sources (PDF/XML des RCP) en texte brut. Ce texte est ensuite découpé en "chunks" (segments ou paragraphes) avec un recouvrement (overlap) pour préserver le contexte. Chaque chunk est converti en vecteur via un petit modèle d'embedding local (ex: all-MiniLM-L6-v2 ou un modèle multilingue plus récent) et inséré dans la table virtuelle vec0 de SQLite.

5.1.2. Moteur d'Inférence LLM : BioMistral Quantifié

Le "cerveau" qui formule la réponse est un Grand Modèle de Langage (LLM) spécialisé.

- **Modèle Sélectionné : BioMistral 7B**.[25] Ce modèle open-source est basé sur l'architecture Mistral mais a été spécifiquement fine-tuné sur la base de données médicale PubMed, lui conférant une compréhension supérieure du vocabulaire et des concepts médicaux par rapport aux modèles généralistes.
- **Quantification GGUF :** La plupart des PC de pharmacie ne disposent pas de cartes

graphiques (GPU) puissantes avec beaucoup de VRAM. Pour tourner de manière fluide sur des CPU grand public, le modèle doit être **quantifié**. Nous utiliserons le format **GGUF** avec une quantification **4-bit (Q4_K_M)**.[27] Ce niveau de compression réduit la consommation de RAM nécessaire à environ 4 à 6 Go, ce qui reste acceptable pour des machines modernes équipées de 8 à 16 Go de RAM, tout en préservant l'essentiel des capacités de raisonnement du modèle.

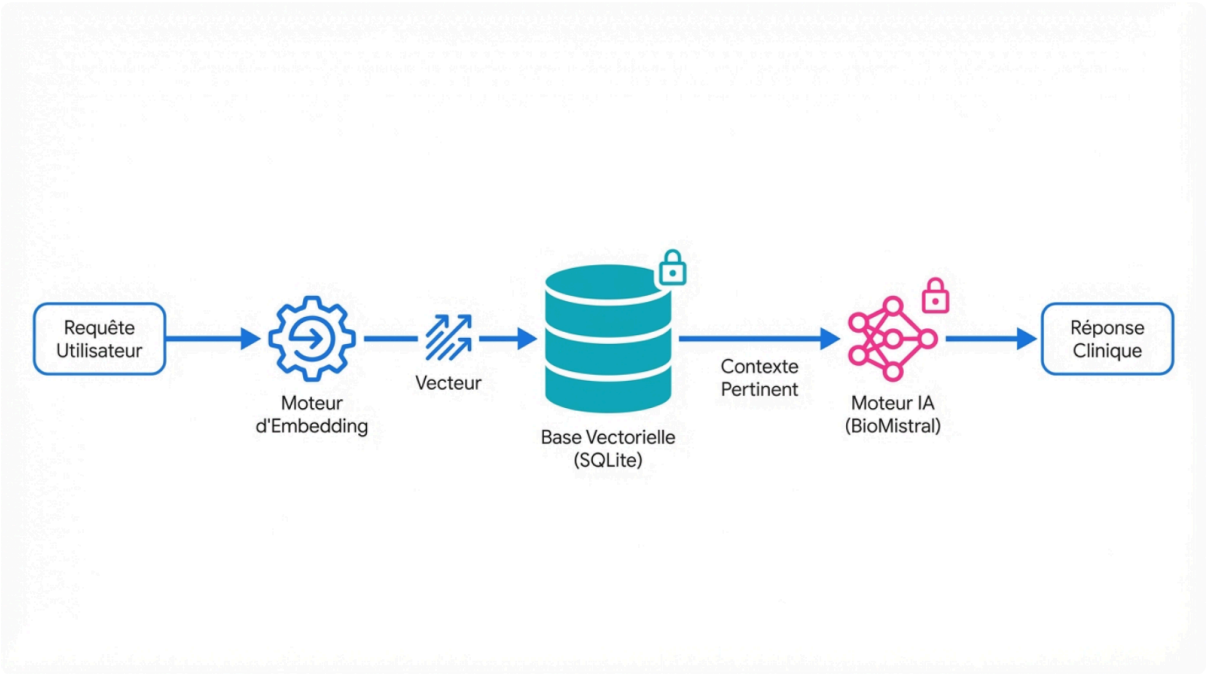
- **Moteur d'Exécution :** L'intégration se fera via **node-llama-cpp**, un binding Node.js performant pour le projet llama.cpp, permettant de charger et d'exécuter le modèle GGUF directement au sein de l'application Electron.[1] Une alternative architecturale consistant à lancer un binaire ollama ou llama-server en tant que processus sidecar local et à communiquer via une API REST locale (localhost) sera évaluée pour sa stabilité (si le modèle crash, il ne fait pas tomber toute l'application).

5.2. Workflow de la Requête Clinique

Le processus de réponse à une question clinique suit un cheminement de données rigoureux :

1. **Déclenchement (Trigger) :** Le pharmacien active l'assistant via un raccourci global (ex: Ctrl+Espace). Une barre de recherche flottante type "Spotlight" apparaît au-dessus du LGO.
2. **Requête (Query) :** Le pharmacien saisit sa question en langage naturel, par exemple : *"Quels sont les effets secondaires du doliprane sur le foie?"*.
3. **Vectorisation (Embedding) :** La question est immédiatement convertie en vecteur numérique par le modèle d'embedding local.
4. **Recherche Vectorielle (Retrieval) :** Ce vecteur est utilisé pour interroger la base sqlite-vec. Le système identifie et récupère les 3 à 5 "chunks" de texte de la base RCP qui sont les plus proches sémantiquement de la question.
5. **Génération (Generation) :** Un "Prompt" est construit dynamiquement, intégrant la question du pharmacien et les chunks récupérés comme contexte. L'instruction donnée au LLM est stricte : *"En utilisant uniquement le contexte médical fourni ci-dessous [Chunk 1, Chunk 2...], réponds à la question suivante :..."*. Ce prompt est envoyé au moteur BioMistral.
6. **Restitution :** La réponse générée est affichée (streamée token par token pour une perception de vitesse accrue) dans l'interface overlay. Chaque affirmation est accompagnée de citations ou de liens vers le document PDF source local, permettant au pharmacien de vérifier l'information.

Pipeline RAG-Pharma : De la Question à la Réponse Sécurisée



Le processus se déroule entièrement hors ligne. La requête est vectorisée, comparée à la base de connaissance (SQLite-vec), et le contexte pertinent est fourni à BioMistral (GGUF) pour générer une réponse clinique sourcée.

---

6. Module 4 : " PhiVision" - Intelligence Visuelle & Aide à la Décision

Ce module remplace la simple automatisation de saisie par un véritable **cockpit clinique augmenté**. Il ne s'agit plus seulement de lire pour écrire, mais de **lire pour comprendre, sécuriser et conseiller**. Ce module analyse en temps réel les éléments affichés à l'écran (ordonnances scannées, panier de produits dans le LGO) pour projeter instantanément des informations critiques et des opportunités de conseil.

6.1. Philosophie : De la Saisie à l'Augmentation

L'objectif n'est plus le mimétisme clavier (RPA), mais l'enrichissement cognitif du pharmacien via trois axes de valeur :

- **Sécurité Clinique :** Détection immédiate des contre-indications et interactions (ex: "Alerte Hépatique" sur du Paracétamol).
- **Conseil Patient Standardisé :** Fourniture de "scripts" de conseils oraux clairs pour homogénéiser la qualité de service de l'équipe.
- **Développement Éthique :** Suggestion contextuelle de produits complémentaires pertinents (ex: Probiotiques avec antibiotiques) pour améliorer le soin global.

6.2. Architecture Technique "API-First" (Mistral & Vision)

Contrairement à l'approche OCR locale complexe (Tesseract/TrOCR) mais qui, pourra quand même rester envisageable dans un second temps, dans le développement., “PhiVision” s'appuie sur la puissance des modèles multimodaux via API.

- **Capture Contextuelle (Le "Lasso") :** Le pharmacien cible une zone (l'ordonnance) ou l'application détecte automatiquement le contenu du LGO via capture d'écran ciblée (`desktopCapturer` Electron).
- **Analyse Multimodale (Mistral Pixtral / GPT-4o) :** L'image est envoyée sécurisée vers l'API d'IA. Le modèle ne fait pas que de l'OCR ; il "comprend" l'image. Il extrait les médicaments, analyse la posologie, et croise ces données avec le profil patient (âge, sexe) s'il est visible.
- **Pipeline de Règles (Rules Engine) :** Les données extraites sont passées dans un moteur de règles hybride (IA + Règles métiers fixes `rules.yml`) pour générer un **Payload Vision** structuré JSON.

6.3. Interface Overlay : Les Cartes Intelligentes

L'affichage repose sur le système de "Cartes" de PhiGenix, intégré dans le volet latéral d'Axora. Le rendu est dynamique et réactif.

Structure des Cartes Affichées :

1. **Carte "Conseil Immédiat" :**
    - **Conseil Oral (Verbatim) :** Une phrase simple à dire au patient (ex: *"Attention, respectez bien 4h entre chaque prise"*).
    - **Points Clés :** Liste à puces des vigilances (Conduite, Soleil, Alcool).
2. **Carte "Opportunités & Soin Global" :**
    - **Produits Complémentaires :** Liste des produits associés logiques (Cross-selling).
    - **Argumentaire ("Pourquoi ?") :** Justification clinique courte pour aider le pharmacien à proposer le produit (ex: *"Pour prévenir la déshydratation liée à la diarrhée"*).
3. **Badges & Alertes de Sécurité :**
    - Tags visuels colorés (Vert/Orange/Rouge) pour qualifier l'ordonnance : `[Ordonnance 2 lignes]`, `[⚠️ Alerte hépatique]`, `[Nouveau Patient]`.

6.4. Flux de Données (Workflow)

Le processus suit le schéma réactif éprouvé de l'architecture PhiVision :

1. **Trigger :** Capture d'image (Manuelle ou Automatique).
2. **Processing (Cloud) :** Envoi API -> Analyse IA -> Structuration JSON (DCI, Posologie, Alertes).

Dispatch (Local) : Le renderer reçoit le payload JSON standardisé :
JSON

```
{
  "advice_oral": "Conseil verbal...",
  "produits_cross": ["Probiotiques"],
  "alertes": [{"type": "warn", "label": "Interaction Alcool"}]
}
```

3.
4. **Rendu (UI) :** Mise à jour instantanée du volet latéral Axora avec les nouvelles cartes, sans bloquer la saisie dans le LGO.

7. UX/UI et Intégration : L'Architecture "Dual-Mode" Adaptative

L'acceptabilité d'Axora par les pharmaciens ne repose pas seulement sur sa performance technique, mais sur son respect strict de l'ergonomie de travail. Pour garantir une accessibilité visuelle optimale (ratio de contraste, lisibilité) sans perturber le flux transactionnel du LGO, l'interface abandonne la superposition transparente globale au profit d'une stratégie d'affichage à deux états : le **Compagnon Discret** et le **Hub Immersif**.

7.1. Structure d'Interface : Le Concept du Volet Latéral

Plutôt qu'une fenêtre "fantôme" recouvrant l'écran (source de confusion et de problèmes de clics), Axora structure l'écran en réservant un espace dédié et maîtrisé.

- **Mode 1 : Le "Compagnon" (L'Onglet Latéral)** En veille, Axora se matérialise sous la forme d'un **onglet rétractable (Side-Drawer)** ou d'une pastille ancrée sur le bord de l'écran.
    - **Discrétion Absolue :** Cet élément occupe une surface minimale (ex: 50px de large) et possède un fond opaque garantissant un contraste parfait, résolvant les problèmes d'accessibilité liés à la transparence sur des interfaces LGO chargées.
    - **Interaction "À la demande" :** Au survol ou au clic, le volet se déplie latéralement pour afficher les notifications rapides (Lecture Vitale, Alerte FMD). Cette zone est techniquement une fenêtre `alwaysOnTop`, mais configurée pour ne jamais masquer le cœur de l'application métier tant que l'utilisateur ne le sollicite pas.
- **Mode 2 : Le "Hub Axora" (Espace de Travail)** Lorsque le pharmacien active une fonction complexe (analyse approfondie d'ordonnance, chat IA complet), l'application bascule en mode **"Focus"**.
    - **Immersion :** L'interface s'étend pour occuper l'espace central (Overlay Modal opaque), offrant une ergonomie riche pour les tableaux de bord et les réponses détaillées.
    - **Isolation :** Ce mode capture temporairement les entrées clavier/souris, permettant une interaction fluide et sécurisée avant de redonner instantanément la main au LGO à la fermeture.

7.2. Intelligence Contextuelle (Context Awareness)

Bien que l'affichage soit moins intrusif, la capacité d'écoute d'Axora reste active en arrière-plan pour "pousser" la bonne information dans l'onglet au bon moment.

- **Monitoring Non-Intrusif :** Le module continue d'utiliser des bibliothèques légères comme `monitor-active-window` (ou des appels API Windows via FFI) pour analyser le titre de la fenêtre active.

- **Scénario d'Usage Révisé :** Si le titre de la fenêtre du LGO passe de "Accueil" à "Dossier Patient : DUPONT Jean", Axora détecte le changement. Au lieu d'afficher un popup bloquant, l'**onglet compagnon** s'anime discrètement (badge de notification ou légère pulsation) pour signaler que des informations contextuelles (droits, rappels, analyse prédictive) sont prêtes à être consultées. Le pharmacien garde la maîtrise totale du moment où il souhaite consommer cette information.