Rapport : Architectures Ouvertes et Interopérabilité pour l'Officine 4.0 : Analyse Technique des Ressources Open Source

Introduction : La Nécessité d'une Rupture Technologique au Comptoir

L'écosystème pharmaceutique français traverse une période de mutation technologique sans précédent, caractérisée par une tension croissante entre des systèmes historiques monolithiques et des exigences opérationnelles de plus en plus agiles. Le "comptoir", point névralgique de l'interaction entre le professionnel de santé et le patient, est traditionnellement tributaire des Logiciels de Gestion d'Officine (LGO). Ces systèmes, bien que robustes pour les tâches administratives lourdes telles que la télétransmission SESAM-Vitale, souffrent souvent d'une architecture fermée (propriétaire) qui freine 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.

L'analyse approfondie des ressources disponibles sur GitHub révèle une opportunité architecturale majeure : l'émergence du **modèle "Sidecar"**. Plutôt que de tenter la refonte intégrale d'un LGO — tâche titanesque soumise à des certifications réglementaires strictes (agrément SESAM-Vitale, DMP compatible) — les développeurs peuvent aujourd'hui s'appuyer sur une constellation de **bibliothèques open source pour construire des modules satellites**. **Ces modules, fonctionnant en parallèle du logiciel principal, ont pour vocation d'augmenter l'expérience du pharmacien en lui offrant des capacités de lecture directe de la Carte Vitale, de décodage avancé des sérialisations (Datamatrix), et d'interrogation sémantique des bases de données médicamenteuses.**

Ce rapport technique a pour objectif de cartographier de manière exhaustive ces **ressources open source**, d'analyser leur viabilité technique pour un déploiement en production, et de

définir les architectures logicielles permettant leur intégration sécurisée au cœur du poste de travail officinal. Nous explorerons successivement les couches matérielles (lecture de cartes à puce et périphériques), les couches de données (intelligence pharmacologique et standards FHIR), et enfin les cadres applicatifs (frameworks Electron/React) permettant de fusionner ces éléments en une interface cohérente.

Partie I : La Couche Matérielle et l'Abstraction des Cartes à Puce

La pierre angulaire de l'identité numérique en santé en France est la **Carte Vitale**. **Pour qu'une application tierce puisse apporter de la valeur au comptoir, elle doit impérativement être capable de dialoguer avec ce support physique, indépendamment du LGO**. Cette indépendance repose sur la maîtrise des **protocoles PC/SC** (Personal Computer/Smart Card) et sur l'utilisation de bibliothèques capables de faire le pont entre le matériel et les interfaces web modernes.

1.1 L'Écosystème PC/SC : Fondations et Middleware

Le **standard PC/SC** est l'interface universelle permettant aux applications informatiques de communiquer avec des **lecteurs de cartes à puce**. Dans le monde open source, l'implémentation de référence est **pcsc-lite**, maintenue historiquement par Ludovic Rousseau. Cette pile logicielle est omniprésente sur les systèmes Unix (Linux, macOS) et possède ses équivalents sur **Windows via l'API winscard.dll**.

1.1.1 L'Interface Node.js : node-pcsclite

Pour le développement d'applications modernes orientées interface utilisateur, le langage JavaScript (via l'environnement d'exécution Node.js) s'est imposé. Le **projet santigimeno/node-pcsclite** constitue à ce titre une ressource critique. Il ne s'agit pas d'une réécriture du protocole, mais d'un "binding" (une liaison) vers les bibliothèques natives du système.

L'analyse du code source et de la documentation de **node-pcsclite** révèle une **architecture événementielle** particulièrement adaptée aux flux de travail en pharmacie. Contrairement aux approches procédurales classiques où le programme doit interroger le lecteur en boucle ("polling"), cette bibliothèque permet de **souscrire à des événements systèmes**. Lorsqu'un patient insère sa carte, l'événement *reader.on('status')* est déclenché quasi instantanément, permettant à l'application de réagir (par exemple, en ouvrant automatiquement le dossier patient ou en affichant les droits à jour) sans intervention manuelle du pharmacien.

Cependant, l'utilisation de **node-pcsclite** impose des prérequis techniques spécifiques selon le système d'exploitation hôte. Sur des distributions Linux couramment utilisées dans les serveurs d'officine (Debian/Ubuntu), l'installation des paquets de développement libpcsclite-dev est nécessaire pour la compilation des modules natifs lors de l'installation du paquet npm. Cette dépendance aux binaires systèmes est un point de vigilance pour le déploiement massif sur des parcs hétérogènes, mais elle garantit en contrepartie une performance optimale et un accès direct aux capacités bas niveau du matériel.

1.1.2 L'Alternative Python : pyscard

En parallèle de l'écosystème Node.js, la communauté Python propose une alternative robuste avec pyscard. Ce projet, également supervisé par Ludovic Rousseau, offre une abstraction de haut niveau sur la couche PC/SC.

Bien que moins adapté au développement d'interfaces graphiques réactives que l'écosystème JavaScript, pyscard excelle dans les tâches de fond, l'analyse de données ou le prototypage rapide de commandes APDU. Des projets comme EMVConsole ou LL-Smartcard du MIT Lincoln Laboratory démontrent la capacité de pyscard à gérer des interactions complexes avec des cartes à puce, y compris pour des tâches de cryptographie ou d'authentification.

Dans une architecture officinale, pyscard pourrait être privilégié pour la conception de "services windows" ou de démons Linux tournant en arrière-plan pour logger les activités des lecteurs ou effectuer des mises à jour de cartes en lot, sans nécessiter d'interface utilisateur directe.

1.2 Protocoles d'Interaction et Commandes APDU

Disposer d'une bibliothèque d'accès au lecteur n'est que la première étape. Pour extraire des informations utiles de la Carte Vitale, l'application doit parler le **langage de la carte** : les **APDU (Application Protocol Data Unit)**, définis par la **norme ISO 7816-4**.

1.2.1 Identification du Support (ATR)

La première information transmise par une carte lors de sa mise sous tension est **l'ATR (Answer To Reset)**. C'est une **séquence d'octets** qui permet d'identifier le type de carte et ses paramètres de communication. L'analyse des dépôts comme **pcsc-tools** fournit une base de connaissance inestimable via le **fichier smartcard_list.txt**. Ce fichier contient des milliers de signatures ATR associées à leurs descriptions textuelles.

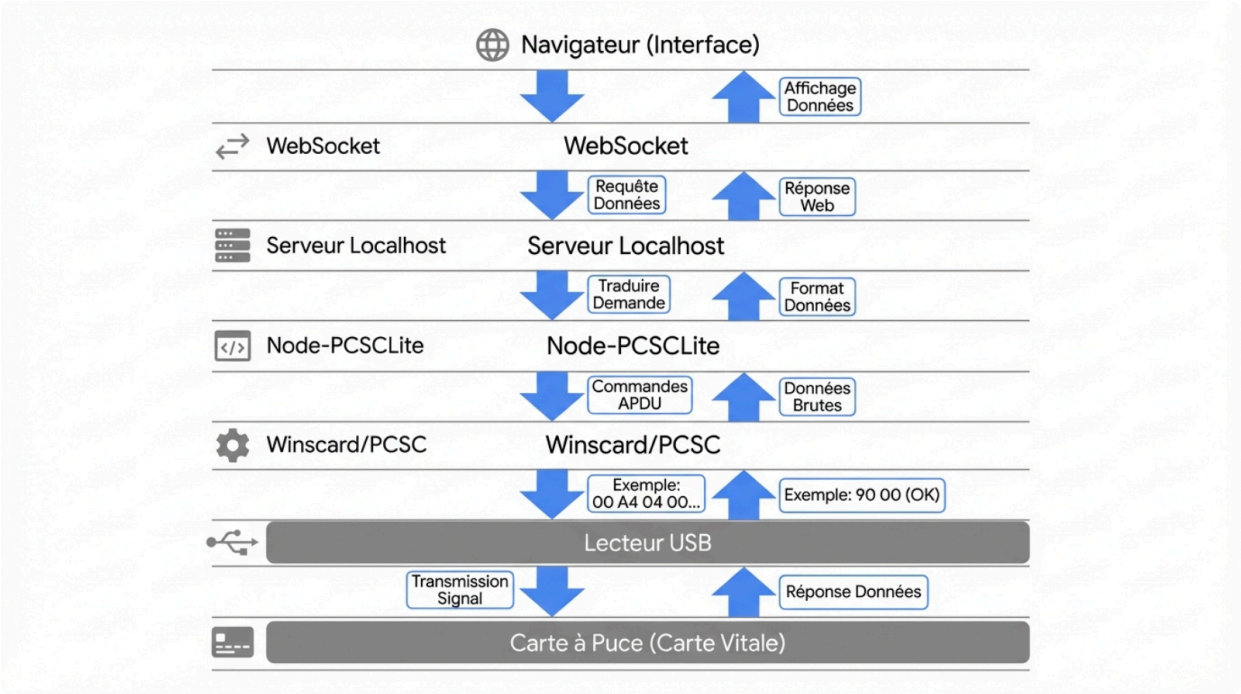
Pour une application visant à améliorer l'expérience au comptoir, l'intégration de cette base de données permet une reconnaissance contextuelle immédiate. Si l'ATR commence par des séquences spécifiques identifiées comme "Carte Vitale 2" ou "Sesam Vitale" (par exemple 3B 75 13...), l'application peut basculer en mode "Délivrance Patient". À l'inverse, si l'ATR correspond à une Carte de Professionnel de Santé (CPS), l'application peut déverrouiller des fonctionnalités d'administration ou de validation pharmaceutique. Cette **détection passive**, sans clic, est un élément clé de l'ergonomie.

1.2.2 Navigation dans le Système de Fichiers (Zone Claire)

Contrairement à une idée reçue, **la lecture des données administratives de la Carte Vitale (NIR, Nom, Prénom, Ayants-droits) ne nécessite pas systématiquement l'accès aux zones chiffrées protégées par les composants propriétaires SESAM. Ces informations résident souvent dans ce que l'on appelle la "Zone Claire".**

Les snippets de code et les discussions techniques illustrent la séquence de commandes nécessaires pour naviguer dans l'arborescence de la carte. La commande ***SELECT FILE (instruction 00 A4)*** est utilisée pour sélectionner l'application Vitale (via son AID spécifique) ou des fichiers élémentaires. Une fois le fichier ciblé, la commande ***READ BINARY (instruction 00 B0)*** permet de récupérer les données brutes.

Architecture du Pont Web-vers-Carte Vitale



Flux d'interaction : L'application web envoie une requête via WebSocket au pont local, qui traduit la demande en commandes APDU natives via node-pcsclite pour interroger la Carte Vitale.

Il est important de noter que l'interprétation sémantique de ces données binaires (décodage TLV - Type Length Value) nécessite le respect de spécifications précises. Les ressources comme **emvlab** ou les **dépôts traitant du format TLV fournissent les algorithmes nécessaires pour "parser" ces réponses**, transformant une suite d'octets incompréhensible en chaînes de caractères lisibles (Nom, Prénom).

1.3 L'Architecture "Pont Local" et l'Accès Web

L'une des **contraintes majeures du développement web moderne est l'isolation (sandboxing) des navigateurs**, qui empêche par défaut une page web d'accéder aux périphériques USB comme les lecteurs de cartes. Or, la tendance des logiciels métiers est au SaaS (Software as a Service) et aux interfaces web.

1.3.1 Le Serveur WebSocket Local

Pour contourner cette limitation tout en restant dans un paradigme web, l'architecture prédominante identifiée dans les projets open source est celle du **"Pont Local" (Local Bridge)**. Le **projet potpiejimmy/pcsc-server** en est l'exemple type.

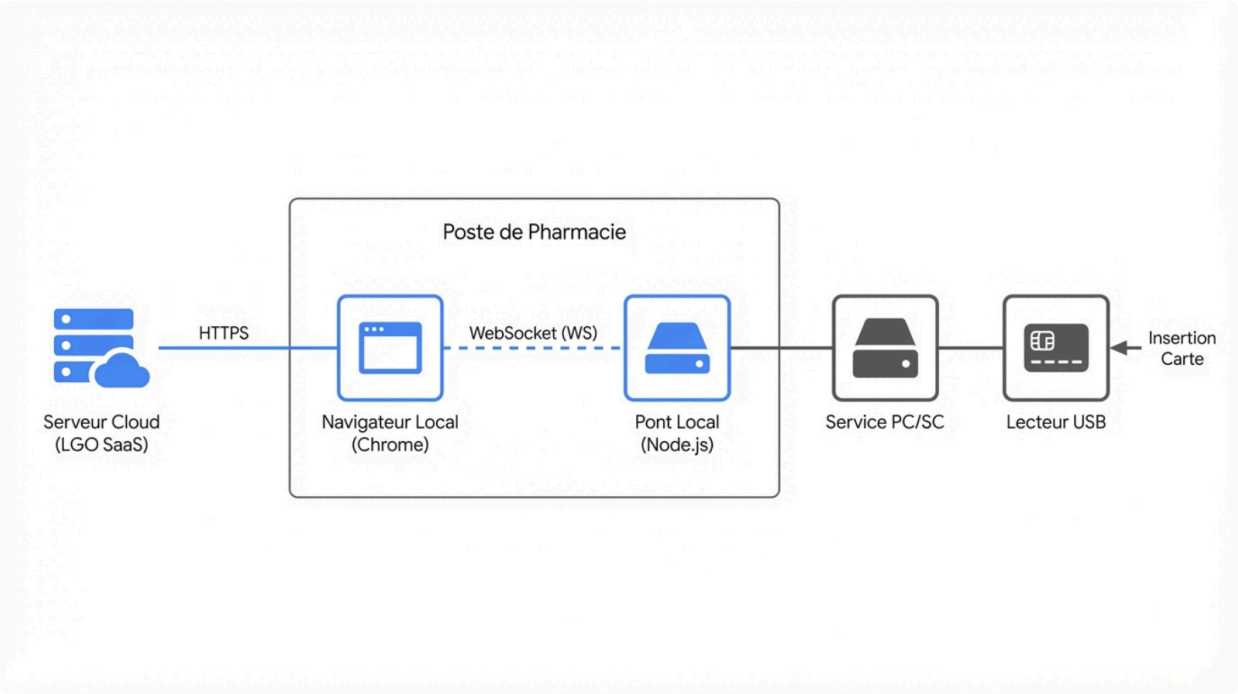
Le principe consiste à installer un petit exécutable (basé sur **Node.js** et **node-pcsclite**) sur le poste du pharmacien. Ce programme n'a pas d'interface graphique ; il lance simplement un serveur **WebSocket** écoutant sur un port local (ex: ws://localhost:8080). L'application web du pharmacien, chargée dans Chrome ou Firefox, se connecte à ce WebSocket.

Cette architecture offre une fluidité remarquable : le pont peut "pousser" (push) des informations vers le navigateur. Dès que la carte est insérée, le serveur local détecte l'événement via **node-pcsclite** et envoie immédiatement un message JSON au navigateur contenant l'ATR ou les données lues. Cela élimine le besoin de rafraîchir la page ou de cliquer sur un bouton "Lire la carte", fluidifiant considérablement l'acte au comptoir

1.3.2 Extensions de Navigateur et Native Messaging

Une alternative plus intégrée, utilisée par des acteurs de la sécurité comme Thales ou Axiad, repose sur les Extensions de Navigateur combinées à l'API Native Messaging. Bien que plus complexe à déployer (nécessitant l'installation d'une extension ET d'un hôte natif), cette méthode est souvent jugée plus sécurisée car elle permet de restreindre la communication à des domaines (origines) spécifiques, empêchant n'importe quel site web malveillant d'interroger le serveur local. Le projet webcard explore cette voie, tentant de standardiser l'accès aux cartes à puce via une extension qui expose une API JavaScript unifiée (navigator.webcard).

Architecture du Pont WebSocket pour Lecteur Vitale



Le 'Pont Local' (Node.js) agit comme un intermédiaire : il traduit les événements matériels (Insertion Carte) en messages WebSocket standards que l'application web peut comprendre.

Partie II : L'Intelligence Périphérique - Scanner et Impression

Au-delà de la carte vitale, l'efficacité au comptoir repose sur la gestion rapide des flux physiques : **boîtes de médicaments** et **documents papier**. L'open source offre des solutions puissantes pour transformer des périphériques standards en outils intelligents.

2.1 Le Défi de la Sérialisation (FMD) et le Parsing GS1

Depuis l'entrée en vigueur de la **Directive sur les Médicaments Falsifiés (FMD)**, chaque boîte de médicament délivrée doit être scannée pour vérification. Le code utilisé est un **Datamatrix 2D encodé selon le standard GS1**.

2.1.1 La Complexité Cachée du Flux de Données

Contrairement à un code-barres classique (EAN13) qui ne contient qu'une référence produit, le **Datamatrix GS1** contient quatre informations critiques : **le GTIN (Code Produit), la Date d'Expiration, le Numéro de Lot, et le Numéro de Série Unique**.

Le défi technique réside dans le format de ces données. Comme l'expliquent les spécifications GS1, certains champs comme le Numéro de Lot (AI 10) ou le Numéro de Série (AI 21) sont de longueur variable. Pour que le logiciel sache où s'arrête le numéro de lot et où commence le champ suivant, un caractère spécial non-imprimable est utilisé comme séparateur : le **FNC1** (Function Code 1), souvent transmis comme le caractère ASCII 29 (Group Separator).

La plupart des scanners de code-barres sont configurés en mode ***"émulation clavier"***. Lorsqu'ils lisent ce caractère spécial, ils peuvent soit l'ignorer, soit envoyer une séquence de touches inattendue, ce qui corrompt la chaîne de caractères reçue par l'application. Une simple expression régulière (Regex) échoue souvent à parser correctement ces chaînes si le séparateur est mal géré.

2.1.2 Solutions Open Source de Parsing

Pour fiabiliser cette lecture, l'utilisation de bibliothèques dédiées est indispensable.

Le **projet gs1/interpretGS1scan** se distingue comme une solution de référence en JavaScript.

Cette bibliothèque implémente la logique rigoureuse des Identifiants d'Application (AI). Elle est capable de détecter la présence des séparateurs FNC1 (qu'ils soient sous forme de caractères bruts ou encodés) et de découper la chaîne en un objet structuré { gtin, expiry, batch, serial }.

Une autre ressource pertinente est BarcodeParser de Peter Brockfeld, qui offre une approche légère pour le parsing côté client. En intégrant ces bibliothèques dans l'interface du pharmacien, le scan d'une boîte ne se contente plus d'ajouter une ligne à la facture. Il permet :

- De vérifier instantanément la date de péremption (comparaison de l'AI 17 avec la date du jour).
- D'automatiser la traçabilité en enregistrant le numéro de lot sans saisie manuelle.
- De détecter les erreurs de lecture ou les codes malformés avant même l'envoi au serveur de sérialisation.

Anatomie d'un Datamatrix Médicament (GS1)

Structure du Flux de Données

Identifiant (AI) Donnée Séparateur (FNC1)

FLUX BRUT (RAW STRING)

INTERPRÉTATION SÉMANTIQUE

01 FIXE	17 FIXE	10 VAR		21 VAR
GTIN (Code Article)	Date d'Exp.	Numéro de L...	GS	N° Série
03400912345678	251231	H5522		SN98765

Survolez un bloc pour voir les détails de la spécification GS1.

Décodage d'un flux GS1 : La chaîne brute contient des Identifiants d'Application (01, 17, 10) qui définissent le format des données suivantes. Le caractère 'FNC1' agit comme séparateur pour les champs à longueur variable (Lot, Série).

Data sources: GS1 Sweden, GS1 Reference, GS1 Global, GitHub (GS1 Parser)

2.2 OCR et Reconnaissance d'Ordonnances

L'augmentation de l'expérience au comptoir passe aussi par la numérisation des ordonnances papier. Bien que l'ordonnance électronique (e-prescription) progresse, le papier reste prédominant. L'**OCR** (Reconnaissance Optique de Caractères) open source offre des pistes intéressantes pour pré-remplir les délivrances.

Le moteur **Tesseract** (maintenu par Google) est la référence open source. Cependant, comme le soulignent les recherches académiques et techniques, Tesseract "brut" performe mal sur l'écriture manuscrite des médecins ("pattes de mouche"). Les projets récents tentent de pallier cela en utilisant des modèles de Deep Learning (comme CNN-LSTM) ou en "fine-tunant" Tesseract sur des corpus d'écritures médicales.

Pour une application officinale, l'approche la plus pragmatique identifiée dans les dépôts GitHub consiste à utiliser Tesseract pour extraire le texte imprimé (en-têtes, noms des patients si imprimés) et à coupler cela avec des bibliothèques de traitement d'image (OpenCV) pour nettoyer le bruit (lignes, tampons) avant l'analyse. Des projets comme Medical-Prescription-OCR proposent des pipelines complets en Python qui tentent de structurer la sortie en JSON, isolant le nom du médicament et la posologie. Intégrer un tel module (via un micro-service Python local) permettrait de proposer une fonctionnalité de "Scan Rapide" où l'ordonnance est convertie en brouillon de délivrance, réduisant le temps de saisie.

2.3 Impression Thermique WebUSB

Enfin, la délivrance se conclut souvent par l'impression d'un ticket ou d'un plan de prise. Traditionnellement, cela requiert des pilotes d'impression lourds. La norme **WebUSB** permet aujourd'hui de piloter des imprimantes thermiques (Epson, Star) directement depuis le navigateur.

Le projet WebUSBReceiptPrinter de Niels Leenheer est une ressource clé. Il fournit une interface JavaScript pour envoyer des commandes **ESC/POS** (le langage standard des imprimantes thermiques) directement via le port USB. Cela signifie qu'une application web de pharmacie peut imprimer un ticket de carte bancaire ou un QR code de traçabilité sans passer par le gestionnaire d'impression de Windows, éliminant une source fréquente de pannes et de lenteurs au comptoir.

Partie III : La Donnée Pharmaceutique Enrichie et l'Interopérabilité

Une fois le patient identifié et le produit scanné, la valeur ajoutée du logiciel réside dans la qualité de l'information médicale affichée. Les données brutes (codes CIP, codes CIS) sont insuffisantes. Le pharmacien a besoin de contexte clinique, réglementaire et économique.

3.1 Structuration de la Base de Données (BDPM vs GraphQL)

La source officielle d'information en France est la **Base de Données Publique des Médicaments (BDPM)**, gérée par l'ANSM. Bien que ces données soient "Open Data", elles sont distribuées sous forme de fichiers textes tabulés plats, difficiles à exploiter en temps réel.

3.1.1 La Révolution GraphQL avec api-bdpm-graphql

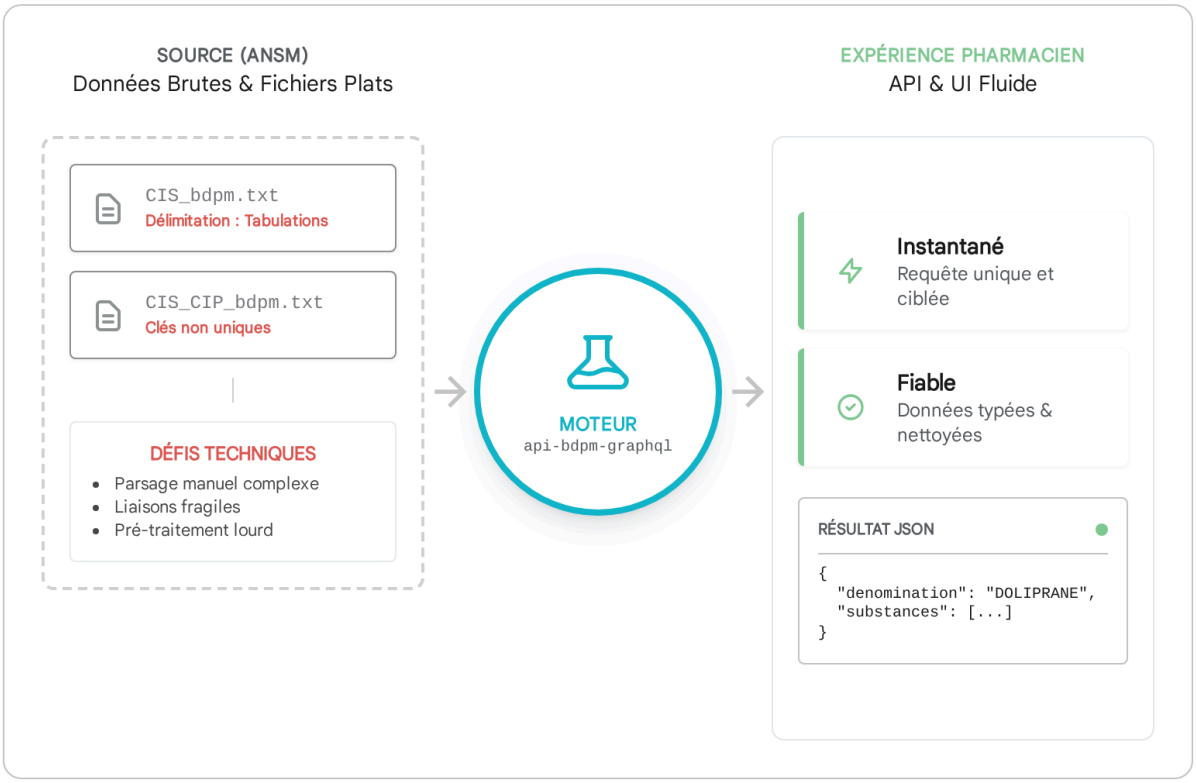
Le projet axel-op/api-bdpm-graphql représente une avancée majeure pour les développeurs. Ce projet open source ingère les fichiers bruts de l'ANSM et les expose via une API **GraphQL**. L'avantage architectural de GraphQL par rapport à une API REST classique est déterminant pour une application de comptoir, où la connexion internet peut être instable ou latente. Au lieu de faire plusieurs requêtes successives (une pour obtenir le nom du médicament, une autre pour son prix, une troisième pour ses génériques), l'application peut envoyer une requête unique et précise :

GraphQL

```
query {
  medicament(cis: "60234100") {
    denomination
    titulaire
    prix
    groupeGenerique {
      libelle
    }
  }
}
```

Cette approche réduit la bande passante et accélère l'affichage de la fiche produit ("Fiche Comptoir"). De plus, ce projet est fourni avec une configuration Docker, ce qui permet de l'héberger sur un serveur local dans l'officine (Edge Computing), garantissant un accès ultra-rapide et une indépendance vis-à-vis des pannes internet.

Transformation de la Donnée Médicament : Du Fichier Plat à l'API Graph



À gauche, la complexité traditionnelle : téléchargement et parsing de fichiers plats hétérogènes. À droite, l'approche moderne : une requête GraphQL unique cible précisément les champs nécessaires (CIS, Prix, Substances) pour l'affichage au comptoir.

Data sources: GitHub (api-bdpm-graphql), GitHub (REST API), data.gouv.fr

3.2 Sécurité Clinique : Le Parsing du Thésaurus des Interactions

La détection des interactions médicamenteuses est une obligation légale et morale. L'ANSM publie ces règles dans un document PDF complexe : le "Thésaurus des interactions médicamenteuses". La recherche manuelle dans ce document est incompatible avec le rythme du comptoir.

Le projet axel-op/parseur-thesaurus-interactions-ansm 32 apporte une réponse technique élégante. C'est une bibliothèque Java capable d'analyser la structure sémantique du PDF de l'ANSM pour en extraire un graphe d'objets informatiques.

En intégrant ce parseur, une application open source peut construire un moteur d'alerte autonome. Lorsqu'un pharmacien scanne deux boîtes, le logiciel peut vérifier localement si les substances actives appartiennent à des classes interagissant entre elles (ex: "Anti-inflammatoires" + "Anticoagulants"). L'utilisation de données sources officielles (ANSM) garantit la fiabilité des alertes, un critère critique pour tout outil de santé.

3.3 Standards d'Interopérabilité : FHIR France

Enfin, aucune application de santé moderne ne peut ignorer l'interopérabilité avec les plateformes nationales (Mon Espace Santé, DMP). Le standard international **FHIR** (Fast Healthcare Interoperability Resources) est la norme retenue par l'État français.

Les dépôts de l'association **Interop'Santé** fournissent les profils FHIR adaptés au contexte français (hl7.fhir.fr.core). Ces ressources définissent comment structurer informatiquement un patient (avec son NIR), un praticien (RPPS), ou une mesure de santé. Pour une application qui viserait, par exemple, à enregistrer des mesures de tension ou de vaccination au comptoir, l'utilisation de ces modèles open source est indispensable pour garantir que les données produites pourront être envoyées vers le DMP sans rejet. Le dépôt IG-fhir-mesures-de-sante [35] offre des exemples concrets de ressources JSON pour ces cas d'usage.

Partie IV : Architecture de Synthèse et Intégration

Comment assembler ces briques disparates (Node.js pour la carte, GraphQL pour la donnée, React pour l'interface) en un outil cohérent? L'architecture "Sidecar" (compagnon) basée sur le framework Electron apparaît comme la solution la plus pertinente.

4.1 Le Choix d'Electron et React

Le framework **Electron** permet de construire des applications de bureau (Windows/Mac/Linux) en utilisant les technologies web (HTML/CSS/JS), tout en conservant un accès aux API natives du système d'exploitation.
Le "boilerplate" (modèle de démarrage) electron-react-boilerplate [36] est une fondation solide, largement adoptée par la communauté open source. Il offre une configuration pré-établie intégrant React pour l'interface utilisateur et une chaîne de compilation optimisée.

- **Pourquoi ce choix?** Electron possède un "Processus Principal" (Main Process) qui s'exécute avec les privilèges Node.js complets. C'est ici que l'on intégrera node-pcsclite pour communiquer avec le lecteur USB sans restriction de navigateur. Le "Processus de Rendu" (Renderer Process), quant à lui, affiche l'interface React et communique avec le Main Process via un mécanisme sécurisé (IPC - Inter-Process Communication).

4.2 Le Modèle "Sidecar" (Compagnon)

L'idée n'est pas de remplacer le LGO existant (ce qui impliquerait de gérer la facturation, la télétransmission, les stocks, etc.), mais de faire tourner cette application Electron en parallèle, comme un compagnon intelligent.
Scénario d'Architecture Unifiée :

1. **Détection Passive :** L'application Electron tourne en tâche de fond. Grâce à node-pcsclite, elle détecte l'insertion d'une Carte Vitale.

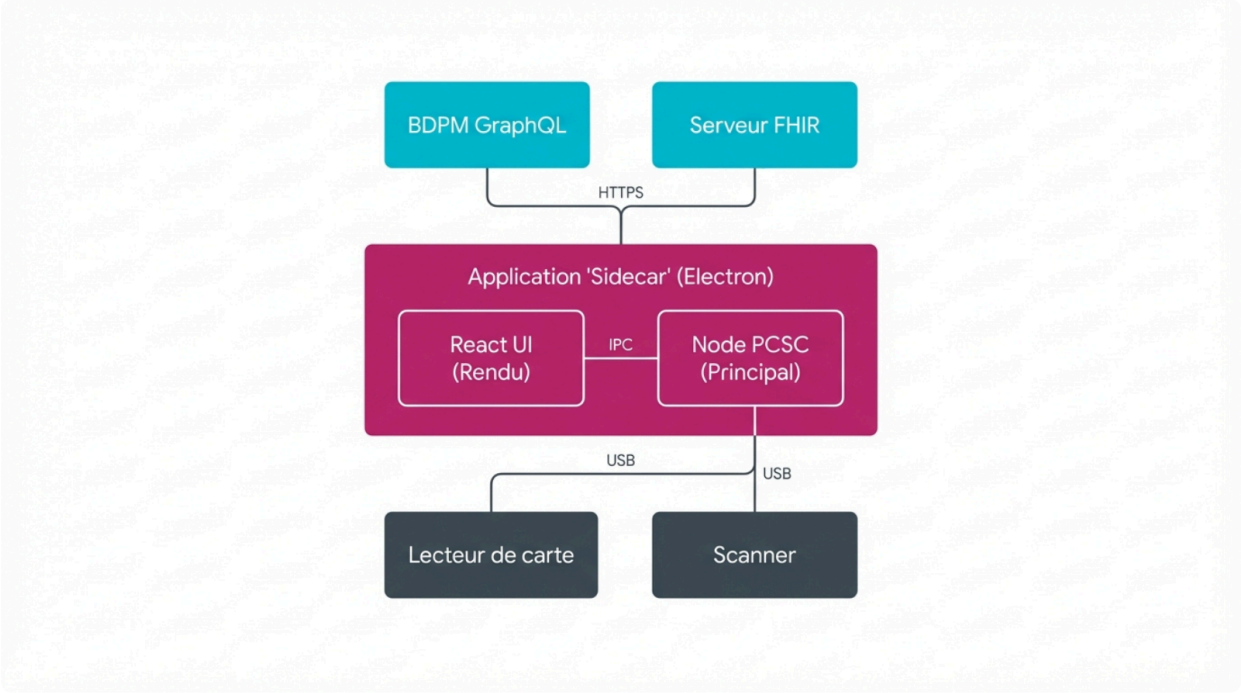
2. **Réveil Contextuel :** Elle identifie l'ATR de la carte vitale et "réveille" l'interface (fenêtre "Always on Top" ou notification) affichant le nom du patient et ses droits (lus via APDU).

3. **Enrichissement :** En parallèle, elle interroge l'API locale api-bdpm-graphql pour pré-charger les alertes sanitaires ou les rappels de lots récents.

4. **Assistance à la Délivrance :** Lorsque le pharmacien scanne une boîte, le module interpretGS1scan (intégré dans l'interface React) décode le Datamatrix. Si le lot est périmé, une alerte visuelle rouge bloque l'écran. Si une interaction est détectée via le moteur thesaurus, une notification apparaît.

5. **Indépendance :** Tout cela se passe sans interférer avec le logiciel de facturation. Le pharmacien utilise son LGO pour la vente, mais s'appuie sur le Sidecar pour l'expertise clinique et l'identité patient.

L'Architecture 'Open Sidecar' pour l'Officine



Vue d'ensemble : L'application 'Sidecar' (basée sur Electron) gère directement les périphériques (Lecteurs, Scanners) via Node.js, tout en interrogeant les sources de données ouvertes (BDPM, FHIR). Elle complète le LGO existant sans le remplacer.

Conclusion et Perspectives

L'analyse des projets disponibles sur GitHub confirme qu'il existe aujourd'hui une "pile technique" open source mature capable de transformer radicalement l'expérience au comptoir. Si la barrière réglementaire de la facturation reste le domaine réservé des éditeurs propriétaires, l'espace de l'intelligence clinique et de l'interaction matériel est largement ouvert à l'innovation.

En assemblant des briques comme node-pcsclite pour l'abstraction matérielle, api-bdpm-graphql pour l'intelligence des données, et les parseurs GS1 pour la conformité FMD, des développeurs ou des pharmaciens technophiles peuvent dès aujourd'hui déployer des outils "compagnons" puissants. Ces outils ne se contentent pas d'afficher de l'information ; ils redonnent au pharmacien la maîtrise de son outil de travail, en introduisant de l'interopérabilité et de la transparence là où régnaient l'opacité et la fermeture. L'avenir du logiciel officinal ne passera peut-être pas par un remplacement brutal des LGO, mais par cette augmentation progressive, modulaire et ouverte de l'écosystème existant.