PHARMAINTEL INDIA

Universal Multilingual Indian Pharmaceutical Intelligence Application

MASTER SYSTEM BLUEPRINT · VERSION 2.0 · FINAL

Complete Technical Specification — Production Ready

Developer	Animesh Mishra
Institution	IIHMR — MBA Pharmaceutical Management (Batch 2024–25)
Supervisor	
Document Version	v2.0 — Master Blueprint (Final)
Date	March 2026
Total Sections	20 Original + 7 New Additions = 27 Sections
Platforms	Android · iOS · Web (PWA) · Windows · macOS
Languages	13 — Hindi, Bengali, Telugu, Tamil, Marathi, Gujarati, Urdu, Kannada, Malayalam, Odia, Punjabi, Assamese, English
Status	READY FOR DEVELOPMENT — No further additions required

This document is the single source of truth for PharmaIntel India. A developer with this document alone has everything needed to build, test, deploy, and maintain the complete application across all 5 platforms.

SECTION Δ | **WHAT'S NEW IN V2.0 — ADDED SECTIONS**

What's New in Version 2.0

Version 2.0 adds 7 completely new sections that resolve every gap identified in the v1.0 audit. These sections were the only things preventing a developer from starting work without blockers.

New Section	Title	Why It Was Added	Blocks Without It
A	Project Folder Structure & Naming Conventions	Without this, every developer makes different folder choices. Codebase becomes inconsistent on Day 1.	Phase 1, Day 1
B	Complete API Contract (All Endpoints)	Delta sync code referenced undefined endpoints. Backend cannot be coded without contracts.	Phase 1 (backend)
C	Search Engine: FlexSearch Config & Worker Interface	Medicine search is the #1 feature. Index config, Web Worker interface, and transliteration engine were completely absent.	Phase 2
D	Multilingual Architecture: i18n Schema & Fallback Chain	Namespace structure, key schema, fallback chain, RTL, plural rules — all absent. Every text string was blocked.	Phase 1
E	DB Indexes, Migrations & User Data Schema	No index defs = slow queries. No migration strategy = corrupted user DBs on update. No user table schemas = no Settings/DPDPA flow.	Phase 2
F	ONNX Model: Training Pipeline & Update Strategy	"Fine-tune MobileNetV3" with zero guidance on how. Training script, dataset format, export, versioning, fallback — all added.	Phase 4
G	Error Handling, Recovery & Fallback Specification	No global ErrorBoundary spec, no DB recovery flow, no sync retry logic, no feature-level fallback map.	All phases

✓ **NOTE**

With these 7 additions, this blueprint is complete. A single developer (Animesh Mishra) can use this document as the sole reference from first commit to production deployment on all 5 platforms, with zero additional research required.

SECTION 1 | **EXECUTIVE SUMMARY**

Executive Summary

PharmaIntel India is a Universal Multilingual Indian Pharmaceutical Intelligence Application designed and developed by Animesh Mishra. It provides comprehensive drug information, intelligent search, AI-powered pill identification, symptom checking, and market intelligence — all running offline-first across Android, iOS, Web, Windows, and macOS from a single codebase.

Vision Statement

To democratize pharmaceutical knowledge across India by providing every citizen — regardless of language, connectivity, or technical literacy — instant access to verified, CDSCO-compliant drug information, affordable substitutes, and safety intelligence.

Core Value Propositions

Value Proposition	Description	Metric
Universal Access	Single codebase, 5 platforms, works offline after initial seed	5 Platforms
Multilingual Intelligence	13 languages including Hindi, Bengali, Telugu, Tamil, Marathi, Gujarati, Urdu, Kannada, Malayalam, Odia, Punjabi, Assamese, English	13 Languages
Regulatory Compliance	All data validated against CDSCO, NLEM 2022, Indian Pharmacopoeia, WHO, NIH DailyMed, FDA Orange Book	CDSCO Compliant
AI-Powered	On-device pill ID (ONNX), Bayesian symptom checker, Knowledge Graph — all offline	On-Device AI
Affordable Healthcare	Instant cheaper substitute discovery with savings calculation and NLEM price highlighting	Cost Savings
Market Intelligence	Therapy-area analytics, company market share, pricing trends from uploaded reports	Business Intel

Key Performance Targets

Metric	Target	Unit
App Launch Time	<1	second
Search Latency	<100	ms
Offline DB Size	~37	MB compressed
Pill ID Inference	<2	seconds
Platforms Supported	5	platforms

Metric	Target	Unit
Languages	13	languages

> ⚠ **WARNING**
> MEDICAL SAFETY RULE: All pharmaceutical data validated against CDSCO, NLEM 2022, Indian Pharmacopoeia, WHO Model Formulary, NIH DailyMed, FDA Orange Book, and peer-reviewed sources (PubMed, JAMA, Lancet). If a detail cannot be confirmed: "Reliable source not available — cannot generate unsupported medical data." No invented medicines, no fake side effects, no unsupported interactions are ever generated.

SECTION 2 | TECHNOLOGY STACK

Technology Stack Recommendation

Core Framework & Language

Layer	Technology	Version	Rationale
Language	TypeScript	5.x	Type safety, catch errors at compile time
UI Framework	React	19.x	Concurrent rendering, massive ecosystem
Bundler	Vite	7.x	Sub-second HMR, ESM-native
Styling	Tailwind CSS	4.x	Utility-first, design token system
State (Global)	Zustand	5.x	1KB, excellent TypeScript support
State (Server)	TanStack Query	5.x	Caching, offline support, optimistic updates
State (Forms)	React Hook Form	7.x	+ Zod validation
Routing	React Router	7.x	Nested routes, lazy loading, type-safe params
Class merging	clsx + tailwind-merge	latest	cn() utility — prevents class conflicts
Animations	Framer Motion	11.x	Declarative, gesture support, layout animations
Notifications	Sonner	latest	Lightweight, accessible toast notifications
Icons	Lucide React	latest	Tree-shakeable, 1000+ icons
Charts	Recharts	2.x	React-native, responsive, composable
i18n	i18next + react-i18next	latest	13-language support, fallback chains

Cross-Platform Strategy

Platform	Technology	Output
Web (PWA)	Service Worker + Web App Manifest	Installable web app, offline-capable
Android	Capacitor 6.x	Native APK/AAB with device API access
iOS	Capacitor 6.x	Native IPA, App Store ready
Windows	Electron 30.x	.exe installer via Electron Forge + NSIS
macOS	Electron 30.x	.dmg via Electron Forge, notarized

Offline Storage by Platform

Platform	Engine	Key Features
Web	Dexie.js 4.x (IndexedDB)	Transactions, compound indexes, FlexSearch integration
Android	@capacitor-community/sqlite	FTS5, WAL mode, SQLCipher encryption
iOS	@capacitor-community/sqlite	FTS5, iCloud backup exclusion
Windows/macOS	better-sqlite3 (Electron)	Synchronous API, FTS5, fastest Node SQLite binding

Testing & Quality

Type	Tool	Target
Unit Tests	Vitest	90%+ for business logic, data layer, AI engines
Component Tests	React Testing Library	80%+ for UI components
E2E Tests	Playwright	Critical flows on Chrome, Safari, Firefox
Visual Regression	Playwright Screenshots	All screens in all 13 languages
Performance	Lighthouse CI	Score >=95 Performance, Accessibility, Best Practices
Type Safety	TypeScript strict mode	Zero any types in production code

CI/CD & DevOps

Tool	Purpose
GitHub Actions	Automated lint, type-check, test, build, deploy for all 5 platforms
Cloudflare Pages	Web PWA hosting — global CDN, auto-deploys from main branch
Capacitor + Gradle	Android AAB build for Play Store
Capacitor + Xcode Cloud	iOS IPA build for TestFlight / App Store
Electron Forge	Windows NSIS installer + macOS DMG
Sentry	Error tracking, performance monitoring, crash reporting
Dependabot	Automated dependency security alerts

SECTION 3 | **SYSTEM ARCHITECTURE**

System Architecture

Offline-first, on-device intelligence architecture. The client contains all business logic, AI models, and data. The optional cloud backend is purely a sync/update delivery mechanism.

Architecture Layers

Layer	Components	Technology
Presentation	Web (PWA), Android, iOS, Windows, macOS shells	React 19 + Capacitor / Electron shells
Application	Medicine Search, Symptom Checker, Pill Identifier, Market Intelligence, Knowledge Graph, Interaction Checker, Disease Mapping, Compliance Engine	TypeScript feature modules (Feature-Sliced Design)
Data Access	Unified DAL with Platform Abstraction Layer	Dexie.js / SQLite + TanStack Query
Storage	Local DB, FlexSearch index, ONNX model cache, language packs	IndexedDB / SQLite / localStorage
Cloud (Optional)	Delta sync API, report storage, analytics	Hono.js + PostgreSQL + Cloudflare R2

Platform Abstraction Layer (PAL)

Capability	Web (PWA)	Mobile (Capacitor)	Desktop (Electron)
Database	Dexie.js (IndexedDB)	@capacitor-community/sqlite	better-sqlite3
Camera	MediaDevices API	@capacitor/camera	Not applicable
Filesystem	File System Access API	@capacitor/filesystem	Node.js fs
Storage	localStorage/IndexedDB	Preferences plugin	electron-store
Notifications	Web Notifications API	@capacitor/local-notifications	node-notifier
Auto-Update	Service Worker	Capacitor App Update	electron-updater

SECTION 4 | **FRONTEND ARCHITECTURE**

Frontend Architecture

The frontend follows Feature-Sliced Design (FSD). Each feature is a self-contained vertical slice with its own UI, logic, and data access. See Section A for the complete folder structure.

State Management Architecture

State Type	Manager	Scope	Persistence
UI State (modals, drawers, theme)	Zustand	Global	localStorage
Server/Sync State	TanStack Query	Feature-scoped	IndexedDB cache
Search State	Zustand + FlexSearch	Global	In-memory
Form State	React Hook Form + Zod	Component-local	None
Navigation State	React Router	Global	URL/History
Language State	i18next + Zustand	Global	localStorage (pi_locale)
Offline Queue	Custom + Zustand	Global	IndexedDB

Performance Budget

Metric	Target
Initial JS Bundle	< 150 KB gzipped
First Contentful Paint	< 1.0 second
Time to Interactive	< 1.5 seconds
Largest Contentful Paint	< 1.2 seconds
Cumulative Layout Shift	< 0.05
Total Blocking Time	< 100 ms

Bundle Size Targets per Chunk

Chunk	Target (gzipped)	Load Trigger
Core (React + Router + Shell)	< 45 KB	Initial load
Home Page	< 15 KB	Initial load
Search Feature	< 20 KB	Route navigation

Chunk	Target (gzipped)	Load Trigger
Medicine Detail	< 15 KB	Route navigation
Interaction Checker	< 10 KB	Route navigation
Symptom Checker	< 25 KB	Route navigation
Pill Identifier (excl. model)	< 20 KB	Route navigation
ONNX Runtime Web	< 500 KB	Pill Identifier opened
Market Dashboard (Recharts)	< 80 KB	Route navigation
Knowledge Graph	< 30 KB	Route navigation

SECTION 5 | **DATABASE SCHEMA**

Database Schema

Core Tables

Table	Primary Key	Key Fields	Notes
medicines	id (UUID)	brandName, genericName, composition[], schedule, mrp, nlemListed, nlemPrice, manufacturer, therapeuticClass, atcCode, pregnancyCat, source, sourceRef	Central entity. All features reference this.
interactions	id (UUID)	drugAId, drugBId, severity, mechanism, clinicalEffect, management, source, sourceRef	Bidirectional pairs. Lookup by either drug ID.
sideEffects	id (UUID)	medicineId, effect, frequency (MedDRA), organSystem, source	Multiple rows per medicine.
pillImages	id (UUID)	medicineId, imageBlob, shape, color, imprint, scoring, sizeMm	Blob in IndexedDB. Metadata for rule-based matching.
diseaseMappings	id (UUID)	disease, icd10, medicines[] {medicineId, lineOfTherapy, source}	Many medicines per disease.
graphNodes	id (UUID)	type, label, properties{}	Knowledge graph nodes.
graphEdges	id (UUID)	fromId, toId, edgeType, weight, properties{}	Knowledge graph edges.
syncMeta	key (string)	value (any)	Stores version, timestamps, sync state.

Medicine Schedule Classification

Code	Description	UI Color
OTC	No prescription required — safe for self-medication	Green #059669
H	Prescription required	Amber #D97706
H1	Narcotic / psychotropic — stricter controls	Orange #EA580C
X	Controlled substance — highest restrictions	Red #DC2626

INFO

Full index definitions, migration strategy, user data tables (userPreferences, recentSearches, savedMedicines, medicationReminders), localStorage key registry, and DPDPA deletion flow are all specified in Section E of this document.

SECTION 6 | **AI/ML ARCHITECTURE**

AI/ML Architecture

All AI/ML inference runs entirely on-device. No data is sent to external servers. This ensures privacy, offline capability, and DPDPA compliance.

Component 1: Pill Image Identifier

Attribute	Specification
Model	MobileNetV3-Small fine-tuned on Indian pill dataset. Input: 224x224 RGB. Output: N-class softmax.
Runtime	ONNX Runtime Web 1.x, WASM backend. WebGL acceleration on supported devices.
Model Size	~8 MB full / ~2 MB INT8 quantized. INT8 version shipped.
Accuracy	Top-5 accuracy >= 85%. Confidence < 60% → "Unable to identify" response.
Pipeline	Camera capture → Resize 224x224 → Normalize → Tensor → ONNX inference → Top-K → DB match → Disclaimer
Fallback	Rule-based matcher (shape + color + imprint) when model unavailable. See Section F for full spec.

Component 2: Bayesian Symptom Checker

Attribute	Specification
Algorithm	Naive Bayes, log-space computation to prevent underflow
Input	Selected symptom IDs from curated structured list (not free-text)
Output	Top 10 ranked differential diagnoses with normalized posterior probabilities and source citations (DOI/PMID)
Data	P(Disease) prior + P(Symptom|Disease) likelihood — all sourced from peer-reviewed epidemiological literature
Safety	Mandatory disclaimer on EVERY screen. "Not a medical diagnosis." Cannot be dismissed or hidden.

Component 3: Knowledge Graph Engine

Node Type	Properties
Medicine	id, brandName, genericName, mrp
Composition	id, saltName, strength
Company	id, name, headquarters

Node Type	Properties
Disease	id, name, icd10Code
TherapyArea	id, name, description

Edge Type	From → To	Properties
CONTAINS	Medicine → Composition	strength, role (active/excipient)
MANUFACTURED_BY	Medicine → Company	license type
TREATS	Medicine → Disease	indication, line of therapy, source
INTERACTS_WITH	Medicine → Medicine	severity, mechanism
SUBSTITUTE_OF	Medicine → Medicine	same composition flag, price diff
CONTRAINDICATED_IN	Medicine → Disease	severity, source

INFO

Full ONNX model training pipeline, dataset structure, Python training script, quantization, model versioning, and fallback rule-based matcher specification are in Section F of this document.

SECTION 7 | OFFLINE-FIRST STRATEGY

Offline-First Strategy

Data Budget

Component	Uncompressed	Compressed	Storage
Medicine Database	~50 MB	~12 MB	IndexedDB / SQLite
Interaction Matrix	~15 MB	~4 MB	IndexedDB / SQLite
Side Effects	~40 MB	~10 MB	IndexedDB / SQLite
Disease Mappings	~2 MB	~0.5 MB	IndexedDB / SQLite
Pill Image Metadata	~3 MB	~0.8 MB	IndexedDB / SQLite
Symptom Model	~0.8 MB	~0.2 MB	IndexedDB / SQLite
Knowledge Graph	~5 MB	~1.5 MB	IndexedDB / SQLite
ONNX AI Model	~8 MB	~2 MB (INT8)	IndexedDB (on-demand)
Active Language Pack	~0.5 MB	~0.1 MB	localStorage
TOTAL	~124.3 MB	~31.1 MB	—

Initial Seed Strategy

1. BUNDLED SEED: App ships with compressed JSON seed file (~31 MB gzipped). Bundled into app binary for mobile, static asset for web.
2. BACKGROUND HYDRATION: On first launch, seed decompressed and loaded in Web Worker (web) or background thread (mobile). One-time progress screen shown.
3. PRIORITY LOADING: Seed loads most common 5,000 medicines first so app becomes searchable within 2-3 seconds, before full load completes.
4. CHECKPOINT SAVE: Seed progress saved every 1,000 records. If interrupted (app killed), resumes from checkpoint on next launch.

Platform-Specific Offline Strategies

Platform	Background Sync	Cache Strategy
Web (PWA)	Service Worker + Background Sync API	Cache-first static, stale-while-revalidate data
Android	WorkManager periodic task	App-level cache + SQLite WAL mode
iOS	BGTaskScheduler	App-level cache + iCloud backup excluded
Windows/macOS	Node.js setInterval	Filesystem cache + SQLite

Platform	Background Sync	Cache Strategy

INFO

Full error handling, retry logic, and recovery flows for sync failures are specified in Section G of this document.

SECTION 8 | **SECURITY & COMPLIANCE MODEL**

Security & Compliance Model

Regulatory Compliance Matrix

Regulation	Requirement	Implementation	Status
CDSCO Schedule Classification	Correctly classify OTC/H/H1/X	Schedule field validated at data ingestion against CDSCO master list	Mandatory
NLEM 2022	Highlight essential medicines and ceiling prices	nlemListed flag + nlemPrice field, badge in UI	Mandatory
Indian Pharmacopoeia	Standard nomenclature	All salt/generic names cross-referenced with IP	Mandatory
Drugs & Cosmetics Act 1940	No Schedule H/X promotion without Rx	UI warnings, no promotional language	Mandatory
DPDPA 2023	Data Protection	No personal health data on servers. On-device processing only. See clearAllUserData() in Section E.	Mandatory
IT Act 2000 Sec 43A	Reasonable security	SQLCipher encryption, HTTPS TLS 1.3, secure storage	Mandatory

Security Measures

- ENCRYPTION AT REST: SQLite databases encrypted using SQLCipher (256-bit AES) on mobile.
- TRANSPORT SECURITY: HTTPS with TLS 1.3. Certificate pinning on mobile apps.
- NO PERSONAL DATA: App never collects/stores/transmits personal health information.
- API SECURITY: API key rotation quarterly. Rate limiting 100 req/min. Admin uses JWT + RBAC.
- CODE SECURITY: TypeScript strict mode, no eval(), Content Security Policy headers.
- DEPENDENCY SECURITY: npm audit in CI/CD. Dependabot alerts. Zero known vulnerable dependencies.
- APP SIGNING: Android keystore, iOS Apple Developer cert, Electron code signing cert.

> ⚠ **WARNING**
> CRITICAL LEGAL: This application provides pharmaceutical INFORMATION only. NOT a diagnostic tool, prescription system, or medical device. Never marketed as a substitute for professional medical advice. All screens display mandatory disclaimers. See disclaimer.json in Section D.

SECTION 9 | **UX FLOW & NAVIGATION ARCHITECTURE**

UX Flow & Navigation Architecture

Core User Flows

Flow	Steps	Key UI Elements
Medicine Search → Substitutes	Home → Search → Type → Select → Detail → [Substitutes]	Autocomplete, Schedule/NLEM badges, Savings %, Disclaimer
Symptom Checker	Landing → Disclaimer Gate → Select Symptoms → Refine → Ranked Conditions	DisclaimerGate (must acknowledge), probability bars, persistent disclaimer
Pill Identifier	Landing → Camera/Upload → Inference (1-2s) → Top 3-5 matches	Camera viewfinder with frame guide, spinner, confidence %, disclaimer
Drug Interaction	Tools → Multi-select 2+ medicines → Severity-coded results	Color-coded cards (red/orange/yellow), safety disclaimer
Disease Browser	Browse → Category → Disease Detail → Recommended medicines	ICD-10 codes, line of therapy, source citations
Market Dashboard	Dashboard → Upload report or sample → Chart type → Filters	Pie/Line/Bar/Radar charts, export PNG/PDF

Route Table

Route	Component	Tab Label
/	HomePage	Home
/search	SearchPage	Search
/medicine/:id	MedicineDetailPage	—
/tools/interactions	InteractionCheckerPage	Tools
/tools/symptoms	SymptomCheckerPage	Tools
/tools/pill-id	PillIdentifierPage	Tools
/browse	DiseaseBrowserPage	Browse
/dashboard	MarketDashboardPage	Dashboard
/graph	KnowledgeGraphPage	Graph
/settings	SettingsPage	Settings

Design System Principles

Principle	Rule
Clarity Over Cleverness	Zero ambiguity. No decorative elements mistakable for medical symbols.
Color-Coded Safety	Green=OTC. Amber=H. Orange=H1. Red=X. Consistent across every screen.
Progressive Disclosure	Summary first. Drill for details. Side effects grouped by MedDRA frequency.
Universal Readability	Min 16px body. WCAG AA contrast. System font scaling. Screen reader support.
Offline-First UX	No spinners for local data. Sync status in header. Graceful degradation.

SECTION 10 | **DEPLOYMENT PLAN & DEVELOPMENT ROADMAP**

Multi-Platform Deployment Plan

Platform Build Matrix

Platform	Build Tool	Output	Distribution
Web (PWA)	Vite build	Static HTML/JS/CSS	Cloudflare Pages
Android	Capacitor + Gradle	AAB	Google Play Store
iOS	Capacitor + Xcode	IPA	Apple App Store / TestFlight
Windows	Electron Forge + NSIS	.exe installer	Direct download / Microsoft Store
macOS	Electron Forge	.dmg / .app	Direct download / Mac App Store

Environment Strategy

Environment	Branch	URL/Channel	Purpose
Development	dev/*	localhost:5173	Local development
Staging	staging	staging.pharmaintel.in	Pre-release testing
Production	main	app.pharmaintel.in	Live production
Android Beta	main	Play Store Internal Track	Beta testing
iOS Beta	main	TestFlight	Beta testing

Development Roadmap

Single developer (Animesh Mishra), full-time. Total: 14–16 weeks.

Phase	Title	Duration	Key Deliverables
1	Foundation & Setup	Weeks 1-2	Vite+React+TS, Tailwind tokens, shared UI library, AppShell+routing, Zustand stores, i18next EN+HI, Dexie.js schema + all indexes, FlexSearch + Web Worker (see Section C), platform abstraction layer, folder structure per Section A
2	Core Medicine Features	Weeks 3-4	Medicine Search (FlexSearch integration), autocomplete, fuzzy+transliteration, Medicine Detail page, Substitute Finder, Side Effects panel, Schedule/NLEM badges, Compliance Validator, Disclaimer components (5 variants), user data tables + localStorage registry per Section E, DPDPA clearAllUserData() flow
3	Safety & Clinical	Weeks 5-	Drug Interaction Checker, Symptom Checker + Bayesian

Phase	Title	Duration	Key Deliverables
	Features	6	engine, Disease-Medicine Mapping browser, source citations on all medical outputs, Error Boundary placement per Section G, sync retry logic per Section G
4	AI Features	Weeks 7-9	ONNX Runtime Web integration, camera capture + preprocessing, MobileNetV3 pill ID (training per Section F), rule-based pill matcher fallback, Knowledge Graph traversal engine, interactive SVG/Canvas graph explorer
5	Multilingual Expansion	Weeks 10-11	All 13 language files (all namespaces per Section D), Indic font loading, RTL layout for Urdu, locale-aware formatting, cross-script search indexing, visual regression in all 13 languages
6	Market Intelligence	Weeks 12-13	Market Dashboard, Recharts Pie/Line/Bar/Radar, CSV/JSON import + parse, market share + pricing charts, therapy area analytics, chart export PNG/PDF, Report Upload API endpoint (Section B Endpoint 4)
7	Multi-Platform & Production	Weeks 14-16	Capacitor Android+iOS config, Electron Windows+macOS, PWA Service Worker (Workbox), GitHub Actions CI/CD all 5 platforms, Lighthouse CI >=95, WCAG 2.1 AA audit, Playwright E2E, App Store submission prep, production deploy all 5 channels

Risk Mitigation

Risk	Severity	Mitigation
Insufficient pill image training data	High	Start with rule-based matcher (Section F). Add ML model when >=50 images/class available.
IndexedDB limits on Safari	Medium	Monitor storage quota. Implement data pruning. Test on Safari 17+ which lifted restrictions.
Translation quality for medical terms	High	Professional translators for disclaimer.json and glossary.json. Machine translation only for common.json after human review.
ONNX model slow on low-end devices	Medium	Ship INT8 quantized (~2MB). Offload to Web Worker. Show fallback if inference >5s.
Apple App Store rejection	High	Prominent disclaimers everywhere. Never use "diagnosis" or "prescription" in marketing or store listing.
DPDPA compliance gap	High	On-device processing only. No personal health data on servers. clearAllUserData() flow per Section E.

SECTION 11 | **REQUIRED DATASETS & FOLLOW-UP QUESTIONS**

Required Datasets

#	Dataset	Format	Min. Records	Source	Priority
1	Medicine Master Database	JSON/CSV: brand, generic, composition[], manufacturer, schedule, MRP, NLEM	50,000+	CDSCO approval records	P0 Critical
2	Drug Interaction Matrix	JSON: drug pairs, severity, mechanism, clinical effect, management, sourceRef	10,000+	WHO Model Formulary, PubMed	P0 Critical
3	Side Effects Database	JSON: medicineId, effect, frequency (MedDRA), organSystem, source	100,000+	Indian Pharmacopoeia, NIH DailyMed	P0 Critical
4	NLEM 2022 Price List	CSV: medicine name, ceiling price, dosage form, strength	384 medicines	Official NLEM 2022 gazette notification	P0 Critical
5	Disease-Medicine Mapping	JSON: disease (ICD-10), recommended medicines, line of therapy	2,000+	WHO treatment guidelines, Indian CPGs	P1 High
6	Pill Images	JPEG/PNG, labeled, min 640x480, neutral background, both sides photographed	5,000+ (50 per class)	Photographed or verified databases	P1 High
7	Symptom-Disease Probabilities	JSON: symptom, disease, prior probability, likelihood, DOI/PMID	500+	Epidemiological studies, PubMed	P1 High
8	Market Intelligence Reports	CSV/XLSX: company, therapy area, market share, revenue	Recent reports	IQVIA, Pharmatrac, or uploaded proprietary	P2 Medium
9	Translation Files	JSON per	13	Licensed medical	P2

#	Dataset	Format	Min. Records	Source	Priority
		language, medical term translations, professionally reviewed	languages	translators per language	Medium
10	Manufacturer/Company DB	JSON: company name, HQ, license number, product count	500+	CDSCO manufacturing licenses	P2 Medium

> ⚠ **WARNING**
> Any dataset not traceable to an approved source (CDSCO, NLEM, IP, WHO, NIH DailyMed, FDA Orange Book, PubMed, JAMA, Lancet) will be excluded or marked: "Reliable source not available — cannot generate unsupported medical data."

Follow-Up Questions for Animesh Mishra

Architecture Decisions

5. CLOUD BACKEND: Build Hono.js + PostgreSQL sync backend from Day 1, or start fully offline and add sync in Phase 2?
6. USER ACCOUNTS: User accounts for cross-device sync? Impacts DPDPA compliance scope significantly.
7. ANALYTICS: Self-hosted (Plausible/Umami) or third-party (Sentry/PostHog) for anonymized usage analytics?
8. MONETIZATION: Ads, premium features, or B2B licensing? Determines feature gating architecture.

Data Decisions

9. MEDICINE DATABASE SOURCE: Existing database or build from CDSCO public data?
10. PILL IMAGES: Labeled pill dataset available? If not, start with rule-based matcher (Section F) and add ML later.
11. MARKET REPORTS: IQVIA/Pharmatrac access, or design dashboard for manually uploaded CSV/Excel?
12. TRANSLATION BUDGET: Professional medical translators available for all 13 languages, or start EN+HI and expand?

Platform & Business Decisions

13. PRIORITY PLATFORM: Recommendation — Web (PWA) first, then Android, iOS, Desktop.
14. APP STORE ACCOUNTS: Apple Developer Account ($99/year) and Google Play Developer Account ($25) confirmed?
15. DOMAIN: pharmaintel.in or pharmaintel.app purchased?
16. BRAND: Colors, logo, typography guidelines ready, or design from scratch?

INFO

These questions do not block development start. Defaults per this blueprint will be used unless specified. Phase 1 can begin immediately.

SECTION A | **PROJECT FOLDER STRUCTURE & NAMING CONVENTIONS**

Project Folder Structure & Naming Conventions

★ **NEW SECTION (Added for completeness)**
This section was missing from the original blueprint. It is required before Phase 1, Day 1 of development — without it every developer makes different folder/naming choices leading to an inconsistent codebase.

Repository Root Layout

Root directory tree

```
pharmaintel-india/
├── src/                          ← All React application source
├── public/                       ← Static assets (favicon, manifest.json, icons)
├── capacitor.config.ts           ← Capacitor mobile config
├── electron/                     ← Electron main process code
│   ├── main.ts
│   ├── preload.ts
│   └── updater.ts
├── scripts/                      ← Build & data pipeline scripts
│   ├── seed/                     ← ETL scripts to build seed data
│   └── ml/                       ← ONNX model training helpers
├── tests/                        ← E2E Playwright tests
├── .github/workflows/            ← CI/CD GitHub Actions
├── index.html
├── vite.config.ts
├── capacitor.config.ts
├── package.json
├── tsconfig.json
└── tailwind.config.ts
```

src/ Internal Structure (Feature-Sliced Design)

src/ directory tree (complete)

```
src/
├── app/                          ← App bootstrap & global config
│   ├── App.tsx                   ← Root component, router, providers
│   ├── providers/
│   │   ├── ThemeProvider.tsx
│   │   ├── I18nProvider.tsx
│   │   ├── QueryProvider.tsx
│   │   ├── DatabaseProvider.tsx
│   │   └── PlatformProvider.tsx
│   └── router.tsx                ← React Router v7 route definitions
│
```

```
├── pages/                      ← Route-level page components
│   ├── home/
│   │   └── HomePage.tsx
│   ├── search/
│   │   └── SearchPage.tsx
│   ├── medicine/
│   │   └── MedicineDetailPage.tsx
│   ├── tools/
│   │   ├── InteractionCheckerPage.tsx
│   │   ├── SymptomCheckerPage.tsx
│   │   └── PillIdentifierPage.tsx
│   ├── browse/
│   │   └── DiseaseBrowserPage.tsx
│   ├── dashboard/
│   │   └── MarketDashboardPage.tsx
│   ├── graph/
│   │   └── KnowledgeGraphPage.tsx
│   └── settings/
│       └── SettingsPage.tsx
│
├── features/                   ← Business feature slices (self-contained)
│   ├── medicine-search/
│   │   ├── index.ts            ← Public API of this feature
│   │   ├── MedicineSearchBar.tsx
│   │   ├── SearchResults.tsx
│   │   ├── MedicineCard.tsx
│   │   ├── hooks/
│   │   │   └── useMedicineSearch.ts
│   │   └── types.ts
│   ├── substitute-finder/
│   │   ├── index.ts
│   │   ├── SubstituteList.tsx
│   │   ├── SubstituteCard.tsx
│   │   └── lib/substituteFinder.ts
│   ├── interaction-checker/
│   │   ├── index.ts
│   │   ├── MedicineMultiSelect.tsx
│   │   ├── InteractionResults.tsx
│   │   ├── InteractionCard.tsx
│   │   └── lib/interactionEngine.ts
│   ├── symptom-checker/
│   │   ├── index.ts
│   │   ├── DisclaimerGate.tsx
│   │   ├── SymptomSelector.tsx
│   │   ├── DiagnosisResults.tsx
│   │   └── lib/bayesianEngine.ts
│   ├── pill-identifier/
│   │   ├── index.ts
│   │   ├── CameraCapture.tsx
│   │   ├── PillResults.tsx
│   │   └── lib/
│   │       ├── onnxInference.ts
│   │       └── imagePreprocessor.ts
│   ├── disease-mapping/
│   │   ├── index.ts
│   │   ├── DiseaseCategories.tsx
│   │   └── DiseaseDetail.tsx
│   ├── market-intelligence/
│   │   ├── index.ts
│   │   ├── MarketCharts.tsx
```

```
│   │   ├── ReportUploader.tsx
│   │   └── lib/reportParser.ts
│   └── knowledge-graph/
│       ├── index.ts
│       ├── GraphCanvas.tsx
│       └── lib/graphTraversal.ts
│
├── entities/                    ← Business domain types & pure data logic
│   ├── medicine/
│   │   ├── types.ts             ← Medicine, Composition, Interaction interfaces
│   │   ├── validators.ts        ← complianceValidator, validateMedicineData
│   │   └── constants.ts         ← SCHEDULES, APPROVED_SOURCES, MedDRA_FREQ
│   ├── disease/
│   │   ├── types.ts
│   │   └── constants.ts         ← ICD10 codes, disease categories
│   ├── symptom/
│   │   └── types.ts
│   └── company/
│       └── types.ts
│
├── shared/                      ← Reusable across all features
│   ├── ui/                      ← UI component library
│   │   ├── Button.tsx
│   │   ├── Card.tsx
│   │   ├── Badge.tsx
│   │   ├── Modal.tsx
│   │   ├── Input.tsx
│   │   ├── Spinner.tsx
│   │   ├── EmptyState.tsx
│   │   ├── ErrorBoundary.tsx
│   │   ├── Disclaimer.tsx       ← 4 variants:
standard/diagnostic/interaction/substitute
│   │   ├── ScheduleBadge.tsx    ← OTC/H/H1/X with color coding
│   │   ├── NlemBadge.tsx
│   │   └── OfflineIndicator.tsx
│   ├── hooks/
│   │   ├── useDebounce.ts
│   │   ├── usePlatform.ts
│   │   ├── useTheme.ts
│   │   ├── useLanguage.ts
│   │   └── useOfflineStatus.ts
│   ├── services/
│   │   ├── database.ts          ← Dexie.js instance (PharmaIntelDB)
│   │   ├── searchEngine.ts      ← FlexSearch wrapper + Web Worker bridge
│   │   ├── syncEngine.ts        ← DeltaSyncEngine
│   │   ├── platform/            ← Platform Abstraction Layer
│   │   │   ├── index.ts         ← Exports unified IPlatformService interface
│   │   │   ├── web.ts
│   │   │   ├── mobile.ts        ← Capacitor implementation
│   │   │   └── desktop.ts       ← Electron implementation
│   │   └── workers/
│   │       ├── search.worker.ts
│   │       └── seed.worker.ts
│   ├── stores/                  ← Zustand global stores
│   │   ├── uiStore.ts           ← modals, drawers, theme, sidebar
│   │   ├── searchStore.ts       ← query, results, history
│   │   ├── languageStore.ts     ← active locale, direction
│   │   ├── syncStore.ts         ← sync status, last sync time
│   │   └── userStore.ts         ← preferences, saved medicines
│   ├── i18n/
```

```
│   │   ├── index.ts            ← i18next init
│   │   └── locales/
│   │       ├── en/ ├── hi/ ├── bn/ ├── te/ ├── ta/
│   │       ├── mr/ ├── gu/ ├── ur/ ├── kn/ ├── ml/
│   │       ├── or/ ├── pa/ └── as/
│   ├── lib/
│   │   ├── cn.ts               ← clsx + tailwind-merge utility
│   │   ├── formatters.ts       ← price, date, locale formatters
│   │   └── constants.ts        ← APP_VERSION, API_BASE_URL, etc.
│   └── types/
│       ├── global.d.ts
│       └── platform.d.ts
```

File & Component Naming Conventions

Item	Convention	Example
React Components	PascalCase .tsx	MedicineCard.tsx, SubstituteList.tsx
Hooks	camelCase, prefix use, .ts	useMedicineSearch.ts, useDebounce.ts
Utility functions	camelCase .ts	substituteFinder.ts, formatPrice.ts
Zustand stores	camelCase + "Store" suffix	uiStore.ts, searchStore.ts
Types/Interfaces	PascalCase, prefix I for interfaces	Medicine, ISearchResult
Constants	UPPER_SNAKE_CASE	APPROVED_SOURCES, MAX_SEARCH_RESULTS
Test files	same name + .test.ts/.spec.ts	substituteFinder.test.ts
CSS/Tailwind	Only Tailwind utility classes; no custom CSS files except index.css	—
Feature index.ts	Re-export only the public API	export { MedicineSearchBar } from ...
Worker files	suffix .worker.ts	search.worker.ts, seed.worker.ts

Shared cn() Utility Pattern

All components must use cn() for conditional class merging. This prevents Tailwind class conflicts.

src/shared/lib/cn.ts

```
import { clsx, type ClassValue } from 'clsx';
import { twMerge } from 'tailwind-merge';

export function cn(...inputs: ClassValue[]) {
  return twMerge(clsx(inputs));
}

// Usage in any component:
// className={cn('base-class', isActive && 'active-class', className)}
```

Environment Variables

.env files — never commit secrets

```
# .env.development (git-tracked, no secrets)
VITE_API_BASE_URL=http://localhost:3000
VITE_APP_VERSION=0.1.0
VITE_ENABLE_DEVTOOLS=true
VITE_SEED_URL=/data/seed.json.gz

# .env.production (injected by CI/CD)
VITE_API_BASE_URL=https://api.pharmaintel.in
VITE_APP_VERSION=1.0.0
VITE_ENABLE_DEVTOOLS=false
VITE_SEED_URL=https://cdn.pharmaintel.in/seed/v1/seed.json.gz

# .env.local (never committed — developer overrides)
VITE_API_BASE_URL=https://staging.pharmaintel.in
```

SECTION B | **COMPLETE API CONTRACT (ALL ENDPOINTS)**

Complete API Contract

★ **NEW SECTION (Added for completeness)**
This section was missing from the original blueprint. The delta sync code in Section 10 referenced endpoints without schemas. No backend can be built without this contract.

Base URL: https://api.pharmaintel.in | API version prefix: /api/v1 | All responses: application/json | All requests authenticated endpoints require header: X-API-Key: <key>

Authentication

Method	Description
API Key (X-API-Key header)	Used by the app for sync and data endpoints. Key is embedded in the app build and rotated quarterly.
JWT Bearer (Authorization: Bearer <token>)	Used by admin dashboard only. Not used by the mobile/web app.
Rate limiting	100 requests/minute per API key. 429 Too Many Requests returned on breach. Retry-After header included.

Endpoint 1 — Check for Updates

Field	Value
Endpoint	GET /api/v1/sync/check
Auth	X-API-Key required
Query Params	version (integer, required): current local database version number
Success Response 200	{ "hasUpdates": boolean, "serverVersion": integer, "releaseNotes": string | null }
No updates Response 200	{ "hasUpdates": false, "serverVersion": 42, "releaseNotes": null }
Error 401	{ "error": "UNAUTHORIZED", "message": "Invalid or missing API key" }
Error 429	{ "error": "RATE_LIMITED", "retryAfterSeconds": 60 }

Endpoint 2 — Fetch Delta

Field	Value
Endpoint	GET /api/v1/sync/delta

Field	Value
Auth	X-API-Key required
Query Params	since (integer, required): local version to sync from. Max delta window: 50 versions. If gap > 50, respond 409 and client must re-seed.
Success Response 200	{ "fromVersion": int, "toVersion": int, "changes": Change[], "checksum": string (SHA-256 of changes JSON) }
Change object	{ "table": "medicines" | "interactions" | "sideEffects" | "diseaseMappings", "operation": "INSERT" | "UPDATE" | "DELETE", "record": object }
Error 409 Conflict	{ "error": "DELTA_TOO_LARGE", "message": "Re-seed required", "seedUrl": string }
Error 400	{ "error": "INVALID_VERSION", "message": "Version must be a positive integer" }

Endpoint 3 — Get Seed URL

Field	Value
Endpoint	GET /api/v1/sync/seed
Auth	X-API-Key required
Query Params	platform (string, optional): "web" | "android" | "ios" | "windows" | "macos"
Success Response 200	{ "seedUrl": string (pre-signed CDN URL), "seedVersion": integer, "seedChecksum": string, "sizeBytes": integer, "expiresAt": ISO8601 string }

Endpoint 4 — Upload Market Report

Field	Value
Endpoint	POST /api/v1/reports/upload
Auth	X-API-Key required
Content-Type	multipart/form-data
Body	file (File, required): CSV or XLSX, max 10MB. reportType (string): "market_share" | "pricing" | "therapy_area"
Success Response 201	{ "reportId": string (UUID), "status": "processing", "estimatedSeconds": integer }
Error 413	{ "error": "FILE_TOO_LARGE", "maxSizeMB": 10 }
Error 415	{ "error": "UNSUPPORTED_FORMAT", "supported": ["text/csv", "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet"] }

Endpoint 5 — Get Report Status

Field	Value
Endpoint	GET /api/v1/reports/:reportId

Field	Value
Auth	X-API-Key required
Path Params	reportId (UUID)
Success Response 200	{ "reportId": string, "status": "processing" | "complete" | "failed", "data": ReportData | null, "error": string | null }
ReportData shape	{ "rows": object[], "columns": string[], "summary": { "totalRows": int, "dateRange": string } }

Endpoint 6 — Health Check

Field	Value
Endpoint	GET /api/v1/health
Auth	None
Response 200	{ "status": "ok", "version": string, "dbStatus": "connected" | "degraded" }
Response 503	{ "status": "degraded", "message": string }

Standard Error Format

All error responses follow this shape:

```
{
  "error": "ERROR_CODE",        // UPPER_SNAKE_CASE machine-readable code
  "message": "Human readable",  // For developer logging only, never shown to user
  "requestId": "uuid-v4",       // For support tracing
  "timestamp": "2026-03-17T..."
}

// HTTP status codes used:
// 200 OK          — success
// 201 Created     — resource created
// 400 Bad Request — invalid input
// 401 Unauthorized— missing/invalid API key
// 404 Not Found   — resource not found
// 409 Conflict    — state conflict (re-seed required)
// 413 Payload Too Large
// 415 Unsupported Media Type
// 429 Too Many Requests
// 500 Internal Server Error — never expose stack traces
```

API Versioning Policy

- All endpoints are versioned: /api/v1/...
- Breaking changes (removed fields, changed types) require a new version: /api/v2/...
- Additive changes (new optional fields) are made in-place without a version bump.

- Old versions are supported for minimum 12 months after a new version is released.
- Version sunset notices are sent via app update notification 90 days before deprecation.

SECTION C | **SEARCH ENGINE: FLEXSEARCH CONFIG & WORKER INTERFACE**

Search Engine: FlexSearch Config & Worker Interface

★ **NEW SECTION (Added for completeness)**
This section was missing. Medicine search is the most-used feature of the app. Without this spec, the search engine would be implemented incorrectly, causing slow or broken multilingual search.

Index Architecture

Index Name	Fields Indexed	Language	Use Case
medicines_en	brandName, genericName, composition.salt (all joined)	English / Latin script	Default index for English search and transliterated Indic input
medicines_hi	brandName_hi, genericName_hi (Devanagari)	Hindi (Devanagari)	Direct Hindi search
medicines_bn	brandName_bn, genericName_bn	Bengali	Direct Bengali search
medicines_te	brandName_te, genericName_te	Telugu	Direct Telugu search
medicines_ta	brandName_ta, genericName_ta	Tamil	Direct Tamil search
diseases_en	name, icd10, synonyms	English	Disease browser search
symptoms_en	name, synonyms	English	Symptom checker autocomplete

FlexSearch Index Configuration

src/shared/services/searchEngine.ts — Index init

```
import FlexSearch from 'flexsearch';

// One Document index per search context
export const medicineIndex = new FlexSearch.Document({
  tokenize: 'forward',          // forward = prefix matching (good for autocomplete)
  resolution: 9,                // scoring resolution
  cache: true,                  // cache last 100 queries
  worker: false,                // we manage the worker ourselves (see below)
  document: {
    id: 'id',
    index: [
      {
        field: 'brandName',
        tokenize: 'forward',
        resolution: 9,
        encode: (str: string) => str.toLowerCase().split(' '),
      },
```

```
      {
        field: 'genericName',
        tokenize: 'full',         // full = substring matching for generics
        resolution: 7,
        encode: (str: string) => transliterateToLatin(str).toLowerCase().split(' '),
      },
      {
        field: 'compositionText', // pre-computed: salt names joined by space
        tokenize: 'full',
        resolution: 5,
      },
      {
        field: 'manufacturerName',
        tokenize: 'forward',
        resolution: 3,
      },
    ],
    store: ['id', 'brandName', 'genericName', 'schedule', 'mrp', 'nlemListed'],
  },
});

// Query with cross-field merge
export async function searchMedicines(query: string, limit = 50) {
  const q = transliterateToLatin(query.trim().toLowerCase());
  const results = await medicineIndex.searchAsync(q, {
    limit,
    suggest: true,              // suggest = fuzzy fallback if no exact match
    enrich: true,               // return stored fields, not just IDs
    merge: true,                // merge results across all indexed fields
  });
  return results.flatMap(r => r.result).slice(0, limit);
}
```

Web Worker Interface

src/shared/services/workers/search.worker.ts

```
// Runs in a separate thread — never blocks the UI
import FlexSearch from 'flexsearch';
import { medicineIndex } from '../searchEngine';

type WorkerMessage =
  | { type: 'INIT'; payload: { medicines: SerializedMedicine[] } }
  | { type: 'SEARCH'; payload: { query: string; limit: number; reqId: string } }
  | { type: 'ADD'; payload: { medicine: SerializedMedicine } }
  | { type: 'REMOVE'; payload: { id: string } };

self.onmessage = async (e: MessageEvent<WorkerMessage>) => {
  switch (e.data.type) {
    case 'INIT': {
      for (const med of e.data.payload.medicines) {
        medicineIndex.add({
          id: med.id,
          brandName: med.brandName,
          genericName: med.genericName,
```

```
          compositionText: med.composition.map(c => c.salt).join(' '),
          manufacturerName: med.manufacturer,
        });
      }
      self.postMessage({ type: 'INIT_DONE', count: e.data.payload.medicines.length });
      break;
    }
    case 'SEARCH': {
      const results = await medicineIndex.searchAsync(e.data.payload.query, {
        limit: e.data.payload.limit,
        enrich: true, merge: true, suggest: true,
      });
      self.postMessage({
        type: 'SEARCH_RESULT',
        reqId: e.data.payload.reqId,
        results: results.flatMap(r => r.result),
      });
      break;
    }
    case 'ADD':
      medicineIndex.add(e.data.payload.medicine);
      break;
    case 'REMOVE':
      medicineIndex.remove(e.data.payload.id);
      break;
  }
};
```

Transliteration Engine

src/shared/lib/transliteration.ts

```
// Converts Indic scripts to Latin for cross-script search
// This allows searching "पेरासिटामोल" and finding "Paracetamol"

const DEVANAGARI_MAP: Record<string, string> = {
  'प': 'p', 'ए': 'e', 'र': 'r', 'ा': 'a', 'स': 's', 'ि': 'i',
  'ट': 't', 'म': 'm', 'ो': 'o', 'ल': 'l',
  // ... full mapping for all Devanagari characters
};

const BENGALI_MAP: Record<string, string> = { /* ... */ };
const TELUGU_MAP: Record<string, string> = { /* ... */ };
const TAMIL_MAP: Record<string, string> = { /* ... */ };

export function transliterateToLatin(input: string): string {
  // Detect script and apply appropriate map
  if (/[ऀ-ॿ]/.test(input)) return applyMap(input, DEVANAGARI_MAP);
  if (/[ঀ-৿]/.test(input)) return applyMap(input, BENGALI_MAP);
  if (/[ఀ-౿]/.test(input)) return applyMap(input, TELUGU_MAP);
  if (/[஀-௿]/.test(input)) return applyMap(input, TAMIL_MAP);
  return input; // already Latin
}

function applyMap(input: string, map: Record<string, string>): string {
```

```
  return input.split('').map(char => map[char] ?? char).join('');
}
```

Search Index Rebuild Strategy

Trigger	Action	Where
App first launch (after seed load)	Build full index from all medicines in DB	seed.worker.ts, background thread
Delta sync applied	Add/update/remove only changed medicine IDs	syncEngine.ts, calls worker ADD/REMOVE
App re-launch (subsequent)	Load serialized index from IndexedDB cache (skip rebuild)	searchEngine.ts init
User clears app data	Full index rebuild on next launch	SettingsPage → clearData()

Search Performance Targets

Scenario	Target Latency	Notes
Autocomplete (first 2+ chars)	< 30ms	Debounced at 150ms so actual perceived latency is 150ms + 30ms
Full search (50 results)	< 100ms	Including field-merge and score sorting
Index build (50,000 medicines)	< 5 seconds	In Web Worker, does not block UI
Index load from cache	< 500ms	IndexedDB read on app start
Transliteration per query	< 2ms	Synchronous, pure function

SECTION D | **MULTILINGUAL ARCHITECTURE: I18N SCHEMA & FALLBACK CHAIN**

Multilingual Architecture: i18n Schema & Fallback Chain

★ **NEW SECTION (Added for completeness)**
This section was missing. All UI text, all disclaimer text, and all medical term translations depend on this schema. Without it, developers will build conflicting translation key structures.

i18next Configuration

src/shared/i18n/index.ts

```
import i18n from 'i18next';
import { initReactI18next } from 'react-i18next';
import LanguageDetector from 'i18next-browser-languagedetector';

i18n
  .use(LanguageDetector)
  .use(initReactI18next)
  .init({
    // Fallback chain: if key missing in 'te', try 'hi', then 'en'
    fallbackLng: {
      'bn': ['hi', 'en'],  'te': ['hi', 'en'], 'ta': ['hi', 'en'],
      'mr': ['hi', 'en'],  'gu': ['hi', 'en'], 'ur': ['hi', 'en'],
      'kn': ['hi', 'en'],  'ml': ['hi', 'en'], 'or': ['hi', 'en'],
      'pa': ['hi', 'en'],  'as': ['hi', 'en'], 'hi': ['en'],
      'default': ['en'],
    },
    ns: ['common', 'medicine', 'disclaimer', 'symptom', 'error', 'glossary'],
    defaultNS: 'common',
    detection: {
      order: ['localStorage', 'navigator'],
      caches: ['localStorage'],
    },
    interpolation: { escapeValue: false }, // React handles XSS
    react: { useSuspense: true },
  });

export default i18n;
export const RTL_LANGUAGES = ['ur'];           // Right-to-left languages
export const SUPPORTED_LOCALES = [
  'en','hi','bn','te','ta','mr','gu','ur','kn','ml','or','pa','as'
];
```

Namespace Definitions

Namespace	File	Contents	Translation Type
common	common.json	Navigation labels, buttons, generic UI strings (Search, Clear, Cancel, Save, Settings, Home, etc.)	UI — machine translation acceptable after human review
medicine	medicine.json	Medicine-specific UI: dosage form names, therapeutic class names, frequency labels, manufacturer labels	UI + some medical — requires medical professional review
disclaimer	disclaimer.json	ALL disclaimer text verbatim. Symptom checker, pill ID, interaction, substitute, general safety	CRITICAL: requires professional medical translator
symptom	symptom.json	Symptom names, body system names, severity labels used in symptom checker UI	Medical — requires professional medical translator
error	error.json	User-facing error messages, offline notices, sync error messages	UI — machine translation acceptable after human review
glossary	glossary.json	Medical term translations: salt names, drug class names, ICD-10 disease names, organ system names	CRITICAL: requires licensed medical professional per language

Translation Key Schema — common.json (example)

src/shared/i18n/locales/en/common.json

```
{
  "nav": {
    "home": "Home",
    "search": "Search",
    "tools": "Tools",
    "browse": "Browse",
    "dashboard": "Dashboard",
    "settings": "Settings"
  },
  "search": {
    "placeholder": "Search medicine, generic, or salt...",
    "noResults": "No medicines found for \"{{query}}\"",
    "resultsCount": "{{count}} result",
    "resultsCount_plural": "{{count}} results",
    "searching": "Searching..."
  },
  "buttons": {
    "search": "Search",
    "clear": "Clear",
    "cancel": "Cancel",
    "save": "Save",
    "viewDetails": "View Details",
    "findSubstitutes": "Find Substitutes",
    "checkInteractions": "Check Interactions",
    "export": "Export"
  },
  "medicine": {
```

```
    "mrp": "MRP",
    "nlemPrice": "NLEM Ceiling Price",
    "savings": "You save {{percent}}%",
    "schedule": "Schedule",
    "manufacturer": "Manufacturer",
    "composition": "Composition",
    "sideEffects": "Side Effects",
    "interactions": "Drug Interactions",
    "substitutes": "Cheaper Substitutes"
  },
  "offline": {
    "banner": "You are offline. Data may not be up to date.",
    "syncPending": "{{count}} update pending sync"
  }
}
```

Translation Key Schema — disclaimer.json (example)

src/shared/i18n/locales/en/disclaimer.json

```
{
  "standard": {
    "title": "Important Information",
    "body": "This information is for educational purposes only. Always consult a
qualified healthcare professional before making any medical decisions."
  },
  "diagnostic": {
    "title": "Not a Medical Diagnosis",
    "body": "This symptom checker provides informational suggestions only. It is NOT a
medical diagnosis. Results are based on statistical probabilities and may not apply to
your situation. Always consult a qualified doctor."
  },
  "interaction": {
    "title": "Drug Interaction Information",
    "body": "Interaction data is for informational purposes only. Do not start, stop,
or change medications without first consulting your doctor or pharmacist."
  },
  "substitute": {
    "title": "Substitute Medicine Information",
    "body": "Substitutes shown contain the same active ingredients at the same
strength. Always confirm with your pharmacist or doctor before switching medicines. Do
not switch without medical advice."
  },
  "pillId": {
    "title": "Pill Identification — Not Definitive",
    "body": "AI-based pill identification is an aid only. Results may be incorrect.
Never take any medicine based solely on this identification. Always verify with a
pharmacist."
  }
}
```

RTL Support for Urdu

src/app/providers/I18nProvider.tsx — RTL switching

```
import { useEffect } from 'react';
import { useLanguageStore } from '@/shared/stores/languageStore';
import { RTL_LANGUAGES } from '@/shared/i18n';

export function I18nProvider({ children }: { children: React.ReactNode }) {
  const { locale } = useLanguageStore();

  useEffect(() => {
    const isRTL = RTL_LANGUAGES.includes(locale);
    document.documentElement.dir = isRTL ? 'rtl' : 'ltr';
    document.documentElement.lang = locale;
    // Tailwind RTL: add 'rtl' class so rtl: variants activate
    document.documentElement.classList.toggle('rtl', isRTL);
  }, [locale]);

  return <>{children}</>;
}
```

Indic Font Loading Strategy

index.html — font loading for 13 scripts

```
<!-- Load Noto Sans for Indic scripts only when needed -->
<!-- Base fonts loaded eagerly -->
<link rel="preconnect" href="https://fonts.googleapis.com">
<link href="https://fonts.googleapis.com/css2?
family=Inter:wght@400;500;600&display=swap" rel="stylesheet">

<!-- Indic fonts loaded lazily per language selection -->
<!-- Loaded via JS when user selects a language: -->
<!--
  Hindi/Marathi:  Noto Sans Devanagari
  Bengali/Assamese: Noto Sans Bengali
  Telugu:         Noto Sans Telugu
  Tamil:          Noto Sans Tamil
  Gujarati:       Noto Sans Gujarati
  Urdu:           Noto Nastaliq Urdu
  Kannada:        Noto Sans Kannada
  Malayalam:      Noto Sans Malayalam
  Odia:           Noto Sans Oriya
  Punjabi:        Noto Sans Gurmukhi
-->

// In languageStore.ts — load font when language changes:
async function loadFontForLocale(locale: string) {
  const fontMap: Record<string, string> = {
    hi: 'Noto+Sans+Devanagari',
    mr: 'Noto+Sans+Devanagari',
    bn: 'Noto+Sans+Bengali',
    as: 'Noto+Sans+Bengali',
    te: 'Noto+Sans+Telugu',
    // ... etc
```

```
  };
  const font = fontMap[locale];
  if (!font) return;
  const link = document.createElement('link');
  link.href = `https://fonts.googleapis.com/css2?family=${font}:wght@400;500&display=swap`;
  link.rel = 'stylesheet';
  document.head.appendChild(link);
}
```

Plural Forms by Language

Language	Plural Rule	i18next Key Pattern
English	singular / plural	"key": "{{count}} result", "key_plural": "{{count}} results"
Hindi	singular / plural (same as English)	"key": "{{count}} परिणाम", "key_plural": "{{count}} परिणाम"
Bengali	singular / plural	"key": "{{count}} ফলাফল", "key_plural": "{{count}} ফলাফল"
Tamil	singular / plural	"key": "{{count}} முடிவு", "key_plural": "{{count}} முடிவுகள்"
Urdu (RTL)	singular / plural	"key": "{{count}} نتیجہ", "key_plural": "{{count}} نتائج"

SECTION E | **DATABASE: INDEXES, MIGRATIONS & USER DATA SCHEMA**

Database: Indexes, Migrations & User Data Schema

★ **NEW SECTION (Added for completeness)**
This section was missing. Without index definitions, queries on 50,000+ records will be slow. Without a migration strategy, releasing a schema change will corrupt existing user databases. Without the user data schema, the Settings "Clear my data" DPDPA flow cannot be built.

Dexie.js Index Definitions (Web / IndexedDB)

src/shared/services/database.ts — complete schema

```
import Dexie, { Table } from 'dexie';

export class PharmaIntelDB extends Dexie {
  medicines!: Table<Medicine>;
  interactions!: Table<Interaction>;
  sideEffects!: Table<SideEffect>;
  pillImages!: Table<PillImage>;
  diseaseMappings!: Table<DiseaseMapping>;
  graphNodes!: Table<GraphNode>;
  graphEdges!: Table<GraphEdge>;
  syncMeta!: Table<{ key: string; value: unknown }>;
  // USER DATA TABLES (see Section below)
  userPreferences!: Table<UserPreferences>;
  recentSearches!: Table<RecentSearch>;
  savedMedicines!: Table<SavedMedicine>;
  medicationReminders!: Table<MedicationReminder>;

  constructor() {
    super('PharmaIntelDB');

    this.version(1).stores({
      // FORMAT: 'primaryKey, index1, index2, [compound+index]'
      // & = unique index,  ++ = auto-increment,  * = multi-entry (array field)
      medicines:         '&id, brandName, genericName, schedule, manufacturer,
therapeuticClass, nlemListed, [genericName+schedule]',
      interactions:      '&id, drugAId, drugBId, severity, [drugAId+drugBId]',
      sideEffects:       '&id, medicineId, organSystem, [medicineId+organSystem]',
      pillImages:        '&id, medicineId, shape, color',
      diseaseMappings:   '&id, icd10, disease',
      graphNodes:        '&id, type, label',
      graphEdges:        '&id, fromId, toId, edgeType, [fromId+edgeType]',
      syncMeta:          '&key',
      // User data tables
      userPreferences:   '&id',              // single row, id='default'
      recentSearches:    '&id, query, searchedAt',
      savedMedicines:    '&id, medicineId, savedAt',
      medicationReminders: '&id, medicineId, nextReminderAt',
    });
```

```
    // Future schema version example:
    // this.version(2).stores({
    //   medicines: '&id, brandName, ..., atcCode',  // added atcCode index
    // }).upgrade(async tx => {
    //   // backfill new field for existing records
    //   await tx.table('medicines').toCollection().modify(m => { m.atcCode = m.atcCode
?? ''; });
    // });
  }
}

export const db = new PharmaIntelDB();
```

Index Design Rationale

Index	Table	Why It Exists
brandName	medicines	FlexSearch still needs Dexie for filtered lookups by name prefix when index is warming up
genericName	medicines	Substitute finder queries equalsIgnoreCase(genericName) — must be indexed for sub-10ms response
[genericName+schedule]	medicines	Compound: "find all OTC paracetamol generics" — common filter combo
[drugAId+drugBId]	interactions	Bidirectional lookup: find interaction between two specific drugs in one query
[medicineId+organSystem]	sideEffects	Group side effects by organ system for the side effects panel
[fromId+edgeType]	graphEdges	Graph traversal: "all edges of type TREATS from this medicine node"
searchedAt	recentSearches	Sort recent searches by time; prune oldest when over 50 entries
nextReminderAt	medicationReminders	Background task queries: "find all reminders due in next 24 hours"

Migration Strategy

Dexie.js handles migrations via versioned schema upgrades. The rules below prevent data loss when a new app version ships a schema change.

Migration rules (enforced in code review)

```
RULE 1: Never modify version(1).stores() after shipping to production.
        Always add a new version(N+1) block.

RULE 2: version(N+1).upgrade() must handle BOTH new installs and existing data.
```

```
         Existing users have the old schema; new users skip straight to latest.

RULE 3: Adding a new indexed field:
         → Add to stores() string, then backfill with upgrade() if needed.

RULE 4: Removing a field (deprecation):
         → Keep the column for 2 versions (set to null), remove in version N+2.

RULE 5: Renaming a field:
         → Add new field + migrate data in version N+1.
         → Remove old field in version N+2.

RULE 6: Test upgrades with a real v1 IndexedDB dump before shipping.
         Use: scripts/test-migration.ts to replay migrations on real data.

// Example: version 2 adds atcCode to medicines
this.version(2).stores({
  medicines: '&id, brandName, genericName, schedule, manufacturer, therapeuticClass,
nlemListed, atcCode, [genericName+schedule]',
  // ... all other tables repeated exactly
}).upgrade(async tx => {
  await tx.table('medicines').toCollection().modify(med => {
    if (!med.atcCode) med.atcCode = '';
  });
});
```

User Data Schema

User data TypeScript interfaces

```
// Stored in IndexedDB table: userPreferences (single row, id='default')
interface UserPreferences {
  id: 'default';           // always 'default' — only one row ever
  locale: string;          // 'hi', 'en', 'bn', etc.
  theme: 'light' | 'dark' | 'system';
  textSizeScale: 1 | 1.25 | 1.5;    // for accessibility
  onboardingComplete: boolean;
  dataConsentGiven: boolean;         // DPDPA consent flag
  consentGivenAt: string | null;     // ISO8601
}

// Stored in IndexedDB table: recentSearches (max 50 entries)
interface RecentSearch {
  id: string;              // UUID
  query: string;           // raw search string
  resultCount: number;     // how many results were found
  searchedAt: string;      // ISO8601 — used for sorting and pruning
}

// Stored in IndexedDB table: savedMedicines
interface SavedMedicine {
  id: string;              // UUID
  medicineId: string;      // FK → medicines.id
  medicineSnapshot: {      // stored so it shows even if DB changes
```

```
    brandName: string;
    genericName: string;
    mrp: number;
    schedule: string;
  };
  note: string;            // user's optional note
  savedAt: string;         // ISO8601
}

// Stored in IndexedDB table: medicationReminders (Phase 2 feature)
interface MedicationReminder {
  id: string;
  medicineId: string;
  medicineName: string;    // snapshot
  dosage: string;          // user-entered: "1 tablet"
  frequency: 'once' | 'twice' | 'three_times' | 'custom';
  timesPerDay: number;
  reminderTimes: string[]; // ["08:00", "20:00"] — 24h format
  nextReminderAt: string;  // ISO8601 — indexed for background query
  active: boolean;
  createdAt: string;
}
```

DPDPA Data Deletion Flow

The "Clear My Data" button in Settings must delete ALL user data in compliance with India's Digital Personal Data Protection Act 2023.

src/features/settings/lib/clearUserData.ts

```
export async function clearAllUserData(): Promise<void> {
  // Step 1: Delete all user-generated data from IndexedDB
  await db.userPreferences.clear();
  await db.recentSearches.clear();
  await db.savedMedicines.clear();
  await db.medicationReminders.clear();

  // Step 2: Clear localStorage (theme, language, onboarding flags)
  const KEEP_KEYS: string[] = [];  // nothing survives
  Object.keys(localStorage)
    .filter(k => !KEEP_KEYS.includes(k))
    .forEach(k => localStorage.removeItem(k));

  // Step 3: Re-initialize default preferences (blank slate)
  await db.userPreferences.put({
    id: 'default',
    locale: 'en',
    theme: 'system',
    textSizeScale: 1,
    onboardingComplete: false,
    dataConsentGiven: false,
    consentGivenAt: null,
  });

  // Step 4: Log deletion event (local only — no server call)
```

```
  console.info('[DPDPA] User data cleared at', new Date().toISOString());

  // NOTE: The medicine/interaction/sideEffects DB is NOT cleared here.
  // That data is public reference data, not personal data.
  // User can optionally clear it via "Reset App Data" (separate, destructive action).
}
```

localStorage Key Registry

All localStorage keys used by the app must be registered here. No component should use arbitrary keys.

Key	Value Type	Purpose	Cleared on DPDPA?
pi_theme	string	Active theme (light/dark/system)	Yes
pi_locale	string	Active language code	Yes
pi_onboarding	boolean	Has user completed onboarding?	Yes
pi_consent	boolean	DPDPA data consent given?	Yes
pi_consent_date	string (ISO8601)	When consent was given	Yes
i18nextLng	string	i18next detected language (auto-set)	Yes
pi_search_history_v2	DEPRECATED — moved to IndexedDB	—	N/A

SECTION F | **ONNX MODEL: TRAINING PIPELINE & UPDATE STRATEGY**

ONNX Model: Training Pipeline & Update Strategy

★ **NEW SECTION (Added for completeness)**
This section was missing. The blueprint specified "fine-tune MobileNetV3-Small" but gave no guidance on how. A developer cannot build the Pill Identifier feature without knowing how the model is trained, exported, and updated.

Model Training Overview

Attribute	Specification
Base model	MobileNetV3-Small (pre-trained on ImageNet via torchvision)
Fine-tuning approach	Transfer learning: freeze all layers except the final classifier head. Replace head with custom 256-class (or N-class) head.
Training framework	PyTorch 2.x + torchvision
Export format	ONNX opset 17, INT8 quantized via torch.quantization.quantize_dynamic
Target runtime	ONNX Runtime Web 1.x with WASM backend
Input shape	[1, 3, 224, 224] — batch=1, RGB, 224x224
Output shape	[1, N] — softmax probabilities for N pill classes
Training environment	Google Colab Pro or local GPU. Python 3.11, CUDA 12.x

Training Dataset Structure

scripts/ml/dataset/README — how to organize pill images

```
pill-images-dataset/
├── train/
│   ├── paracetamol_500mg_round_white/     ← folder name = class label
│   │   ├── img_001.jpg
│   │   ├── img_002.jpg
│   │   └── ...  (minimum 50 images per class)
│   ├── amoxicillin_500mg_capsule_red_yellow/
│   │   └── ...
│   └── ... (one folder per pill class)
├── val/
│   ├── paracetamol_500mg_round_white/     ← 20% of images held out
│   │   └── ...
│   └── ...
├── class_map.json                         ← Maps folder name → medicine ID in DB
└── metadata.csv                           ← img_path, class_label, medicine_id, source

# class_map.json format:
```

```
{
  "paracetamol_500mg_round_white": "med_uuid_001",
  "amoxicillin_500mg_capsule_red_yellow": "med_uuid_002"
}

# Minimum requirements:
# - At least 50 training images per class
# - At least 10 validation images per class
# - Images: JPEG, minimum 640x480, neutral background, good lighting
# - Both sides of tablet photographed (obverse + reverse)
# - Multiple lighting conditions per pill
```

Training Script

scripts/ml/train.py — PyTorch fine-tuning

```
import torch
import torchvision
from torchvision import datasets, transforms, models
from torch import nn, optim
import json, os

# Config
DATASET_DIR = './pill-images-dataset'
MODEL_OUT   = './models/pill_classifier.pth'
ONNX_OUT    = './models/pill_classifier.onnx'
NUM_CLASSES = len(os.listdir(f'{DATASET_DIR}/train'))
EPOCHS      = 20
LR          = 0.001
BATCH_SIZE  = 32

# Data transforms
transform_train = transforms.Compose([
    transforms.Resize((256, 256)),
    transforms.RandomCrop(224),
    transforms.RandomHorizontalFlip(),
    transforms.ColorJitter(brightness=0.2, contrast=0.2),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]),
])
transform_val = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]),
])

train_ds = datasets.ImageFolder(f'{DATASET_DIR}/train', transform=transform_train)
val_ds   = datasets.ImageFolder(f'{DATASET_DIR}/val',   transform=transform_val)
train_dl = torch.utils.data.DataLoader(train_ds, batch_size=BATCH_SIZE, shuffle=True)
val_dl   = torch.utils.data.DataLoader(val_ds,   batch_size=BATCH_SIZE)

# Model: MobileNetV3-Small, replace classifier
model = models.mobilenet_v3_small(weights='IMAGENET1K_V1')
for param in model.parameters(): param.requires_grad = False  # freeze base
model.classifier[3] = nn.Linear(model.classifier[3].in_features, NUM_CLASSES)
```

```
model = model.cuda()

optimizer = optim.Adam(model.classifier.parameters(), lr=LR)
criterion = nn.CrossEntropyLoss()

# Training loop
for epoch in range(EPOCHS):
    model.train()
    for imgs, labels in train_dl:
        imgs, labels = imgs.cuda(), labels.cuda()
        optimizer.zero_grad()
        loss = criterion(model(imgs), labels)
        loss.backward()
        optimizer.step()
    print(f'Epoch {epoch+1}/{EPOCHS}')

torch.save(model.state_dict(), MODEL_OUT)
print(f'Model saved: {MODEL_OUT}')

# Export to ONNX
model.cpu().eval()
dummy = torch.randn(1, 3, 224, 224)
torch.onnx.export(model, dummy, ONNX_OUT,
    opset_version=17,
    input_names=['input'], output_names=['output'],
    dynamic_axes={'input': {0: 'batch'}, 'output': {0: 'batch'}})
print(f'ONNX exported: {ONNX_OUT}')

# Save class map (folder index → medicine ID mapping)
idx_to_class = {v: k for k, v in train_ds.class_to_idx.items()}
with open('./models/class_index.json', 'w') as f:
    json.dump(idx_to_class, f)
print('Class index saved')
```

INT8 Quantization (reduces 8MB → ~2MB)

scripts/ml/quantize.py

```
from onnxruntime.quantization import quantize_dynamic, QuantType

quantize_dynamic(
    model_input='./models/pill_classifier.onnx',
    model_output='./models/pill_classifier_int8.onnx',
    weight_type=QuantType.QInt8,
)
print('Quantized model saved: pill_classifier_int8.onnx')

# Validate quantized model accuracy on val set
# Run: python validate_onnx.py --model ./models/pill_classifier_int8.onnx
# Target: top-5 accuracy >= 85% on validation set
```

Model Hosting & Update Strategy

Concern	Specification
Model hosting	Stored on Cloudflare R2 / AWS S3. URL format: https://cdn.pharmaintel.in/models/v{N}/pill_classifier_int8.onnx
Model versioning	Semantic version in URL path (v1, v2, v3). App checks model version in sync/check endpoint response.
First download trigger	Model is NOT bundled in the app. Downloaded the first time user navigates to Pill Identifier page.
Caching	Model binary cached in IndexedDB after first download. Key: "onnx_model_v{N}". Survives app restarts.
Update trigger	sync/check response includes modelVersion field. If serverModelVersion > localModelVersion, download new model on next Pill Identifier open.
Fallback if no model	Show rule-based pill search (shape + color + imprint) with a notice: "AI identification unavailable — use manual search"
Model file size	~8MB full precision ONNX, ~2MB INT8 quantized. Ship INT8 version.

Fallback: Rule-Based Pill Matching

When the ONNX model is unavailable (no internet on first load, model download failed, or device too slow), fall back to rule-based attribute matching.

src/features/pill-identifier/lib/ruleBasedMatcher.ts

```
export interface PillAttributes {
  shape: PillShape;
  color: PillColor;
  imprint?: string;     // optional — many Indian pills have no imprint
  scoringLine: boolean;
  sizeMm?: number;
}

export type PillShape =
  | 'round' | 'oval' | 'oblong' | 'capsule' | 'triangle'
  | 'square' | 'diamond' | 'pentagon' | 'hexagon';

export type PillColor =
  | 'white' | 'off_white' | 'yellow' | 'orange' | 'red'
  | 'pink' | 'brown' | 'green' | 'blue' | 'purple'
  | 'grey' | 'black' | 'transparent' | 'multicolor';

export async function ruleBasedMatch(attrs: PillAttributes): Promise<Medicine[]> {
  // Query IndexedDB: find pills matching shape + color (imprint optional)
  let query = db.pillImages.where('shape').equals(attrs.shape);
  const candidates = await query.toArray();

  return candidates
    .filter(p => {
      const colorMatch = p.color === attrs.color;
      const imprintMatch = !attrs.imprint || p.imprint.includes(attrs.imprint);
      const scoreMatch = p.scoring === attrs.scoringLine;
```

```
      return colorMatch && imprintMatch && scoreMatch;
    })
    .map(p => p.medicineId)  // then load medicine details
    .slice(0, 10);
}
```

SECTION G | **ERROR HANDLING, RECOVERY & FALLBACK SPECIFICATION**

Error Handling, Recovery & Fallback Specification

★ **NEW SECTION (Added for completeness)**
This section was missing. Without it, every developer handles errors differently, leading to inconsistent UX — some errors crash the app, others silently fail, and users get no feedback.

Global Error Boundary

src/shared/ui/ErrorBoundary.tsx

```
import { Component, ErrorInfo, ReactNode } from 'react';

interface State { hasError: boolean; error: Error | null; }

export class ErrorBoundary extends Component<{ children: ReactNode; fallback?:
ReactNode }, State> {
  state: State = { hasError: false, error: null };

  static getDerivedStateFromError(error: Error): State {
    return { hasError: true, error };
  }

  componentDidCatch(error: Error, info: ErrorInfo) {
    // Log to Sentry (non-PII only — no medicine names or search queries in error
context)
    console.error('[ErrorBoundary]', error.message, info.componentStack);
  }

  render() {
    if (this.state.hasError) {
      return this.props.fallback ?? (
        <div className="flex flex-col items-center justify-center min-h-screen p-8
text-center">
          <h2 className="text-xl font-semibold text-gray-800 mb-3">Something went
wrong</h2>
          <p className="text-gray-500 mb-6">Please restart the app. If the problem
persists, try clearing app data in Settings.</p>
          <button onClick={() => window.location.reload()}
            className="px-6 py-2 bg-blue-600 text-white rounded-lg">
            Restart App
          </button>
        </div>
      );
    }
    return this.props.children;
  }
}

// PLACEMENT in component tree:
```

```
// <ErrorBoundary>                    ← wraps entire App (catches everything)
//   <AppShell>
//     <ErrorBoundary fallback={<FeatureError />}> ← wraps each feature route
//       <MedicineDetailPage />
//     </ErrorBoundary>
//   </AppShell>
// </ErrorBoundary>
```

Database Error Recovery

src/shared/services/database.ts — recovery flow

```
// Called once on app start
export async function initDatabase(): Promise<void> {
  try {
    await db.open();
  } catch (err) {
    if (err.name === 'DatabaseClosedError' || err.name === 'InvalidStateError') {
      // IndexedDB is corrupted — wipe and rebuild
      console.warn('[DB] Corruption detected, rebuilding...');
      await Dexie.delete('PharmaIntelDB');
      await db.open();
      // Trigger reseed in background
      triggerReseed();
    } else {
      throw err;  // Unknown error — propagate to ErrorBoundary
    }
  }
}

function triggerReseed() {
  // Show one-time "Rebuilding database..." progress bar
  // Run seed.worker.ts to reload from bundled seed file
  const worker = new Worker(new URL('./workers/seed.worker.ts', import.meta.url));
  worker.postMessage({ type: 'SEED', seedUrl: import.meta.env.VITE_SEED_URL });
}
```

Sync Error Handling & Retry Logic

src/shared/services/syncEngine.ts — retry with backoff

```
const RETRY_DELAYS = [2000, 4000, 8000]; // ms — exponential backoff

export async function syncWithRetry(): Promise<void> {
  for (let attempt = 0; attempt < RETRY_DELAYS.length; attempt++) {
    try {
      await syncDelta();
      return; // success
    } catch (err) {
      const isLast = attempt === RETRY_DELAYS.length - 1;
      if (isLast) {
```

```
        // All retries failed — mark sync as failed, show non-blocking toast
        syncStore.setState({ syncStatus: 'error', lastError: err.message });
        toast.warning('Sync failed. Data may be slightly out of date.');
        return;
      }
      await sleep(RETRY_DELAYS[attempt]);
    }
  }
}

// Network status check before sync
export function startBackgroundSync() {
  window.addEventListener('online', () => syncWithRetry());
  setInterval(() => {
    if (navigator.onLine) syncWithRetry();
  }, 15 * 60 * 1000); // every 15 minutes when online
}
```

Seed Load Failure Recovery

src/shared/services/workers/seed.worker.ts — failure handling

```
self.onmessage = async (e) => {
  if (e.data.type !== 'SEED') return;
  try {
    const response = await fetch(e.data.seedUrl);
    if (!response.ok) throw new Error(`HTTP ${response.status}`);

    // Stream-decompress the gzipped seed
    const ds = new DecompressionStream('gzip');
    const stream = response.body!.pipeThrough(ds);
    const text = await new Response(stream).text();
    const records: Medicine[] = JSON.parse(text);

    // Load in priority batches (common medicines first)
    const BATCH_SIZE = 1000;
    for (let i = 0; i < records.length; i += BATCH_SIZE) {
      const batch = records.slice(i, i + BATCH_SIZE);
      await db.medicines.bulkPut(batch);
      self.postMessage({
        type: 'SEED_PROGRESS',
        loaded: Math.min(i + BATCH_SIZE, records.length),
        total: records.length,
      });
    }
    // Save progress checkpoint — if interrupted, resume from here next launch
    await db.syncMeta.put({ key: 'seedLoadedCount', value: records.length });
    self.postMessage({ type: 'SEED_DONE', count: records.length });
  } catch (err) {
    self.postMessage({ type: 'SEED_ERROR', error: err.message });
    // Worker will retry on next app launch automatically
    // App is still usable — search just returns no results until seed completes
  }
};
```

Feature-Level Fallback Map

Feature	Primary Failure	Fallback Behavior	User Message
Medicine Search	FlexSearch index not built yet	Show "Database loading..." progress, disable search input	"Setting up your medicine database..."
Substitute Finder	No substitutes in DB	Show empty state with explanation	"No cheaper substitutes found for this medicine."
Drug Interaction	Interaction data missing for a drug pair	Show partial results + disclaimer	"No interaction data available for this combination. Consult your pharmacist."
Symptom Checker	Bayesian model JSON missing	Disable feature, show maintenance notice	"Symptom checker temporarily unavailable."
Pill Identifier — AI	ONNX model not downloaded or failed	Switch to rule-based attribute search	"AI identification unavailable. Use manual pill search below."
Pill Identifier — Rule	No pill images in DB	Disable entire Pill Identifier	"Pill database not available. Update the app to enable this feature."
Market Dashboard	No report uploaded	Show demo data with banner	"Showing sample data. Upload your report to see real analytics."
Knowledge Graph	Graph data not loaded	Show spinner, then empty state	"Knowledge graph is loading..."
Delta Sync	API unreachable after 3 retries	Continue offline, show sync warning	"Could not sync. Using cached data."

⚠ **WARNING**

NEVER show raw error messages (stack traces, SQL errors, JSON parse errors) to users. All errors must be translated to human-readable messages using the error namespace in i18n. Log raw errors to console.error() only.

SECTION ✓ | **PRE-DEVELOPMENT CHECKLIST — EVERYTHING YOU NEED TO START**

Pre-Development Checklist

Before writing the first line of code, verify all items below are in place. This checklist ensures zero blockers during development.

Accounts & Access

- GitHub repository created (pharmaintel-india or similar)
- Google Play Developer Account ($25 one-time) — needed for Phase 7
- Apple Developer Account ($99/year) — needed for Phase 7 iOS
- Cloudflare account (free tier sufficient for Pages + R2)
- Sentry account for error monitoring

Development Environment

- Node.js 22.x LTS installed
- Python 3.11+ installed (for ML training in Phase 4)
- VS Code + extensions: ESLint, Prettier, Tailwind CSS IntelliSense, TypeScript
- Git configured with SSH key

Data (Minimum for Phase 1-2 Development)

- At least 1,000 medicine records in the correct JSON schema (enough to test search)
- NLEM 2022 price list (384 medicines — available from official gazette)
- Basic drug interaction dataset for testing (50-100 pairs minimum)
- English and Hindi translation files seeded with at least common namespace keys

Architecture Decisions Resolved

- Cloud backend: build now or offline-only first? (Answer before Phase 1)
- User accounts: yes or no in v1? (Answer before Phase 1 — affects DB schema)
- Priority platform confirmed (Recommendation: Web PWA first)

Legal & Compliance

- Privacy Policy drafted (required before App Store submission)
- Terms of Service drafted
- Medical disclaimer text reviewed by a qualified healthcare professional
- DPDPA compliance checklist reviewed with legal counsel (if budget allows)

✓ NOTE

Once all items above are checked, you are ready to begin Phase 1. The blueprint in this document covers everything from first commit to production. No additional documentation is needed.