PotatoGuard

Potato Disorder Image Diagnosis Application

ジャガイモ生育障害 画像診断アプリ　開発仕様書

ドキュメント種別	開発仕様書 (Development Specification)
バージョン	v1.0 — Draft
作成日	2026 年 3 月
対象プラットフォーム	iOS 16+ / Android 10+ (React Native)
開発環境	macOS 15.5 / VSCode + Claude / GitHub
対応言語	英語（基本）・日本語・ネパール語 他
通信要件	オフライン優先設計（低帯域環境対応）

目次 / Table of Contents

01 Project Overview プロジェクト概要

1.1 目的と背景

PotatoGuard は、ジャガイモ農家（特にネパール・日本・南米・欧州などの遠隔地農村）が、スマートフォンで撮影した写真をもとにジャガイモの生育障害を診断し、適切な対策を即座に得られる意思決定支援ツールです。

低い通信環境でも動作するオフライン優先設計とし、Web 翻訳 API を一切使わず静的翻訳バンドルで多言語対応します。診断結果はローカルに保存・PDF 出力でき、後日クラウド同期も可能にします。

1.2 対象ユーザー

ユーザー層	詳細
一次ユーザー（農家）	ジャガイモ生産農家（ネパール・日本・欧州・南米）
二次ユーザー（普及員）	農業普及員、農協担当者、NGO フィールドスタッフ
三次ユーザー（研究者）	植物病理専門家、DB 管理者、研究機関

1.3 設計原則

原則	内容
Offline-First（オフライン優先）	コア機能はすべてネット不要。AI モデル・DB・翻訳を端末内に保持
Low-Bandwidth（低帯域対応）	画像は WebP 圧縮。初回ダウンロード後は差分更新のみ（数 MB）
No Runtime Translation	Web 翻訳 API 未使用。静的 i18n JSON バンドルで完全オフライン翻訳
Multi-Image Diagnosis	1〜5 枚の写真を入力し複数アングルのスコアを統合して精度向上
Decision Support Only	あくまで農家の判断を補完する提案ツール。最終判断は農家が行う
Extensible DB	障害 DB は差分更新パッケージで追加・修正可能な設計
Privacy by Default	写真のクラウドアップロードは明示的オプトインのみ。アカウント不要

02 System Architecture システムアーキテクチャ

2.1 全体構成（4 層アーキテクチャ）

レイヤー	担当	主要技術
UI Layer	画面表示・カメラ・ナビゲーション	React Native + Expo SDK 52 (TypeScript)
AI Engine	オンデバイス画像推論	TensorFlow Lite（EfficientNet-V2-S、INT8 量子化）
Local DB	障害 DB・診断履歴・翻訳	SQLite（expo-sqlite v14）+ WebP 画像ファイル
Sync Layer	DB 更新・クラウドバックアップ（任意）	REST API + S3 互換ストレージ / Firebase（オプション）

2.2 技術スタック一覧

フロントエンド

- React Native 0.74 + Expo SDK 52（TypeScript strict モード）
- Expo Router v3（ファイルベースナビゲーション）
- React Native Paper v5（Material Design UI コンポーネント）
- React Native MMKV（高速 KV ストレージ：設定・言語選択保持）

AI・機械学習

- TensorFlow Lite Runtime（react-native-fast-tflite）
- モデル：EfficientNet-V2-S を PlantVillage + PotatoMD で Fine-tune
- INT8 後処理量子化によりモデルサイズ < 15 MB を目標
- 推論時間：中位 Android (Snapdragon 665) で < 3 秒

データベース・ストレージ

- SQLite（expo-sqlite）：障害マスタ・診断履歴・翻訳テキスト
- expo-file-system：参照画像（WebP）・診断写真の管理
- 差分更新パッケージ：.zip + SHA-256 チェックサム検証

レポート・共有

- react-native-html-to-pdf：完全オフライン PDF 生成
- expo-sharing：OS ネイティブ共有シート（WhatsApp・SMS・メール等）

開発環境・CI/CD

- macOS 15.5 / VSCode + Claude Extension / GitHub monorepo
- GitHub Actions：lint → test → EAS Build → App Store / Play Store
- EAS Build（Expo Application Services）：クラウドビルド

03 Disorder Database Design 障害データベース設計

3.1 カバー範囲

カテゴリ	障害例	目標件数
栄養欠乏（常量）	N・P・K・Ca・Mg・S 欠乏	10〜15
微量要素欠乏	Fe・Zn・Mn・B・Cu・Mo 欠乏	10〜12
菌類病害	疫病(P. infestans)・早期疫病・黒あざ病・粉状そうか病・バーティシリウム萎凋病	18〜25
細菌性病害	そうか病・黒あし病・輪腐病・軟腐病・青枯病(Ralstonia)	8〜12
ウィルス病	PVY・PLRV・PVX・PVS・PMTV（モップトップ）・ファイトプラズマ	8〜12
害虫・食害	コロラドハムシ・ハリガネムシ・アブラムシ・ポテトガ・シストセンチュウ(PCN/RKN)	10〜14
非生物的ストレス	干ばつ・過湿・霜害・高温障害・日焼け・雹害	8〜12
生理障害	空洞芋・内部褐変・緑化・裂開・くず芋	8〜10

初期ターゲット：80 障害・320 枚以上の参照画像（各障害に最低 4 枚）。データ出典：EPPO DB、FAO 害虫シート、PlantVillage（CC ライセンス）、EPPO 画像ギャラリー、各国植物防疫機関。

3.2 SQLite スキーマ設計

テーブル構成

テーブル名	役割
disorders	障害マスタ（名称・症状・原因・対策・農薬・地域等）
disorder_images	参照画像（WebP blob または外部ファイルパス）
pesticides	農薬マスタ（有効成分・用量・使用時期・国別登録情報）
diagnosis_sessions	診断履歴（撮影写真・推論結果・農家メモ等）
translations	UI 翻訳文字列テーブル（i18n JSON の DB 版）
db_meta	DB バージョン・更新履歴・チェックサム管理

disorders テーブル（主要カラム）

カラム名	型	説明
id	INTEGER PK	オートインクリメント主キー
uuid	TEXT UNIQUE	同期用安定 ID（例：dis_lateblight_001）
name_en / name_ja / name_ne	TEXT	英語・日本語・ネパール語名称

category	TEXT	fungal | bacterial | viral | pest | nutrient | abiotic | physiological
pathogen_name	TEXT	病原体・欠乏元素の学名
symptoms_en/ja/ne	TEXT	症状説明（各言語）
causes_en/ja/ne	TEXT	原因分析（各言語）
treatment_cultural	TEXT (JSON)	営農的対処法リスト
pesticide_ids	TEXT (JSON)	pesticides テーブルへの参照 ID 配列
regions	TEXT (JSON)	発生多発地域リスト（国コード）
severity	TEXT	low | medium | high | critical
growth_stage	TEXT (JSON)	発生しやすい生育ステージ
image_ids	TEXT (JSON)	disorder_images への参照 ID 配列
confidence_threshold	REAL	この障害を表示する最低信頼度（例：0.35）
updated_at	TEXT	ISO 8601 タイムスタンプ
db_version	INTEGER	レコード作成時の DB スキーマバージョン

diagnosis_sessions テーブル（主要カラム）

カラム名	型	説明
id	INTEGER PK	オートインクリメント
session_uuid	TEXT UNIQUE	クライアント生成 UUID（クラウド同期用）
created_at	TEXT	ISO 8601
location_lat / lng	REAL	GPS 座標（任意）
field_notes	TEXT	農家入力メモ（自由記述）
photo_paths	TEXT (JSON)	撮影写真のローカルパス配列
top_result_uuid	TEXT	最上位診断結果の disorder UUID
all_results	TEXT (JSON)	[{disorder_uuid, confidence, rank}, ...]
action_taken	TEXT	農家が取った対処（後日記録可）
report_path	TEXT	生成された PDF のローカルパス
synced_at	TEXT	クラウド同期日時（NULL=未同期）

04 AI Diagnosis Engine AI 診断エンジン

4.1 モデル仕様

パラメータ	仕様
ベースモデル	EfficientNet-V2-S（ImageNet 事前学習済み）
ファインチューニングデータ	PotatoMD データセット + PlantVillage + 現地圃場収集画像
クラス数	～90 クラス（拡張可能）
入力解像度	224 × 224 px、RGB
量子化方式	INT8 後処理量子化（Post-Training Quantization）
モデルファイルサイズ	目標 < 15 MB
推論レイテンシ	中位 Android (Snapdragon 665) で < 3 秒
出力形式	Softmax 確率値（Top-3 クラス + 信頼度スコア）
多画像統合	最大 5 枚の信頼度スコアを Average Pooling で統合

4.2 診断フロー（処理ステップ）

1. 農家がカメラ撮影または端末ギャラリーから 1～5 枚の写真を選択
1. アプリがぼけ検出・解像度チェック・クロップ補助を実施
2. 各画像を 224×224 にリサイズ → TFLite モデルで推論
3. 各画像の Softmax 出力を平均プーリングで統合
4. 信頼度閾値以上の障害 Top-3 を SQLite から取得
5. 結果画面へ遷移（症状・原因・対策・参照画像を表示）
6. 診断セッションを自動保存（diagnosis_sessions テーブル）

4.3 フォールバック設計

推論の信頼度が全クラスで閾値（デフォルト 0.35）を下回る場合、「確定診断なし」と表示し、症状カテゴリ別のチェックリスト（キー症状の有無を農家が入力）によるルールベース絞り込みへ自動フォールバックします。これにより、モデルが苦手なケース（画像品質不良・珍しい障害）でも有用な提案が可能です。

4.4 モデル学習・更新パイプライン

- 学習環境：Python 3.11 + TensorFlow 2.16 + Google Colab Pro / ローカル GPU
- アノテーション：Label Studio（バウンディングボックス + クラスラベル）
- 量子化 + エクスポート：tflite_convert（INT8 キャリブレーションデータセット使用）
- モデル配布：アプリアップデートに同梱 OR 単独モデル更新パッケージとして配布
- バージョン管理：db_meta テーブルに model_version を記録、クラッシュは Firebase Crashlytics で検出

05 App Screens & UX Flow アプリ画面・UX フロー

5.1 画面一覧

画面名	役割	オフライン動作
スプラッシュ / 初回設定	言語選択・権限リクエスト・初期 DB ダウンロード	△（DL 時のみ通信）
ホームダッシュボード	診断開始・履歴・DB 閲覧・設定への 4 つのアクセス	✅
カメラ / 写真選択	1〜5 枚撮影または選択・トリミング・回転	✅
診断処理中	AI 推論プログレスバー・推定残り時間	✅
診断結果（Top-3）	信頼度バー付き上位 3 候補・タップで詳細	✅
障害詳細	症状・原因・対策・農薬・参照画像ギャラリー	✅
診断履歴	日付・場所・サムネイル・トップ結果一覧	✅
レポート表示 / 出力	PDF プレビュー・共有ボタン・農家メモ追記	✅
障害 DB 閲覧	カテゴリ別一覧・全文検索・フィルタ（重症度・地域・季節）	✅
設定	言語・地域（農薬絞込用）・エキスパートモード・DB 更新・アプリ情報	✅
DB 更新（オンライン時）	差分パッケージ確認・ダウンロード・適用	オンライン要
クラウド同期（任意）	診断データのバックアップ・他端末参照	オンライン要

5.2 診断結果画面の構成要素

要素	詳細
① 信頼度バッジ	色分けバー（緑 >70%、黄 40-70%、赤 <40%）+ 数値表示
② 障害名	主言語名（大）+ 学名（小・斜体）
③ 症状タブ	葉・茎・塊茎・全株の視覚的症状を平易な文章で記述
④ 原因タブ	土壌・環境・病原体・害虫別の原因解説（簡潔に 3〜5 行）
⑤ 対策タブ	営農作業（ポイント箇条書き）+ 農薬リスト（ユーザーの地域でフィルタ済み）
⑥ 参照画像	スワイプ可能なギャラリー（早期/中期/重症 × 各部位）
⑦ アクションボタン	レポート保存 | メモ追加 | 共有 | DB 全文表示

5.3 農薬情報の表示ルール

農薬リストはユーザーが設定で選択した「地域（国）」を基準にフィルタし、その国で登録済みの製品のみ表示します。各農薬には「有効成分・希釈倍率・散布時期・最終確認年」を表示し、「最終確認年が 2 年以上前」の場合は「登録状況を農協・普及員に確認してください」の警告を添えます。

06 Multilingual System 多言語対応

6.1 対応言語と翻訳範囲

言語	コード	優先度	翻訳対象
英語（English）	en	必須・基本	UI 全文・障害 DB 全テキスト（一次言語）
日本語	ja	必須	UI 全文・障害 DB 全テキスト・農薬登録名
ネパール語（नेपाली）	ne	必須	UI 全文・障害 DB 全テキスト
スペイン語（Español）	es	将来対応	UI テキスト（南米展開時に追加）
ヒンディー語（हिन्दी）	hi	将来対応	UI テキスト（インド展開時に追加）

6.2 実装方式

- ライブラリ：react-i18next + expo-localization
- 翻訳ファイル：assets/i18n/{locale}.json として静的バンドル（ビルド時に同梱）
- 障害 DB コンテンツ：disorders テーブルに各言語カラム（name_en, name_ja, name_ne）
- フォント：Noto Sans JP（日本語）/ Noto Sans Devanagari（ネパール語）を assets に同梱 ≈ 4 MB
- フォールバック：翻訳キーが存在しない場合は英語を表示
- 言語設定：MMKV に永続化、設定画面でいつでも切替可能、アプリ再起動不要

> Web 翻訳 API（Google Translate 等）は一切使用しません。すべての翻訳テキストはビルド時にバンドルするため、オフライン環境でも完全に動作します。

07 Offline & Low-Bandwidth Design オフライン・低帯域設計

7.1 オフライン動作の範囲

機能	オフライン可否	備考
画像診断（AI モデル推論）	✅ 完全オフライン	TFLite モデルは端末内蔵
障害 DB 参照・検索	✅ 完全オフライン	SQLite は端末ストレージ
診断結果・対策・農薬表示	✅ 完全オフライン	全テキストを DB 保持
多言語表示	✅ 完全オフライン	静的 i18n バンドル
PDF レポート生成	✅ 完全オフライン	HTML-to-PDF 端末処理
診断履歴閲覧	✅ 完全オフライン	SQLite ローカル保存
共有（WhatsApp 等）	✅ ローカルファイル共有のみ	受信側の通信は不要
障害 DB の更新	⚡ 通信時のみ	差分パッケージ HTTPS ダウンロード
クラウド同期（任意）	⚡ 通信時のみ	キュー方式で再試行

7.2 初回セットアップ戦略

2. アプリインストール時：最小限データ（モデル + 20 件の主要障害）を APK/IPA に同梱
3. 初回起動ウィザードで言語・地域を設定後、WiFi 接続を推奨する通知を表示
4. ユーザーが任意タイミングでフル DB（推定 150〜300 MB）をダウンロード
5. ダウンロードは中断・再開可能（Resume 対応）
6. フル DB 未取得でも最小データで診断は動作する（精度は制限あり）

7.3 画像・データの軽量化施策

- 参照画像：WebP 形式、最大 800 px 辺、品質 75（約 30〜80 KB/枚）
- 診断写真：撮影後に自動で 1280px にリサイズしてローカル保存
- DB パッケージ：zstd 圧縮で転送サイズを削減
- フォント：可変フォントを使用しサブセット化（使用文字のみ）

08 Diagnosis Records & Reports 診断記録・レポート機能

8.1 ローカル履歴管理

全診断セッションは自動的に SQLite の diagnosis_sessions テーブルに保存されます。農家はネット環境なしで過去の診断を閲覧・再出力できます。

- 診断履歴一覧：日付・圃場（GPS）・サムネイル・トップ診断結果を表示
- 詳細表示：当時の写真・結果・対策・農家メモを再表示
- 削除：任意の履歴を削除可（クラウド同期済みデータは要確認ダイアログ）
- 保存容量管理：設定で「保存する最大診断件数」を指定可（デフォルト 200 件）

8.2 PDF レポート機能

レポート要素	内容
ヘッダー情報	診断日時・GPS 位置（地図サムネイル）・農場メモ
撮影写真	診断に使用した写真（最大 5 枚、縮小表示）
診断結果	Top-3 障害名・信頼度・症状説明
原因分析	各 Top 候補の原因解説
対策提案	営農作業のポイント箇条書き
農薬リスト	適用農薬名・有効成分・希釈倍率・散布時期（地域フィルタ済み）
免責事項	「本結果は参考情報です。最終判断は専門家または農協にご相談ください」
フッター	アプリ名・DB バージョン・モデルバージョン

- 出力形式：PDF（主）+ PNG 画像（メッセージアプリ向け）
- 生成ライブラリ：react-native-html-to-pdf（完全オフライン）
- 共有方法：OS ネイティブ共有シート（WhatsApp・LINE・SMS・メール・印刷・クラウドドライブ）
- 言語：アプリ設定言語でレポート生成

8.3 クラウドバックアップ（任意機能）

クラウド同期は完全任意のオプトイン機能です。アカウント未作成でもアプリの全機能が使えます。同期を有効にした場合、診断メタデータを Firestore に、写真は Firebase Storage（暗号化）に保存します。通信が断絶した場合は同期キューに積み、接続回復時に自動送信します。GDPR および各国のデータプライバシー法を遵守します。

09 DB Update Workflow データベース更新フロー

9.1 更新の種類と規模

更新タイプ	ユースケース	転送サイズ目安
差分更新（Delta）	障害 1〜10 件の追加・修正・画像差替え	2〜15 MB
メジャー更新	カテゴリ追加・スキーマ変更・モデル更新	20〜80 MB
フルリセット	DB 破損回復・大規模再構成	150〜300 MB
モデル単独更新	AI モデル差替え（TFLite ファイルのみ）	10〜20 MB

9.2 更新プロセス

7. アプリが更新サーバーに問合せ（週次チェック or 手動）
7. サーバーからマニフェスト取得：{version, date, delta_size_mb, sha256}
8. ユーザーに通知：「新しい障害データが利用可能です（XX MB）」
9. ユーザーが承認 → HTTPS で差分パッケージ(.zip)をダウンロード
10. SHA-256 チェックサム検証 → 不一致なら破棄して再取得
11. SQLite マイグレーションスクリプトを適用（INSERT / UPDATE / DELETE）
12. 更新完了通知 → 旧 DB は 7 日間バックアップとして保持（ロールバック用）

9.3 DB 管理・編集ツール（db-tools CLI）

- 言語：Python 3.11
- import_disorders.py：CSV + 画像フォルダ → SQLite への一括インポート
- build_update_package.py：前バージョンとの差分 .zip パッケージ生成
- validate_db.py：スキーマ・必須フィールド・画像整合性チェック
- export_for_review.py：農学専門家レビュー用 HTML レポート生成

障害 DB の更新には農学専門家によるレビューフローを必須とします。投稿→査読→承認→パッケージ化の 4 ステップを db-tools スクリプトでトラッキングします。

10 Repository Structure GitHub リポジトリ構成

10.1 モノレポ構成

パス	内容
potato-guard/	リポジトリルート（モノレポ）
app/	React Native / Expo アプリ本体
app/src/screens/	画面コンポーネント（React）
app/src/components/	再利用 UI コンポーネント
app/src/i18n/	翻訳 JSON バンドル（en/ja/ne）
app/src/db/	SQLite フック・マイグレーション・クエリ
app/src/ai/	TFLite 推論モジュール
app/assets/models/	バンドル済み .tflite モデル
app/assets/fonts/	NotoSans JP / Devanagari フォント
db-tools/	Python 製 DB 管理 CLI ツール群
db-tools/data/raw/	CSV インポート元データ・画像
db-tools/data/packaged/	生成された更新パッケージ(.zip)
ml-tools/	モデル学習・変換スクリプト
ml-tools/train.py	EfficientNet ファインチューニング
ml-tools/export_tflite.py	INT8 量子化 + エクスポート
docs/	開発者ドキュメント・API 仕様・設計図
.github/workflows/	CI: lint → test → EAS Build → release

10.2 ブランチ戦略

- main：プロダクション（保護ブランチ・直接 push 禁止）
- develop：フィーチャー統合ブランチ
- feature/XXX：機能開発ブランチ
- db/XXX：障害データ追加・修正
- model/XXX：AI モデル更新
- hotfix/XXX：緊急修正
- リリースタグ：v{major}.{minor}.{patch}（例：v1.0.0）

11 Development Phases & Schedule 開発フェーズ・スケジュール

11.1 フェーズ別ロードマップ

フェーズ	テーマ	期間	主な成果物
Phase 0	基盤構築・環境整備	1〜2W	モノレポ・CI/CD・デザイントークン・EAS 設定
Phase 1	アプリシェル・多言語	3W	ナビゲーション・i18n・設定画面・SQLite スキーマ
Phase 2	障害 DB v1 構築	4W	50+件障害レコード・200+参照画像・import CLI ツール
Phase 3	AI モデル v1	4W	TFLite モデル・RN 推論ブリッジ・テストパイプライン
Phase 4	診断フロー	3W	カメラ画面・推論統合・結果画面・障害詳細
Phase 5	レポート・履歴	2W	セッション保存・履歴画面・PDF 生成・共有
Phase 6	オフライン・DB 更新	2W	差分更新機構・オプションクラウド同期
Phase 7	フィールドテスト・QA	3W	ネパール/日本での現地テスト・翻訳 QA・パフォ調整
Phase 8	ストア申請・リリース	2W	App Store / Play Store 申請・メタデータ・コンプライアンス

総期間目安：約 24〜26 週（6 ヶ月）。Phase 2〜4 は並列作業可能なため、チーム体制次第で短縮できます。

12 Risks & Mitigations リスクと対策

リスク	影響	発生率	対策
AI 精度が圃場条件で不足（日光・影・泥汚れ等）	高	中	ルールベース絞り込みフォールバック実装。現地写真でモデル再学習
旧型 Android 端末でのストレージ不足	中	高	Essential DB（50 MB）と Full DB（300 MB）のティア提供
ネパール語農業専門用語の翻訳品質	中	中	ネパール農業省または農大と翻訳レビュー提携
農薬登録情報の陳腐化・地域差	高	高	地域・年度フィルタ必須化。年次レビューサイクルを運用規定化
画像ライセンス侵害（参照画像）	高	低	CC ライセンスまたは独自撮影画像のみ使用。全画像に出典記録
App Store 規制（医療・農薬表示）	中	低	「判断補助ツール」として位置付け・免責事項を全結果に掲載
現地通信環境での初回 DL 失敗	中	高	Resume 対応ダウンロード。最小バンドル版を APK 内同梱

13 Open Issues & Recommendations 未決事項・提案

13.1 技術的未決事項

#	問い	推奨案
Q1	クラウドバックエンドの選択	Phase 1 は Firebase（迅速）、ユーザー規模拡大後にインド/ネパールリージョンの自前サーバーへ移行を検討
Q2	AI モデルのゼロからの学習 vs 既存モデルライセンス購入	PlantVillage 事前学習 + PotatoMD Fine-tuning で開始。CIP（国際ジャガイモセンター）との共同研究提携を並行交渉
Q3	誤診断フィードバックの収集・再学習ループ	アプリ内「誤診断報告」ボタン → 農学専門家レビュー → 承認済み修正を次回 DB 更新に反映
Q4	ユーザー写真データのクラウド利用（モデル改善目的）	明示的オプトイン同意フォームを実装。匿名化・リージョンロックを条件として研究機関と共有検討
Q5	ビジネスモデル	農家向け無料。NGO・農協・研究機関向け機関ダッシュボード（有料サブスクリプション）として収益化

13.2 推奨する次のアクション（着手順）

8. GitHub モノレポ作成・ブランチ保護・CI/CD パイプライン構築（第 1 週）
13. 農学専門家パートナーと障害 DB スキーマ・農薬情報の仕様確定（第 1〜2 週）
14. PlantVillage・EPPO からライセンス画像の調達と権利確認（第 2〜3 週）
15. db-tools CLI 実装 → シード障害 30 件を SQLite に投入（第 3〜4 週）
16. React Native + TFLite ダミーモデルで推論パイプラインの動作確認（第 2〜4 週）
17. Phase 1〜3 を可能な限り並行進行（第 4〜10 週）
18. ネパール現地農家 5〜10 名による第一回フィールドテスト（第 12 週目標）

本仕様書は随時改訂されるリビングドキュメントです。開発進行・フィールドフィードバックに応じて各セクションを更新します。 / This is a living document subject to revision as development progresses and field feedback is gathered.