SkinCheck — User and Developer Manual

Project: SkinCheck — AI-Assisted Skin Lesion Monitoring (Senior Design Project, Spring 2026)
Version: 1.0.0 **Platforms:** iOS, Android, Web (demo) **Demo Video:**
https://youtu.be/Bz28Oo7YwBY

Table of Contents

PART I — USER MANUAL

1. Introduction

1.1 What is SkinCheck?

SkinCheck is a **privacy-first mobile application** for tracking and monitoring suspicious skin lesions over time. Using an on-device AI model (TensorFlow Lite), the app captures a photo of a mole or lesion and returns an instant classification — *benign*, *malignant*, or *uncertain* — together with a confidence score and a recommended follow-up date.

SkinCheck is **not a diagnostic tool**. It is an educational screening aid that helps users decide when to consult a dermatologist.

1.2 Main Benefits at a Glance

Feature	Benefit to the User
On-device AI inference	Photos never leave the phone unless the user opts in
Scan history with photos, notes, and dates	Easy comparison of lesions over time
Automatic follow-up reminders	12-month reminder for benign, 3-month for malignant results (configurable)
PDF export	Shareable report for dermatologist visits
Face ID / Touch ID app lock	Health data stays private even if the phone is unlocked
Optional 2FA (TOTP)	Extra account protection that works offline
Multi-language UI	English, Spanish, French (auto-detected from device locale)
Insights dashboard	Charts for scan frequency, risk trend, follow-up adherence

1.3 Demo Video

A full walkthrough of the main features is available on YouTube: **https://youtu.be/Bz28Oo7YwBY**

---

2. Installation & First-Launch Setup

2.1 Supported Devices

- **iOS 15** or later (iPhone 8 and newer recommended)
- **Android 8.0 (API 26)** or later
- **Web preview** (limited — no camera/ML inference)

2.2 Installing the App

Because SkinCheck is distributed through Expo Application Services (EAS) during this senior design cycle, users currently install builds via TestFlight (iOS) or a signed APK (Android).

iOS (TestFlight):

1. Open the invite email and tap **View in TestFlight**.
2. Install TestFlight from the App Store if prompted.
3. Tap **Install** next to *SkinCheck*.
4. Once installed, open the app. Grant **Camera**, **Photo Library**, and **Notifications** permissions when asked.

Android (APK):

1. Download `skincheck.apk` from the distribution link.
2. Enable *Install from Unknown Sources* (Settings → Security).
3. Tap the APK to install.
4. Launch SkinCheck and grant the requested permissions.

2.3 First-Launch Onboarding

On first launch SkinCheck presents a 4-slide onboarding flow:

1. **Welcome** — project overview and medical disclaimer.
2. **How It Works** — camera capture and on-device AI explainer.
3. **History & Reminders** — how past scans and follow-ups are tracked.
4. **Privacy** — local-first storage and the optional sync toggle.

After onboarding, users can:

- **Create an account** (email + password) — required to keep scan history scoped to the device/user.
- **Try it first** — a test account is available in development builds (`test@skincheck.com` / `password`).

2.4 Optional Protections (Recommended)

From **Settings**, users are encouraged to enable:

- **Biometric App Lock** (Face ID / Touch ID) — auto-locks after 30 s in background.
- **Two-Factor Authentication (TOTP)** — pair with Google Authenticator or any compatible authenticator app.
- **Follow-up Notifications** — keeps reminders active for benign / malignant intervals.

---

3. Core Features Walkthrough

3.1 Dashboard

The **Dashboard** is the home tab. It summarizes:

- Total scan count and days since the last scan.
- A "Quick Start" card that opens the camera.
- A preview of the last 3 scans.
- Insight teasers (average confidence, benign-vs-malignant distribution).
- A **Sync Status Badge** showing pending/failed uploads when backend sync is enabled.

3.2 Capturing a Scan

1. Tap the **floating blue camera button** at the bottom of any tab.
2. The **Camera** screen opens in full-screen modal with guidance overlays:
    - **Distance indicator** — warns if the lesion is too close/far.
    - **Lighting indicator** — warns if the scene is too dark.
3. Tap the shutter (or pick a photo from the library).
4. The app runs **on-device inference** and transitions to the **Results** screen.

3.3 Reading Results

The **Results** screen shows:

- **Classification** — *Benign*, *Malignant*, or *Uncertain* with a color-coded badge.
- **Confidence score** — percentage (0–100%).
- **Risk level** — Low / Moderate / High.
- **Probability breakdown** — benign vs. malignant bars.
- **Next check-up date** — auto-scheduled.

- **Notes field** — add a location tag (e.g., "left forearm").
- **Save to History** and **Share PDF** actions.

3.4 Scan History

The **History** tab lists all saved scans (local-first). Tap any row to open **Scan Detail** with:

- Full-size captured photo.
- Notes (editable).
- Delete action (with confirmation).
- Re-export individual scan as PDF.

3.5 Insights

The **Insights** screen visualizes trends using `victory-native` charts:

- Scan frequency (weekly/monthly).
- Risk trend over time.
- Follow-up adherence rate.
- Distribution pie chart.

3.6 Educational Content

The **Learn More** screen describes the **ABCDE rule** (Asymmetry, Border, Color, Diameter, Evolution) and prevention tips. Accessible from the Dashboard or Settings.

3.7 Notifications

- **Follow-up reminders** at user-defined intervals (default: benign = 12 months, malignant = 3 months).
- **Inactivity nudges** after 2/4/8 weeks without a scan.
- **Weekly tips** (opt-in).
- **UV/seasonal alerts** (opt-in, May–September in Northern Hemisphere).

3.8 PDF Export

From History or a single scan, tap **Export PDF** to generate a formatted report containing the thumbnail, prediction, confidence, date, and notes. Uses the native Share Sheet (`expo-sharing`).

---

4. User Scenarios

SkinCheck is designed to be useful for several distinct audiences. Each scenario below maps a typical workflow.

4.1 End User — The Self-Monitoring Patient

Persona: *Alex, 34, fair-skinned, has several moles they want to track.*

Workflow:

1. Registers an account with email/password; enables Face ID lock.
2. Each month, opens the camera, photographs a tracked mole, and adds a note like "*right shoulder blade*".
3. Reviews the **History** tab side-by-side to see if the lesion's appearance has changed.
4. Receives a notification 12 months after a benign result prompting a re-check.
5. Before an annual dermatologist appointment, taps **Export PDF** from History and emails the report to the doctor.

Tips for this user:

- Always capture in good natural light with the phone ~10–15 cm above the skin.
- Use consistent naming in the notes field (e.g., "RA-01", "LA-02") to correlate moles.

4.2 Concerned Family Member — The Caregiver

Persona: *Jordan monitors moles on behalf of an elderly parent.*

Workflow:

1. Installs SkinCheck on the parent's phone *or* uses a shared device account.
2. Enables **Biometric App Lock** so the app auto-locks after being idle.
3. Captures scans at each visit and uses the **Notes** field to record who the scan is for and which body region ("Mom — upper back, mole #3").
4. Turns on **Inactivity Nudges (4 weeks)** to be reminded to do follow-ups.
5. Uses **Insights** to show the parent the benign-vs-malignant distribution over time.
6. Exports a **PDF report** before the parent's next doctor visit.

Tips:

- Disable biometric lock only if the parent uses the app independently; otherwise keep it on.
- Use the **Activity Log** (Settings → Activity Log) to see when scans were taken.

4.3 Wellness / Health-Conscious User

Persona: *Sam, a runner who spends many hours outdoors and wants to monitor sun exposure.*

Workflow:

1. Enables **UV/Seasonal Alerts** in Notification Preferences.
2. Turns on **Weekly Tips** to receive educational content every Monday.
3. Uses the **Dashboard** as a weekly check-in tool; taps the camera button whenever any mole looks different.
4. Opens **Learn More** to review the ABCDE rule periodically.
5. Keeps **Data Sync** OFF (default) — all data remains on-device.

Tips:

- Review **Insights** → **Risk Trend** monthly.
- Combine SkinCheck with an annual in-person dermatologist exam; the app is a screening aid, not a replacement.

4.4 Administrator / Developer / Power User

Persona: *The senior-design team or a future maintainer validating the app in the field.*

Workflow:

1. Logs in with an admin account and opts in to **Data Sync** (Settings → Privacy → Data Sync).
2. Configures `EXPO_PUBLIC_API_URL` to point at a staging backend (via `.env` during development builds).
3. Uses **Settings** → **Activity Log** to audit 18 event types (login, logout, scan created, 2FA enabled, biometric unlock, session expiry, etc.).
4. Checks the **Sync Status Badge** on the Dashboard for queue counts; taps to trigger a manual sync.
5. Verifies build version, ML model version, and commit hash on **Settings** → **About**.
6. Uses the **Developer Tools section** (in dev builds only) for model diagnostics printed to the Dashboard.

Admin-only considerations:

- Administrators should periodically review the **Audit Log** to detect anomalous patterns (repeated failed logins, lockouts).
- The **15-minute account lockout** automatically triggers after 5 failed login attempts.
- Sessions expire after **7 days**; the app force-logs out and prompts re-authentication.

---

5. Settings, Privacy, and Security

Setting	Where to Find It	What It Does
Theme (Light/Dark/System)	Settings → Appearance	Controls color scheme
Language	Settings → Language	English, Spanish, French
Biometric Lock	Settings → Security	Face ID / Touch ID after 30 s inactive
Two-Factor Authentication	Settings → Security → 2FA	TOTP via any authenticator app
Data Sync (opt-in)	Settings → Privacy	Sync scan metadata to backend over HTTPS
Delete Account	Settings → Account	Permanently removes local + server data
Notification Preferences	Settings → Notifications	Toggle reminders and nudges
Activity Log	Settings → Activity Log	View 200 most recent audit events
Privacy Policy	Settings → Privacy Policy	Full privacy disclosure
Medical Disclaimer	Settings → Medical Disclaimer	Clinical limitations explained

Privacy Guarantees:

- Images and ML inference run **entirely on-device** by default.
- Sensitive tokens use **SecureStore** (iOS Keychain / Android Keystore).
- Passwords are hashed with **SHA-256 + per-user salt** on-device; the backend uses **bcrypt**.
- Data sync is **off by default** and is explicitly opt-in.

---

6. Troubleshooting & FAQ

Problem	Fix
Camera button does nothing	Settings → Privacy → Camera: enable access for SkinCheck
"Loading AI model..." never completes	Restart the app; re-open from background. On web the mock model is used automatically

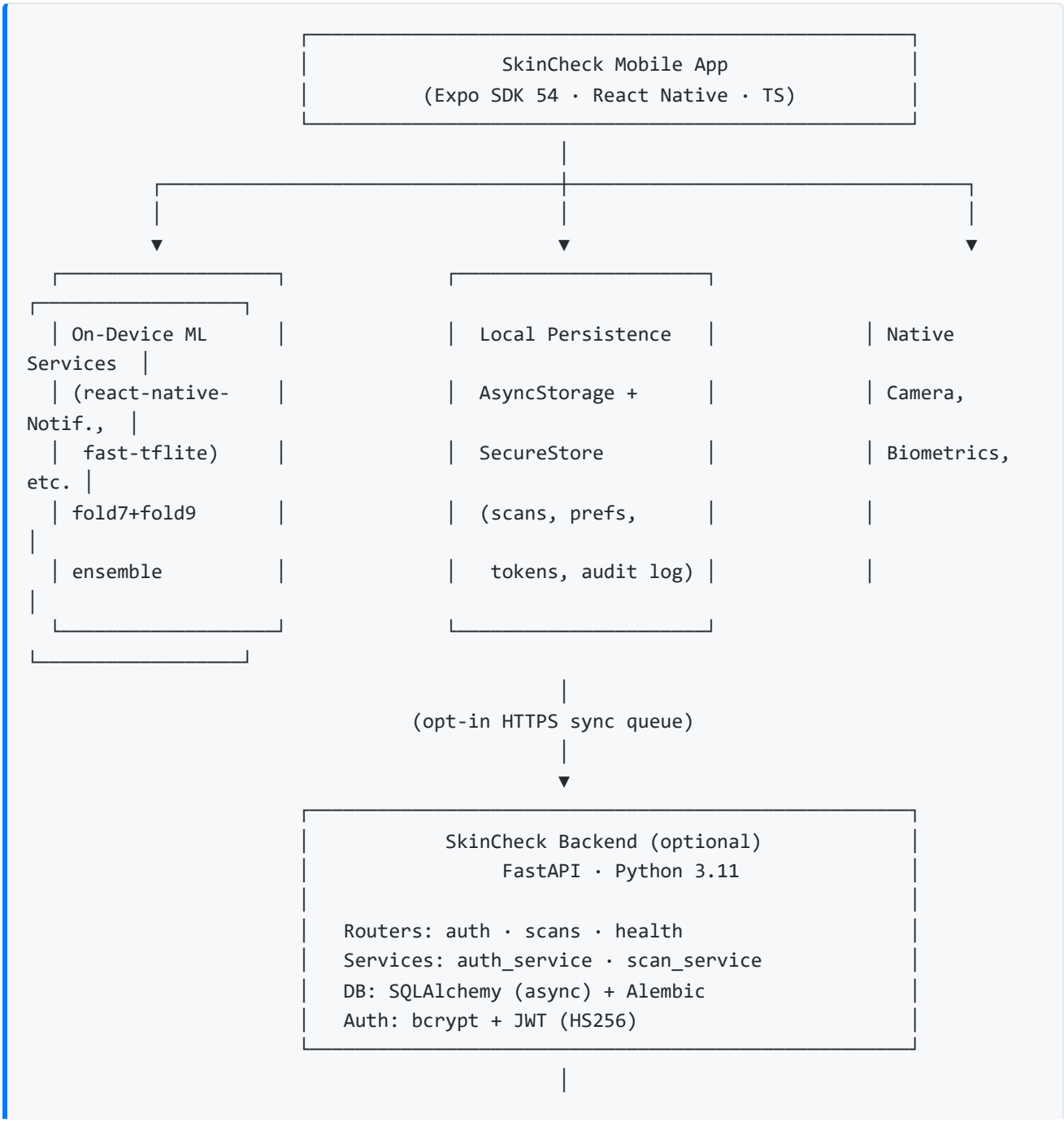
Problem	Fix
Face ID prompt repeats	Disable Biometric Lock from Settings → Security, then re-enable
Account locked for 15 minutes	You exceeded 5 failed logins. Wait 15 min or reset via 2FA recovery
Notifications not firing	Ensure OS-level notification permission is granted; check Settings → Notifications in-app
PDF export does nothing	Ensure the Share Sheet has at least one enabled destination
Scan history not syncing	Enable Data Sync and verify `EXPO_PUBLIC_API_URL` (developer builds only)
"Session expired" alert	Sessions are 7 days; simply log back in
App on web shows mock predictions	TFLite is not supported on web; use iOS or Android for real inference

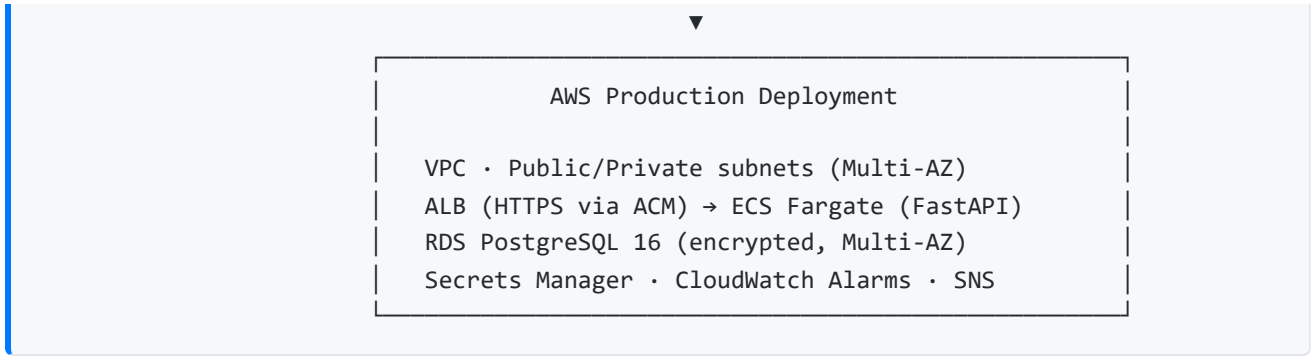
Contact: For bug reports during senior design, open an issue on the course repository or use the in-app crash reporting (Sentry is integrated but DSN is not set by default).

PART II — DEVELOPER MANUAL

7. System Architecture

7.1 High-Level Architecture





7.2 Architecture Description

SkinCheck is a **local-first mobile app with an optional cloud backend**.

- **Frontend (React Native + Expo 54):** handles all user interaction, camera capture, ML inference, storage, and UI. The app is organized around a **root stack navigator** that wraps a **bottom-tab navigator** (Dashboard, History, Settings). A floating camera button over the tab bar opens the modal Camera screen.
- **On-device ML runtime:** `react-native-fast-tflite` loads two TFLite models bundled in `assets/models/` (`fold7_model.tflite` , `fold9_model.tflite`) and runs a simple averaging ensemble. The ensemble threshold and an "uncertain" margin are configured in `assets/models/model_config.json` .
- **Local storage:** `AsyncStorage` for scans and preferences, `expo-secure-store` for tokens, passwords salts, and 2FA secrets.
- **Optional backend (FastAPI):** adds multi-device sync. When `EXPO_PUBLIC_API_URL` is set, the app queues mutations (create/update/delete scans) into a **sync queue** that is flushed automatically on connectivity and periodically (every 60 s in foreground).
- **Cloud deployment:** the `backend/infra/template.yaml` CloudFormation stack provisions an ECS Fargate service fronted by an Application Load Balancer, with an encrypted RDS PostgreSQL 16 database and AWS Secrets Manager for JWT/DB secrets.
- **Observability:** Sentry (opt-in via `EXPO_PUBLIC_SENTRY_DSN`) on the client; CloudWatch alarms (CPU, unhealthy hosts) with SNS email notification on the server.
- **OTA updates:** Expo Updates with two channels — `preview` (stable demo) and `next` (active development).

7.3 Runtime Startup Sequence

When the app launches (`App.tsx`):

1. `initSentry()` → crash reporting is initialized (no-ops if DSN missing).
2. `i18nReady` → device language loaded and UI strings populated.

3. `notificationService.requestPermissions()` → OS notification permission.
4. `mlModelService.initializeModel()` → TFLite models loaded, shapes validated, up to 2 retries.
5. `syncProcessor.processQueue()` → pending sync jobs flushed.
6. `connectivityService.start()` → foreground/online listener registered.
7. `AuthProvider` mounts → stored session restored (if valid).
8. `RootNavigator` decides: **Onboarding** → **Auth** → **Main tabs** (+ biometric auto-lock).

---

8. Source File Layout

8.1 Repository Tree (top-level)

```
Mobile App/
├── App.tsx                     # App entry, providers, startup sequence
├── index.js                    # Expo entrypoint → registers App
├── app.json                    # Expo config (permissions, plugins, EAS)
├── eas.json                    # EAS build/update profiles
├── package.json                # Scripts, deps, Jest config
├── tsconfig.json               # TypeScript strict mode + path aliases
├── babel.config.js / metro.config.js
├── jest.setup.ts               # Global test mocks
│
├── screens/                    # One file per screen (17 screens)
├── components/                 # Reusable UI primitives (Button, Card, …)
├── navigation/                 # RootNavigator, MainTabNavigator, types
├── hooks/                      # useAuth, useTheme, useOTAUpdate, …
├── utils/                      # Business logic (mlModel, auth, storage, …)
├── constants/                  # Theme, typography, spacing
├── assets/                     # Models, images, fonts, locales
├── locales/                    # i18n JSON (en, es, fr)
├── docs/                       # Project documentation (this file)
├── scripts/                    # Helpers (model conversion, i18n check)
├── .maestro/                   # E2E flows
├── __tests__/                  # Jest unit/integration tests
│
└── backend/
    ├── app/
    │   ├── main.py             # FastAPI app factory
    │   ├── config.py           # Settings (pydantic-settings)
    │   ├── database.py         # Async SQLAlchemy session
    │   ├── logging_config.py
    │   ├── models/             # SQLAlchemy ORM (user, scan, base)
    │   ├── schemas/            # Pydantic request/response schemas
    │   ├── routers/            # FastAPI routers (auth, scans, health)
    │   ├── services/           # Business logic (auth_service, scan_service)
    │   └── middleware/         # JWT auth middleware
    ├── alembic/                # Database migrations
    ├── infra/                  # CloudFormation + deploy docs
    ├── Dockerfile
    ├── docker-compose.yml
    ├── requirements.txt
    └── tests/                  # pytest suite
```

8.2 Frontend Screens (`screens/`)

File	Purpose
`OnboardingScreen.tsx`	4-slide first-launch flow; persists completion flag.
`AuthScreen.tsx`	Login / register forms, lockout handling, 2FA entry.
`DashboardScreen.tsx`	Home tab: totals, recent scans, insight preview, sync badge.
`CameraScreen.tsx`	Full-screen camera with quality indicators; can also pick from library.
`ResultsScreen.tsx`	Post-scan prediction view; save/share/PDF actions.
`InsightsScreen.tsx`	Analytics charts (frequency, trend, distribution, adherence).
`HistoryScreen.tsx`	List of past scans; pull-to-refresh; delete/export.
`ScanDetailScreen.tsx`	Per-scan detail view with editable notes.
`SettingsScreen.tsx`	Root of preferences; entry points to all settings sub-screens.
`ProfileEditScreen.tsx`	Change name/email/password; delete account.
`NotificationPreferencesScreen.tsx`	Toggle follow-ups, nudges, UV, weekly tips.
`TwoFactorSetupScreen.tsx`	TOTP setup with QR/deep-link, secret key, verification.
`AuditLogScreen.tsx`	Renders last 200 audit events from `auditService` .
`LanguagePickerScreen.tsx`	Switch language; saves to AsyncStorage.
`EducationalContentScreen.tsx`	Static ABCDE content.

File	Purpose
`MedicalDisclaimerScreen.tsx` / `PrivacyPolicyScreen.tsx` / `AppInfoScreen.tsx`	Informational screens.

8.3 Reusable Components (`components/`)

File	Purpose
`ThemedText.tsx` / `ThemedView.tsx`	Consume `useTheme()` for consistent colors.
`Button.tsx`	Animated button with `primary` / `secondary` / `destructive` variants.
`Card.tsx`	Pressable card with spring press animation.
`Spacer.tsx`	Layout helper reading spacing tokens.
`ScreenScrollView.tsx` / `ScreenKeyboardAwareScrollView.tsx` / `ScreenFlatList.tsx`	Safe-area + keyboard-aware wrappers.
`HeaderTitle.tsx`	Custom nav header text.
`ErrorBoundary.tsx` / `ErrorFallback.tsx`	Crash-guard around the app tree; renders fallback UI.
`SyncStatusBadge.tsx`	Dashboard badge showing pending/failed sync counts.
`UpdateBanner.tsx`	OTA update-available banner.

8.4 Navigation (`navigation/`)

- `RootNavigator.tsx` — native stack; chooses between Onboarding, Auth, and the Main tabs; defines 15 screens including modals for Camera, Results, Insights, and ScanDetail.
- `MainTabNavigator.tsx` — bottom tabs (Dashboard, History, Settings) + `FloatingActionButton` that navigates to the Camera modal.
- `types.ts` — `RootStackParamList`, `MainTabParamList`, and typed props (`RootStackScreenProps`, `MainTabScreenProps`).

8.5 Utilities (`utils/`) — Core Business Logic

File	Purpose
`mlModel.ts`	TFLite load, preprocess, predict; mock fallback. Exports `mlModelService` .
`auth.ts`	Email/password auth, session, lockout, password hashing. `authService` .
`storage.ts`	Scan CRUD, notification prefs, data-sync toggles. `storageService` .
`api.ts`	HTTPS client to FastAPI backend with JWT refresh.
`syncQueue.ts`	Persistent queue of pending backend mutations.
`syncProcessor.ts`	Flushes the queue with retry/backoff.
`connectivity.ts`	Online/offline detection and listeners.
`notifications.ts`	Schedules follow-ups, nudges, UV alerts, weekly tips.
`crypto.ts`	PBKDF2 key derivation + AES-GCM encrypt/decrypt.
`biometric.ts`	Wraps `expo-local-authentication` .
`totp.ts`	Offline HMAC-SHA1 TOTP generator (RFC 6238).
`audit.ts`	Append-only local event log (cap 200).
`analytics.ts`	Computes `AnalyticsSummary` from the scan list.
`pdfExport.ts`	Builds HTML and converts via `expo-print` .
`sentry.ts`	`initSentry` + `captureError` helpers.
`i18n.ts`	i18next initialization + `i18nReady` .

8.6 Hooks (`hooks/`)

File	Returns
`useAuth.tsx`	`{ user, isAuthenticated, isLoading, sessionExpired, login, register, logout }`
`useTheme.ts`	`{ theme, colors }` based on system color scheme
`useColorScheme.ts` / `.web.ts`	Platform-specific color scheme hook

File	Returns
`useScreenInsets.ts`	Safe-area insets utility
`useSyncStatus.ts`	Pending/failed counts + `triggerSync`
`useOTAUpdate.ts`	Checks Expo Updates; exposes `isReady`, `applyUpdate`, `dismissUpdate`

8.7 Backend (`backend/app/`)

File	Purpose
`main.py`	FastAPI app factory; startup/shutdown; CORS; router registration.
`config.py`	`Settings` (prefix `SKINCHECK_`): DB URL, JWT, CORS, rate limits.
`database.py`	Async engine + session dependency.
`logging_config.py`	Structured JSON logging.
`models/user.py`	`User` ORM.
`models/scan.py`	`Scan` ORM with FK to user.
`schemas/auth.py`	Pydantic DTOs (register, login, tokens, update, change pwd).
`schemas/scan.py`	`ScanCreate`, `ScanRead`, `ScanListResponse`, `ScanUpdate`.
`routers/auth.py`	`/auth/register`, `/login`, `/refresh`, `/me`, `/change-password`, `/delete-account`.
`routers/scans.py`	CRUD + export endpoints for scans.
`routers/health.py`	`/health` liveness probe.
`services/auth_service.py`	Password hashing (bcrypt), JWT access/refresh, in-memory rate limiter.
`services/scan_service.py`	Cursor-paginated list, create/update/delete, owner scoping.

File	Purpose
`middleware/auth_middleware.py`	`get_current_user` dependency that validates JWT.
`alembic/`	Migration environment and revisions.
`infra/template.yaml`	CloudFormation (VPC, ALB, ECS, RDS, Secrets Manager, alarms).
`Dockerfile` / `entrypoint.sh` / `docker-compose.yml`	Container build and local stack.

---

9. Key Methods and Functions

Below are the most important methods in each subsystem. Full signatures use TypeScript (frontend) or Python (backend).

9.1 `utils/mlModel.ts` — `mlModelService`

```
initializeModel(): Promise<boolean>
```

Loads all configured TFLite runtimes, validates input/output shapes, and retries twice before falling back to mock predictions. Returns `true` if at least one runtime loaded.

```
preprocessImage(imageUri: string): Promise<Float32Array | null>
```

Resizes the input to the model's expected size (e.g. 224×224), decodes the JPEG, and normalizes RGB values to `[-1, 1]` (standard ImageNet/MobileNet preprocessing).

```
predict(imageUri: string): Promise<SkinLesionPrediction>
```

Runs every loaded model, averages malignant probabilities, and classifies as `benign`, `malignant`, or `uncertain` based on `threshold` and `uncertain_margin` from `model_config.json`. Returns `SkinLesionPrediction` with `classification`, `confidence`, `riskLevel`, `source`, and `details`.

```
getModelInfo(): Promise<{ loaded, version, source, loadedModels, ensembleThreshold, ...
}>
```

Diagnostics used by the Dashboard in development builds.

9.2 `utils/storage.ts` — `storageService`

- `getScanHistory(): Promise<ScanResult[]>` — reads from AsyncStorage; merges with backend when sync is on.
- `saveScan(scan: ScanResult): Promise<void>` — local insert; enqueues a `create_scan` sync job.
- `deleteScan(id: string): Promise<void>` — local delete; enqueues `delete_scan`.
- `updateScanNotes(id, notes): Promise<void>` — patches local entry; enqueues `update_scan`.
- `getNotificationPreferences(): Promise<NotificationPreferences>` — returns defaults if unset.
- `saveNotificationPreferences(p): Promise<void>` — persists user toggles.
- `getDataSyncEnabled() / setDataSyncEnabled()` — opt-in sync toggle.

9.3 `utils/auth.ts` — `authService`

- `register(email, password, name): Promise<AuthResult>`
- `login(email, password, twoFactorCode?): Promise<AuthResult>` — enforces 15-min lockout after 5 failures.
- `logout(): Promise<void>` — clears session + audit log entry.
- `isSessionValid(): Promise<boolean>` — 7-day expiry; called periodically every 5 minutes.
- `enableTwoFactor(secret): Promise<void>` / `disableTwoFactor(): Promise<void>`
- `changePassword(current, new): Promise<void>` — uses SHA-256 + salt on device.

9.4 `utils/notifications.ts` — `notificationService`

- `requestPermissions(): Promise<boolean>`
- `scheduleFollowUpNotification(scan, monthsInterval): Promise<string>` — returns the Expo notification id.
- `cancelAllScheduled(): Promise<void>`
- `scheduleInactivityNudge(weeks): Promise<string>` — internal; respects preferences.
- `scheduleUvAlert(): Promise<void>` — May–September only.
- `scheduleWeeklyTip(): Promise<void>`

9.5 `utils/syncQueue.ts` + `syncProcessor.ts`

- `syncQueue.enqueue(op: SyncOp): Promise<void>`

- `syncQueue.getPendingCount(): Promise<number>`
- `syncProcessor.processQueue(): Promise<void>` — drains the queue; on failure marks items as `failed` and backs off.

9.6 `utils/analytics.ts`

- `computeSummaryStats(scans: ScanResult[]): AnalyticsSummary` Produces totals, `scanFrequency`, `riskTrend`, prediction distribution, `adherenceRate`, average confidence, and `daysSinceLastScan`.

9.7 Backend — `services/auth_service.py`

- `hash_password(password: str) -> str` — bcrypt with generated salt.
- `verify_password(password: str, hashed: str) -> bool`
- `create_user(session, email, password, name) -> User` — raises `AuthError` on duplicate email.
- `authenticate_user(session, email, password) -> User` — respects `InMemoryRateLimiter`; raises `RateLimitError` after N attempts.
- `create_access_token(user_id) -> (token, expires_in)` — HS256 JWT, short-lived.
- `create_refresh_token(user_id) -> token` — long-lived.
- `decode_token(token, expected_type) -> user_id` — raises `AuthError` on bad/expired tokens.

9.8 Backend — `services/scan_service.py`

- `list_scans(session, user_id, cursor=None, limit=20) -> (scans, next_cursor)` — cursor pagination by timestamp.
- `create_scan(session, user_id, payload: ScanCreate) -> Scan`
- `get_scan(session, user_id, scan_id) -> Scan | None` — enforces ownership.
- `update_scan(session, user_id, scan_id, payload: ScanUpdate) -> Scan | None`
- `delete_scan(session, user_id, scan_id) -> bool`

9.9 Backend — `middleware/auth_middleware.py`

- `get_current_user(request, session)` — FastAPI dependency: reads `Authorization: Bearer <jwt>`, decodes, loads `User`, raises 401 on failure.

---

10. Backend API Reference

Base URL: `http://localhost:8000/api` (local) or `https://api.your-domain.com/api` (prod).

All endpoints except `/auth/register`, `/auth/login`, `/auth/refresh`, and `/health` require `Authorization: Bearer <access_token>`.

10.1 Auth

Method	Path	Request	Response	Notes
POST	`/auth/register`	`UserCreate` (email, password, name)	`AuthResponse` (user + tokens)	201 on success; 400 on duplicate
POST	`/auth/login`	`LoginRequest`	`AuthResponse`	401 on bad creds; 429 when rate-limited
POST	`/auth/refresh`	`{ refresh_token }`	`TokenPair`	401 on expired refresh
GET	`/auth/me`	—	`UserRead`	
PATCH	`/auth/me`	`UserUpdate` (email?, name?)	`UserRead`	400 if email already used
POST	`/auth/change-password`	`{ current_password, new_password }`	204	400 if current password wrong
DELETE	`/auth/delete-account`	—	204	Cascade-deletes scans

10.2 Scans

Method	Path	Request	Response	Notes
GET	`/scans?cursor=…&limit=20`	—	`ScanListResponse`	Cursor pagination
POST	`/scans`	`ScanCreate`	`ScanRead`	201
GET	`/scans/{scan_id}`	—	`ScanRead`	404 on not found/not owner

Method	Path	Request	Response	Notes
PATCH	`/scans/{scan_id}`	`ScanUpdate`	`ScanRead`	
DELETE	`/scans/{scan_id}`	—	204	
GET	`/scans/export/all`	—	`{ items: ScanRead[] }`	Up to 1000 scans

10.3 Health

- `GET /health` → `{ status: "ok" }`. Used by ALB for target-group health checks.

10.4 Error Format

FastAPI + custom handlers return:

```
{ "detail": "Human-readable message" }
```

Validation errors include a `loc` path; the frontend (`utils/api.ts`) flattens this into field-scoped errors for forms.

---

11. Data Models

11.1 Frontend (TypeScript)

```
interface User {
  id: string;
  email: string;
  name: string;
  createdAt: string;
  twoFactorEnabled: boolean;
}

interface ScanResult {
  id: string;
  imageUri: string;
  prediction: "benign" | "malignant" | "uncertain";
  confidence: number;            // 0..1
  timestamp: string;             // ISO8601
  notes?: string;
  nextCheckupDate?: string;
}

interface SkinLesionPrediction {
  classification: "benign" | "malignant" | "uncertain";
  confidence: number;
  riskLevel: "low" | "moderate" | "high";
  source: "model" | "mock";
  modelsUsed: string[];
  details: { malignantProbability: number; benignProbability: number };
}

interface NotificationPreferences {
  enabled: boolean;
  benignInterval: 1 | 3 | 6 | 12;
  malignantInterval: 1 | 3 | 6 | 12;
  nudgeEnabled: boolean;
  nudgeWeeks: 2 | 4 | 8;
  uvAlertsEnabled: boolean;
  weeklyTipsEnabled: boolean;
}
```

11.2 Backend (SQLAlchemy)

```
class User(Base, TimestampMixin):
    id: str (UUID, PK)
    email: str (unique, indexed)
    name: str
    hashed_password: str         # bcrypt
    is_active: bool = True
    two_factor_enabled: bool = False
    scans: list[Scan]            # cascade delete

class Scan(Base, TimestampMixin):
    id: str (UUID, PK)
    user_id: str (FK users.id, ondelete=CASCADE)
    image_uri: str
    prediction: str              # benign | malignant | uncertain
    confidence: float
    timestamp: datetime
    notes: str | None
    next_checkup_date: datetime | None
```

`TimestampMixin` adds `created_at` and `updated_at`.

---

12. Build, Test, and Deployment

12.1 Local Development

Frontend:

```
npm install
npm run dev            # expo start --dev-client
npm run ios            # iOS simulator
npm run android        # Android emulator
```

Backend:

```
cd backend
python -m venv .venv
. .venv/Scripts/activate          # Windows
pip install -r requirements.txt
uvicorn app.main:app --reload --host 0.0.0.0 --port 8000
```

Docker (full stack):

```
cd backend
docker compose up --build
```

12.2 Environment Variables

Frontend (`.env`):

- `EXPO_PUBLIC_API_URL` — e.g. `https://api.your-domain.com` (optional; local-only if unset).
- `EXPO_PUBLIC_SENTRY_DSN` — optional crash reporting.

Backend (`.env` , prefix `SKINCHECK_`):

- `SKINCHECK_DATABASE_URL`
- `SKINCHECK_JWT_SECRET_KEY`
- `SKINCHECK_ACCESS_TOKEN_EXPIRE_MINUTES`
- `SKINCHECK_REFRESH_TOKEN_EXPIRE_DAYS`
- `SKINCHECK_CORS_ORIGINS` (JSON array)
- `SKINCHECK_LOGIN_RATE_LIMIT_ATTEMPTS`

12.3 Testing

- **Unit/Integration (Jest + React Native Testing Library):** `npm test`
- **E2E (Maestro):** `npm run test:e2e:ios` / `:android`
- **Backend (pytest):** `npm run test:backend`
- **Translation completeness:** `npm run i18n:check:strict`
- **Formatting:** `npm run format` / `npm run check:format`
- **Linting:** `npm run lint`

Test suites cover all utils (`auth` , `storage` , `mlModel` , `notifications` , `crypto` , `totp` , `audit` , `analytics` , `pdfExport` , `sentry` , `syncQueue` , `syncProcessor` , `api` , `biometric` , `i18n`) and every screen. Backend tests cover auth (including edge cases), scans, services, health, and config.

12.4 OTA Release Channels

- `preview` — frozen demo track (senior design demo).
- `next` — active development.
- `production` — reserved for future public launch.

Commands:

```
npm run update:preview -- "hotfix message"
npm run update:next -- "dev update message"
npm run build:preview:ios
npm run build:next:ios
```

12.5 AWS Deployment

Run the CloudFormation template in `backend/infra/template.yaml`:

1. Push a Docker image to ECR (`ApiEcrRepository` output).
2. Deploy/update the stack (creates VPC, subnets, ALB, ECS service, RDS, Secrets Manager, CloudWatch alarms, SNS).
3. Point DNS at the ALB (optionally a Route53 alias record if `ApiDomainName` + `HostedZoneId` are provided).
4. Configure ACM cert ARN for HTTPS.
5. Set `EXPO_PUBLIC_API_URL` to the published `ApiUrl` output.

Detailed steps: `backend/infra/DEPLOY.md`.

---

13. Extending the System

13.1 Swapping the ML Model

1. Train and export a new TFLite model.
2. Copy the file to `assets/models/<key>_model.tflite`.
3. Add an entry to `assets/models/model_config.json` (`key`, `threshold`). Update `ensemble.threshold` and `uncertain_margin` if needed.
4. Register the asset in `mlModel.ts#loadConfiguredModels` (`modelAssets[key] = require(...)`).
5. Run the app — shape validation runs automatically and reports mismatches.

13.2 Adding a New Screen

1. Create `screens/MyScreen.tsx`.
2. Add a name + params type to `navigation/types.ts`.
3. Register the screen in `navigation/RootNavigator.tsx` (or a tab in `MainTabNavigator.tsx`).
4. Add localized strings to all files under `locales/<lang>/common.json`.
5. Add a test in `__tests__/screens/MyScreen.test.tsx`.

13.3 Adding a New API Endpoint

1. Add a Pydantic schema in `backend/app/schemas/`.
2. Add a service function in `backend/app/services/`.
3. Add a router in `backend/app/routers/` (or extend an existing one).
4. Register the router in `backend/app/main.py`.
5. Add tests in `backend/tests/`.
6. Extend `utils/api.ts` on the frontend to call it.

13.4 Adding a New Audit Event

1. Append the event name to the `AuditEventType` union in `utils/audit.ts`.
2. Call `auditService.log('my_event')` where appropriate.
3. Add a translated label under `locales/<lang>/common.json → audit.events`.

13.5 Adding a New Language

1. Create `locales/<lang>/common.json` by copying an existing file.
2. Translate every key.
3. Run `npm run i18n:check:strict` to verify completeness.
4. Add the language option to `LanguagePickerScreen.tsx`.

Appendix A — Demo Video

Watch the feature walkthrough here: **https://youtu.be/Bz28Oo7YwBY**

Appendix B — Medical Disclaimer

SkinCheck is an educational tool and **not a substitute for professional medical diagnosis**. Always consult a licensed dermatologist for any concerning skin lesion. The app's predictions should be treated as a supplementary signal only.

Appendix C — License & Attribution

SkinCheck was built as a senior design project (Spring 2026). Third-party libraries retain their original licenses (Expo, React Navigation, FastAPI, SQLAlchemy, Victory, etc.). TFLite model weights are

property of the training data provider and are bundled for academic demonstration purposes only.

End of SkinCheck User and Developer Manual — v1.0.0.