UrbanFix AI — Full Application System Report

Date: 2026-02-20
Workspace: `appv1`
Scope: End-to-end documentation of the currently built application (frontend + backend + data model + operational flows)

---

1) Executive Summary

UrbanFix AI is a civic issue reporting and municipal collaboration platform with:

- A **React Native (Expo)** mobile app (`frontend`) for citizens, admins, and field workers.
- A **Node.js + Express** backend (`backend`) exposing REST APIs.
- **Supabase PostgreSQL** as the primary datastore.
- **JWT-based app auth** plus **Supabase Auth integration** (OTP + Google OAuth).
- **Role-based UX and permissions** for:
  - `citizen`
  - `admin` / `super_admin`
  - `field_worker`
- Built-in systems for:
  - issue lifecycle tracking,
  - map-based discovery,
  - social interactions (upvotes/comments/following),
  - municipal page publishing,
  - gamification,
  - notifications and push token registration.

This document describes **how the app is built today**, what each module does, and what is expected for each role/use case.

---

2) Product Goals & Operating Model

2.1 Core Product Goal

Enable citizens to report civic issues quickly and transparently while giving authorities structured workflows to assign, resolve, and communicate updates.

2.2 Main Product Pillars

1. **Report with context** (title, category, media, GPS location).
2. **Route and prioritize** issues by severity/category/department.
3. **Track progress** through status timeline and proof of resolution.
4. **Engage community** with feed interactions and follow systems.
5. **Incentivize participation** with points, badges, leaderboard.

2.3 Current Role Model

- **Citizen:** reports issues, engages with feed, follows municipal pages/issues, tracks updates.
- **Admin / Super Admin:** triages priority queue, updates statuses, assigns departments/workers, can create municipal pages and official posts.
- **Field Worker:** receives assigned tasks, marks progress, uploads proof, resolves field tasks.
- **Municipal Page (entity):** content-author profile represented in feed/posts; pages are created/managed by admins.

---

3) System Architecture

3.1 High-Level Architecture

- **Mobile Client:** Expo React Native app with role-aware navigation.
- **API Layer:** Express server with route modules by domain.
- **Data Layer:** `backend/data/store.js` wraps Supabase table operations.
- **Auth Layer:**
  - app JWT for backend route protection,
  - Supabase Auth token exchange endpoint for OAuth/OTP sessions.
- **Notification Layer:**
  - in-app notification records in DB,
  - optional FCM push send path via Firebase Admin.

3.2 Runtime / Deployment Notes

- Backend local default port: `5000`.
- Frontend API base selection:
  - in dev: LAN IP (`src/services/api.ts`) for physical device testing,
  - prod fallback: `https://urban-fix-ai.onrender.com`.

3.3 Technology Stack

Frontend

- React Native `0.81.5`
- Expo SDK `54`
- TypeScript
- React Navigation (stack + tabs)
- Axios
- Expo modules: notifications, location, auth-session, image-picker, camera
- react-native-maps

Backend

- Node.js (>=18)
- Express `5.x`
- Supabase JS client
- JWT
- bcrypt
- Firebase Admin

Database

- Supabase PostgreSQL
- PostGIS extension for geospatial support

---

4) Repository Structure and Responsibilities

4.1 Root

- Root scripts delegate to backend start/build.

4.2 Backend (`/backend`)

- `server.js`: app bootstrap, middleware, route mounting.
- `routes/`: API domains (`auth`, `issues`, `workflows`, `gamification`, `notifications`, `municipal`, `users`).
- `middleware/authMiddleware.js`: JWT validation + role guards.
- `data/store.js`: all DB operations (users/issues/comments/follows/etc.).
- `services/notificationService.js`: DB notifications + FCM delivery.
- `services/ai/*`: placeholder/mocked AI analysis helpers.
- SQL migrations: schema + push token tables.

4.3 Frontend (`/frontend/src`)

- `navigation/`: role-based tab flows + root stack.
- `context/AuthContext.tsx`: auth lifecycle, session restore, setup flags, Supabase callbacks.
- `screens/`: Auth/Main/Admin/Municipal/Worker experiences.
- `services/api.ts`: axios client + typed API wrappers.
- `services/locationService.ts`: location permissions/retries/reverse geocode/cache.
- `services/notificationService.ts`: push registration and backend token registration.

---

5) Authentication, Session, and Onboarding

5.1 Auth Methods Supported

1. Email + password (backend auth endpoints).
2. Email OTP / magic link (Supabase Auth).
3. Google OAuth (Supabase Auth with deep-link callback handling).

5.2 Session Model

- Supabase-authenticated users call `/api/auth/supabase-login` to obtain app JWT.
- App JWT stored in AsyncStorage (`token`), attached to API requests by interceptor.
- User payload persisted as `user` in AsyncStorage.

5.3 First-Run / Setup Gates

After login, user flow is gated by:

1. **Location setup** (`LocationSetupScreen`) — requested on fresh login/open paths.
2. **Profile setup** (`ProfileSetupScreen`) if incomplete profile (`username/city/ward` missing).

5.4 Profile Setup Expectations

- Username validation rules: min length + character restrictions + uniqueness check.
- Location confirmation: city + ward.
- Interest selection: at least one civic topic.

---

6) Navigation and Role-Based UX

6.1 Root Navigation Decisioning

- No user: onboarding + login/register/OTP.
- Authenticated but setup incomplete: location/profile setup screens.
- Setup complete:
  - Admin/Super Admin → `AdminTabs`
  - Field worker → `WorkerTabs`
  - Otherwise → `CitizenTabs`

6.2 Citizen Tabs

- Feed
- Map
- Report (floating central action opens Report Issue screen)
- Alerts (notification badge polling)
- Profile

6.3 Admin Tabs

- Dashboard
- Map
- Alerts
- Profile

6.4 Worker Tabs

- Tasks dashboard
- Map
- Alerts
- Profile

---

7) Feature Catalogue (Full)

7.1 Issue Reporting

Screen: `ReportIssueScreen`
Capabilities:

- Category selection.
- Title/description.
- Photo/video/gallery evidence.
- Location detection priority:
    1. EXIF GPS from media,
    2. live GPS,
    3. cached location.
- Anonymous toggle.
- Emergency toggle.

Backend behavior (`POST /api/issues`):

- Creates issue with AI-like severity placeholder logic.
- Initializes status timeline as `Submitted`.
- Rewards points (+10) for non-anonymous reports.
- May trigger first-report badge notification.

7.2 Feed System

Screen: `HomeFeed`
Modes:

- Community feed
- Municipal feed

Filter options:

- all
- trending
- following
- high_priority
- resolved
- my_posts
- nearby (UI filter option; server filtering primarily via existing route logic)

Engagement actions:

- upvote/downvote
- comment navigation
- share
- open issue detail
- municipal profile entry

7.3 Municipal Content

- Municipal pages searchable/suggested/followable.
- Page profile with Updates/About tabs.
- Official updates are created as issues authored by `MunicipalPage` type and marked resolved-style records.

7.4 Map Intelligence

Screen: `MapScreen`

- Live location acquisition (best effort).
- Optional nearby seed generation for demo data.
- Category + status filters.
- 50km proximity filtering (when user location available).
- Marker severity styling + emergency marker hint.
- Heatmap-like circles by severity.

7.5 Issue Detail and Public Traceability

Screen: `IssueDetailScreen`

- Shows media, severity, tags, priority score, metadata.
- Full status timeline.
- Resolution proof (before/after + worker remarks) when resolved.
- Social actions and comments.
- Follow issue for updates.

7.6 Notifications

Frontend: polling + swipe interactions.
Backend: CRUD + read state endpoints.

Capabilities:

- Fetch with unread count.
- Mark one/all as read.
- Delete one/all.
- Auto refresh intervals.
- Swipe-to-delete UX.

7.7 Gamification

- Leaderboard (`citizen` ranking by points).
- Badge earned state.
- Platform issue stats summary.
- Profile level progression representation.

7.8 Admin Operations

Screen: `AdminDashboard`

- Pull high-priority queue.
- Update status (`Acknowledged`, `InProgress`, `Resolved`).
- Assign department/worker metadata.
- View aggregate platform stats.

7.9 Field Worker Operations

Screen: `FieldWorkerDashboard`

- Load assigned tasks.
- Start work (`InProgress`).
- Complete work (`Resolved`) with proof photo upload.
- Open map directions.

7.10 Profile and Settings

- User profile with stats, badge browser, level progress, logout.
- Settings toggles persisted to AsyncStorage (dark mode toggle retained but light mode not active).
- Support/legal links and danger zone actions.

---

8) API Surface (As Built)

8.1 Auth

- `POST /api/auth/register`
- `POST /api/auth/login`
- `POST /api/auth/supabase-login`

8.2 Issues

- `GET /api/issues`
- `POST /api/issues`
- `GET /api/issues/:id`
- `POST /api/issues/seed-nearby`
- `GET /api/issues/geojson`
- `PUT /api/issues/:id/upvote`
- `PUT /api/issues/:id/downvote`
- `PUT /api/issues/:id/follow`
- `POST /api/issues/:id/comments`
- `GET /api/issues/:id/comments`

8.3 Workflows

- `PUT /api/workflows/:id/status` (admin)
- `PUT /api/workflows/:id/assign` (admin)
- `PUT /api/workflows/:id/worker-update` (field worker/admin)
- `GET /api/workflows/assigned/:workerId`

8.4 Gamification

- `GET /api/gamification/leaderboard`
- `GET /api/gamification/badges`
- `GET /api/gamification/stats`

8.5 Notifications

- `GET /api/notifications`
- `PUT /api/notifications/read-all`
- `PUT /api/notifications/:id/read`
- `DELETE /api/notifications/:id`
- `DELETE /api/notifications`

8.6 Municipal Pages

- `POST /api/municipal/create` (admin)
- `GET /api/municipal/search`
- `GET /api/municipal/suggested`
- `GET /api/municipal/:id`
- `GET /api/municipal/:id/followers`
- `POST /api/municipal/:id/follow`
- `POST /api/municipal/:id/unfollow`
- `POST /api/municipal/:id/post` (admin)
- `PATCH /api/municipal/:id` (admin)

8.7 Users

- `GET /api/users/check-username/:username`
- `GET /api/users/profile`
- `PUT /api/users/profile`
- `POST /api/users/push-token`

---

9) Data Model (Supabase)

9.1 Primary Tables

- `users`
- `issues`
- `municipal_pages`
- `comments`
- `status_timeline`
- `resolution_proofs`
- `notifications`
- `follows`
- `issue_upvotes`
- `issue_downvotes`
- `issue_followers`
- `badges`
- `rewards`
- `push_tokens`
- `levels`

9.2 Geospatial Model

- `issues.location_coords` as `GEOGRAPHY(Point, 4326)`.
- Also stores `location_longitude` and `location_latitude` for direct usage.

9.3 Referential Integrity

- most relation tables use foreign keys with cascade behaviors where appropriate (e.g., comments/upvotes tied to issues/users).

---

10) Role-Based Use Cases and Expected Outcomes

10.1 Citizen — Core Use Cases

Use Case A: Submit a normal civic issue

Flow: Report issue → includes title/category/location/media → submit.
Expected outcome:

- issue created with status `Submitted`,
- timeline entry created,
- user gets points (+10),
- report appears in feed/map,
- notification may be generated.

Use Case B: Submit emergency issue

Expected outcome:

- issue flagged `emergency=true`,
- severity tends toward critical in current logic,
- appears visually emphasized in map/feed/detail.

Use Case C: Track progress

Expected outcome:

- user sees status timeline updates,
- gets notifications on updates/comments/upvotes,
- if resolved, sees proof + remarks.

Use Case D: Engage socially

Expected outcome:

- upvote/downvote and comment affect engagement counters,
- issue owner receives activity notifications,
- follow issue enables update tracking.

10.2 Admin / Super Admin — Core Use Cases

Use Case A: Triage queue

Flow: open dashboard high-priority feed.
Expected outcome: quick visibility into critical/high issues.

Use Case B: Update status

Flow: set `Acknowledged` / `InProgress` / `Resolved`.
Expected outcome:

- issue status changes,
- timeline entry appended with admin context,
- owner notified,
- owner rewards incremented on resolved path.

Use Case C: Assign work

Flow: assign department and optional worker/deadline.
Expected outcome:

- assignment metadata stored,
- status auto-acknowledged if previously submitted,
- assigned worker receives notification.

Use Case D: Manage municipal communication

Flow: create municipal page and post official updates.
Expected outcome:

- update appears in municipal feed/profile,
- citizens can follow pages and consume official communications.

10.3 Field Worker — Core Use Cases

Use Case A: Start assigned task

Expected outcome: status moves to `InProgress`, timeline updated.

Use Case B: Complete with proof

Flow: upload completion photo + resolve status.
Expected outcome:

- issue marked `Resolved`,
- resolution proof stored,
- worker performance counters updated,
- citizen receives resolution benefits/points.

Use Case C: Navigate to site

Expected outcome: opens native map intent with issue coordinates.

10.4 Municipal Page Entity — Expected Use

- Represents official civic authority voice in-app.
- Publishes announcements/updates/completion notices.
- Gains followers; users receive regular visibility via municipal feed and page view.

---

11) Notification and Messaging Behavior

11.1 In-App Notifications

Generated for:

- onboarding welcome,
- badge events,
- report submission confirmations,
- status updates,
- upvotes/comments,
- assignments.

11.2 Push Notifications (FCM)

- Device push tokens are registered via frontend and stored in `push_tokens`.
- Backend notification service attempts multicast send through Firebase Admin.
- Invalid tokens are cleaned up when known failure codes are returned.

---

12) Security and Access Control

12.1 Backend Route Protection

- `protect` middleware validates JWT and injects normalized user object.
- `admin` guard: allows `admin` + `super_admin`.
- `fieldWorker` guard: allows `field_worker` + `admin`.

12.2 Important Security Expectations

- Supabase frontend uses publishable/anon key (expected for client apps with proper RLS).
- Backend auth and RBAC must be authoritative for privileged actions.
- SQL schema and RLS policy hardening should align for production-grade data isolation.

---

13) AI Layer Status (Current vs Intended)

13.1 Current Implementation

AI modules in backend are **mock/stub behavior**:

- issue detection returns always valid with randomized category,
- severity scoring uses static mapping,

- duplicate check returns false.

13.2 Product Expectation

Intended architecture hints at integrating true AI services (vision, duplicate clustering, severity ranking), but this is currently placeholder logic.

---

14) Observed Operational Characteristics

1. **Graceful fallback patterns** are present in many flows (dummy municipal posts, cached location, polling re-sync).
2. **Role-driven navigation** is strongly implemented in client routing.
3. **Feature completeness is high for MVP+**, especially across citizen/admin/worker lifecycle.
4. **Some hardcoded/dev-oriented values exist** (LAN IP, demo municipal content, static badge definitions fallback).

---

15) Expected End-to-End Scenario Walkthroughs

15.1 Citizen report to closure

1. Citizen logs in and completes setup.
2. Reports pothole with photo + GPS.
3. Issue appears in feeds/map with status `Submitted`.
4. Admin acknowledges and assigns roads team.
5. Worker marks `InProgress` then `Resolved` with proof.
6. Citizen sees timeline progression and before/after evidence.
7. Citizen receives points and notifications.

15.2 Municipal communication cycle

1. Admin creates municipal page.
2. Admin publishes official update post.
3. Citizens follow the page.
4. Updates appear in municipal feed and profile.

15.3 Worker task execution cycle

1. Worker opens assigned tasks dashboard.
2. Starts work, then marks complete with photo.
3. System records resolution proof and updates counters.

---

16) Non-Functional Expectations (Current State)

- **Responsiveness:** polling and optimistic UI patterns used in key screens.
- **Reliability:** retry/fallback for location and notification UI re-sync on API failure.
- **Scalability readiness:** modular route/data layers support growth, though some flows are still MVP-level implementations.

- **Auditability:** status timeline provides transparent change history at issue level.

---

17) Known Gaps / Improvement Opportunities

1. **AI engine is mocked** and should be replaced with production inference services.
2. **Notification service file shows iterative comments/cleanup opportunities** but functional intent is clear.
3. **Some frontend demo data/hardcoded values** should be externalized for production.
4. **Dark mode setting currently fixed to dark-only experience.**
5. **Security hardening checklist** should include strict Supabase RLS validation and secret handling audit.

---

18) Module-by-Module Inventory Checklist

Backend Modules

- Auth routes ✅
- Issue routes ✅
- Workflow routes ✅
- Gamification routes ✅
- Notification routes ✅
- Municipal routes ✅
- User routes ✅
- Auth middleware ✅
- Store data access layer ✅
- Notification service + Firebase config ✅
- SQL migrations (core + push token) ✅

Frontend Modules

- Auth flow screens ✅
- Setup flow (location + profile) ✅
- Role-based navigators ✅
- Home feed (community/municipal/reels/filter) ✅
- Report issue workflow ✅
- Map screen ✅
- Issue detail + timeline/proof/comments ✅
- Admin dashboard ✅
- Worker dashboard ✅
- Municipal profile ✅
- Notifications center ✅
- Profile and settings ✅
- API service wrappers ✅
- Location and notification service helpers ✅

---

19) Final Assessment

UrbanFix AI is a **substantially implemented civic operations platform** with a clear role-segmented architecture and full issue lifecycle support from reporting to resolution proof.

The app is already strong for a production-oriented MVP in:

- role-based operations,
- issue workflow orchestration,
- civic engagement features,
- map and timeline visibility,
- gamification and notifications.

Main next evolution areas are:

- replacing mocked AI with real services,
- tightening production config/security details,
- reducing hardcoded/demo data surfaces,
- formalizing operational observability and QA automation.

---

20) Appendix: Key Concepts Dictionary

- **Issue:** Any reported civic problem or official municipal update record.
- **Status Timeline:** Ordered history of status transitions and comments.
- **Resolution Proof:** Post-resolution media + remarks evidence.
- **Municipal Page:** Admin-created official authority profile for public updates.
- **Following (Issue):** User subscribes to issue updates.
- **Following (Page):** User subscribes to municipal page updates.
- **Priority Score:** Numeric urgency signal used in queueing/visibility.
- **AI Severity:** Classified severity bucket (`Low/Medium/High/Critical`).

End of report.