EPIC: Citizen Issue Reporting & Resilience

Executive Summary

A mobile-first reporting flow that captures accurate, verifiable infrastructure issues with strong location integrity, offline resilience, and clean evidence handling.

Stakeholders & Value

User Personas

- **Primary Persona:** Citizen (concerned resident, daily commuter).
- **Stakeholders:** Government Admins, Field Workers, System Admin.

User Value

Citizens can report issues in seconds with high trust that their report is accurate, accepted, and tracked to completion.

Goal & Vision

Deliver a seamless, location-locked reporting experience that avoids spam, enforces evidence integrity, and supports low-connectivity conditions.

Scope

In Scope

- GPS-locked reporting with accuracy indicators and a 5m confirmation constraint.
- Photo upload (Camera or Gallery) with EXIF validation (time <= 7 days; location within 5m of device location).
- Silent duplicate detection and report-count aggregation (no duplicate disclosure in reporting UI).
- Offline store-and-forward.
- Dynamic category fetch from backend.
- "My Reports" dashboard with status tracking and resolution feedback (like/dislike).
- Email OTP login and Google OAuth only (no SMS/phone OTP).
- OpenStreetMap base tiles (Mapbox optional for enhanced tiles).

Out of Scope

- SMS/Email notifications (in-app only).
- Video reporting.
- Remote pin placement outside the 5m radius.
- City-wide analytics or heatmaps (handled in analytics spec).

Success Metrics

- High upload success rate in low-bandwidth zones.
- Reduced duplicate ticket creation via backend deduplication.
- Fast report submission time.

Stories Under This Epic

USER STORY 1: Location Confirmation & Silent Duplicate Check

Executive Summary

Confirm the user's current location within a strict 5m radius and perform duplicate detection in the background without exposing existing issues on the reporting screen.

User Persona & Problem Statement

Who: As a Citizen standing in front of a pothole... **Why:** I want to report quickly without seeing other people's reports, while the system still prevents duplicates behind the scenes.

Scope (In & Out)

In Scope

- Location confirmation map with 5m lock.
- GPS accuracy validation.
- Silent duplicate check and report-count incrementing.
- OpenStreetMap base tiles (Mapbox optional).

Out of Scope

- Displaying nearby issue markers or report counts on the reporting screen.
- Editing or commenting on existing reports.

Features & Acceptance Criteria

Feature: Location Confirmation (5m Lock)

User Story: As a Citizen, I want to confirm my exact location, so the report is tied to the correct spot.

Acceptance Criteria:

- [ ] Verify that when the user opens the report flow, the map centers on current GPS location.
- [ ] Verify that the UI shows a "Confirm Location" step with address text (Swiggy-like confirmation).
- [ ] Verify that the pin is fixed to current location and cannot be moved beyond 5m.
- [ ] Verify that if GPS accuracy is worse than 5m, the user is prompted to retry or wait.

Feature: Silent Duplicate Check

User Story: As a System, I want to detect nearby duplicates without showing them to the user, so reporting remains simple while data stays clean.

Acceptance Criteria:

- [ ] Verify that the reporting screen does NOT display any existing issue markers or report counts.
- [ ] Verify that the system checks for existing issues within a 5m radius before creating a new issue.
- [ ] Verify that if a match is found, the report is appended to the existing issue and the report_count increments by 1.
- [ ] Verify that the user sees a normal success confirmation without disclosure that the issue already exists.
- [ ] Verify that if no match is found, a new issue is created.

UI/UX Design & User Flow

Flow: Home -> Report Issue -> Confirm Location -> Capture/Select Photo -> Submit -> Success.

Functional Requirements

Location Services:

- Subscribe to the device GPS provider.
- Ignore cached locations older than a defined freshness threshold.

Duplicate Matching (Silent):

- Call `GET /issues/nearby` with `latitude` , `longitude` , `radius=5` , `status_exclude=CLOSED,RESOLVED` .
- Do not render the response in the UI. Use it only for matching.
- If a match exists, use `POST /issues/{id}/evidence` to append evidence and increment `report_count` (evidence_type=REPORT).

Map Constraints:

- Validate that distance between device GPS and confirmed pin is <= 5m.

Map Provider:

- Use OpenStreetMap tiles by default; Mapbox can be configured as an optional provider.

Edge Cases

- **GPS Drift:** If GPS drift exceeds 5m, the user is blocked from submitting until accuracy improves.
- **Hidden Duplicate:** Users may unknowingly report an existing issue; backend merges into the existing ticket.
- **Permission Denied:** If location permissions are denied, block access to the report flow.

USER STORY 2: Create New Infrastructure Report

Executive Summary

Capture high-quality evidence for a new report while enforcing EXIF-based validation for time and location integrity.

User Persona & Problem Statement

Who: As a Citizen... **Why:** I want to report a hazard with a valid photo so it gets accepted and routed correctly.

Scope (In & Out)

In Scope

- GPS-only location capture.
- Photo upload from Camera or Gallery.
- EXIF validation (timestamp <= 7 days; location within 5m of device GPS).
- Category selection and optional description.

Out of Scope

- Video upload.
- Audio notes.

Features & Acceptance Criteria

Feature: Evidence & Location Capture

User Story: As a Citizen, I want to submit a photo and category so the correct department is notified.

Acceptance Criteria: Location:

- [ ] Verify that the issue location is set to device GPS coordinates.
- [ ] Verify that the UI displays an accuracy indicator (e.g., "Accuracy: +/- 5m").
- [ ] Verify that the user cannot submit if the location accuracy is worse than 5m.

Photo:

- [ ] Verify the user can select an image from the Gallery or capture via Camera.
- [ ] Verify at least one photo is mandatory.
- [ ] Verify EXIF timestamp is within 7 days of submission.
- [ ] Verify EXIF location is within 5m of current device GPS.
- [ ] Verify photos with missing/invalid EXIF metadata are rejected with a recapture prompt.

Category:

- [ ] Verify categories are fetched dynamically from `GET /categories`.
- [ ] Verify the form cannot be submitted without a category selected.

Assignment:

- [ ] Verify backend calculates zone based on lat/long and assigns to the correct authority.

UI/UX Design & User Flow

Flow: Capture/Select Photo -> Report Details Form -> Submit -> Success or Error.

Functional Requirements

Category Management:

- Call `GET /categories` on form load.
- Cache the response for a defined duration.
- Fallback to a safe default list if API fails.

Data Payload Construction:

- Submit a multipart/form-data POST to `/report` with fields:
  - `latitude`, `longitude`, `accuracy_meters`
  - `category_id`, `description`, `photo`
  - `timestamp`, `exif_timestamp`, `exif_lat`, `exif_long`

Image Processing:

- Compress images to stay within size limits.
- Device GPS remains the source of truth for issue location.
- EXIF metadata is used only for validation (not for location placement).

Backend Assignment Logic:

- Perform point-in-polygon to determine assigned zone.
- Create ticket with status `REPORTED` and assign to organization.

Edge Cases

- **Gallery Mismatch:** Gallery photos from another location fail EXIF validation and are rejected.
- **Offline Capture Delay:** If upload is delayed beyond 7 days, the backend rejects the report.

USER STORY 3: Offline Reporting & Sync

Executive Summary

Ensure reports are not lost when connectivity drops by storing payloads locally and syncing later.

User Persona & Problem Statement

Who: As a Citizen in a low-connectivity zone... **Why:** I do not want to lose a report when the network fails.

Scope (In & Out)

In Scope

- Local persistence of reports.
- Background retry logic.
- Pending upload status indicators.

Out of Scope

- Peer-to-peer sync.

Features & Acceptance Criteria

Feature: Store-and-Forward

User Story: As a System, I want to queue reports locally if the API is unreachable, so data is not lost.

Acceptance Criteria:

- [ ] Verify failed `POST /report` calls are saved to local storage.

- [ ] Verify "Pending Upload" status is shown in "My Reports".
- [ ] Verify background retry occurs on connectivity restore.
- [ ] Verify local copies are cleared after successful upload.

UI/UX Design & User Flow

Flow: Submit -> Offline Detected -> Saved to Outbox -> Auto Sync -> Status Updated.

Functional Requirements

Local Persistence:

- Use a local DB (SQLite/Realm/AsyncStorage).
- Store payload with `retry_count` initialized to 0.

Network Detection & Retry:

- Listen for connectivity changes.
- Retry queued uploads on reconnect.

Storage Management (Minimal NFR):

- Cap local storage and warn user before accepting new offline reports.

Edge Cases

- **Long Offline Period:** Reports older than 7 days will fail EXIF validation when synced.

USER STORY 4: Citizen Dashboard - My Reports & Feedback

Executive Summary

Provide a personal dashboard to track issue status and submit feedback on resolved work.

User Persona & Problem Statement

Who: As a Citizen... **Why:** I want to see progress on my reports and rate the quality of fixes.

Scope (In & Out)

In Scope

- List of user reports with status tracking.
- Detail view with before/after photos.
- Like/Dislike feedback on resolved or closed issues.

Out of Scope

- Direct chat with authorities.
- City-wide analytics maps.

Features & Acceptance Criteria

Feature: Status Tracking

User Story: As a Citizen, I want to track my reported issues.

Acceptance Criteria:

- [ ] Verify the list shows issues submitted by the logged-in user.
- [ ] Verify each item shows status: REPORTED -> ASSIGNED -> IN_PROGRESS -> RESOLVED -> CLOSED.
- [ ] Verify resolved/closed items show the "After" photo.
- [ ] Verify dismissed items show the rejection reason if available.

Feature: Resolution Feedback (Like/Dislike)

User Story: As a Citizen, I want to rate the completed fix so authorities can be evaluated.

Acceptance Criteria:

- [ ] Verify Like/Dislike controls appear only for RESOLVED or CLOSED issues.
- [ ] Verify a user can submit only one vote per issue.
- [ ] Verify the vote is stored with user_id and issue_id.
- [ ] Verify aggregated like/dislike counts are available to authority and analytics dashboards.

UI/UX Design & User Flow

Flow: My Reports -> Select Report -> View Timeline -> Submit Like/Dislike.

Functional Requirements

Data Retrieval:

- `GET /user/{user_id}/reports` with pagination.
- Sort by `created_at` desc.

Status Enumeration:

- REPORTED -> "Report Sent"
- ASSIGNED -> "Authority Notified"
- IN_PROGRESS -> "Work Started"
- RESOLVED -> "Fix Verified"
- CLOSED -> "Closed"
- DISMISSED -> "Rejected" (show reason)

Feedback Submission:

- `POST /issues/{id}/feedback` with payload `{ user_id, vote: LIKE|DISLIKE }`.
- Prevent duplicate votes per user.

Success Metrics

- Higher citizen re-engagement after resolution.
- Increased feedback submission rate for closed issues.