International Journal of Scientific Research in Engineering & Technology
Volume6, Issue6 (November- December 2026), PP: 11-19.
https://www.doi.org/10.59256/ijsreat.20260606003
www.ijsreat.com
ISSN No: 2583-1240

Smart City Assistance and Discovery Platform: GIS Discovery and Grievance Management

Aditya S Kallur[1] , Eshwar Ashok[2], Gagan P S[3], Sheryas S Hatture[4], Chaitra K G[5]
[1, 2, 3, 4]Department of Information Science and Engineering Bapuji Institute of Engineering and Technology, Davanagere, Karnataka, India.
[5] Project Guide, Department of Information Science and Engineering, Bapuji Institute of Engineering and Technology, Davanagere, Karnataka, India.

To Cite this Article: *Aditya S Kallur[1] , Eshwar Ashok[2], Gagan P S[3], Sheryas S Hatture[4], Chaitra K G[5], "Smart City Assistance and Discovery Platform: GIS Discovery and Grievance Management", International Journal of Scientific Research in Engineering & Technology, Volume 06, Issue 06, November-December 2026, PP:11-19.*

Copyright: ©2026 This is an open access journal, and articles are distributed under the terms of the Creative Commons Attribution License; Which Permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

Abstract: *Residents of growing Indian cities struggle to find reliable information about public services and to get civic complaints noticed, prioritised and resolved. This paper presents the Smart City Assistance and Discovery Platform, a full-stack web system built as a pilot for Davanagere, India, that joins both functions in one application. Citizens search 31 verified civic facilities in eight sectors on a Leaflet.js and OpenStreetMap map, report issues with GPS coordinates and photographs, and track each complaint from Pending to Resolved. A rule-based Civic Intelligence Engine analyses every complaint on submission: it scores severity, routes the complaint to a municipal department, flags nearby duplicates using the Haversine distance, computes a priority, assigns a service-level deadline of 24 to 336 hours, and proposes an action checklist. Staff dashboards add hotspot detection, a civic score and weekly trends. All rules, weights and thresholds are explicit constants, so every decision can be reproduced by hand. The platform uses Node.js, Express, MongoDB, JWT and bcrypt. An automated suite of 39 assertions and a 23-check verification script passed, and 19 of 20 API test cases passed; one case that needs a live database was not run. The work contributes toward SDG 11 (Sustainable Cities and Communities); it has not yet been evaluated with real municipal data or in a user study.*

Keywords: *smart city, grievance redressal, GIS service discovery, rule-based decision support, duplicate detection, hotspot detection, SDG 11*

I. INTRODUCTION

Cities depend on many public services: hospitals, police stations, transport hubs, colleges, markets, government offices and utilities. In most cities, information about these services is spread across printed notices, word of mouth and general web searches, so a citizen who needs a service urgently has no single reliable place to look. Everyday civic problems such as potholes, broken streetlights, uncollected garbage, drainage blockages and water-supply failures are reported in the same informal way, by phone calls, visits to the municipal office or messages. Such complaints are hard to track, and the citizen rarely learns whether the problem was noticed, assigned or resolved.

Research on smart-city complaint systems addresses parts of this, such as authenticated submission, automatic routing and escalation [1], but few combine service discovery, evidence-based reporting, prioritisation and municipal analytics in one lightweight platform. The proposed Smart City Assistance and Discovery Platform is a full-stack web application developed for Davanagere with one design rule: every decision the system makes about a complaint is produced by explicit, inspectable code, with published weights and thresholds, and not by an opaque model.

A. Problem Statement

Three problems recur. First, discovery is fragmented: contact numbers, working hours and locations of facilities are kept in different places and may be out of date. Second, reporting is informal: a complaint made by phone does not capture the exact location or photographic evidence, and it cannot be followed afterwards. Third, handling is unstructured: when every complaint is treated alike, a dangerous open drain near a school can wait as long as a cosmetic defect, wrongly routed complaints are delayed further, and repeated complaints from one area are not noticed as a pattern. This project thereforebuilds one platform for service discovery, evidence-based reporting, rule-based prioritisation and municipal analytics.

B. Objectives

The main objectives of the platform are:

(1) To provide map-based discovery of verified civic facilities (31 facilities in eight sectors) with address, contact number, hours and services.

(2) To let citizens report issues with a category, description, GPS location and photograph, and track each complaint from Pending to Resolved.
(3) To implement a Civic Intelligence Engine of pure functions for severity scoring, department routing, duplicate detection, priority, service-level deadlines and recommended actions, with all parameters published.
(4) To give municipal staff hotspot detection, a civic score, weekly category trends and a public summary free of personal data.
(5) To secure the system with JWT and bcrypt authentication, role-based access, validated uploads, rate limiting and an audit log, and to keep main flows usable if the database is unreachable.
(6) To evaluate the platform with automated assertions, a system verification script, API test cases and worked scenarios, and to report the results exactly as obtained.

C. Scope

The work is a pilot for Davanagere only. It covers six complaint categories (Road & Potholes, Streetlight, Garbage & Sanitation, Water Supply, Drainage, Other), citizen and staff roles, a keyword-based civic assistant named Nagara, and staff analytics. The engine supports human decisions: it flags and recommends, and never merges or deletes complaints or acts on its own. The platform is not a replacement for official municipal grievance procedures and has not been deployed with a municipality.

D. Contributions and Organization

The contributions are: (i) a unified platform joining GIS discovery with a complaint lifecycle that keeps a timeline, using one REST API and one database for citizens and staff; (ii) a reproducible, rule-based Civic Intelligence Engine; (iii) municipal analytics that return "insufficient data" instead of an invented number; (iv) a security and reliability layer; and (v) a reproducible verification record of 39 unit assertions, 23 system checks and 20 API test cases. Section II reviews related work, Section III presents the materials and methods, Section IV the results, Section V discusses limitations, and Section VI concludes.

II. RELATED WORK

A. Smart-City Complaint Management

Shanmugam and Dharani present an intelligent complaint management system for smart cities that uses authenticated submission, automatic routing of each complaint to the concerned department, and escalation [1]. This is close to the lifecycle and routing of our platform, which adds explicit priorities, deadlines and duplicate flags. Shama et al. describe CitySolution, a mobile application that classifies complaint photographs with a deep learning model and lets authorities monitor the complaints [2]. Image classifiers need labelled photographs and trained models; our platform needs neither, and stores the photograph as evidence, leaving image classification as future work.

B. Road Damage Detection

Anand et al. detect road cracks and potholes with computer vision [3], Maeda et al. classify road damage from smartphone images with deep neural networks [4], and Mertz et al. discuss city-wide road distress monitoring with smartphones [5]. These studies show that road problems can be found automatically from images. Our severity module reads only the text of a complaint, so these works mark a direction for photo-based issue detection.

C. Civic Chatbots

A damopoulou and Moussiades review rule-based and machine-learning chatbots [6]. Rule-based interaction suits focused domains, which supports Nagara, our civic assistant, answering guided questions from the facility data. Chaidrata et al. describe an intent-matching chatbot with natural language understanding [7], the route we expect to take to move Nagara beyond keyword rules.

D. Open Geodata and Web Foundations

Maier describes OpenStreetMap as a freely available geospatial source [8], which makes it suitable for the map layer without licence cost. Chopra et al. discuss a token-based authentication scheme for mobile applications [9], supporting our use of JWT. Romaniuk and Shapovalova discuss JavaScript full-stack development with MongoDB, Express and Node.js [10], and Fielding's architectural style underlies the REST interface [11]. The OWASP Top 10 [12] guided input validation, rate limiting, secure headers and password hashing.

Table I compares capabilities as described in the two closest studies [1], [2] with this work. A dash means the capability is not described in the summaries we reviewed, not that the system lacks it.

Capability	[1]	[2]	This work
Authenticated complaint submission	Yes	–	Yes (JWT)
Automatic department routing	Yes	–	Yes (rules)
Escalation	Yes	–	No; deadline status only
Photo classification (deep learning)	–	Yes	No; photo kept as evidence
Authority monitoring	–	Yes	Yes (dashboard)
GIS service discovery	–	–	Yes (31 facilities)
Duplicates, hotspots, civic score	–	–	Yes

Table I. Capability comparison

In summary, existing work treats routing, image-based classification, road-damage detection, chatbots and mapping separately. The platform here integrates service discovery and grievance handling in a lightweight system that needs no training data or special hardware.

III. MATERIAL AND METHODS

A. Existing System

In most municipalities, service information is found through printed notices and general web searches, and complaints are made by telephone, office visits or messages. Computerised complaint systems in the literature, such as [1] and [2], cover submission, routing and monitoring; as far as the studies we reviewed describe, they do not also offer map-based facility discovery, duplicate detection or area-level analytics.

The limitations addressed by the proposed platform are:

- Fragmented service information whose contact details and hours may be out of date.
- Informal complaints with no exact location, photographic evidence or tracking.
- Uniform handling of complaints, with no severity, priority or service-level deadline.
- Wrongly routed complaints, and repeated complaints from one area not recognised as a pattern.
- Separate tools for service discovery and grievance handling.

These are problem areas identified from the literature and the authors' observation; no comparison with deployed municipal systems was carried out.

B. Proposed System

The proposed platform joins both functions in one web application. Citizens search facilities on a map, report problems with GPS coordinates and photographs, and track complaints through a timeline. When a complaint is submitted, the Civic Intelligence Engine scores its severity, routes it to a department, flags nearby duplicates, computes a priority, assigns a service-level deadline and proposes an action checklist. Staff see these results in a command-center dashboard with hotspots, a civic score and weekly trends. Every rule, weight and threshold is an explicit constant, so each decision can be reproduced by hand.

C. Technologies Used

Technology	Purpose
HTML5, CSS3, JavaScript	Browser interface: map, report form, staff panel and Nagara assistant
Leaflet.js 1.9.4, OpenStreetMap	Interactive map and map tiles; the only external service
Node.js, Express	Application layer with 17 REST endpoints over JSON
MongoDB, Mongoose	Storage of User, Report, Poi and AuditLog (local or Atlas)
JWT, bcrypt	Token-based login with role claim; password hashing
Helmet, express-validator, rate limiting	Secure headers, input validation and request throttling
Civic Intelligence Engine	Pure JavaScript functions for severity, routing, duplicates, priority, deadlines and actions
Haversine formula	Distance for duplicate detection and hotspot clustering
Node.js assertions, verification script	Automated testing of the engine and the system

Table II. Technologies used

D. System Architecture

The platform is a four-layer web application (Fig. 1). The presentation layer runs in the browser and is written in HTML5, CSS3 and vanilla JavaScript [13]; it holds the interactive map, the report form, the staff panel and the Nagara assistant. The application layer is a Node.js [14] and Express service exposing 17 REST endpoints over JSON for authentication, reports, facilities (points of interest, POIs) and analytics. The Civic Intelligence layer holds the rule-based modules and is called by the reports route at submission and by the analytics route on request. The data layer stores four Mongoose models, User, Report, Poi and AuditLog, in MongoDB [15], locally or on MongoDB Atlas; complaint photographs are kept in an uploads folder, and OpenStreetMap tiles are the only external service.

The flow of one complaint through these layers is shown in Fig. 2: the citizen logs in and receives a token, submits the report, the engine analyses it, and the citizen later tracks its status and timeline. In the code, the analysis runs inside the creation request and its result is stored with the complaint in one database write.

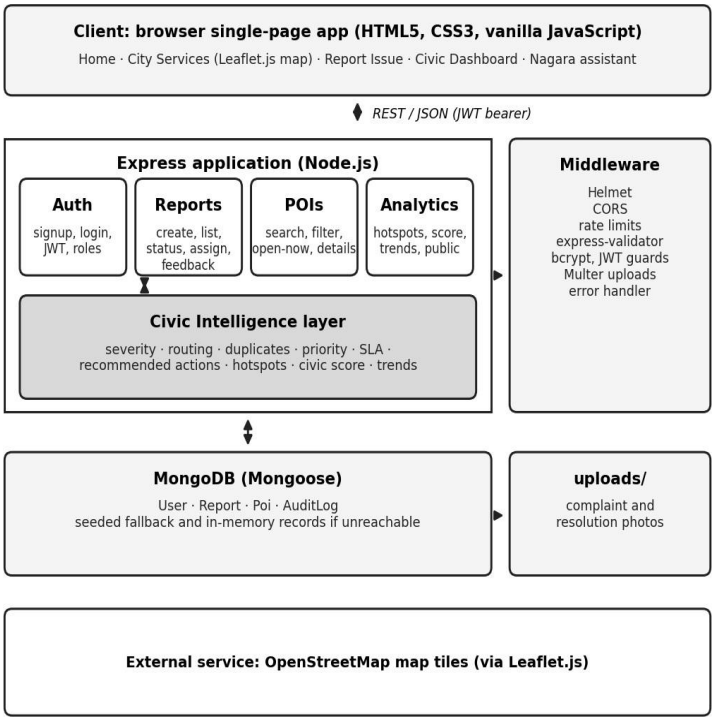


Fig. 1. System architecture of the Smart City platform. The Civic Intelligence layer is the core of the system.

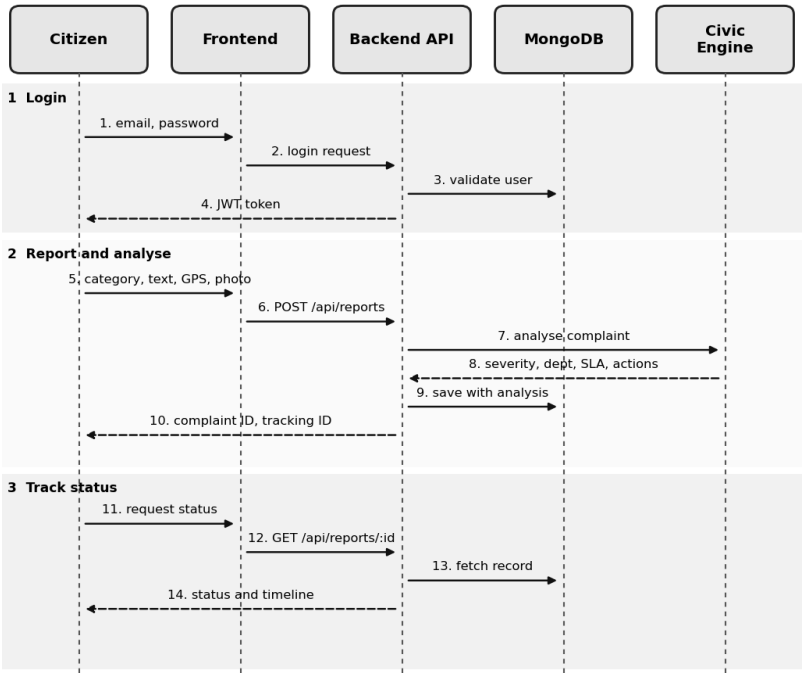


Fig. 2. Sequence of login, complaint submission, analysis and status tracking.

Table III lists the four collections, and Table IV groups the 17 endpoints by who may call them.

Collection	Main fields
User	Name, unique email, hashed password, role (citizen or staff), creation time
Report	Title, category, description, location, photo URLs, severity and priority with scores, department, SLA hours and deadline, recommended actions, possible duplicates, status, timeline, resolution time, feedback, reporter, assignee
Poi	Name, category, coordinates, address, contact, hours, services, rating, personnel, aliases, category-specific fields
Audit Log	Action, entity type and id, user, role, details, time

Table III. Database Collections and Main Fields

Access	Count	Endpoints
Public	9	Signup, login; facility list and details; complaint list, statistics and single complaint; public analytics summary; health check
Logged-in	2	Create complaint; feedback on a resolved complaint (original reporter only)
Staff only	6	Status update; department assignment; hotspots; civic score; trends; command center

Table IV. Rest endpoints by access level

Security follows common practice [12]. Passwords are hashed with bcrypt (10 salt rounds). A login returns a JWT with a seven-day lifetime carrying the user id and role, and two guards enforce access: one requires a valid token and the other requires the staff role. Helmet sets secure headers, cross-origin access is limited to the configured client origin, and rate limits allow 1,000 requests per 15 minutes in general and 60 per 15 minutes on the authentication routes. JSON bodies are limited to 1 MB and inputs are validated with express-validator. Photographs must be JPEG, PNG, WebP or GIF files of at most 5 MB; the stored name is random and the extension comes from the verified MIME type, not from the client's file name. The error handler does not return stack traces. If MongoDB is unreachable, the server retries the connection every five seconds and meanwhile serves the seeded facilities and in-memory records, so the main flows stay usable. Report creation, status changes, assignment and feedback are written to an append-only audit collection; a failed audit write does not stop the request.

The code has about 1,600 lines of backend application code (routes, models, middleware and intelligence modules), about 7,300 lines of frontend HTML, CSS and application JavaScript, and about 1,100 lines of test and verification scripts.

E. Implemented Modules

- **Service Discovery:** searches 31 verified facilities in eight sectors on a Leaflet.js map, with an open-now status computed on the server.
- **Nagara Assistant:** answers guided questions by matching keywords against the facility list loaded from the API.
- **Authentication and Access Control:** JWT and bcrypt login with citizen and staff roles enforced on the server.
- **Complaint Management:** submission with category, GPS location and photograph, status lifecycle with timeline, and citizen feedback.
- **Civic Intelligence Engine:** severity, department routing, duplicate detection, priority, deadline and recommended actions.
- **Municipal Analytics:** hotspots, civic score, weekly trends, command center and a public summary without personal data.
- **Audit and Reliability:** append-only audit log and a fallback mode when the database is unreachable.

F. Service Discovery and Complaint Lifecycle

Discovery. The interface uses a State, District and City selection model so that other jurisdictions can be added; at present only Davanagere is configured, with the map centred on (14.4644, 75.9218). The 31 facilities belong to eight sectors (hospital, police, education, transport, market, government, utility and emergency). Each record holds address, contact number, hours, services, rating, coordinates and, where available, a staff directory. A search builds a case-insensitive filter over a facility's name, category, address, description, services and aliases. The server computes an open-now flag from the current hour: a 24×7 facility is open; a facility with unknown hours returns unknown rather than a guess; if the opening hour is not later than the closing hour the facility is open when the hour lies in [open, close); otherwise the window crosses midnight and the facility is open when the hour is at least the opening hour or below the closing hour. Results are sorted with open facilities first and then by rating. If the database returns nothing, the seeded list is filtered by the same rules.

Lifecycle. A logged-in citizen submits a title, one of six categories, a description, a location label, optional GPS coordinates and an optional photograph. The record starts as Pending with a "Report submitted" timeline entry. Staff change the status to In Progress or Resolved; each change appends the status, a note, the staff id and the time, and resolution sets the resolution time and may attach a resolution photograph. A citizen can rate a resolved complaint, which the API accepts only from the original reporter. The tracking ID shown to the citizen is the last six characters of the record id.

G. Civic Intelligence Engine

Fig. 3 shows how a complaint is processed. When it is submitted, the reports route runs six pure functions in a fixed order inside the same request and stores their results together with the complaint in one database write.

1) *Severity scoring.* The title and description are joined and lower-cased. The score s counts matches in three keyword lists (25 hazard terms such as open drain, exposed wire, gas leak, flood and children; 17 medium terms such as overflow, broken, not working and blocked; 8 minor terms such as small, minor and cosmetic), plus scale modifiers w_m, as in (1), where n_h, n_m and n_l are the match counts.

$$s = \min(100,\ 30n_h + 15n_m + 3n_l + w_m) \qquad (1)$$

A modifier adds 10 for words such as entire or multiple, 8 for a problem pending for days or weeks, and 8 for a main road or junction. If only minor terms match, s is capped at 20. The level is high if any hazard term matches or $s \geq 45$, medium if any medium term matches or $s \geq 15$, and low otherwise.

2) Department routing. Hazard patterns are checked first and override the category: an exposed or live wire or transformer goes to Electrical & Lighting, a gas leak or fire to Emergency Services, a sinkhole or structural collapse to Roads & Infrastructure, and a flood or open drain to Drainage & Stormwater. Otherwise a table maps Road & Potholes to Roads &

Infrastructure, Streetlight to Electrical & Lighting, Garbage & Sanitation to Sanitation, Water Supply to Water Works, Drainage to Drainage & Stormwater, and Other to General Administration. The route and its reason are stored.

3) Duplicate detection. A complaint with coordinates is compared with complaints of the same category from the last 72 hours. The great-circle distance d uses the Haversine formula in (2), with Earth radius $R = 6371$ km and latitudes φ and longitudes λ.

$$d = 2R \arcsin \sqrt{(\sin^2(\Delta\varphi/2) + \cos \varphi_1 \cos \varphi_2 \sin^2(\Delta\lambda/2))} \quad (2)$$

A candidate lies within 0.15 km (about 150 m). It is kept if the title overlap, the number of shared words longer than two letters divided by the smaller word count, is at least 0.5, or if it lies within 40% of the radius (60 m). The system only flags duplicates; it never merges or deletes complaints. Complaints without coordinates are skipped.

4) Priority and service-level deadlines. Priority P combines severity, category, age and duplicates as in (3), rounded and limited to 0–100, where w_{sev} is 5, 15 or 30 for low, medium or high severity, w_{cat} is the category weight, a is the age in hours and k the number of duplicates.

$$P = w_{sev} + 0.3s + w_{cat} + \min(0.5a, 25) + \min(5k, 20) \quad (3)$$

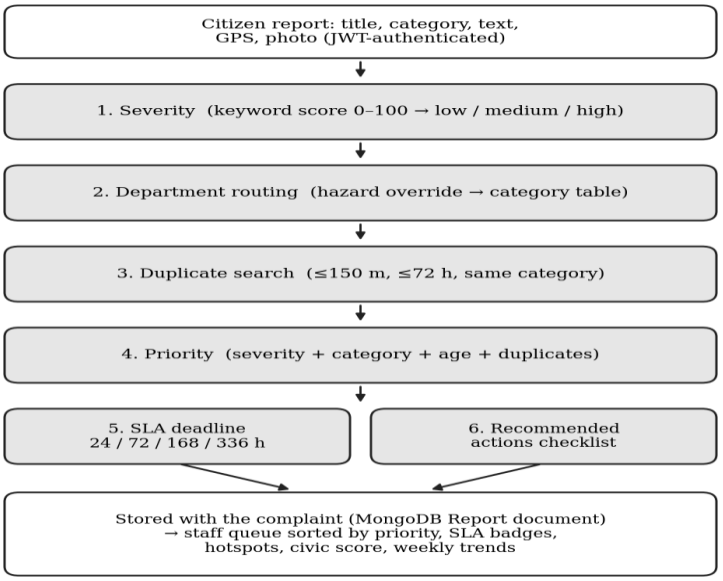


Fig. 3. Civic Intelligence pipeline executed within the complaint-creation request. Every stage is a pure function.

Table V lists the weights and Table VI the levels. The deadline is the creation time plus the hours in Table VI. The deadline status is overdue if it has passed, due soon if 24 hours or fewer remain, and on track otherwise.

Component	Value
Severity weight	Low 5, medium 15, high 30; plus 0.3 × severity score
Category weight	Water Supply 7; Road & Potholes 6; Drainage 6; Garbage & Sanitation 5; Streetlight 4; Other 3
Age	0.5 per hour, at most 25
Duplicates	5 per similar report, at most 20

Table V. Priority Weights

Level	Condition	SLA (hours)
Critical	High severity and $P \geq 55$	24
High	$P \geq 40$	72
Medium	$P \geq 18$	168
Low	Otherwise	336

Table VI. Priority Levels and Deadlines

5) Recommended actions. A static table gives three base actions for the category (for Road & Potholes: dispatch an inspection team, assess for temporary barricading, schedule a repair crew). High severity adds three actions, such as escalation to the duty supervisor, and critical priority adds two, such as dispatch within the hour. The list is advice for a human dispatcher; the system takes no action by itself.

H. Hotspots, Civic Score and Trends

Unresolved complaints with coordinates are grouped by greedy proximity clustering with a 0.4 km radius; a group of at least three is a hotspot, centred on the mean location. The civic score C needs at least five complaints, and otherwise returns

"insufficient data" and no number. With resolution rate r, on-time rate t among resolved complaints that have deadlines, and o overdue critical complaints, it is given by (4); if t is undefined, $80r$ replaces $50r + 30t$, and C is limited to 0–100.

$$C = 50r + 30t + 20 - \min(5o, 20) \quad (4)$$

The public summary rates C as Optimal (80 or more), Moderate (60 or more) or Needs Attention. Trends compare this week with last week per category, marking a change above 10% as up or down, and require at least four complaints. The public summary returns counts, the score and hotspot centres only, with no personal data.

I. Testing Methodology

The platform was tested against its functional requirements at three levels. At unit level, Node.js assertions exercise the intelligence modules and the open-now logic. At system level, a verification script checks the facility data, the location model, the engine, the complaint lifecycle, the civic score and hotspots, and the absence of hard-coded secrets in the frontend. At API level, the backend was started and called over HTTP with 20 test cases. Each test case records the input, the expected outcome, the actual outcome and a pass, fail or not-run status. The pass rate is calculated as (passed test cases ÷ total test cases) × 100, and a case that could not be executed is reported as not run, never as passed. Eight complaints written by the authors were also passed through the engine to show its behaviour. Because there is no field deployment and no labelled municipal data, severity precision and recall and user-level measures are not reported.

IV. RESULTS

A. Functional Testing

The Atlas database was not reachable in the test environment, so the server ran in fallback mode with in-memory data. Table VII lists the API test cases by group with the expected outcome and the result obtained.

Test IDs	Scenario	Expected outcome	Result
TC-01–07	Authentication and role checks	Wrong password, missing token and citizen access to a staff route are refused	7 / 7 Pass
TC-08–09	Facility listing and search	Facilities are listed and search returns matches	2 / 2 Pass
TC-10–11	Complaint with photo and GPS; invalid upload	Valid complaint is stored with its analysis; invalid upload is rejected	2 / 2 Pass
TC-12–15, 18	Severity, routing, deadline, duplicates, hotspots	Engine outputs follow the specified rules	5 / 5 Pass
TC-16–17	Status update and tracking	Staff status change is recorded and the citizen can track it	2 / 2 Pass
TC-19	Analytics access control and data	Staff-only analytics are protected; public summary returns data	1 / 1 Pass
TC-20	Feedback and audit records	Feedback from the reporter is accepted and audit entries are written	Not run (needs MongoDB)

Table VII. API test cases and results

B. Results Summary

Test level	Total	Passed	Failed	Not run
Unit assertions	39	39	0	0
System checks	23	23	0	0
API test cases	20	19	0	1

Table VIII. Summary of test results

All 39 assertions and all 23 checks passed, and both scripts were re-executed while preparing this paper. For the API tests, 19 of 20 cases passed (95% of the planned cases); every executed case passed and none failed, and the remaining case needs MongoDB and was not run. Manual assignment and the retrieval of neighbouring complaints for duplicate detection also read the database and were not exercised through the API. Each intelligence function is pure and has no database access, which is what makes direct testing possible. These results show that the system behaves as specified, not that it improves complaint resolution in practice.

C. Worked Scenarios

To show the engine's behaviour, eight complaints written by the authors were passed through the implemented functions (Table IX). The values are the program's output, not field data, and can be reproduced by hand from Tables V and VI. The open-drain case matches test case TC-10: severity 100, priority 66, critical, deadline 24 hours and eight actions.

#	Complaint	Severity	Dept.	Priority	SLA h
1	Small pothole, minor crack	Low (12)	Roads	Low (15)	336
2	Streetlight not working, broken pole	Medium (30)	Electrical	Medium (28)	168
3	Garbage pileup, foul smell, overflowing bins	High (60)	Sanitation	High (53)	72
4	Open drain near school gate, flooding	High (100)	Drainage	Critical (66)	24
5	No water since morning, entire street	Medium (25)	Water Works	Medium (30)	168
6	Live wire near bus stop, filed as Road	High (60)	Electrical	High (54)	72
7	Gas leak and fire risk, filed as Other	High (75)	Emergency	Critical (56)	24
8	Park bench, faded colour	Low (3)	General Admin.	Low (9)	336

Table IX. Engine output for eight scenarios

Scenario 6 shows the hazard override: a live wire filed under Road & Potholes goes to Electrical & Lighting. Its priority is 30 + 0.3 × 60 + 6 = 54, one point below the critical threshold of 55, which shows how sharply the thresholds act. Age and duplicates change the outcome as well. A fresh medium-severity complaint in Garbage & Sanitation (severity score 30) has priority 29, becomes high priority (41) after 24 hours unresolved, since age adds 12, and reaches high (44) when three duplicates are flagged.

D. Observed Application Functionality

Fig. 4 shows the staff Command Center with the six seeded sample reports: the lifecycle strip from citizen report to tracking, the totals (six reports, four pending, two resolved) and a resolution rate of 33%. The figures are sample data, not real complaints.

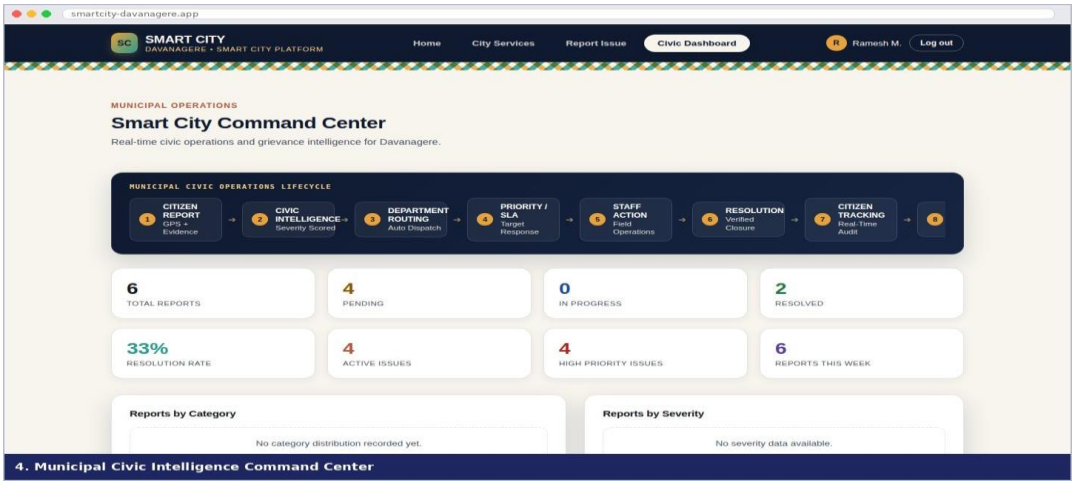


Fig. 4. Municipal Civic Intelligence Command Center populated with seeded sample reports.

V. DISCUSSION AND LIMITATIONS

Keeping the decision logic in explicit, deterministic functions nefits: any score can be reproduced by hand, a failure can be traced to a named rule, and the engine needs no training datgave three practical bea or special hardware. The integration of discovery, reporting, prioritisation and analytics in one platform gives citizens and staff a single point of access, which supports SDG 11 (Sustainable Cities and Communities).

The approach also has clear limits.

- **No field evaluation.** The platform has not been tested with real municipal complaints, staff or citizens. Weights, thresholds and deadlines are hand-set policy values, not calibrated against resolution outcomes.
- **Keyword rules.** Severity depends on English keywords, so unusual phrasing or non-English (for example Kannada) text can be scored wrongly. Overlapping phrases also add twice: “gas leak” and “leak”, or “overflow” and “overflowing”. Precision and recall have not been measured.
- **Actions follow the category.** Recommended actions are keyed to the complaint category, so a live wire filed under Road & Potholes is routed to Electrical & Lighting but receives road-repair actions.
- **Unverified database paths.** Feedback, manual assignment, audit writes and the retrieval of neighbouring complaints for duplicate detection need MongoDB and were not run in the API tests.
- **Access control.** The role is taken from the signup request, so staff accounts are not created through an approval step,

which a production deployment must add. There is no screen for feedback or for viewing the audit log, and data created in fallback mode is not persistent.

- **Scope.** The deployment covers Davanagere only and needs an internet connection. A service-provider role appears in the design diagrams but is not implemented; only citizen and staff roles exist. Citizens with poor connectivity or no GPS may supply weaker evidence, and complaints without coordinates are not checked for duplicates.

Privacy, ethics and reproducibility. The public analytics summary exposes only counts, the civic score and hotspot centres, never names or complaint text. Passwords are stored only as bcrypt hashes, protected routes check the role on the server, and sensitive actions are logged. Decisions stay with people: the engine flags duplicates but never deletes complaints, and its recommended actions are suggestions for a dispatcher. To reproduce the results, install Node.js 18 or newer and MongoDB (local or Atlas), set the connection string, token secret, port and client origin in the environment file, load the seed data, start the API on its default port and serve the frontend folder statically; the two test scripts then run from the command line.

VI. CONCLUSION

The Smart City Assistance and Discovery Platform shows that service discovery and grievance management can be combined in one lightweight, secure web application for a mid-sized city. Citizens find 31 verified facilities on a map and report problems with location and photographs, while a rule-based engine gives each complaint a severity, department, priority, deadline and action checklist that staff can verify by hand. Automated tests, API tests and worked scenarios support the claim that the platform is consistent and auditable, but they do not replace evaluation with real municipal data. Future work includes a pilot with municipal staff to calibrate thresholds against resolution times, machine-learning severity and duplicate detection trained on labelled complaints, deep learning photograph classification [2]–[5], an intent-based Nagara assistant [7], native mobile applications with SMS and WhatsApp notifications, multi-city support, municipal ERP integration, a service-provider module, and approval-based staff accounts with feedback and audit screens.

Acknowledgment

The authors thank their guide, Mrs. Chaitra K G, and the project coordinators, Dr. Arun Kumar G H and Mr. Santhosh T, for constant guidance; Dr. Vinutha H P, Head of the Department of Information Science and Engineering; and the Principal and management of Bapuji Institute of Engineering and Technology, Davanagere, for the opportunity to carry out this work. Generative AI (Claude, by Anthropic) was used to help draft and format this manuscript from the authors' project report, source code and test results; the authors reviewed and verified the content, results and references.

REFERENCES

1. G. Shanmugam and A. Dharani, "Revolutionizing civic complaints with intelligent complaint management system for smart city," Int. Res. J. Modernization Eng., Technol. Sci. (IRJMETS), vol. 7, no. 5, pp. 7518–7523, May 2025.
2. F. Shama, A. Aziz, and L. B. M. Deya, "CitySolution: A complaining task distributive mobile application for smart city corporation using deep learning," SoftwareX, vol. 27, Art. no. 101829, 2024, doi: 10.1016/j.softx.2024.101829.
3. S. Anand, S. Gupta, V. Darbari, and S. Kohli, "Crack-Pot: Autonomous road crack and pothole detection," in Proc. Digital Image Computing: Techniques and Applications (DICTA), 2018, pp. 1–6.
4. H. Maeda, Y. Sekimoto, T. Seto, T. Kashiyama, and H. Omata, "Road damage detection and classification using deep neural networks with smartphone images," Comput.-Aided Civil Infrastruct. Eng., vol. 33, no. 12, pp. 1127–1141, 2018.
5. C. Mertz, S. Varadharajan, S. Jose, K. Sharma, L. Wander, and J. Wang, "City-wide road distress monitoring with smartphones," in Proc. ITS World Congress, 2014, pp. 1–9.
6. E. Adamopoulou and L. Moussiades, "Chatbots: History, technology, and applications," Mach. Learn. Appl., vol. 2, 2020.
7. A. Chaidrata et al., "Intent matching based customer services chatbot with natural language understanding," in Proc. 5th Int. Conf. Commun. Inf. Syst. (ICCIS), 2021; arXiv:2202.00480, 2022.
8. G. Maier, "OpenStreetMap, the Wikipedia map," Region, vol. 1, no. 1, pp. R3–R10, 2014.
9. S. Chopra, A. Singh, and A. Singh, "An authentication based scheme for mobile applications using THJWT," in CEUR Workshop Proc., vol. 3244, paper 02, 2022.
10. O. V. Romaniuk and O. O. Shapovalova, "Use of the MERN stack in web application development," in Proc. Vinnytsia Nat. Tech. Univ. Conf., 2024.
11. R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, Univ. California, Irvine, CA, USA, 2000.
12. OWASP Foundation, "OWASP Top 10 – 2021: The ten most critical web application security risks," 2021.
13. Ecma International, ECMAScript 2015 Language Specification, ECMA-262, 6th ed., Geneva, Switzerland, 2015.
14. T. Hughes-Croucher and M. Wilson, Node.js: Up and Running. Sebastopol, CA, USA: O'Reilly Media, 2012.
15. Chodorow, MongoDB: The Definitive Guide, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2013.