ARJST
- Academic Research Journal of Science and Technology
e-ISSN: 3048-9644
publications.ngmc.ac.in

UGC Approved by Transparent Peer Review

Smart Civic Connect: A Role-Based Web Platform for Reporting, Assigning and Tracking Urban Infrastructure Complaints

Tohid Mansur Mulla[1,*]

[1] Dr. J. J. Magdum College of Engineering, Jaysingpur, India, Student, Department of MCA
***Corresponding author:** Tohid Mansur Mulla, tohidmulla834@gmail.com

ARTICLE INFO

Article history
Received: 20 November 2026
Accepted: 20 November 2026

Article ID
MS-2026-0002

Peer review
Double-blind peer review

Keywords
Civic Complaint Management
E-Governance
Smart City
Django
Role-Based Access Control
Geospatial Mapping
Citizen Participation

ABSTRACT

Potholes, overflowing garbage, dead streetlights and burst water mains are reported to local bodies every day, yet citizens rarely learn what happened to their complaint. The problem is less a shortage of repair crews than the lack of a shared, trackable channel between the three parties involved: the citizen who sees the defect, the administrator who decides who should fix it, and the worker who fixes it. This paper presents Smart Civic Connect, a full-stack web platform that connects citizens, municipal administrators and maintenance workers in one workflow. It is built with Python and Django, with Bootstrap, Leaflet.js and Chart.js on the client and SQLite or PostgreSQL for storage. It introduces a structured complaint model with ten civic categories, four priority levels, map coordinates and photographs; a seven-state lifecycle with a complete history of every transition; server-enforced role-based access; evidence-based resolution through a completion photograph; a public community feed and colour-coded map; an in-app notification service with a fail-safe e-mail fallback; and an administrative analytics dashboard with CSV export. We describe the architecture and implementation, summarise the automated test suite, and discuss limitations and future work.

How to cite this article: Tohid Mansur Mulla (2026). Smart Civic Connect: A Role-Based Web Platform for Reporting, Assigning and Tracking Urban Infrastructure Complaints. *ARJST*, *5*(07).
Copyright and licence: © 2026 The Author(s). This is an open access article distributed under the terms of the Creative Commons Attribution 4.0 International License (CC BY 4.0), which permits unrestricted use, distribution and reproduction in any medium, provided the original work is properly cited.

I. Introduction

Urban infrastructure fails in small, visible ways: a pothole on a main road, a streetlight that stays dark, a drain that overflows after every shower. Local bodies depend largely on citizens to notice such defects, and the common reporting channels are telephone helplines, written applications and visits to a ward office. These channels share three weaknesses. They leave no reliable record that the citizen can inspect, they give no indication of who is responsible for the repair, and they offer no proof when the work is finished. Smart-city initiatives have repeatedly identified responsive, information-driven local services as a central goal [1], and the gap between a reported problem and a verified repair is where that goal is most often lost.

A phone call or an e-mail can work for one complaint, but it does not scale. Reports get buried, two citizens report the same defect without knowing it, the administrator cannot see which crews are overloaded, and the citizen has no way to check progress. A worker, in turn, receives instructions verbally and has no structured way to report what was done.

Smart Civic Connect addresses this with a purpose-built platform. A citizen files a complaint in a few minutes with a category, description, photographs and a map location. An administrator reviews it, sets its priority and assigns it to a maintenance worker. The worker accepts the task, reports progress and uploads a completion photograph. At each step the citizen receives a notification, and every transition is stored in a permanent history. Complaints that the citizen marks as public also appear on a community feed and an interactive map, so that residents can see what is being reported and repaired in their area.

II. Objectives

The objectives of this paper are:

1) To give citizens a simple, structured way to report civic defects with photographs and an accurate location.

2) To make every complaint traceable from submission to closure through a defined lifecycle and a permanent history.

3) To give administrators one place to triage, prioritise, assign and review complaints and the workers handling them.

4) To require verifiable evidence of completed work and keep the citizen informed at each stage.

5) To protect private complaints and restrict each action to the role that is allowed to perform it.

III. Literature Review

Civic reporting systems generally fall into three groups: traditional channels such as helplines, written applications and ward-office visits; municipal portals and mobile applications that accept a complaint and return a reference number; and public, map-based reporting services in which every report is visible to the community. The best-known example of the third kind is the FixMyStreet service in the United Kingdom, whose use has been studied as a way for citizens to voice local public-service concerns [2].

A recent review of smart complaint redressal research in India covers work from 2019 to 2025 and groups it into artificial intelligence, blockchain, the Internet of Things, mobile applications and governance frameworks [3]. It finds that mobile platforms widen access by letting citizens register complaints at any time, but that long-term engagement, verification of complaints and prevention of misuse remain open problems. It also reports persistent barriers in scalability, privacy, multilingual language processing and digital literacy, and concludes that no single technology is a complete solution. Table 1 compares the main approaches with the platform presented here.

Table 1. Comparison of Complaint-Handling Approaches

Approach	*Strengths*	*Challenges*
Traditional channels (helpline, letter, ward visit)	Familiar to all citizens; no technology needed.	Slow; little or no record; no visible progress or proof of repair.
Municipal portals and mobile apps	Anytime access; reference number; some tracking.	Engagement and complaint verification are hard to sustain [3].
Public map-based reporting	Reports visible to the whole community; encourages transparency [2].	Depends on citizen participation; needs moderation.
Smart Civic Connect (this paper)	Role-based workflow; photo evidence; status history; public map; notifications.	No automatic prioritisation; English only; not yet piloted with a real municipal office.

IV. System Architecture and Methodology

System Architecture: Smart Civic Connect uses a three-tier client–server architecture, summarised in Table 2 and Fig 1. The server follows the model–template–view organisation of the Django framework [4] and exposes selected data as JSON in line with REST principles [5]. The project is divided into six Django applications: accounts, complaints, workers, notifications, community and analytics.

Table 2. System Layers

Layer	*Technology*	*Responsibility*
Presentation	Django templates, Bootstrap 5.3, Leaflet.js, Chart.js, JavaScript	Citizen, worker and admin interfaces; map; charts; light and dark theme
Application	Python, Django, role decorators, form validation, notification service	Authentication, business rules, complaint workflow, JSON endpoints, CSV export
Data	SQLite (default) or PostgreSQL; media store with Pillow	Users, complaints, images, history, comments, announcements, notifications

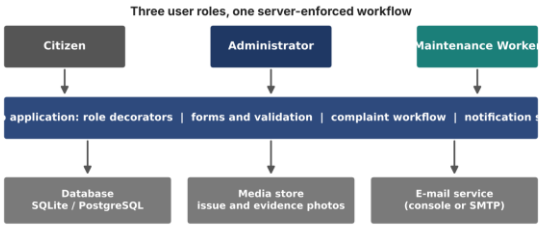


Fig 1. Three user roles served by one server-enforced workflow

User Roles:

- **Citizen:** registers, files complaints with photographs and a map location, tracks progress on a personal dashboard, comments on complaints, and views the public feed and map.
- **Maintenance Worker:** sees only the complaints assigned to them, accepts a task, logs progress with notes, and uploads progress or completion photographs. Workers are created by an administrator and belong to a department such as roads, sanitation, water or electrical.
- **Administrator:** triages all complaints, changes status and priority, assigns workers, rejects invalid

reports with a reason, creates worker accounts, suspends or reactivates users, publishes announcements, and views analytics.

Complaint Data Model: The central entity is the *Complaint*. Each complaint receives a readable identifier of the form SCC-YYYYMMDD-XXXX, made of the date and four random letters or digits, which the citizen can quote when following up. A complaint stores a title, description, one of ten categories (garbage and dumping, road damage, water leakage, streetlight, drainage, footpath, traffic signal, fallen tree, public property damage, and other), a priority (low, medium, high or urgent), an address, a landmark, and latitude and longitude stored with seven decimal places. It also records the citizen who filed it, the assigned worker, a public-visibility flag, internal administrator notes, a rejection reason, resolution notes, and creation, update and resolution timestamps. Supporting entities are *ComplaintImage* (a photograph typed as initial, in-progress or resolved), *ComplaintHistory* (previous status, new status, the user who made the change, remarks and a timestamp), *Comment*, *Announcement* and *Notification*. The custom *User* entity extends Django's user model with a role, phone number, address, department and profile picture.

Complaint Lifecycle: A complaint moves through the states shown in Fig 2 and Table 3. A new complaint is submitted. The administrator places it under review and then assigns it to a worker. The worker moves it to in progress when work begins and marks it resolved when the repair is complete. The administrator then confirms the outcome by setting the status to closed. A complaint that is invalid or duplicated is rejected, and the rejection reason is shown to the citizen. Keeping "resolved" (reported by the worker) distinct from "closed" (confirmed by the administrator) gives the municipality a review step before a complaint leaves the active queue.

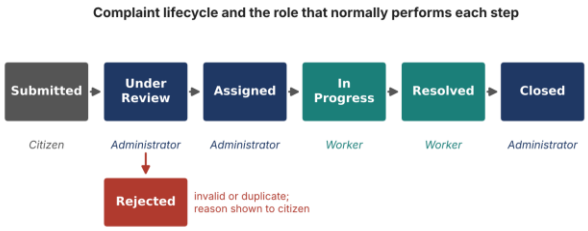


Fig 2. Complaint lifecycle

Table 3. Complaint Status Lifecycle

Status	*Normally set by*	*Meaning*
Submitted	Citizen (automatic)	Complaint registered and queued for review
Under Review	Administrator	Being checked and prioritised
Assigned	Administrator	A worker has been allocated
In Progress	Worker	Work has started on site
Resolved	Worker	Repair reported complete, with notes and photograph
Closed	Administrator	Outcome confirmed; complaint archived
Rejected	Administrator	Invalid or duplicate; reason recorded

Evidence-Based Resolution: When a worker updates a task, a note describing the work is mandatory and a photograph is optional but encouraged. A photograph uploaded with a "resolved" update is stored as resolution evidence, while one uploaded with an "in progress" update is stored as a progress photograph. The complaint page shows the citizen's original photographs next to the worker's completion photographs, so that residents and administrators can compare the before and after state of the site. Setting a complaint to resolved also records the resolution timestamp and the resolution notes.

Community Feed and Interactive Map: Complaints marked public appear on a searchable community feed that can be filtered by category and status and is paginated nine items to a page. The same complaints, provided they carry coordinates, are served as JSON to an interactive map built with Leaflet [7] on OpenStreetMap data [6]. Markers are colour-coded by status, and a popup shows the title, category, priority, address, date, first photograph and a link to the complaint page. Visitors do not need an account to browse the feed or map.

Privacy of Complaints: A citizen can mark a complaint as private. A private complaint is visible only to the citizen who filed it, the worker assigned to it and administrators; any other request receives an HTTP 403 response. The same rule applies to posting comments.

Notification Service: A single service function creates an in-app notification and, when the recipient has an e-mail address, attempts to send a matching e-mail. Notifications are generated when a complaint is submitted, when its status changes, when a worker is assigned, when work starts, when it is resolved, when someone comments, and when an announcement is published. An e-mail failure can never interrupt the action that triggered it, because sending uses a silent-failure mode and is wrapped in exception handling. Announcements, which go to every active citizen, skip e-mail to avoid a mass mailing.

Administrative Workflow and Analytics: The administrator's overview shows the total number of complaints, new complaints today, pending complaints (submitted, under review or assigned), open high-priority complaints, complaints in progress, and resolved

complaints. The complaint table can be searched by title, identifier, address or citizen name and filtered by status, priority and category. The average resolution time is the mean time between submission and resolution over the n complaints that have a resolution timestamp:

$$T_{\text{avg}} = (1/n) \cdot \Sigma\, (t_{\text{res,i}} - t_{\text{sub,i}}) \quad (1)$$

where $t_{\text{sub,i}}$ and $t_{\text{res,i}}$ are the submission and resolution times of complaint i, in hours. The resolution rate shown on the home page is the share of complaints that are resolved or closed:

$$R = 100 \cdot (N_{\text{resolved}} + N_{\text{closed}}) / N_{\text{total}} \quad (2)$$

The analytics dashboard draws Chart.js charts of complaints by category, status and priority, a six-month trend of reports, and the resolved-versus-active ratio. All complaints can be exported to a CSV report with the assigned worker for each row.

Comments and Announcements: Any signed-in user who may see a complaint can comment on it. Comments from workers and administrators are flagged as official, and the complaint owner and the assigned worker are notified of new comments. Administrators can also publish city announcements, which are shown on the community feed, can be pinned, and are delivered as notifications to all active citizens.

V. Implementation

Back End: The server is written in Python with Django, using the framework's object–relational mapper for persistence and its template engine for pages. The accounts application provides the custom user model, registration, login, profile editing and role-based redirection after login. The complaints application provides the landing page, citizen dashboard, complaint submission, detail page and the map-data endpoint. The workers application provides the task dashboard, task acceptance and progress updates. The analytics application provides the administrator overview, complaint management, user management, chart data and CSV export. Django REST Framework is installed and configured with a read-only-for-anonymous default permission, while the current JSON endpoints are implemented as lightweight Django views.

Front End: Pages are rendered from Django templates styled with Bootstrap 5.3 and Bootstrap Icons, using the Inter typeface. A custom stylesheet defines a navy, teal and emerald colour scheme with light and dark themes, and the chosen theme is remembered in the browser. JavaScript handles the theme switch, the image preview on the complaint form, browser geolocation and the Leaflet map. On the complaint form the citizen can click on the map or use the device location to fill in the latitude and longitude.

Notifications and E-mail: In-app notifications are stored in the database and listed on a notification page with an unread counter. E-mail uses Django's mail framework. By default it falls back to the console backend, which prints messages to the terminal during development, and a real SMTP server is selected through environment variables.

Deployment: Configuration is read from a .env file. The database defaults to SQLite and switches to PostgreSQL when DB_ENGINE is set to postgresql. A management command, seed_data, fills the database with demonstration users, complaints, before-and-after photographs and comments so that every role can be tried immediately. For production the guide recommends turning debug mode off, setting a long secret key and allowed hosts, collecting static files, and serving the application with Gunicorn behind an Nginx reverse proxy with HTTPS.

VI. Security and Reliability

- **Authentication:** passwords are stored using Django's salted password hashing and checked against four validators (similarity to user attributes, minimum length, common passwords and all-numeric passwords). Registration rejects an e-mail address that is already in use.
- **Authorisation:** role decorators on every protected view allow only the permitted roles, and the check runs on the server. Workers can open and update only tasks assigned to them. Citizen registration always creates the citizen role, so a user cannot choose to become an administrator. These measures follow the access-control guidance of the OWASP Top Ten [8].
- **Privacy:** private complaints are hidden from everyone except the owner, the assigned worker and administrators, both for viewing and commenting.
- **Request protection:** Django's security, CSRF and clickjacking middleware are enabled, and forms carry CSRF tokens.
- **Upload validation:** images are limited to JPG, JPEG, PNG and WebP files of at most 5 MB, and the image fields also verify that the upload is a real image.
- **Accountability:** every status change is written to the complaint history with the user, remarks and timestamp. Administrators can suspend or reactivate accounts but cannot suspend their own.
- **Fault tolerance:** a failing e-mail delivery is caught and logged so that it can never roll back or block a complaint update.

VII. Testing

The project includes ten automated test cases written with the Django test framework. They cover citizen registration, login with role-based redirection, profile update, complaint submission with generation of a unique identifier, privacy access control for private complaints, status transition and worker assignment, the map-data endpoint, the analytics endpoint, CSV export and the administrator's ability to suspend a user. The expected result of running the suite is that all tests pass with no

system-check issues. These tests check the correctness of the logic and the access rules; they are not a substitute for a field pilot with real citizens, workers and a municipal office.

VIII. Key Features and Benefits

Traceability: every complaint has an identifier, an owner, an assignee and a complete status history, so responsibility is visible at each step.

Verifiable Resolution: the completion photograph is attached to the same record as the original defect, so a repair can be checked rather than merely claimed.

Transparency: the public feed and map show residents what is being reported and repaired in their neighbourhood.

Accessibility: the responsive Bootstrap interface works on desktop and mobile browsers, and citizens can report from the field using the device location.

Informed Citizens: notifications on every significant change remove the need for manual follow-up calls.

Managerial Insight: the dashboard, resolution-time metric and CSV export give administrators data for planning and review.

IX. Scope and Limitations

This is a system-design and implementation paper. Its scope is a working platform for citizen complaint reporting and resolution, and the observations above describe the design rather than measured outcomes. Table 4 summarises the scope and the main limitations.

Table 4. Scope and Limitations

Scope	*Limitations*
Complaint reporting, assignment, resolution and tracking for citizens, workers and administrators.	No field-trial data on resolution times or satisfaction; the demonstration data is synthetic.
Public feed, interactive map and analytics dashboard.	Priority is set manually; no automatic prioritisation, service-level deadline, escalation or duplicate detection.
In-app and e-mail notifications.	No SMS, WhatsApp or push alerts; English-only interface.
Role-based access, privacy and audit history.	The administrator's status form does not enforce a strict order of transitions.
Citizen-supplied map location.	Locations are not verified or converted to ward or address data.
Six-month trend and average resolution time.	The trend uses approximate 30-day month bounds, and the average is computed in application code, which would need database-side aggregation at very large volumes.

X. Future Work

1) Detect duplicate complaints by combining geographic proximity with category, and merge them into one record.

2) Add priority scoring and service-level deadlines with automatic escalation of overdue complaints.

3) Expose a documented REST API, using the installed Django REST Framework, for a mobile application and Progressive Web App.

4) Add SMS, WhatsApp and push notifications, and Hindi and Marathi interfaces to reach more citizens.

5) Use reverse geocoding and ward boundaries, and add heat maps that show recurring problem areas.

6) Classify the category of a complaint from its photograph to reduce mis-filed reports.

7) Collect a rating from the citizen after closure, and run a pilot with a local body to report measured resolution times and satisfaction.

XI. Conclusion

Smart Civic Connect shows that a carefully designed web platform can turn civic complaint handling from an informal, opaque process into a structured, traceable and accountable workflow. By combining a structured complaint record, a defined lifecycle, server-enforced roles, photographic before-and-after evidence, a public map and fail-safe notifications, it addresses the main coordination gaps between citizens, administrators and maintenance crews. The implementation is tested, documented and deployable. The next step is a pilot with a real municipal office to measure its effect on response time and citizen satisfaction, and to refine prioritisation and duplicate handling with actual usage data.

References:

[1] A. Caragliu, C. Del Bo, and P. Nijkamp, "Smart cities in Europe," *J. Urban Technol.*, vol. 18, no. 2, pp. 65–82, 2011.

[2] S. King and P. Brown, "Fix my street or else: Using the internet to voice local public service concerns," in *Proc. 1st Int. Conf. Theory and Practice of Electronic Governance (ICEGOV)*, 2007, pp. 72–80.

[3] A. Pande, Y. Shelar, P. Chapte, G. Shete, and A. Patil, "Smart Complaint Redressal Systems: A Research Review," *Int. J. Res. Publ. Rev.*, vol. 6, no. 10, pp. 1551–1555, Oct. 2025, doi: 10.55248/gengpi.6.1025.3607.

[4] Django Software Foundation, "Django documentation," 2026. [Online]. Available: https://docs.djangoproject.com

[5] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, Univ. of California, Irvine, CA, USA, 2000.

[6] M. Haklay and P. Weber, "OpenStreetMap: User-generated street maps," *IEEE Pervasive Comput.*, vol. 7, no. 4, pp. 12–18, 2008.

[7] V. Agafonkin, “Leaflet: an open-source JavaScript library for mobile-friendly interactive maps.” [Online]. Available: https://leafletjs.com

[8] OWASP Foundation, “OWASP Top Ten Web Application Security Risks,” 2021. [Online]. Available: https://owasp.org/www-project-top-ten

REFERENCES

A. Caragliu, C. Del Bo, and P. Nijkamp, “Smart cities in Europe,” J. Urban Technol., vol. 18, no. 2, pp. 65–82, 2011.

S. King and P. Brown, “Fix my street or else: Using the internet to voice local public service concerns,” in Proc. 1st Int. Conf. Theory and Practice of Electronic Governance (ICEGOV), 2007, pp. 72–80.

A. Pande, Y. Shelar, P. Chapte, G. Shete, and A. Patil, “Smart Complaint Redressal Systems: A Research Review,” Int. J. Res. Publ. Rev., vol. 6, no. 10, pp. 1551–1555, Oct. 2025, doi: 10.55248/gengpi.6.1025.3607.

Django Software Foundation, “Django documentation,” 2026. [Online]. Available: https://docs.djangoproject.com

R. T. Fielding, “Architectural styles and the design of network-based software architectures,” Ph.D. dissertation, Univ. of California, Irvine, CA, USA, 2000.

M. Haklay and P. Weber, “OpenStreetMap: User-generated street maps,” IEEE Pervasive Comput., vol. 7, no. 4, pp. 12–18, 2008.

V. Agafonkin, “Leaflet: an open-source JavaScript library for mobile-friendly interactive maps.” [Online]. Available: https://leafletjs.com

OWASP Foundation, “OWASP Top Ten Web Application Security Risks,” 2021. [Online]. Available: https://owasp.org/www-project-top-ten

2026 The Author(s) https://creativecommons.org/licenses/by/4.0/ DOI: