HUALAOWEI

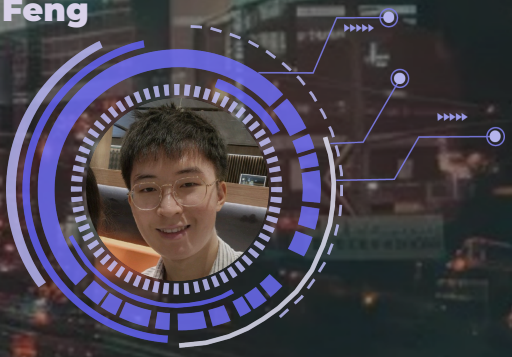
SMART CITY PROPOSAL

TRACK 2: Improve Citizen Engagement and Civic Services

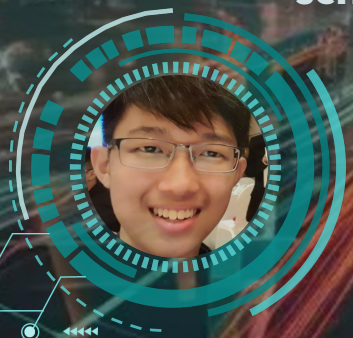
This proposal envisions a future where Artificial Intelligence, Cloud Computing, and Intelligent Systems work hand in hand to transform urban living, paving the way for a smarter, more connected Singapore.

Woo Yan Seun

Tang Ming Feng



Jerick Cheong



Fleming Siow

INTRODUCTION

OneService

The OneService app, developed by the SG government, provides a centralised platform for citizens to report municipal issues. It also features a chatbot for answering queries and routing reports to relevant agencies. While it has greatly streamlined issue reporting, it has notable limitations, such as a lack of **advanced** issue detection, **online** social engagement, and **predictive analysis** for proactive planning.

Problem Statement

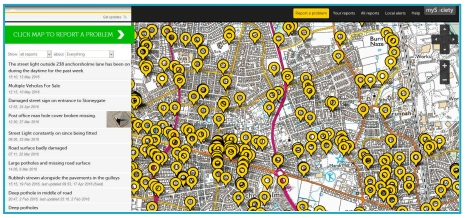
As one of the world's most densely populated cities, SG faces growing urban management challenges. Conventional issue-reporting methods often result in delayed responses and a lack of proactive measures. With rising demands for public services, real-time responsiveness and predictive insights are critical to maintaining efficiency and sustainability.

Open Source Services

While SG does not have readily available municipal data, such issues are **prevalent across the world**, and we can make use of data gathered by **global municipal open data initiatives**, **urban research projects** etc. for **insights** and **AI training**. This method can ensure a localised yet **data-driven approach** to improving civic services in SG.

Open311

Open311 is an **open standard** for civic issue reporting, enabling structured data exchange between citizens, municipalities, and third-party applications. While not adopted by SG's agencies, we can leverage its **standardised format** to improve AI model training, enhance issue classification, and strengthen predictive analytics. Using Open311-compatible data ensures scalability, interoperability, and alignment with global smart city frameworks, while datasets from other cities help refine AI-driven issue detection and prioritisation.

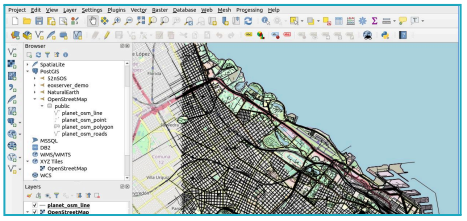


Crowdsourced Data

The internet has a compilation of crowdsourced civic issue reports, capturing real-world citizen feedback on urban problems such as road damage, waste management, and faulty infrastructure. It provides valuable insights into how residents engage with local governments, report issues, and track resolution progress. An example would be **FixMyStreet**, which provides a wealth of **crowdsourced civic issue reports**, widely used in model training (e.g., **OpenPotBot** [Link]) for classifying and enriching street fault data.

OpenStreetMap

OpenStreetMap (OSM) is a collaborative, **open-source mapping platform** that provides detailed geographic data, including street networks, public amenities, and infrastructure layouts. Unlike proprietary mapping services, OSM allows full data access and customisation, making it ideal for AI-powered issue detection and predictive analytics. With OSM, we can accurately geotag reported issues without the use of paid services, and also overlay reports to enhance spatial analysis for smarter municipal planning.



Expected Impact

Through our findings, we believe that by leveraging global open data initiatives and crowdsourced insights, we can harness the power of AI and Cloud Computing to drive more efficient civic engagement, and enhanced municipal services, thus achieving the following:

Improved Response Time

AI-driven categorisation and prioritisation, allows issues to be rectified earlier.

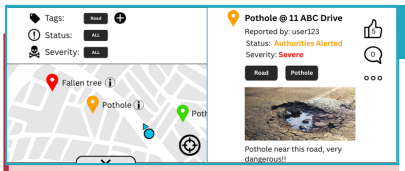
Increased Citizen Participation

Social engagement and transparent issue tracking.

Proactive Urban Planning

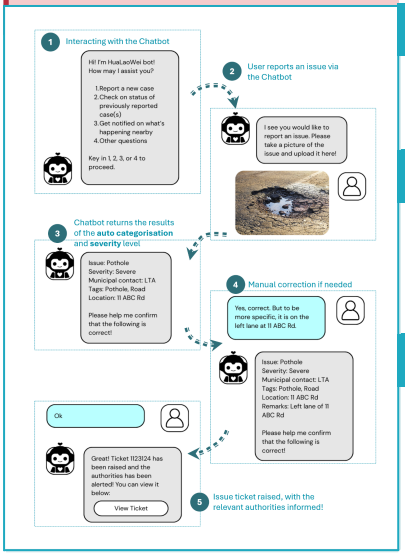
Data-driven insights to predict and prevent recurring issues.

OUR SOLUTION



Core Features

Our solution is a mobile application that consists of several user-centric thought designs which utilises Cloud Computing and Smart City AI technologies to enhance the issue-reporting experience.



Municipal Reporting App

- A cross-platform app to report municipal issues and track status of the reported issue real-time.
- Status of report may include: Processing, Alerted, Viewed, Resolved & Reopened etc.

Public Forum

- A community engagement forum for discussing reported issues (users may choose not to post it under the forum).
- An Upvote, Downvote & Commenting system, that will be visible to all, including any given authorities.

Interactive Chatbot

- NLP/LLM Chatbot, possibly built and deployed within Huawei Cloud.
- Its job is to guide users in municipal queries, submitting a report and ideally assisting with issue categorisation.

Auto Categorisation & Severity

- Computer Vision Model to analyse multiple images or video submission to categorise issue type and severity level.
- Will be trained with crowdsourced data, and refined with newly reported data.

Live Geographical Map

- Displays open reported issues acknowledged by relevant authorities.
- Allows filtering by category, severity, and resolution status.

Notification System

- A real-time notification system to inform users of relevant information such as updates made to their report status, or any issues/hazards near them.

Data Collection

- Essential data will be collected such as the Datetime of Report, Issue Type, Media, Location, Description, Severity Level, & Datetime of Resolved etc.
- Reported data will be made publicly available so others may access it.

Predictive Analytics

- Deep Learning Model to analyse historical data and forecast municipal issue trends.
- Integration with Open311 and OSM is a consideration that we will try to see if it allows for more useful insights.

Authentication & Security

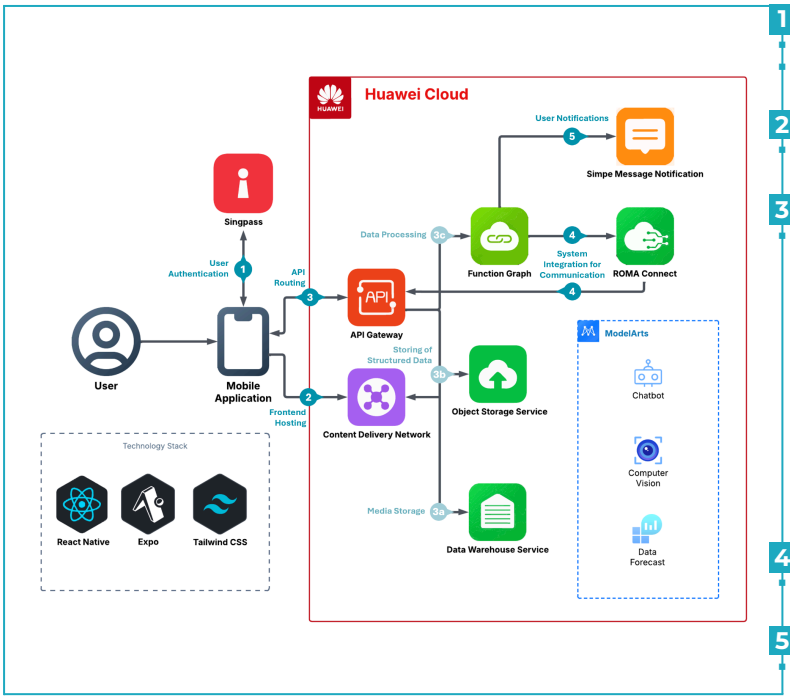
- SingPass integration for secure authentication and authorisation.
- Role-based access control for government agencies.

Optional Features (*If Feasible)

- Offline mode, so users may draft a report even without internet connection.
- Advanced CV model to access the fix's sustainability, by comparing before and after images of the issue/hazard.
- **Real-time tracking of authorities and estimated time of arrival via GPS.*

Architecture Diagram

How we integrate our features and technology into Huawei Cloud.



1 User Authentication via Singpass
Users login to the HuaLaoWei mobile app. The app redirects them to SingPass for authentication via OAuth 2.0 / OpenID Connect.

2 CDN for Frontend Hosting
App fetches static frontend assets (made with REACT Native) from CDN.

3 API Gateway for Backend Services
Requests from the app are handled and routed to the appropriate backend services which includes:

- **a OBS for Media Storage**
  Offers scalable, cost-effective storage of large unstructured data (images & videos).
- **b DWS for Structured Reports**
  Structured data is stored here for efficient querying, analysis & integration with AI models.
- **c FunctionGraph for Data Processing**
  Triggered to process specific tasks like:
  - Using of AI Models
  - Data Transformations & Workflows
  - Calling other Huawei Cloud related services

4 Municipal System Integration with ROMA
Facilitates communication (i.e issue reporting) between the app and municipal backend services

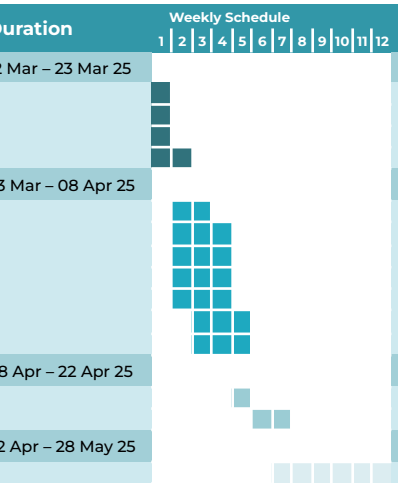
5 SMN for User Alert
Preferences can be configured such that users receive real-time notifications via SMN.

DELIVERABLES

Project Timeline

How we envision our project to span out in the next couple of weeks.

Phase	Duration	Additional Information
1 Proposal	12 Mar – 23 Mar 25	
a Research & Idea Formulation		See what additional features we can add on top.
b Design & Pre-development		UI/UX Mockups, can be included in the proposal.
c App Architecture		
d Proposal Writeup		
2 Prototype Development	23 Mar – 08 Apr 25	
a Implement Frontend Interface		Will likely be done with REACT Native and Expo.
b Building the Chatbot		
c Building the CV Model		
c Building the Forecast Model		
d Setup Huawei Cloud Services		
e Integrating the Backend		
f Create Presentation Deck		
3 User Testing & Refinement	08 Apr – 22 Apr 25	
a Feedback from Mentors & Friends		
b Hotfix & Refinement		
4 Additional Features & Finalisation	22 Apr – 28 May 25	
a Adding Optional Features		Offline Mode, CV to access fix's sustainability etc.



Scalability

As urban populations grow and municipal demands increase, a scalable solution is essential for long-term viability. Our platform leverages several approaches to meet up with evolving user demand and functionalities.

Cloud-Native Architecture

By utilising OBS, DWS and FunctionGraph for data processing and AI, our system dynamically scales based on demand. Moreover, CDN-based frontend hosting optimises performance by serving static assets efficiently, ensuring minimal latency and power consumption. This approach minimises on-premise infrastructure costs, reduces energy consumption, ensuring long-term sustainability.

Community-Driven Management

Through crowdsourced reporting and public engagement features, our platform fosters a self-sustaining civic ecosystem where citizens actively contribute to city upkeep. By empowering residents to report, discuss and track issues, we encourage a culture of shared responsibility, reducing reliance on government monitoring alone.

Standardised and Forward-looking Foundation

Our solution is built on open-standard technologies (e.g., Open311, OpenStreetMap) to ensure long-term adaptability. As more services adopt this foundation, our modular design allows seamless integration with evolving AI models, new data sources, and smart city systems, ensuring that it remains relevant, expandable, and adaptable to future urban challenges.

Enhancing Citizen & Government Collaboration

Singapore's Smart Nation vision emphasises seamless digital interactions between citizens and government services. Our solution strengthens this engagement by integrating AI and Cloud Computing features which includes CV detection, a chatbot, real-time reporting, and transparent case tracking, ensuring faster and more effective municipal response times.

Optimising Public Service Efficiency

By incorporating CV defined categorisation and prioritisation, our system streamlines issue resolution by directing reports to the appropriate agencies with urgency-based classification. This ensures that high-impact urban problems (e.g., hazards near hospitals, schools or transport hubs) receive immediate attention, improving service efficiency and public satisfaction.

Enabling Data-Driven Urban Planning

Our platform aggregates and analyses real-time urban issue reports alongside OpenStreetMap and historical civic data, providing authorities with actionable insights for strategic urban planning. This allows government agencies to make informed, data-backed decisions on infrastructure development, public resource allocation, and long-term sustainability efforts.

Relevance

Singapore's Smart Nation vision prioritises AI-driven governance, digital inclusion, and efficient public services. Likewise, we believe our solution aligns with this by contributing to a smarter, more connected city.

Long Term Vision

Moving forward, as the app continues to grow, we further plan to focus on continuous innovation, and deeper integration with smart city initiatives.

Expansion to ASEAN

Extending to ASEAN would not only enhance urban governance on a regional scale but also provide a richer pool of diverse data, improving insights and analytics for smarter, localised decision-making.

AI Advancements

As more data is collected, we plan to enhance AI models with more sophisticated algorithms, refining predictive capabilities for better issue detection, prioritisation, and urban planning.

Multimodal Data Fusion

Future iterations will integrate real-time data streams from IoT sensors, traffic monitoring systems, and environmental data sources, enabling more precise issue forecasting and smarter resource allocation.

APPENDIX A: Open311 APIs

The Open311 standard allows applications to interact with municipal service request systems in a consistent, standardised way.
While SG's agencies do not currently adopt Open311, we can still explore how this format could be internally leveraged to improve data structure, and reporting flow, providing us the ability to facilitate future interoperability and to streamline integration with third-party services.

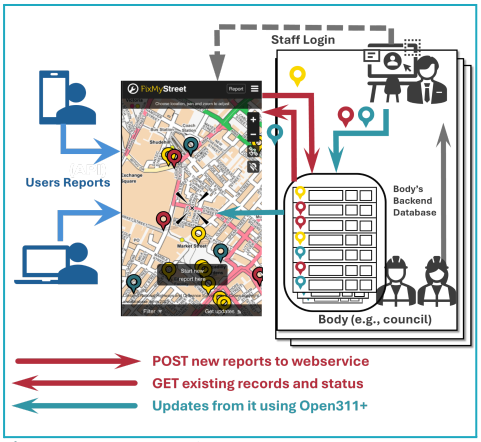


Figure A1: Level 3 Integration of the Open311 API Workflow

FixMyStreet Integration with Open311

Figure A1 illustrates how FixMyStreet (FMS) integrates the Open311 API at varying levels to streamline issue reporting between citizens and municipal authorities [Link].

User Interaction (No Integration)
At the most basic level, users submit civic issue reports via the FMS interface. These are forwarded to authorities via email with no backend connectivity.

Level 1 – Report Injection
Using Open311's POST /requests endpoint, FMS sends structured reports directly to the authority's backend system. This eliminates manual data entry and allows authorities to assign internal reference IDs, which are then stored in FMS's system.

Level 2 – Automatic Status Updates
When a report is updated in the authority's backend, FMS polls for changes via GET /request_updates, reflecting the new status to the user without staff intervention.

Level 3 – Backend-Originated Issues Displayed Publicly
Issues created by municipal staff outside FMS can also be exposed on the platform through GET /requests, offering transparency into all ongoing work.

By adopting Open311, systems like FMS can achieve higher levels of automation, transparency, and scalability in issue resolution. Even if full integration is not possible now, it's flexible, modular design allows progressive adoption to agency readiness.

GET Service Request (API)

Query the current status of an individual request [Link].

Purpose	Query the current status of an individual request
URL	https://[API endpoint]/requests/[service_request_id].[format]
Sample URL	https://api.city.gov/dev/v2/requests/123.xm ?jurisdiction_id=city.gov
Formats	XML (JSON available if denoted by Service Discovery)
HTTP Method	GET
Requires API Key	No
Arguments	service_request_id, jurisdiction_id*

Figure A2: GET Service API Overview


```
[
  {
    "service_request_id":638344,
    "status":"closed",
    "status_notes":"Duplicate request.",
    "service_name":"Sidewalk and Curb Issues",
    "service_code":006,
    "description":null,
    "agency_responsible":null,
    "service_notice":null,
    "requested_datetime":"2010-04-14T06:37:38-08:00",
    "updated_datetime":"2010-04-14T06:37:38-08:00",
    "expected_datetime":"2010-04-15T06:37:38-08:00",
    "address":"8TH AVE and JUDAH ST",
    "address_id":545483,
    "zipcode":94122,
    "lat":37.762221815,
    "long":-122.4651145,
    "media_url":"http://city.sg/media/638344.jpg"
  }
]
```


Figure A3: GET Service API Response Example

Field Name	Description
service_request_id	The unique ID of the service request created.
status	The current status of the service request. Accepts either "open" or "closed.
status_notes	Explanation of why status was changed to current state or more details on current status than conveyed with status alone.
service_name	The human readable name of the service request type
service_code	The unique identifier for the service request type
description	A full description of the request or report submitted. May contain line breaks, but not code. Otherwise, this is free form text limited to 4,000 characters.
agency_responsible	The agency responsible for addressing the service request.
service_notice	Information about the action expected to fulfill the request.
requested_datetime	The date and time when the service request was made. Returned in w3 format, eg 2010-01-01T00:00:00Z
updated_datetime	The date and time when the service request was last modified. Returned in w3 format, eg 2010-01-01T00:00:00Z
expected_datetime	The date and time when the service request can be expected to be fulfilled.
address	Human readable address or description of location.
address_id	The internal address ID used by a jurisdictions master address repository or other addressing system.
zipcode	The postal code for the location of the service request.
lat	latitude using the (WGS84) projection.
long	longitude using the (WGS84) projection.
media_url	A URL to media associated with the request, eg an image.

Figure A4: GET Service API Response List

GET Service Type Request (API)

Provides a list of acceptable 311 service request types [Link].

Purpose	Provide a list of acceptable 311 service request types and their associated service codes. These request types can be unique to the city/jurisdiction.
URL	https://[API endpoint]/services.[format]
Sample URL	https://api.city.gov/dev/v2/services.xml ?jurisdiction_id=city.gov
Formats	XML (JSON available if denoted by Service Discovery)
HTTP Method	GET
Arguments	jurisdiction_id*

Figure A5: GET Service Type API Overview

```
[
  {
    "service_code":001,
    "service_name":"Cans left out 24x7",
    "description":"Garbage or recycling
cans that have been left out for more
than 24 hours. Violators will be cited.",
    "metadata":true,
    "type":"realtime",
    "keywords":"lorem, ipsum, dolor",
    "group":"sanitation"
  },
]
```

Figure A6: GET Service Type Response Example

Field Name	Description
service_code	id for the service request type
service_name	name of the service request type
description	A brief description of the service request type.
metadata	Determines whether there are additional form fields for this service type.
type	**realtime**: The service request ID will be returned immediately **batch**: A token will be returned immediately, which can be later used to return the service request ID. **blackbox**: No service request ID will be returned
keywords	A comma separated list of tags or keywords to help users identify the request type.
group	A category to group this service type within.

Figure A7: GET Service Type Response List

POST Service Request (API)

Create service requests [Link].

Purpose	Create service requests
URL	[https://API endpoint]/requests.[format]
Sample URL	https://api.city.gov/dev/v2/requests.xml
Format Sent	Content-Type: application/x-www-form-urlencoded
Formats	XML (JSON available if denoted by Service Discovery)
HTTP Method	POST
Requires API Key	Yes

Figure A8: POST Service API Overview

Field Name	Description
service_request_id	The unique ID of the service request created.
token	Use this to call GET service_request_id.
service_notice	Addressing the information reported.
account_id	The unique ID of the user reporting.

Figure A9: POST Service Response List

Field Name	Description	Required
jurisdiction_id	This is only required if the endpoint serves multiple jurisdictions	Depends
service_code	This is obtained from GET Service List method	Yes
location parameter	A full location parameter must be submitted.	Yes
attribute	An array of key/value responses based on Service Definitions. This is only required if the service_code requires a service definition with required fields`.	Yes
lat	Latitude using the (WGS84) projection.	No
long	Longitude using the (WGS84) projection.	No
address_string	Human entered address or description of the location. This is required if no lat/long or address_id are provided.	Depends
address_id	The internal address ID used by a jurisdiction's master address repository. This is required if no lat/long or address_string are provided.	Depends
email	The email address of the person submitting the request	No
device_id	The unique device ID of the device submitting the request. This is usually only used for mobile devices.	No
account_id	The unique ID for the user account of the person submitting the request	No
first_name	The given name of the person submitting the request	No
last_name	The family name of the person submitting the request	No
phone	The phone number of the person submitting the request	No
description	A full description of the request or report being submitted. This may contain line breaks, but not html or code. Otherwise, this is free form text limited to 4,000 characters.	No
media_url	A URL to media associated with the request, eg an image.	No

Figure A10: POST Service Arguments

APPENDIX B: Leveraging OpenStreetMap

OpenStreetMap (OSM) is a global open-source mapping platform that offers editable geographic data contributed by volunteers.
For our project, it plays a pivotal role in spatial analysis, urban issue localisation, and custom geovisualization—all while avoiding the licensing restrictions of commercial maps. The Learn OSM website provides easy to understand, step-by-step guides to get started with OSM [Link].

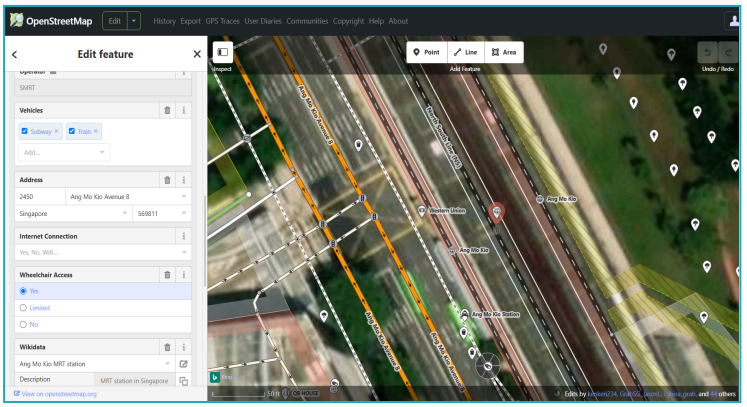


Figure B1: OSM Default On-line Editor

OSM as a Geospatial Layer

OSM provides access to detailed data such as:

- Road Networks
- Building Footprints
- Public Infrastructure (e.g. Parks, Streetlights, Bins)
- Geographical boundaries

By overlaying reported issues on these layers, we may be able to include these into our model analysis:

- Perform spatial clustering of faults (e.g., potholes concentrated in older estates)
- Estimate risk zones based on environmental and infrastructural factors
- Correlate issue types with nearby POIs (e.g., illegal dumping near public bins)

OSM Editing Softwares

There are various softwares — online and offline, which can be used to edit and render OSM data, such as:

iD	The default on-line editor. It's easy to use, and allows mapping from various data sources.
JOSM	An advanced editor, written in Java. Can be used for bulk editing, and extended via plugins.
QGIS	OSM data can be loaded in QGIS as a vector layer, a core functionality or through plugins.
Osmosis	A utility program for performing many tasks at a raw level on OSM data, which includes ETL.
osm2pgsql	A utility program that converts .osm data into PostgreSQL, often used to render via Mapnik

We can use these softwares to:

- Tag specific structures (e.g., "damaged sidewalk")
- Validate automated geotagging from user reports
- Update underlying OSM data where necessary

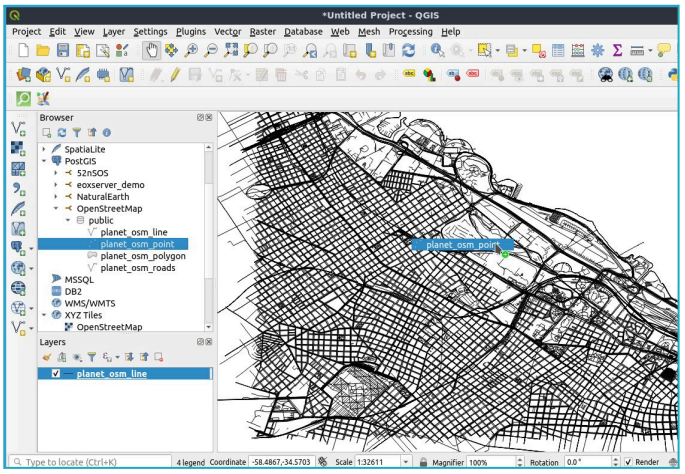


Figure B2: Editing & Customizing with OSM QGIS Software

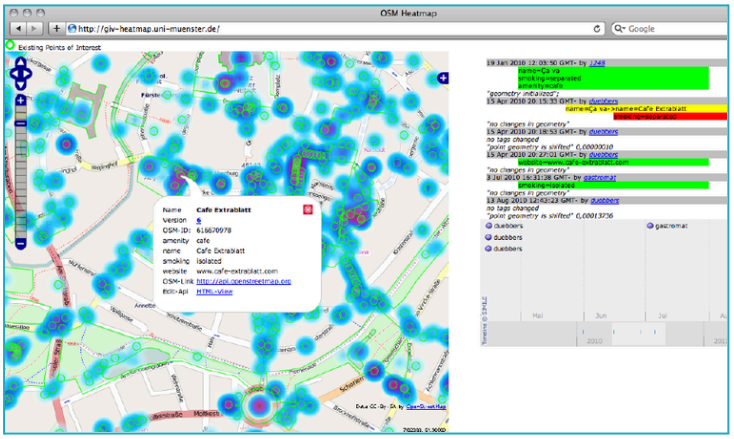


Figure B3: A Heat Map Generated with OSM

Integrating OSM

While integrating OSM data is a new frontier for us, we see it as a valuable opportunity to enhance our spatial analysis. In our implementation, this would involve:

- Backend services fetch OSM tile and vector data dynamically using APIs (such as Overpass API).
- Issues are automatically geotagged using coordinates from device GPS, aligned to OSM layers.
- Custom icons and overlays are rendered over the base map to represent live issue reports.

This will enable our platform to:

- Render a live issue map for users.
- Enable region-specific trend analysis.
- Create heat maps or priority zones to act on.

APPENDIX C: Application Interface

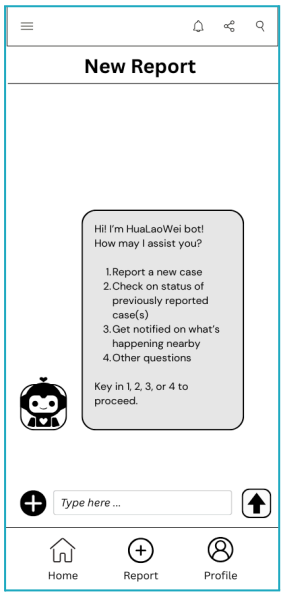


Figure C1: Chatbot Options

Figure C2: Chatbot calling CV model

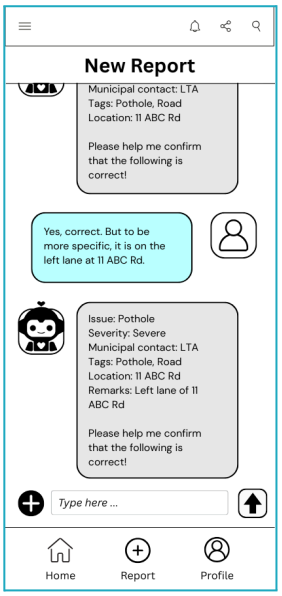


Figure C3: Chatbot manual correction

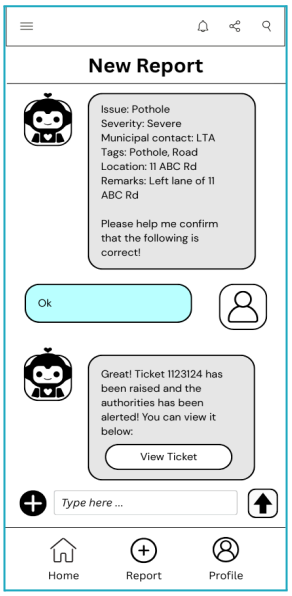


Figure C4: Chatbot raising ticket

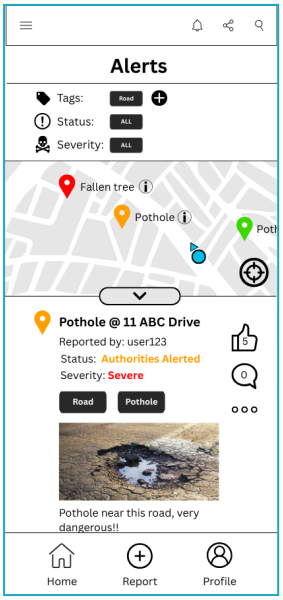


Figure C5: Live Geographical Map

The above images illustrate how a user can report an issue via chatbot interaction.
While a manual form—similar to OneService—will be available, some users may prefer a conversational assistant for a more guided and convenient reporting experience.

Figure C1 shows the chatbot's main options, namely:
(1) Report an issue through chat interaction.
(2) Check the status of previous reports, with the option to revoke any reports.
(3) View nearby conditions, as well as manage notification settings.
(4) Ask municipal-related questions.

Figure C2 presents a sample conversation for reporting an issue.
Upon sending an image (when asked), it triggers the backend services to automatically detect the Issue Type, Severity Level, Municipal Agency, Tags and Location, which then gets filled for the user to view.

Figure C3 continues the interaction.
While fields are auto-filled, users may still edit or add information before submission.

Figure C4 shows the final confirmation.
Once the user verifies the details, the chatbot files the report.
By selecting "View Ticket", users can track the status and queue number.

Figure C5 displays a possible preview of the live geographical map interface.
Each raised and acknowledged ticket appears here with icons representing issue types.
A filter system will be located at the top to allow users to toggle issues by their Tags, Status and Severity.
Clicking on a specific icon opens the forum post related to that issue. Users who choose not to share it publicly will not have their name displayed, and no commenting system will be displayed either.