Citizen AI – Intelligent Citizen Engagement Platform

Team Information

- • Team ID: LTVIP2025TMID21064
  • Team Size: 4 members
  • Team Leader: Batchula Sri Teja Naga Sai
  • Team Members: Anagani Yogananda, Alluri Pranathi Sai, Annavarapu Phaneedra

Table of Contents

1. Project Overview

Citizen AI is designed to be an intelligent assistant that bridges the communication gap between citizens and civic authorities. It focuses on real-time interaction using AI, especially Hugging Face's IBM Granite LLM model, to provide dynamic and context-aware responses to civic queries.

This platform integrates complaint registration, AI-based query resolution, and sentiment analytics to deliver a unified civic management solution. The chatbot simplifies access to policies and local information while the dashboard visualizes live feedback to support better governance.

The core aim is to foster inclusive and responsive digital governance, encourage public participation, and build trust between authorities and citizens.

2. System Architecture

The system architecture of Citizen AI follows a layered and service-oriented model. The frontend provides interfaces for citizens to interact with the chatbot, submit complaints, and view the sentiment dashboard. It is built using HTML, CSS, and JavaScript.

The backend, developed using FastAPI, handles routing, API management, complaint processing, and sentiment analysis. It communicates with Hugging Face's IBM Granite LLM model via REST API to retrieve context-specific responses for user queries.

Real-time updates are made possible through Chart.js, allowing the sentiment analysis results to refresh every 15 seconds. Data flows seamlessly between modules, ensuring fast processing and user feedback. Google Colab is used for LLM prompt experimentation and performance testing.

3. Technology Stack

Citizen AI employs a modern tech stack to ensure modular development and smooth integration with generative AI services.

- Backend: FastAPI for API development, Uvicorn as ASGI server
- AI Layer: IBM Granite LLM via Hugging Face API for generating human-like responses
- Sentiment Engine: TextBlob (Python NLP library) for sentiment classification
- Frontend: HTML, CSS for structure and styling; JavaScript and Chart.js for dashboard visualization
- Colab: Google Colab notebook to experiment with LLM prompts and test API responses

The stack supports quick deployment, scalable modules, and AI integration without needing local heavy infrastructure.

4. Project Structure

The project directory is structured to maintain clarity and separation of concerns.

- app/: Contains FastAPI application files and routers
- routes/: Separate route files for chatbot, complaints, and sentiment APIs
- utils/: Utility scripts for functions like sentiment analysis
- templates/: HTML files for frontend interface
- static/: CSS and JavaScript files including Chart.js logic
- colab/: Google Colab notebook for testing IBM Granite model and prompt performance

This modular structure ensures that each component is independently testable and maintainable.

5. Implementation Details

The implementation involves a three-part process: frontend interaction, backend processing, and AI response generation.

Frontend: Users interact via a web form (for complaints) or chat window (for AI questions). This sends input to the FastAPI backend.

Backend: The backend validates and processes inputs. Complaints are categorized and logged, while user queries are passed to the AI module via Hugging Face's API. Responses are returned as JSON and displayed to the user.

AI Integration: Google Colab was used for testing prompts and evaluating response quality. The model used is IBM Granite 3.3-2b-instruct. For sentiment, TextBlob analyzes the polarity of feedback and classifies it into positive or negative.

Dashboard: The sentiment counts are tracked and updated live on a bar chart created with Chart.js.

6. Development Workflow

The development lifecycle followed an agile model with sprint-based tasks and daily reviews.

• Phase 1: Ideation and requirement gathering—focus on problem areas in civic engagement
• Phase 2: Frontend design and HTML form/dashboard creation
• Phase 3: Backend setup using FastAPI, API route creation
• Phase 4: Hugging Face model setup, Colab experimentation with prompts
• Phase 5: Sentiment engine integration using TextBlob
• Phase 6: Testing, debugging, and UI refinement
• Phase 7: Documentation and final report preparation

Each feature was tested independently and integrated using version control.

7. Setup and Installation

To run Citizen AI locally, follow these steps:

1. Clone the repository:
   `git clone https://github.com/your-username/citizen-ai.git`

2. Navigate to the directory and install dependencies:
   `pip install -r requirements.txt`

3. Run FastAPI server locally:
   `uvicorn app.main:app --reload`

4. Open index.html in any browser to use the frontend. The real-time dashboard updates automatically.

5. For AI integration, open the provided Colab notebook (`model_integration.ipynb`) and set the Hugging Face API key before testing prompts.

8. Features and Functionality

Citizen AI provides several innovative features that enhance civic interaction:

• AI Chat Assistant: Understands and responds to civic queries using IBM Granite LLM
• Complaint Management: Users can log complaints with categories and descriptions
• Sentiment Analysis: All feedback is classified using NLP as positive or negative
• Dashboard Visualization: Real-time feedback is displayed on a bar graph using Chart.js
• Modular API System: Each feature is served through FastAPI for quick performance and independent scaling
• Google Colab Testing: Prompts are pre-tested before deployment ensuring high-quality responses

9. API Documentation

The backend exposes four primary REST API endpoints:

1. `POST /chat`: Sends a user query and receives AI-generated response
2. `POST /complaint`: Accepts complaint category and description, stores data
3. `POST /feedback/sentiment`: Processes input and returns sentiment label
4. `GET /dashboard`: Returns count of positive and negative sentiment entries for chart rendering

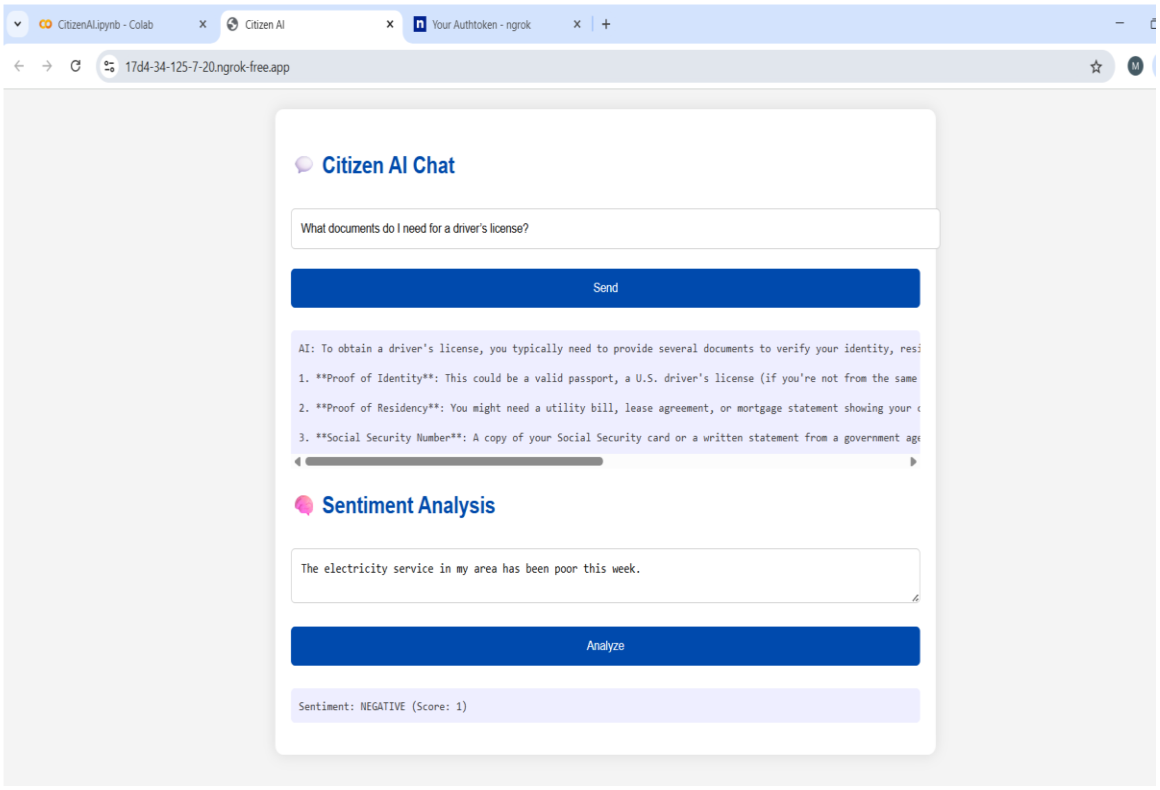
All endpoints return JSON data. Errors are handled using FastAPI exception handlers.

10. Screenshots and Results

Screenshots to be inserted by the user include:

• The chatbot interface showing a query and response
• Complaint form with example data filled in
• Sentiment chart showing current positive and negative counts

• Console logs of API responses in JSON format
• Google Colab output from Hugging Face model test



Citizen AI Chat

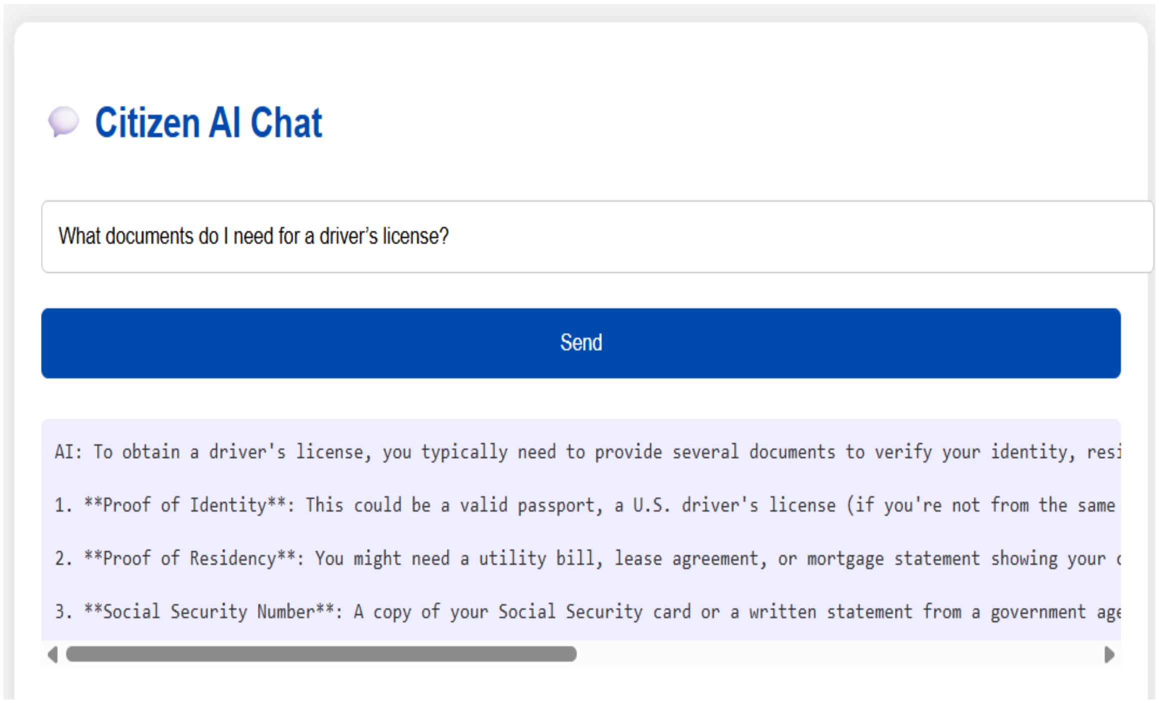
What documents do I need for a driver's license?

```
AI: To obtain a driver's license, you typically need to provide several documents to verify your identity, resi
1. **Proof of Identity**: This could be a valid passport, a U.S. driver's license (if you're not from the same
2. **Proof of Residency**: You might need a utility bill, lease agreement, or mortgage statement showing your c
3. **Social Security Number**: A copy of your Social Security card or a written statement from a government age
```

Sentiment Analysis

The electricity service in my area has been poor this week.

```
Sentiment: NEGATIVE (Score: 1)
```

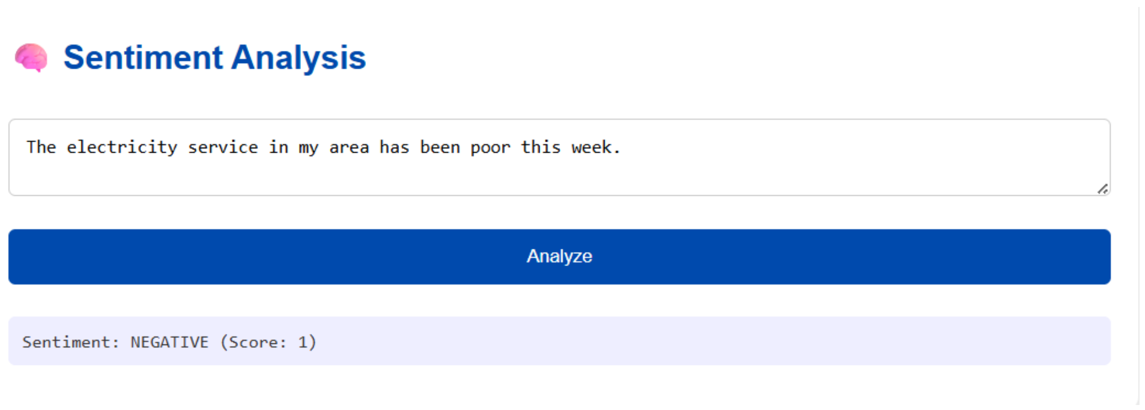


Citizen AI Chat

What documents do I need for a driver's license?

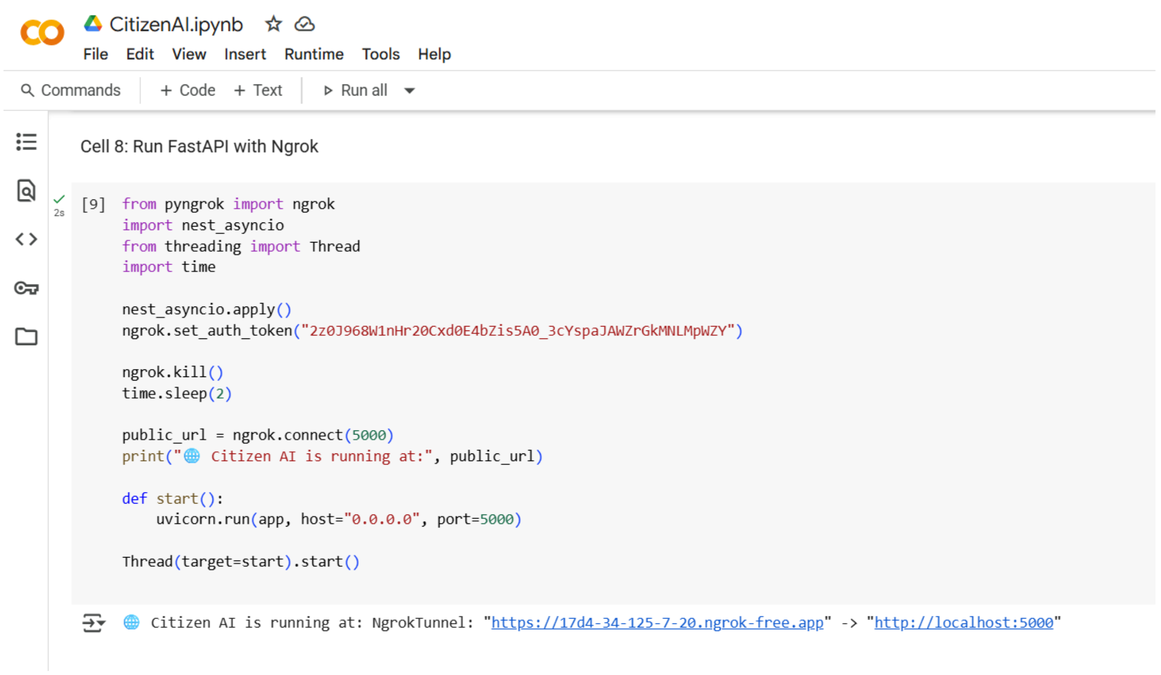
Send

```
AI: To obtain a driver's license, you typically need to provide several documents to verify your identity, resi
1. **Proof of Identity**: This could be a valid passport, a U.S. driver's license (if you're not from the same
2. **Proof of Residency**: You might need a utility bill, lease agreement, or mortgage statement showing your c
3. **Social Security Number**: A copy of your Social Security card or a written statement from a government age
```



```
The electricity service in my area has been poor this week.
```

```
Sentiment: NEGATIVE (Score: 1)
```


CitizenAI.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all

Cell 8: Run FastAPI with Ngrok

```
[9] from pyngrok import ngrok
    import nest_asyncio
    from threading import Thread
    import time

    nest_asyncio.apply()
    ngrok.set_auth_token("2z0J968W1nHr20Cxd0E4bZis5A0_3cYspaJAWZrGkMNLMpWZY")

    ngrok.kill()
    time.sleep(2)

    public_url = ngrok.connect(5000)
    print("🌐 Citizen AI is running at:", public_url)

    def start():
        uvicorn.run(app, host="0.0.0.0", port=5000)

    Thread(target=start).start()
```

```
🌐 Citizen AI is running at: NgrokTunnel: "https://17d4-34-125-7-20.ngrok-free.app" -> "http://localhost:5000"
```

Video Links

Demo link: https://drive.google.com/file/d/1pdjzdwuWuIndoIWtHoXMVLJHUQ078GR-/view?usp=sharing

GitHub link: https://github.com/Pranathi-Sai-Alluri/Citizen-AI

11. Challenges and Solutions

Several challenges were encountered and resolved during development:

• Problem: Live dashboard not refreshing automatically
 Solution: Integrated JavaScript's `setInterval()` to update data every 15 seconds

• Problem: Hugging Face model latency and API key limits

Solution: Optimized prompt structure and used fewer API calls per session

• Problem: Neutral sentiments misclassified
Solution: Refined sentiment logic to exclude polarity range around zero

• Problem: UI formatting inconsistencies
Solution: Separated CSS styling and added validation on form input

12. Future Enhancements

Citizen AI is a scalable platform, and there are several potential enhancements that can increase its usability, accessibility, and reach in the future. These enhancements aim to serve both the general public and administrators with even greater efficiency and automation. Here is a detailed breakdown of the proposed future improvements:

1. **Voice Assistant Integration**:
Voice-based input can be integrated to allow users to speak their complaints or queries instead of typing them. This will make the platform accessible to people with limited literacy or physical impairments and improve usability in rural or underdeveloped areas.

2. **Multilingual Support**:
Adding support for regional languages will help cater to a broader audience, especially in a diverse country like India. The AI model can be fine-tuned or translated to understand and respond in various local dialects to break language barriers.

3. **Mobile Application**:
A dedicated Citizen AI mobile app would increase ease of access for daily users. This app can allow push notifications, complaint status tracking, instant query responses, and geo-tagging complaints on the go.

4. **Policy Document Summarization**:
The platform can be extended to allow PDF or document uploads for civic policy analysis. Using LLM-based summarization, it could generate easy-to-understand summaries for citizens and officials.

5. **Admin Dashboard and Portal**:
An administrative backend portal can be built to help officials monitor complaints, analyze trends, visualize heatmaps of high-complaint areas, and export analytics reports for policy decisions.

6. **Smart Notifications and Alerts**:
Integration with SMS and email systems can enable users to receive automatic updates on

complaint resolution status, new policy changes, or urgent civic alerts (e.g., water cuts, roadblocks).

7. **Integration with IoT Devices**:
IoT sensors can be installed in public infrastructure (e.g., garbage bins, pipelines) and connected to the platform to automatically raise alerts in case of anomalies like overflows, leakage, or fire.

8. **Gamification of Civic Participation**:
Introducing gamified elements like reward points for repeated user engagement, successful complaint closure, and eco-friendly actions can boost public participation and maintain long-term interest in the platform.

9. **Social Media Sentiment Integration**:
Mining Twitter, Reddit, or Facebook posts using keyword monitoring and sentiment detection can give authorities early warnings and deeper insights into public sentiment beyond the platform.

10. **Heatmap Visualization**:
A GIS-based interface can be introduced to allow dynamic visualization of complaint density, sentiment trends, and problem clusters on a city map, helping with faster decision-making.

These enhancements aim to make Citizen AI a powerful, all-in-one civic engagement and decision-support system capable of functioning in real-world urban environments and scaling across municipalities.

13. Conclusion

Citizen AI represents a step toward intelligent civic engagement using natural language understanding and real-time analytics. It empowers both citizens and government bodies to interact more efficiently, address problems quickly, and improve satisfaction through automation and transparency.

Its AI-driven approach makes civic services more responsive, citizen-friendly, and insight-driven. Future enhancements will only further improve its scalability and impact.