# CITIZEN AI: INTELLIGENT CITIZEN ENGAGEMENT PLATFORM

PROJECT DOCUMENTATION :

PROJECT DONE BY :

FATHIMA HASEENA J

FARITHA NEILOFAR S

FATHIMA RUFIYA M

GAYATHRI M

# **Introduction**

**Citizen AI** is an intelligent citizen engagement platform intended to bridge communication between citizens and public service/government agencies. It enables citizens to ask questions, provide feedback, and explore civic procedures; and enables officials to monitor sentiment, feedback trends, and improve responsiveness.

### **Features**

|  |  |
| --- | --- |
| **Feature** | **Description** |
| **Chatbot / Virtual Assistant** | Real-time conversational interface for citizen queries regarding public services, documents, procedures. Context aware. |
| **Sentiment Analysis** | Analyze textual feedback / citizen messages to determine sentiment (positive, negative, neutral) and trends. |
| **Dashboard / Analytics** | Visualize citizen feedback over time, sentiment trends, most asked questions, etc. |
| **Modular Architecture** | Backend using Flask; AI integration modular; database layer abstracted to allow SQLite or PostgreSQL. |
| **Context Tracking** | The system tracks conversation flow so that responses reflect prior context in the interaction. |

### **Architecture**

* **Backend**: Flask app written in Python. Acts as web server, handles routing, logic for processing chat messages, feedback, sentiment processing.
* **AI Components**:  
  • IBM Granite LLM — for generating responses to citizen queries.  
  • IBM Watson NLP — for language processing tasks such as sentiment, intent detection.
* **Frontend**: Static HTML/CSS/JS (Bootstrap) pages: chat interface, dashboard, feedback pages.
* **Database**: either SQLite (for small / dev use) or PostgreSQL (for production, scale). Stores user messages, feedback entries, sentiment data, possibly log of interactions.
* **Deployment**: (optional) Docker for containerization; environment variables via .env file.

### **Setup & Installation (Suggested)**

Here is a suggested flow for installing / running locally:

1. Clone the GitHub repo.
2. Create a Python virtual environment (e.g. python3 -m venv venv).
3. Activate environment, install dependencies via pip install -r requirements.txt.
4. Create a .env file to store configuration:
5. AI service credentials (e.g. IBM Granite API Key, Watson credentials)
6. Database connection string (for SQLite or PostgreSQL)
7. Secrets, other config (port, host)
8. Initialize the database (run migrations or setup script). If using SQLite, may just create file. If PostgreSQL, ensure DB exists.
9. Run the Flask app locally (e.g. flask run or python app.py).
10. Access the UI via browser: chat page, feedback page, dashboard.

For production:

1. Use a more robust server (e.g. Gunicorn or uWSGI) behind a reverse proxy (e.g. Nginx).
2. If using PostgreSQL, ensure connection settings, backups, migrations.
3. Containerization (Docker) for easier deployment.
4. Use HTTPS, environment variables for secrets.

### **APIs / Modules (Hypothetical / To Be Defined)**

If the system exposes APIs, the endpoints may include:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Endpoint** | **Method** | **Purpose** | **Input** | **Output** |
| /chat | POST | Send message from citizen to chatbot | JSON with user\_id, message, context | JSON with response, possibly updated context |
| /feedback | POST | Submit feedback / rating | JSON with user\_id, feedback\_text, optional rating | JSON ACK / status |
| /sentiment | GET or POST | Get sentiment analysis of feedback or messages | feedback text(s) | sentiment result + score |
| /dashboard/data | GET | Data for dashboard visualisations | filters (date range, sentiment types) | JSON with counts, trends, etc. |

These should be documented clearly with input validation, error responses, authentication requirements.

### **Data Model**

Entities likely needed:

* **User / Citizen** (if login / identity): user id, name, possibly contact, preferences
* **Conversation / Message**: message text, timestamp, sender (citizen or system), context or conversation id
* **Feedback**: feedback text, timestamp, rating / tags, linked to user or session
* **Sentiment Record**: for each feedback or message, a sentiment label and score, timestamp
* **Dashboard Metrics**: aggregated counts, trends (may be derived rather than stored)

Data retention, privacy, anonymization policies are important.

### **Security & Privacy**

* Authenticate users (if needed) or ensure anonymized feedback if anonymous.
* Secure API keys / AI service credentials (don’t commit to repo).
* Use TLS/HTTPS in transit.
* Encrypt sensitive data at rest if required (depending on jurisdiction).
* Handle personal identifiable information (PII) carefully: minimize collection, possibly anonymize.
* Privacy policy & terms of service for data usage.
* Audit logs for important events.

### **Limitations / Known Issues**

* Handling of ambiguous or malicious input (in the chatbot)
* AI hallucination / incorrect responses
* Scaling beyond small user base (if using SQLite, or single Flask server)
* Multi-language support (if only configured for one language)
* Real-time performance under load

### **Maintenance & Extensions**

* Monitoring (log errors, usage)
* Model updates / retraining (if applicable)
* Adding support for multiple languages, or integrating more AI services
* Improving UI/UX for accessibility
* Adding more analytic modules: trending topics, keyword extraction, etc.