Citizen AI: Intelligent Citizen Engagement Platform

Project Documentation

1. Introduction

- **Project Title:** Citizen AI: Intelligent Citizen Engagement Platform

- **Team Member :** Naseema Begam M
- **Team Member :** Amreen Begum K
- **Team Member :** Farhana M
- **Team Member :** Revathi V

2. Project Overview

Purpose:

Citizen AI is an intelligent assistant built to improve communication between citizens and civic authorities. The assistant delivers two core capabilities:

1. City Safety Analysis — generates concise, human-friendly reports about a city's crime, accident and safety profile.

2. Citizen Service Help — answers queries about government procedures (for example: how to apply for a ration card, voter ID, birth certificate, complaint process, etc.).

By combining a generative LLM with a simple web interface, Citizen AI improves access to civic information and helps officials see aggregated feedback and basic analytics.

Features:

Each feature below lists the key point, functionality, and a short example of use.

- Conversational Interface
    - *Key Point:* Natural language interaction
    - *Functionality:* Users type questions in plain language; the assistant replies with clear, stepwise answers and suggestions.

    - *Example:* "How do I apply for a driving license?" → lists documents, steps and links.

- Policy Summarization
    - *Key Point:* Simplified policy understanding
    - *Functionality:* Convert long policy text or PDF into short, actionable bullets.
    - *Example:* Upload a government circular and receive a 5-bullet summary.
- Resource Forecasting (future extension)
    - *Key Point:* Predictive analytics
    - *Functionality:* Estimate short-term resource needs (energy/water/waste) from historical CSV data.
    - *Example:* Forecast next month's electricity demand using past usage.
- Eco-Tip Generator (optional)
    - *Key Point:* Personalized sustainability advice
    - *Functionality:* Generate daily/weekly tips tailored to a user's profile or city context.

- Citizen Feedback Loop
    - *Key Point:* Community engagement
    - *Functionality:* Collect feedback via forms and analyze sentiment (aggregate view for officials).
- KPI Forecasting & Anomaly Detection (ML Modules)
    - *Key Point:* Strategic planning support and early warning
    - *Functionality:* Lightweight regression/time-series forecasting for KPIs; flag unusual patterns with simple statistical or ML checks.

- Multimodal Input Support
    - *Key Point:* Flexible data handling
    - *Functionality:* Accepts text queries, PDF uploads (for summarization) and CSVs (for forecasting).
- Streamlit or Gradio UI
    - *Key Point:* User-friendly interface
    - *Functionality:* Single-page or tabbed layout for citizens and officials; quick shareable links via Gradio when using Colab.

3. Architecture

The architecture is modular so the project can start small (Colab + Gradio + LLM) and scale to a production architecture (FastAPI + Vector DB + dashboards).

High-level Components

- Frontend (Gradio / Streamlit)
    - Simple app for interaction, structured into tabs or pages: *City Analysis*, *Citizen Services*, *Upload / Summarize*, *Feedback*.
    - For local/production deployment, Streamlit offers page routing and richer components; Gradio is convenient for Colab demos and a quick public link.
- Backend (Python)
    - LLM wrapper and business logic — prompt construction, safe-response filtering, and result formatting.
    - In production, FastAPI exposes endpoints for UI, summarization, forecasting, and feedback ingestion.
- LLM Integration — IBM Watsonx Granite (Hugging Face)
    - Model: ibm-granite/* family (for example granite-3.2-2b-instruct).
    - The LLM handles NLU/NLG tasks: city reports, summarization, stepwise instructions.

- Optional Vector Search
    - Embeddings: Sentence Transformers or equivalent.
    - Vector DB: Pinecone or similar — used when you want fast semantic search over uploaded policies.
- ML Modules
    - Forecasting and anomaly detection use scikit-learn and pandas. Visualization uses matplotlib (or plotly for interactive dashboards).
- Data & Storage
    - Short-term: CSVs uploaded at runtime (Colab) and in-memory storage.
    - Production: Cloud buckets for files, a small DB (Postgres/Firestore) for feedback and logs, and Pinecone for embeddings.

4. Setup Instructions

Prerequisites:

- Python 3.9+
- pip, virtualenv / venv
- Google Colab account (for demo) with ability to use GPU runtimes
- (Optional) Hugging Face account with access to IBM Granite models
- (Optional production) API keys for Pinecone and IBM Cloud/Watsonx

Installation & Google Colab steps (recommended for demo)

1. Open a new Google Colab notebook.
2. Change runtime: Runtime → Change runtime type → Hardware accelerator → GPU (choose T4).
3. Install packages in the first cell
4. Model load pattern
5. Add the response generation helper and Gradio UI.
6. Run all cells and copy the Gradio public link provided by app.launch(share=True).

5. Folder Structure

app/ – Contains all core Python logic including model loading and response generation.

app/model_loader.py – Handles loading of IBM Granite model and tokenizer.

app/response_generator.py – Contains reusable function to generate text responses from model.

app/city_analysis.py – Function for generating crime & safety analysis for a given city.

app/citizen_services.py – Function for answering public service and civic-related queries.

ui/ – Contains Gradio UI components and layout code.

ui/gradio_interface.py – Defines the Gradio Blocks, Tabs, Inputs, and Outputs for the application.

notebooks/ – Stores Google Colab notebook version of the project.

notebooks/citizen_ai_colab.ipynb – Main notebook to run the app with GPU runtime.

requirements.txt – Lists all dependencies (transformers, torch, gradio).

README.md – Documentation about project setup and usage.

6. Running the Application

To Start the Project:

1.Open Google Colab Notebook

Upload or open citizen_ai_colab.ipynb.

2.Change Runtime

- Go to Runtime → Change runtime type.
- Select T4 GPU as Hardware Accelerator.

3.Install Dependencies

Run the following in the first cell

4.Run the Notebook Cells

- The code will load IBM Granite model from Hugging Face.
- Tokenizer and model are prepared automatically.

5.Launch the Application

- The last cell contains
- This generates a public Gradio link.

6.Navigate the Application

- City Analysis Tab → Enter a city name → Get crime, accident, and safety analysis.
- Citizen Services Tab → Enter a query about policies/services → Get detailed guidance.

7.Close the Session

- Stop runtime in Colab → Public link expires after session ends.

Frontend (Gradio)

The frontend of this project is developed using Streamlit, which provides an interactive and user-friendly web interface. It includes multiple pages such as dashboards, file uploads, chat interface, feedback forms, and report viewers. Navigation across these pages is handled through a sidebar that leverages the streamlit-option-menu library. Each page is modularized, making the system scalable and easy to maintain. The UI is designed to be simple yet functional, enabling users to upload documents, interact with the chat assistant, and visualize real-time results like summaries, predictions, and reports.

Backend (Model + Hugging Face Transformers)

The backend is powered by FastAPI, a high-performance REST framework that exposes API endpoints for core functionalities such as document processing, chat interactions, eco-tip generation, report creation, and vector embedding. It supports asynchronous execution, ensuring faster performance and scalability. FastAPI also provides built-in Swagger documentation, which simplifies API testing and debugging. The backend serves as the engine that processes inputs, communicates with the IBM Granite model, and delivers dynamic outputs to the Streamlit frontend in real time.

7. API Documentation

- POST /chat/ask — Input: {"query": "..."}. Output: LLM response to citizen query.
- POST /city-analysis — Input: {"city": "Mumbai"}. Output: Generated safety & accident analysis.
- POST /upload-doc — Upload policy document for embedding and summarization.
- GET /search-docs?query=... — Returns top semantically similar documents from Pinecone.
- GET /get-eco-tips?topic=energy — Returns generated sustainability tips.
- POST /submit-feedback — Stores citizen feedback and optional metadata (location, priority).

Each endpoint should be documented and available in Swagger UI when using FastAPI.

8. Authentication & Security

Demo mode: open access for demonstration.

Production considerations:

- Use JWT tokens or API keys to protect endpoints.
- Implement role-based access: citizen, official, admin.
- Rate-limit requests and monitor model usage to control costs.
- Sanitize and validate uploaded files; do not execute arbitrary code.
- Secure secrets with environment variables and secret managers.

9. User Interface

Design principles: accessibility, simplicity, and clarity.

Components:

- Sidebar / Tab navigation
- Input boxes with placeholder examples
- Clear action buttons (Analyze / Ask / Upload)
- Output area with structured answers (headings, bullets)
- File upload widgets for PDFs/CSVs
- Report download option (PDF)

10. Testing

Unit tests:

- Prompt engineering functions (ensure prompt templates are correct).
- Tokenization & model wrapper utilities.

API tests:

- Use pytest + requests or FastAPI test client to validate endpoints.
- Run tests for typical and malformed payloads.

Manual testing:

- Upload different PDFs and CSVs to validate summarization & forecasting.
- Try queries that are ambiguous, niche, or deliberately invalid.

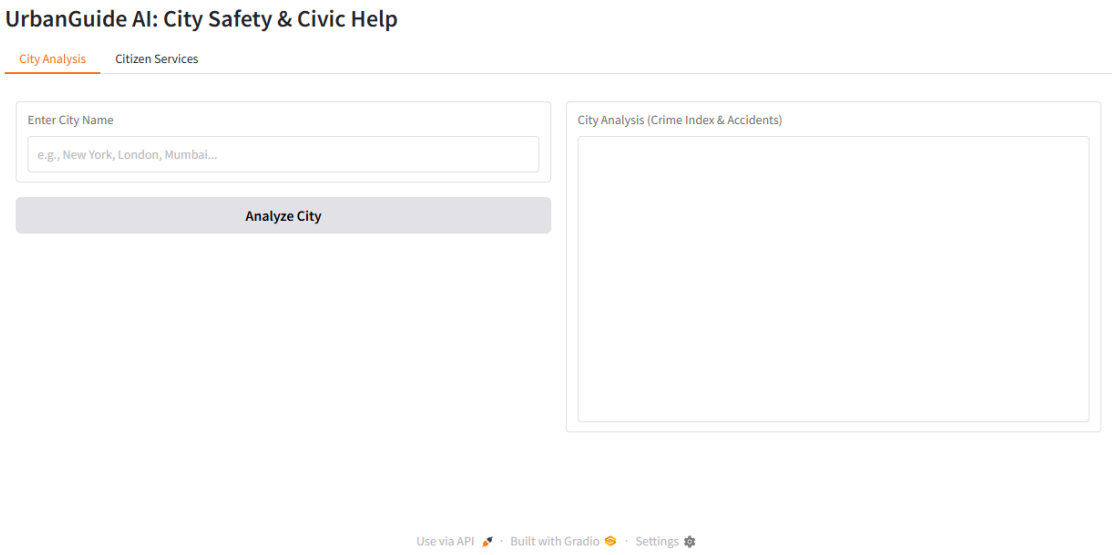
Performance tests:

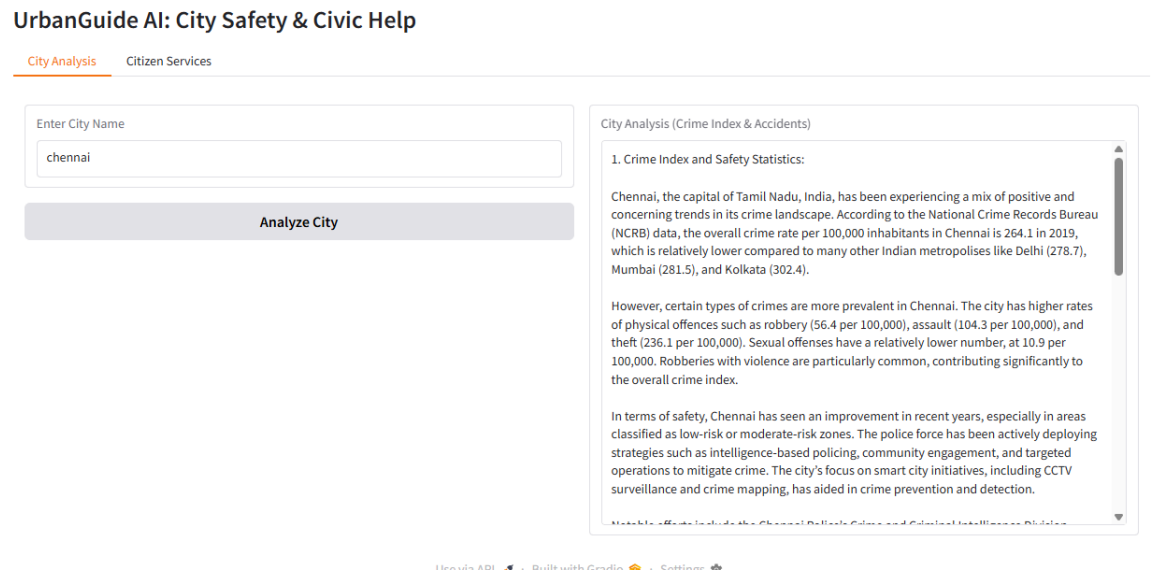
- Measure latency on Colab (T4) for short & medium prompts.
- Test memory usage and ensure the model doesn't OOM.

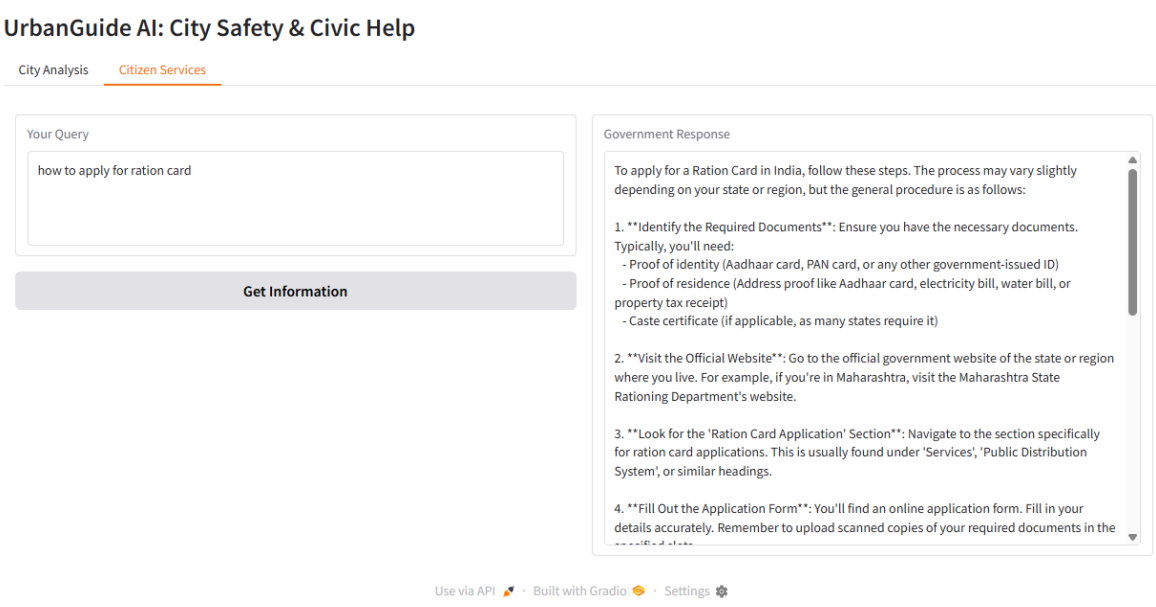
Edge case handling:

- Empty or whitespace-only queries → return helpful guidance.
- Very long documents → chunk and summarize progressively.
- Invalid API keys → clear error responses.

11. Screenshots







12. Known Issues

- Responses are AI-generated and may occasionally be incorrect or outdated. Always advise verification with official sources for legal/critical matters.
- Colab sessions are temporary; the Gradio share link will stop working when the notebook disconnects.
- Large LLM variants may be costly or memory-heavy for local machines.
- No built-in multilingual dataset — quality in regional languages requires fine-tuning or translation pipeline.

13. Future Enhancements

- Real-time data integration: Connect to government APIs for verified crime or traffic data.
- Sentiment analytics dashboard: Time-series charts to show citizen mood/trends.
- Multilingual support: Add Tamil, Hindi and other local languages.
- Persistent deployment: Host the app on a managed cloud for 24/7 availability.
- Access control & logging: Full authentication, user history and audit logs.
- Lightweight mobile UI: Progressive Web App or mobile-optimized pages.