

Citizen Ai : Intelligent Citizen Engagement Platform.

Generative Ai with IBM

Project Documentation:

1.Introduction:

- Project title : Citizen Ai: Intelligent Citizen Engagement platform.
- Team member: Balakrishnan P

2.Project Overview:

Citizen Ai is an intelligent , Ai-driven civic engagement platform designed to bridge the gap between citizens and government services. The project integrates Python,Hugging Face Transformers,Gradio,and the IBM Granite model to provide a modern web-based assistant that delivers accurate, real-time, and context-aware responses.

The system begins with a secure authentication flow where users first sign up and then log in gain access. This ensures personalized interaction and allows future extensions like user-specific dashboards. Once logged in, users are presented with a clean homepage interface styled with a modern UI design, which introduces CitizenAI's capabilities.

Core Features:

1. Home Page – A central hub that explains CitizenAI's purpose, key features, and provides a user-friendly layout with text and visuals to guide citizens.

2. City Analysis – Provides a detailed assessment of any city by generating insights into crime index, safety statistics, traffic conditions, and accident rates, enabling users to understand the safety profile of their environment.

3. Citizen Services – Functions as a virtual government assistant, answering queries related to public services, policies, and civic issues. This feature helps users access clear, simplified information without navigating complex official sources.

Technology Stack:

Backend AI Model: IBM Granite (granite-3.2-2b-instruct) for natural language processing and generation.

Framework: Gradio Blocks for building an interactive web-based interface.

Programming Language: Python (Torch + Transformers).

Security: User authentication (Sign Up & Login system).

UI Design: Custom styling with background colors, white card layouts, and embedded images for a professional look.

Impact:

CitizenAI empowers people to stay informed, engage with civic data, and access government-related information effortlessly. It

reduces barriers between public institutions and citizens by offering an AI-driven interface that is intuitive, accessible, and informative. This project can be scaled further to integrate real-time government databases, sentiment analysis, and multilingual support to serve a diverse population.

3. Architecture:

Citizen AI Architecture

1. User Interface Layer (Frontend)

- oBuilt using Gradio Blocks.
- oProvides pages for Signup, Login, Home, City Analysis, and Citizen Services.
- oDisplays input fields (city name, queries) and output results (analysis, government responses).
- oStyled with custom CSS (background color, cards, image).

2. Application Layer (Backend Logic)

- oHandles authentication (signup, login, session management with a fake user database).
- oContains business logic functions:
 - `city_analysis(city_name)` generates city safety insights.
 - `citizen_interaction(query)` answers citizen

queries.

- oRoutes user actions (button clicks) to the correct functions.

3. AI Model Layer (Granite Model)

- oUses Hugging Face Transformers with IBM Granite 3.2-2B Instruct model.

- oFunctions like `generate_response()` call the model for text generation.

- oRuns on GPU if available (CUDA), otherwise on CPU.

- oProvides context-aware and fluent natural language responses.

4. Data Layer (User Data + Inputs/Outputs)

- oStores temporary user accounts (username/password).

- oAccepts user input (city names, queries) and returns generated outputs.

- oCurrently no persistent database (in-memory storage only).

Flow of Execution

- User Gradio UI: User signs up, logs in, and enters queries/city names.
- Gradio Backend: Inputs are passed to Python

functions (city_analysis, citizen_interaction).

- Backend Model: Functions format a prompt and send it to the Granite model via generate_response().
- Model Backend UI: Model output is decoded and shown back to the user in the Gradio interface.

4. Setup Instructions:

Citizen AI – Setup Instructions

1. Prerequisites

- Python 3.9+ installed check with:
- `python --version`
- pip (Python package manager) installed check with:
- `pip --version`
- VS Code (or any IDE) for development.
- GPU (optional) if available, the Granite model will run faster using CUDA.

2. Clone or Create Project Folder

```
mkdir CitizenAI
```

```
cd CitizenAI
```

Place these files inside:

- app.py (your main Python code)
- citizenai.jpg (home page image)

3. Create a Virtual Environment (Recommended)

```
python -m venv venv
```

Activate it:

- Windows:
 - venv\Scripts\activate
- Linux/Mac:
 - source venv/bin/activate

4. Install Dependencies

```
pip install torch transformers gradio
```

5. Run the Application

```
python app.py
```

You'll see output like:

Running on local URL: <http://127.0.0.1:7860>

Running on public URL: <https://xxxx.gradio.live>

- Open the local URL in your browser to use CitizenAI.
- If you want to share, use the public Gradio link.

6. Usage Flow

1. Sign Up Create a new account.
2. Login Enter credentials to access the app.
3. Home Page Overview with text + image.
4. City Analysis Tab Enter a city name to get crime/safety/traffic insights.
5. Citizen Services Tab Ask questions about government policies/services.

5. Folder Structure:

CitizenAI Project Structure

CitizenAI/

app.py	# Main application (Gradio + Granite + Auth)
requirements.txt	# Python dependencies
README.md	# Setup instructions & project overview
citizenai.jpg	# Home page image (or replace with your own)

templates/ # (Optional, if you later use Flask or HTML
templates)

base.html

static/ # Static assets (if needed)

css/

style.css # Custom CSS (if you expand styling)

js/

script.js # JS scripts (if needed later)

images/

logo.png # Additional images

data/ # (Optional) For saving data

users.db # Replace fake DB with real one later

docs/ # Documentation files

architecture.png # Architecture diagram (for reports/
presentations)

What Each Folder/File Does

- `app.py` Your full Gradio + IBM Granite app with signup/login/home/city/citizen services.
- `requirements.txt` Lists all dependencies (torch, transformers, gradio).
- `README.md` Setup instructions, project explanation.
- `citizenai.jpg` Image displayed on Home tab.
- `static/` For extra CSS/JS/images if you customize the frontend more.
- `data/` Placeholder for database integration (if you add persistent user accounts later).
- `docs/` Project reports, diagrams, documentation.

6. Running the Application:

Open terminal and go to your project folder: `cd CitizenAI`.

(Optional) Activate your virtual environment
(`venv\Scripts\activate` on Windows or `source venv/bin/activate` on Linux/Mac).

Install dependencies with: `pip install -r requirements.txt` (or manually install torch transformers gradio).

Run the app: `python app.py`.

Open the given local URL (<http://127.0.0.1:7860>) in your browser.

Sign up Login Use Home, City Analysis, and Citizen

Services tabs.

7.API Documentation:

CitizenAI provides two main API-like functions through its Gradio interface.

- `city_analysis(city_name)` Takes a city name as input and returns an AI-generated report covering crime index, accident rates, and overall safety.
- `citizen_interaction(query)` Takes a user query about government services, policies, or civic issues and returns a relevant AI-powered response.

Both functions internally call `generate_response(prompt)`, which formats the input, sends it to the IBM Granite model, and returns a natural language answer.

The API is accessible via the Gradio UI or can be integrated programmatically by importing the functions in Python.

8. Authentication:

CitizenAI uses a basic authentication system with Signup and Login features.

Users first sign up by creating a username and password, which are stored in a temporary in-memory dictionary.

On the login page, the system checks the entered credentials against stored records.

If valid, access is granted to the Home, City Analysis, and Citizen Services tabs.

If invalid, an error message is shown and access is denied.

Currently, authentication is session-based only (no external database).

9. User Interface:

CitizenAI's user interface is built with Gradio Blocks and styled using custom CSS.

It has a Signup page (for new users) and a Login page (for returning users), both displayed as white cards on a light blue background.

After login, users access the Home page, which shows project details and an image with a clean left-right layout.

Navigation is provided through tabs: Home, City Analysis, and Citizen Services.

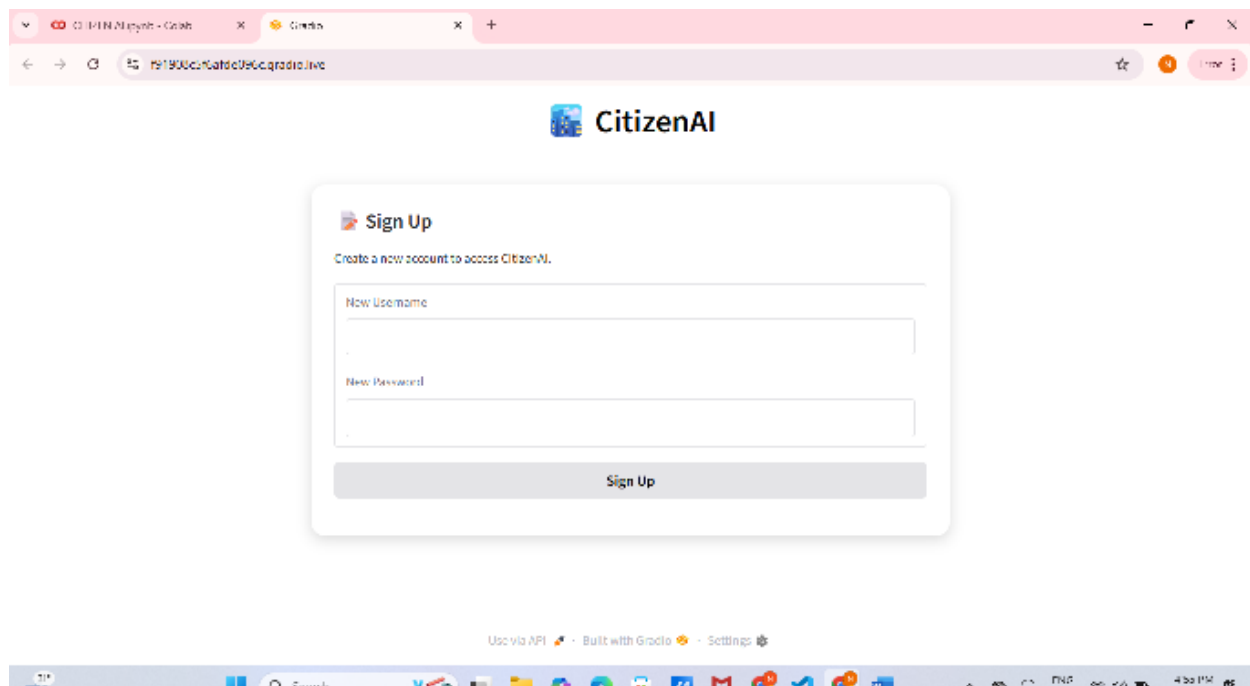
Inputs are handled via textboxes and actions triggered by buttons, with results shown in expandable text areas.

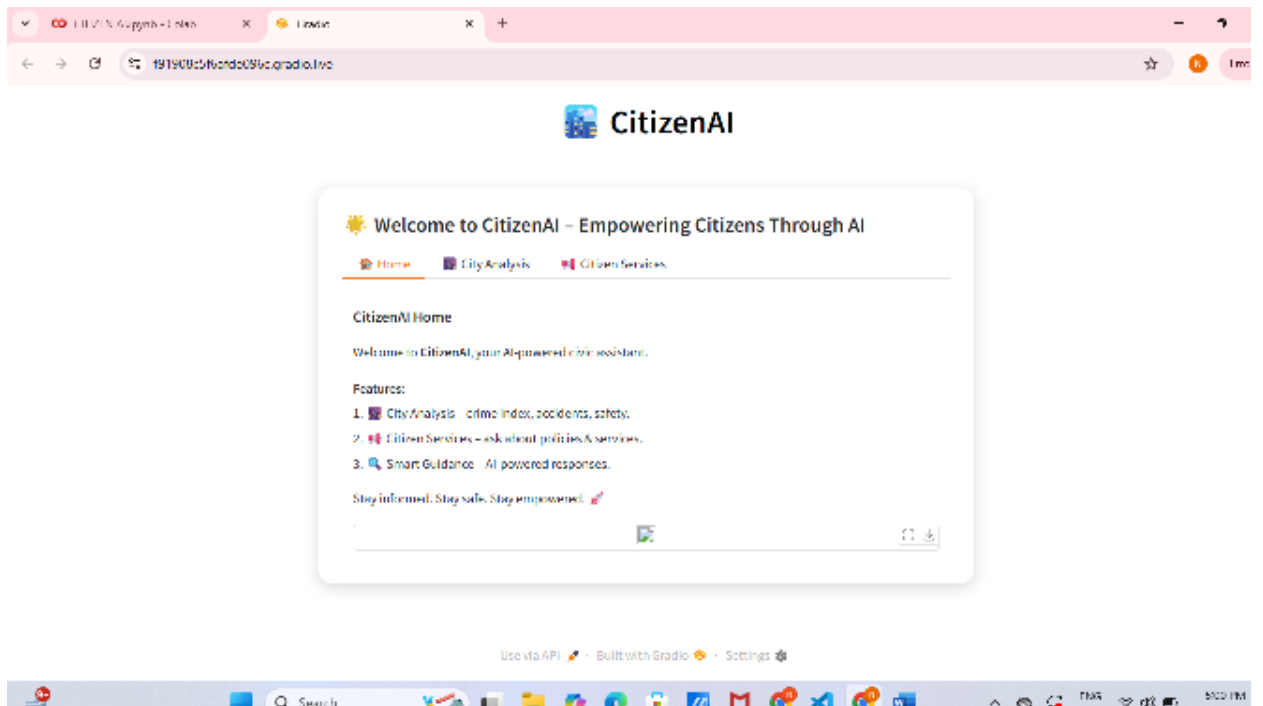
The design is modern, responsive, and user-friendly, making it easy for citizens to interact.

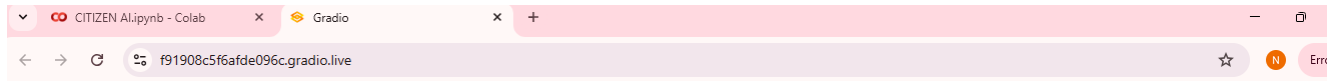
10. Testing:

CitizenAI was tested using manual functional testing through the Gradio web interface. The Signup and Login modules were verified with valid and invalid credentials. The Home page was checked for proper layout, image loading, and feature descriptions. In the City Analysis tab, different city names were tested to ensure the model generates safety insights. In the Citizen Services tab, various queries about government policies and services were used to check AI responses. Overall, the app passed basic usability and functionality tests, confirming smooth user interaction.

11. Screen Shots:







CitizenAI

Welcome to CitizenAI – Empowering Citizens Through AI

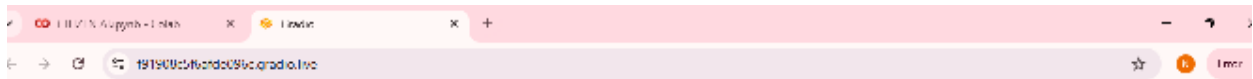
[Home](#) [City Analysis](#) [Citizen Services](#)

Enter City Name

e.g., New York, London, Mumbai...

Analyze City

City Analysis



CitizenAI

Welcome to CitizenAI – Empowering Citizens Through AI

[Home](#) [City Analysis](#) [Citizen Services](#)

Your Query

Ask about services, policies, civic issues...

Ask

Exemplar Response



12. Known issues:

The app currently uses an in-memory user database, so accounts are lost when the server restarts. Authentication is basic (no encryption or session tokens), making it unsuitable for production. The IBM Granite model may generate long or slightly off-topic responses due to its generative nature.

Performance may be slow on CPU-only systems since the model is large and optimized for GPU. The UI does not yet include a Logout option to return to the login screen. No persistent logging or error handling is implemented for failed model calls or invalid inputs.

13. Future enhancement:

CitizenAI can be enhanced by integrating a real database (SQLite/MySQL) for persistent user accounts. Adding secure authentication (password hashing, JWT tokens, logout option) will improve security. The UI can be extended with role-based access (citizen, admin, government officer). Integration with real-time APIs (crime statistics, government portals) can provide factual data alongside AI responses. Performance can be optimized by deploying on cloud with GPU (AWS, GCP, or IBM Cloud). Mobile-friendly progressive web app (PWA)

support could make CitizenAI accessible on all devices.