# Citizen AI – Intelligent Citizen Engagement Platform

## Introduction

Citizen AI is an Intelligent Citizen Engagement Platform designed to help governments and organizations improve communication with citizens. The platform integrates AI-powered features such as chat responses, sentiment analysis, concern reporting, city analysis, and dashboard insights. By leveraging IBM Granite models and Python libraries, the system provides real-time assistance for public services, safety assessments, and policy-related queries. This documentation outlines the milestones, activities, and structured workflow of the project.

## Project Overview

Citizen AI – Intelligent Citizen Engagement Platform

Citizen AI is an intelligent citizen engagement platform that provides AI-powered city analysis, a government assistant for citizen queries, and an interactive dashboard for real-time analytics. This document provides setup instructions, architecture, core functionalities, deployment steps, testing guidelines, and the project milestone plan.

## Key Features

Citizen AI – Intelligent Citizen Engagement Platform

1. Project Overview

Citizen AI is an intelligent citizen engagement platform that provides AI-powered city analysis, a government assistant for citizen queries, and an interactive dashboard for real-time analytics. This document provides setup instructions, architecture, core functionalities, deployment steps, testing guidelines, and the project milestone plan.

• AI chatbot for citizen-service interaction  
• City analysis (crime index, accident rates, safety assessment)  
• Sentiment analysis and feedback routing  
• Real-time analytics dashboard  
• Concern reporting and tracking

## Technology Stack

Citizen AI – Intelligent Citizen Engagement Platform

1. Project Overview

Citizen AI is an intelligent citizen engagement platform that provides AI-powered city analysis, a government assistant for citizen queries, and an interactive dashboard for real-time analytics. This document provides setup instructions, architecture, core functionalities, deployment steps, testing guidelines, and the project milestone plan.

2. Key Features

• AI chatbot for citizen-service interaction  
• City analysis (crime index, accident rates, safety assessment)  
• Sentiment analysis and feedback routing  
• Real-time analytics dashboard  
• Concern reporting and tracking

Backend: Python, Flask  
AI Models: IBM Granite (granite-3.2-2b-instruct) / Hugging Face transformers  
Frontend: Gradio for prototype UI; Flask + Bootstrap for production UI  
Database: SQLite (prototype) or PostgreSQL (production)  
Deployment: Docker, Gunicorn/uvicorn, cloud hosting (AWS/GCP/Azure)

## Architecture

Citizen AI – Intelligent Citizen Engagement Platform

1. Project Overview

Citizen AI is an intelligent citizen engagement platform that provides AI-powered city analysis, a government assistant for citizen queries, and an interactive dashboard for real-time analytics. This document provides setup instructions, architecture, core functionalities, deployment steps, testing guidelines, and the project milestone plan.

2. Key Features

• AI chatbot for citizen-service interaction  
• City analysis (crime index, accident rates, safety assessment)  
• Sentiment analysis and feedback routing  
• Real-time analytics dashboard  
• Concern reporting and tracking

3. Technology Stack

Backend: Python, Flask  
AI Models: IBM Granite (granite-3.2-2b-instruct) / Hugging Face transformers  
Frontend: Gradio for prototype UI; Flask + Bootstrap for production UI  
Database: SQLite (prototype) or PostgreSQL (production)  
Deployment: Docker, Gunicorn/uvicorn, cloud hosting (AWS/GCP/Azure)

High-level components:  
• Frontend (Gradio prototype / Flask templates) – handles user input and displays responses.  
• Backend (Flask app) – manages routing, input validation, orchestration with AI model and DB.  
• Model layer – hosts the language model (locally or via managed inference service).  
• Database – stores reports, user sessions, logs and analytics.  
• Dashboard – visualizes engagement metrics and sentiment.  
  
Note: Include a network diagram in the design phase showing interactions between components, model hosting, and external services.

## Milestones & Activities

Citizen AI – Intelligent Citizen Engagement Platform

Introduction

Citizen AI is an Intelligent Citizen Engagement Platform designed to help governments and organizations improve communication with citizens. The platform integrates AI-powered features such as chat responses, sentiment analysis, concern reporting, city analysis, and dashboard insights. By leveraging IBM Granite models and Python libraries, the system provides real-time assistance for public services, safety assessments, and policy-related queries. This documentation outlines the milestones, activities, and structured workflow of the project.

Milestone 1: Project Setup and Architecture

Activity 1.1: Select and Confirm AI Model

1. Confirm the selection of the IBM Granite model for core AI capabilities.  
2. Specify Python libraries: Flask (web framework), PyTorch (model backend), Hugging Face (transformers, accelerate, bitsandbytes).  
3. Review documentation for model loading, inference, quantization, and device handling.  
4. Explore Flask documentation for routing, templating, and session management.

Activity 1.2: Define the Architecture of the Application

1. Define modules: Frontend, Backend, AI Model, Database, Dashboard.  
2. Prepare architecture diagram to show interactions between components.

Activity 1.3: Set Up the Development Environment

1. Install prerequisites: Python 3.9+, Git, CUDA-enabled GPU (optional), Docker (optional).  
2. Create virtual environment: python -m venv venv  
3. Activate and install dependencies: pip install -r requirements.txt

Milestone 2: Core Functionalities

Activity 2.1: Develop Core Functionalities

• Implement AI-powered city analysis.  
• Create chatbot for citizen queries.

Activity 2.2: Implement Flask Backend

• Manage routing and user input.  
• Integrate AI response generation.

Milestone 3: Application Logic and Data Handling

Activity 3.1: Writing Main Logic

• Implement app.py with model integration, query handling, and database interaction.

Milestone 4: Frontend Development

Activity 4.1: Designing and Developing UI

• Create citizen query input and city analysis dashboard.

Activity 4.2: Creating Dynamic Templates

• Use Flask render\_template and Bootstrap for responsive design.

Milestone 5: Deployment

Activity 5.1: Set Up Virtual Environment

• Configure .env and environment variables for deployment.

Activity 5.2: Testing and Verifying Local Deployment

• Test Flask/Gradio app locally and verify functionality.

Milestone 6: Functional Testing and Verification

• Conduct unit tests, integration tests, load tests.  
• Validate security and privacy compliance.

## Deployment Recommendations

Citizen AI – Intelligent Citizen Engagement Platform

1. Project Overview

Citizen AI is an intelligent citizen engagement platform that provides AI-powered city analysis, a government assistant for citizen queries, and an interactive dashboard for real-time analytics. This document provides setup instructions, architecture, core functionalities, deployment steps, testing guidelines, and the project milestone plan.

2. Key Features

• AI chatbot for citizen-service interaction  
• City analysis (crime index, accident rates, safety assessment)  
• Sentiment analysis and feedback routing  
• Real-time analytics dashboard  
• Concern reporting and tracking

3. Technology Stack

Backend: Python, Flask  
AI Models: IBM Granite (granite-3.2-2b-instruct) / Hugging Face transformers  
Frontend: Gradio for prototype UI; Flask + Bootstrap for production UI  
Database: SQLite (prototype) or PostgreSQL (production)  
Deployment: Docker, Gunicorn/uvicorn, cloud hosting (AWS/GCP/Azure)

4. Architecture

High-level components:  
• Frontend (Gradio prototype / Flask templates) – handles user input and displays responses.  
• Backend (Flask app) – manages routing, input validation, orchestration with AI model and DB.  
• Model layer – hosts the language model (locally or via managed inference service).  
• Database – stores reports, user sessions, logs and analytics.  
• Dashboard – visualizes engagement metrics and sentiment.  
  
Note: Include a network diagram in the design phase showing interactions between components, model hosting, and external services.

5. Installation & Local Setup

Prerequisites:

• Python 3.9+  
• Git  
• CUDA-enabled GPU recommended for local model inference  
• Docker (optional, for containerized deployment)

5.1 Create project environment

1. Clone repository  
 git clone <repo-url>  
2. Create and activate virtual environment  
 python -m venv venv  
 source venv/bin/activate (Windows: venv\Scripts\activate)  
3. Install dependencies  
 pip install -r requirements.txt

5.2 Model & Tokenizer

Example (using Hugging Face transformers):  
  
from transformers import AutoTokenizer, AutoModelForCausalLM  
  
model\_name = 'ibm-granite/granite-3.2-2b-instruct'  
tokenizer = AutoTokenizer.from\_pretrained(model\_name)  
model = AutoModelForCausalLM.from\_pretrained(model\_name, torch\_dtype=torch.float16 if torch.cuda.is\_available() else torch.float32, device\_map='auto' if torch.cuda.is\_available() else None)  
  
Ensure you have proper access/credentials for private model repositories if required.

6. Application Code Overview

Main modules:

• app.py / server.py — Flask routing and API endpoints  
• model\_driver.py — wrappers for model loading and generation  
• ui\_gradio.py — Gradio Blocks-based prototype (for demo)  
• analytics.py — ETL & aggregation for dashboard  
• db.py — database models and migration scripts

6.1 Provided Gradio Prototype

The Gradio prototype provides two tabs:  
• City Analysis — accepts a city name and returns AI-generated analysis.  
• Citizen Services — accepts a citizen query and returns a government-assistant response.  
  
Example: run ui\_gradio.py (or the provided script) to launch locally. Use app.launch(share=True) for an externally reachable demo link.

7. API Endpoints (Example)

POST /api/v1/city-analysis  
Payload: {"city":"Mumbai"}  
Response: {"analysis":"..."}  
  
POST /api/v1/citizen-query  
Payload: {"query":"How to apply for a birth certificate?"}  
Response: {"response":"..."}

8. Data Handling & Security

• Sanitize and validate all citizen inputs before model calls.  
• Rate-limit API endpoints to prevent abuse.  
• Log queries and model outputs for debugging and monitoring, but avoid storing PII in logs.  
• Use HTTPS in production and secure credentials with environment variables or a secrets manager.  
• If using third-party model hosting, review data sharing and retention policies.

9. Testing & Verification

1. Unit tests for routing and data validation.  
2. Integration tests that mock model responses.  
3. Load testing to ensure the API handles expected concurrency.  
4. Manual functional testing of Gradio/Flask UI.

• Containerize the app with Docker. Use a slim base image and multi-stage builds.  
• Use Gunicorn or Uvicorn behind a reverse proxy (NGINX) for concurrency.  
• For model serving, consider managed inference (AWS SageMaker, Vertex AI) or Triton/accelerated runtime.  
• Set up monitoring (Prometheus + Grafana) and centralized logging (ELK or Cloud provider).

## Conclusion

Citizen AI provides a structured approach to intelligent citizen engagement, combining AI-driven responses, real-time city analysis, and user-friendly interfaces. By following the defined milestones and activities, the platform ensures scalability, security, and effective public service delivery. This project demonstrates the application of modern AI models, web technologies, and structured workflows in building impactful civic-tech solutions.