📘 Project Documentation

1. Introduction

- **Project Title:** Citizen AI – Intelligent Citizen Engagement Platform
- **Team Members:**
    - MOHAMED JIYATH R
    - CHANDRA MOHAN A
    - DEEPAK KUMAR N
    - KATHIR D

---

2. Project Overview

Purpose

Citizen AI was developed to address the communication gap between governments and citizens. Traditional systems like call centers, static websites, or manual helpdesks are slow and inefficient. This platform leverages AI to provide real-time assistance, analyze citizen feedback, and present actionable insights to officials.

The purpose is to:

- Improve responsiveness of government services.
- Build trust through transparent communication.
- Enable data-driven decision-making.

Features

- **Conversational Assistant** – AI chatbot using IBM Granite for real-time Q&A.
- **Sentiment Analysis** – Classifies citizen feedback as Positive, Neutral, or Negative.
- **Dynamic Dashboard** – Visualizes citizen interactions and public sentiment.
- **Feedback Loop** – Collects and stores citizen feedback for analytics.
- **Contextual Chat** – Tracks conversation history for personalized responses.
- **Data Storage** – Secure database for all queries, feedback, and insights.

---

3. Architecture

Frontend (HTML, CSS, JavaScript + Bootstrap)

- Provides a chatbot UI, feedback form, and analytics dashboard.
- Responsive design with accessibility features for all user groups.

Backend (Flask – Python)

- RESTful API routes for chat, feedback, sentiment, and dashboard.
- Handles Granite LLM integration and database operations.

LLM Integration (IBM Watsonx Granite)

- Processes natural language queries and generates human-like responses.
- Enhances chatbot with contextual awareness.

Database (SQLite/PostgreSQL)

- Stores citizen queries, feedback, and sentiment results.
- Supports analytics and reporting.

Architecture Flow

Citizen → Chat/Feedback → Flask API → IBM Granite + Database → Dashboard for Admins

---

4. Setup Instructions

Prerequisites

- Python 3.9+
- pip & virtual environment
- IBM Granite API credentials
- PostgreSQL/SQLite setup
- Git for version control

Installation Steps

1. Clone repository from GitHub.
2. Install dependencies from requirements.txt.
3. Configure .env with API keys & database URI.
4. Run backend: python app.py.
5. Launch frontend in browser.
6. Access dashboard and chatbot modules.

---

5. Folder Structure

CitizenAI/

|— app.py # Entry point for Flask

|— config.py # Environment configuration

|— .env # API keys & secrets

|— /routes # Chat, feedback, dashboard APIs

|— /templates # HTML files

|— /static # CSS, JS, images

|— /models # Database schemas

|— /utils # Sentiment analysis, helpers

|— init_db.py # DB initialization

---

6. Running the Application

- Start Flask server: python app.py.
- Open frontend UI in browser (http://localhost:5000).
- Interact with chatbot in **Chat Page**.
- Submit feedback in **Feedback Page**.
- Admins monitor data in **Dashboard Page**.

---

7. API Documentation

- **POST /chat** → Returns AI response for citizen query.
- **POST /feedback** → Stores citizen feedback & sentiment.
- **GET /dashboard** → Provides sentiment stats & user trends.
- **GET /history** → Returns previous chat sessions for context.

---

8. Authentication

- Environment variables (.env) protect API keys.
- Planned enhancements:
    - JWT-based user authentication.
    - Role-based access (Citizen/Admin).
    - Encrypted feedback storage.

---

9. User Interface

- **Chatbot UI** – Real-time AI assistant.
- **Feedback Form** – Simple interface for submitting citizen feedback.
- **Dashboard** – Displays:

- Sentiment trends
- Interaction frequency
- Feedback analysis charts

Design ensures accessibility with clear labels, responsive layout, and minimal clutter.

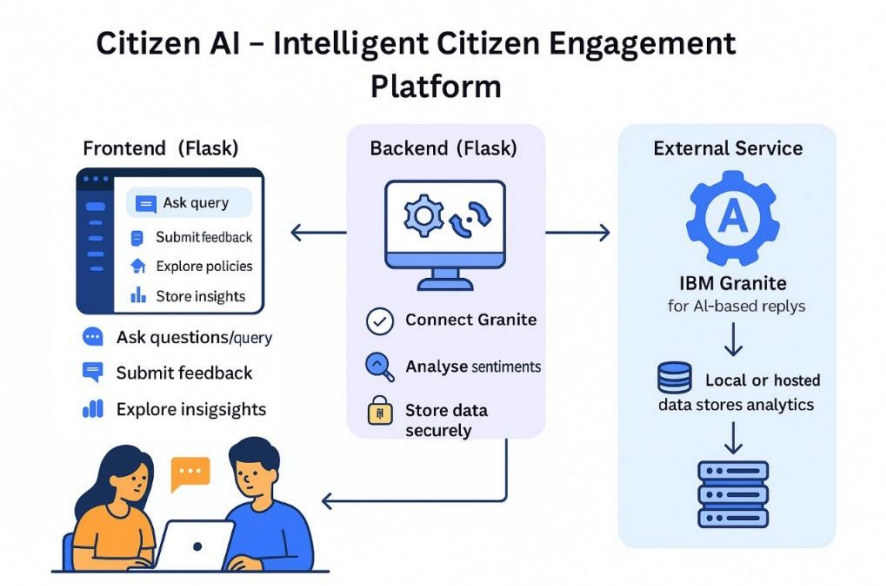
---

10. Testing

- **Unit Testing** – Checked Flask routes and sentiment classifier.
- **Integration Testing** – Verified backend–frontend sync.
- **Performance Testing** – Ensured response <3s per query.
- **Edge Cases** – Invalid inputs, long queries, multi-user load.

---

11. Screenshots



---

12. Known Issues

- Occasional latency in LLM responses.
- Sentiment classifier mislabels neutral feedback in rare cases.
- UI freezes on long API calls (partially fixed with async).

---

13. Future Enhancements

- Add multi-language support for citizens.
- Deploy on cloud with CI/CD pipeline.

- Enable OAuth2/JWT-based authentication.
- Improve sentiment analysis with advanced NLP models.
- Expand dashboard with predictive analytics (e.g., forecasting service demand).