“Citizen AI–Intelligent Citizen Engagement Platform”

Submitted by

Team Members

S.Mukesh

S.Praveen Kumar

D.Sridhar

Team Leader:

P.Sam David

Topics

S.No.	Title
1.	BRAINSTORMING & IDEATION
2.	REQUIREMENT ANALYSIS
3.	PROJECT DESIGN
4.	PROJECT PLANNING
5.	PROJECT DEVELOPMENT
6.	FUNCTIONAL & PERFORMANCETESTING
7.	DEPLOYMENT

BRAINSTORMING & IDEATION

Objective:

Problem Context

"Governments often struggle with delayed and inefficient communication with the public. Traditional systems such as phone calls, static websites, or manual help desks make it difficult for citizens to receive timely and relevant information regarding public services, policies, and civic issues.

Additionally, the government lacks effective tools to understand the overall mood or sentiment of citizens based on their feedback. As a result, citizen trust and engagement with digital governance platforms remain limited.

Purpose of the Project

- Citizen AI was developed to solve these challenges by introducing a real-time AI assistant capable of:
- Answering citizen questions with human-like responses using IBM Granite models.
- Analyzing citizen feedback to detect sentiments such as satisfaction or dissatisfaction.
- Presenting actionable insights on a dynamic dashboard.
- Enhancing government responsiveness, transparency, and public trust through AI-powered interactions.

Key Points:

➢ Problem Statement

- Citizens face delays in receiving information or reporting issues.
- Feedback collected by government departments is rarely analyzed in real time.

- There is no central platform to combine public service interaction, sentiment analysis, and policy responsiveness.

➢ Proposed Solution

- **Citizens:** To ask questions, provide feedback, and get quick answers.
- **Government Officials:** To monitor public sentiment and improve services.
- **Policy Makers:** To use data insights for better governance decisions.
- **Developers/Admins:** To manage system operations and improve AI performance.

➢ Target Users

- **City Residents:** To report civic issues easily and get eco-advice..
- **City Administrators:** To monitor feedback, analyze KPIs, and respond to anomalies.
- **Urban Planners:** To summarize long documents and make informed policy decisions.
- **Teachers & Students:** To explore sustainability practices via the Eco Tips Generator..
- **Government Departments:** For utility monitoring and forecasting resource demands.

➢ Expected Outcomes

- A responsive, intelligent chatbot available 24/7 for public queries.
- Real-time analysis of citizen feedback to detect trends and issues.
- Dashboards that visualize public mood and interaction frequency.
- Enhanced digital governance through AI, leading to improved public satisfaction and trust.

REQUIREMENT ANALYSIS

Objective:

Functional Requirements:

The Citizen AI platform is designed to offer several key functionalities that enhance citizen engagement and government responsiveness. At the core of the system is a real-time chat assistant that allows users to interact with public service information using natural language. The platform processes user queries and returns intelligent responses using IBM Granite large language models (LLMs).

In addition, citizens can submit feedback on government services through a user-friendly web interface. This feedback is stored securely and is automatically analyzed for sentiment—classified as positive, neutral, or negative—to help identify areas of concern or satisfaction.

A dynamic dashboard visualizes this sentiment data along with interaction trends and other metrics, enabling decision-makers to understand public opinion in real time. The system also features a contextual response engine that tracks user interactions and provides personalized replies based on the conversation history, enhancing the overall relevance of the chatbot's responses.

All citizen interactions, feedback entries, and sentiment results are persistently stored in a backend database to support analytics and reporting.

Technical Requirements:

The technical implementation of Citizen AI relies on a modern, scalable, and secure architecture. The backend is built using Python with the Flask framework, allowing for modular API routing and a clean project structure. The frontend is implemented using HTML, CSS, and JavaScript to create responsive user interfaces, including the chatbot, feedback form, and analytics dashboard.

AI functionality is powered by IBM Granite, which handles natural language understanding and response generation. The system connects to a relational database such as SQLite or PostgreSQL to store user feedback, sentiment scores, and session logs. Configuration settings—including API keys and database URIs—are securely managed using .env files and a centralized config.py module.

To ensure reliability and performance, the application is expected to maintain low response latency (under 3 seconds per query) and support concurrent users through optimized backend routes and efficient frontend rendering. Basic security measures like input validation and environment-based configuration are implemented to protect user data and prevent vulnerabilities.

The system also includes provisions for unit and integration testing to maintain code quality and long-term maintainability.

Key points:

➢ Technical Requirements:

- **Backend developed using Flask (Python)** with RESTful routing.
- **Frontend created using HTML, CSS, and JavaScript** for interactive UI.
- **AI integration handled via IBM Granite API** for NLP and chat responses.
- **Database setup using SQLite or PostgreSQL** for structured data storage.
- **Sensitive data and configuration managed** through .env and config.py
- **Chat latency kept under 3 seconds per response** for smooth user experience.
- **Scalable and modular design** to support multiple users and future features.
- **Input sanitization and secure API key handling** for data protection.
- **Unit and integration tests included** to ensure quality and reliability.

➢ Functional Requirements:

- **Enable real-time chatbot interaction** using IBM Granite.
- **Accept citizen queries** and provide human-like responses.
- **Allow feedback submission** through a web form interface.
- **Automatically analyze sentiment** (Positive, Neutral, Negative) from feedback.
- **Display sentiment insights and user trends** on a dynamic dashboard.
- **Provide personalized and context-aware replies** based on session history.
- **Store all user queries, feedback, and sentiment results** in a secure database.

➢ Constraints & Challenges:

- **Granite API Token Limitations:**
  - May limit the length or frequency of messages; prompt optimization is required.
- **Latency in Real-Time Communication:**

    - Long response times may degrade user experience; handled with asynchronous support.
- **Sentiment Classification Accuracy:**
    - Simple models may misclassify neutral or negative tones, requiring refinement.
- **Data Privacy & Consent:**
    - Citizen feedback may include sensitive content; secure storage and consent protocols are essential.

- **User Diversity:**
  - Citizens with varying levels of digital literacy require an intuitive and accessible user interface.
- **Deployment & Infrastructure:**
  - Hosting costs, optional cloud configuration, and domain management may be required for demo readiness and scalability.

PROJECT DESIGN

Objective:

The objective of the Project Design phase is to define the overall structure, flow, and technical organization of the Citizen AI platform. It includes the layout of system components, the interaction between modules, and the architecture that enables real-time chatbot responses, sentiment analysis, feedback handling, and analytics dashboard visualization.

This design ensures that the system remains scalable, modular, and maintainable, while delivering a smooth experience to both citizens and administrators.

Key points:

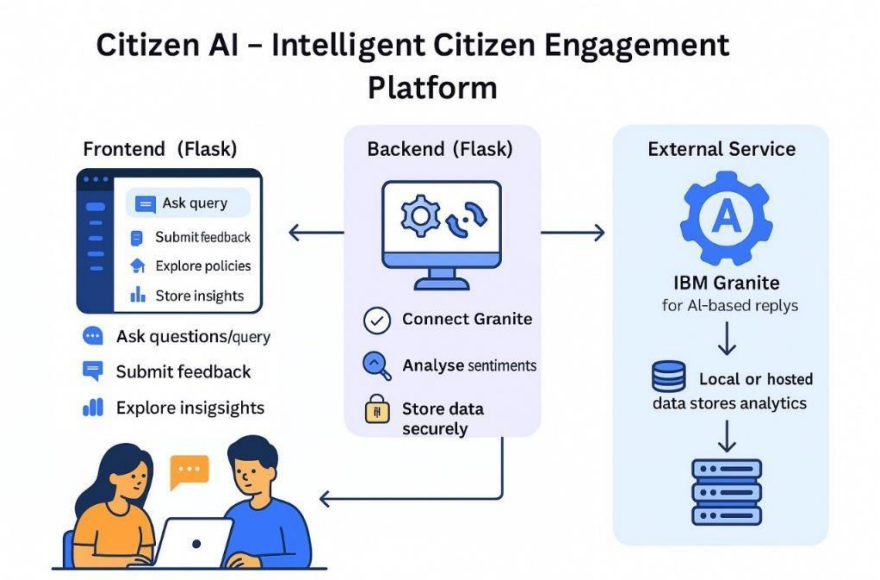
The Citizen AI platform follows a modular Flask architecture with distinct components for routes, models, templates, and static files—ensuring clean structure and maintainability.

User queries are handled via chat_routes.py, processed by granite_interface.py using IBM Granite, and returned in real time through chat.js. Feedback is collected via feedback_routes.py, analyzed by sentiment_analysis.py, and securely stored in the database.

Administrators access insights through dashboard_routes.py, which fetches and visualizes sentiment trends and interaction metrics. Contextual responses are enabled by session tracking, allowing the chatbot to deliver personalized replies.

Configuration settings are securely managed through .env files and the centralized config.py module, supporting smooth, AI-driven workflows across the platform.

➢ System Architecture Diagram:



➢ User flow:

- **A citizen opens the platform** and interacts with the chatbot by typing a query.
- **The message is sent to the backend**, where IBM Granite processes it and returns a response.
- **The citizen may also submit feedback** using a form, which is analyzed for sentiment and stored in the database.
- **An admin logs into the dashboard** to monitor overall sentiment trends and feedback analytics.
- **The system tracks session history**, enabling context-aware, more personalized responses for returning or active users.

➢ UI/UX Consideration:

The Citizen AI platform is designed with a clean, responsive, and user-friendly interface to ensure ease of access for users of all digital literacy levels. The chatbot interface provides real-time feedback through smooth interactions and toast notifications to confirm actions. The layout leverages **Bootstrap** for responsive design, ensuring compatibility across devices such as desktops, tablets, and mobile phones.

High-contrast colors and readable font sizes are used to support accessibility, and input fields are clearly labeled to enhance usability. The **dashboard** presents key metrics through intuitive charts with minimal clutter, allowing administrators to quickly understand public sentiment and engagement.

The overall design prioritizes **simplicity**, **clarity**, and **functionality** to foster user engagement and build trust in the platform

PROJECT PLANNING

Objective:

The objective of the Project Planning phase is to structure the development of the Citizen AI form into manageable tasks and timeframes using an **Agile methodology**. By organizing work into focused sprints and distributing responsibilities across the team, the project ensures steady progress, timely feedback, and successful completion of each module.

This structured plan includes **sprint goals**, **task allocation**, and **milestone tracking**, enabling the team to adapt quickly to changes while maintaining momentum and quality throughout the development lifecycle.

Key Points:

➢ Sprint Planning:

- **Sprint 1: Project Setup**
    - **Initialize Flask project with** app.py**,** config.py**,** .env**, and install dependencies**.
- **Sprint 2: Chatbot Development**
    - **• Build real-time chatbot with** chat_routes.py**,** granite_interface.py**, and frontend (**chat.html**,** chat.js.
- **Sprint 3: Feedback & Sentiment**
    - **Create feedback form**, classify sentiment using sentiment_analysis.py, and store data in the database
- **Sprint 4: Dashboard Integration**
    - **Develop admin dashboard** to visualize sentiment and engagement trends.
- **Sprint 5: Contextual Chat**
    - **Implement session-based contextual replies** using helpers.py.

- **Sprint 6: Database Setup**
    - **Define schemas in** models.py**, initialize database with** init_db.py.
- **Sprint 7: Testing & QA**
    - **Perform unit and integration testing**, and **optimize performance**.
- **Sprint 8: Deployment**
    - **Prepare deployment configs**, **add demo data**, and **finalize UI and documentation**.

PROJECT DEVELOPMENT

Objective:

The objective of the Project Development phase is to implement, integrate, and test all modules of the Citizen AI platform, transforming the design into a fully functional system. This includes building the **chatbot**, **sentiment analysis module**, **feedback system**, **dashboard**, and **database integration**, ensuring that all components work together seamlessly.

Key Points:

➢ Technology Stack Used:

- **Backend Framework:** Python with Flask (for modular API routing)
- **Frontend Technologies:** HTML, CSS, JavaScript (with Bootstrap)
- **AI Integration:** IBM Granite LLM (for chatbot and contextual responses)
- **Sentiment Analysis:** Custom Python module using NLP techniques
- **Database:** SQLite (development) / PostgreSQL (production-ready)
- **Configuration & Secrets:** .env file and config.py for environment-based setup
- **Version Control:** Git and GitHub
- **Deployment (Optional):** Docker, cloud hosting options

➢ Development Process:

The development of Citizen AI followed a structured and modular approach. The project was divided into independent yet connected components, ensuring flexibility and easier debugging. Each major feature was implemented and tested step by step—starting from backend setup to AI integration and frontend user experience.

Below are the key stages of the development process:

Flask Project Initialization:

- **Created the basic Flask structure** with app.py, config.py, and .env for secure settings.
- **Installed necessary dependencies** including Flask, IBM Granite SDK, and SQLAlchemy.

Chatbot Module Implementation:

- **Developed** chat_routes.py to handle incoming chat messages.
- **Integrated IBM Granite API** through granite_interface.py to fetch AI responses.
- **Designed** chat.html **and** chat.js for real-time user interaction on the frontend

FUNCTIONAL&PERFOMANCETESTING

Objective:

The primary objective of the Functional and Performance Testing phase is to ensure that all features of the Citizen AI platform work as intended and that the system performs reliably under expected user conditions. This phase involved testing each module individually and then in combination, checking for accuracy, responsiveness, and stability. The goal was to identify and resolve any bugs, inconsistencies, or performance bottlenecks before deployment.

➢ Deployment