Citizen AI Project Documentation

AI-Powered Citizen Services and City Analysis

Prepared By: Sivagurunathan

Abstract

The Citizen AI project is an AI-powered web application designed to provide government-related information, city analysis, and citizen services. It integrates IBM Granite generative AI with a Flask backend and an HTML/CSS frontend. The system enables natural language interaction between users and the AI, offering intelligent responses to citizen queries and analyzing city-specific data for improved service delivery.

Objectives

- Develop a conversational AI system to answer government and citizen service-related queries.
- Enable city-level analysis for better governance insights.
- Provide a simple and intuitive web-based interface for interaction.
- Ensure scalability with a modular architecture for future improvements.

System Architecture

- Frontend: HTML/CSS templates for web pages (Home, About, Services, Chat, Dashboard, Login).
- Backend: Flask framework to handle routes, authentication, and API calls.
- AI Model: IBM Granite (via HuggingFace Transformers) for text generation.
- Data Handling: In-memory storage for chat history, sentiment, and issues (with future database integration planned).

The architecture follows an MVC-like pattern, with Flask managing requests, the AI model processing inputs, and HTML templates rendering outputs dynamically.

Project Workflow

Activity 1: Project Setup and Architecture

- Select IBM Granite and required libraries (Transformers, Accelerate, BitsAndBytes, PyTorch).
- Define system architecture (Flask backend, HTML/CSS frontend, AI integration).
- Set up development environment.

Activity 2: Backend Core Functionalities

- Implement Flask routes (/, /about, /services, /chat, /dashboard, /login, /logout).
- Add authentication and session management.
- Integrate IBM Granite for response generation.
- Build helper functions (AI responses, sentiment analysis, formatting).

Activity 3: Data Handling and Logic

- Set up in-memory storage for history, sentiment, and issues.
- Implement logic for user input (queries, feedback, concerns).

- Aggregate dashboard data (sentiment counts, recent issues).

Activity 4: Frontend Development

- Create HTML templates (index.html, about.html, services.html, chat.html, dashboard.html, login.html).
- Apply styling using CSS.
- Build forms for input (chat, feedback, login).
- Display backend-generated content dynamically.

Activity 5: Integration and Testing

- Connect frontend templates to Flask routes.
- Test user flows: login, logout, navigation, chat, dashboard.
- Debug errors.

Activity 6: Refinement and Deployment

- Refine UI/UX and optimize inference performance.
- Configure server and database for deployment.
- Deploy application to hosting platform.
- Prepare documentation and user guides.

Implementation Details

- Flask Backend: Core routes, authentication, and API integration with AI model.
- AI Model: IBM Granite loaded via HuggingFace with torch optimizations.
- Frontend: Responsive HTML templates styled with CSS.
- Data Handling: In-memory structures storing chat history and dashboard metrics.

Testing & Results

- Verified successful login/logout flows.
- AI model responds to citizen queries with contextual accuracy.
- Dashboard reflects sentiment analysis and recent user concerns.
- End-to-end testing confirms stable integration of frontend and backend.

Future Enhancements

- Integrate a persistent database (PostgreSQL/MySQL).
- Add role-based access control for admin dashboards.
- Expand AI responses with fine-tuning on government datasets.
- Deploy with containerization (Docker/Kubernetes) for scalability.

Conclusion

The Citizen AI project demonstrates how generative AI can be integrated into government service platforms to improve accessibility, responsiveness, and transparency. With further development, the system can become a robust citizen support platform.