**Project Report: Citizen AI – Intelligent Citizen Engagement Platform**

**Project Description**

Citizen AI is an intelligent citizen engagement platform designed to revolutionize how governments interact with the public. By leveraging a powerful technology stack that includes a **Flask** backend, **IBM Granite** models for conversational AI, and the **TextBlob** library for sentiment analysis, Citizen AI provides real-time, AI-driven responses to citizen inquiries.

The platform integrates Natural Language Processing (NLP) and sentiment analysis to assess public opinion, track emerging issues, and generate actionable insights for government agencies. A dynamic analytics dashboard, powered by **Chart.js**, offers real-time visualizations of citizen feedback, helping policymakers enhance service delivery and transparency. By automating routine interactions and enabling data-driven governance, Citizen AI aims to improve citizen satisfaction, government efficiency, and public trust.

**Scenarios**

* **Scenario 1: Real-Time Conversational AI Assistant** A citizen visits the official government website and uses the chat assistant to ask, "What are the timings for the city library?" The system, powered by IBM Granite, instantly processes the query and provides a precise, human-like response with the library's operating hours and location.
* **Scenario 2: Citizen Sentiment Analysis** After a new public transport route is launched, citizens use the platform's feedback form to share their opinions. The system automatically analyzes each submission using TextBlob, classifying it as "Positive," "Neutral," or "Negative." This allows officials to quickly gauge public reception without manual review.
* **Scenario 3: Dynamic Dashboard** A department head logs into the Citizen AI dashboard and views a bar chart showing a sharp increase in "Negative" sentiment over the past week. By reviewing the raw feedback, they identify the cause—a recurring issue with waste management in a specific district—and can dispatch a team to address it promptly.

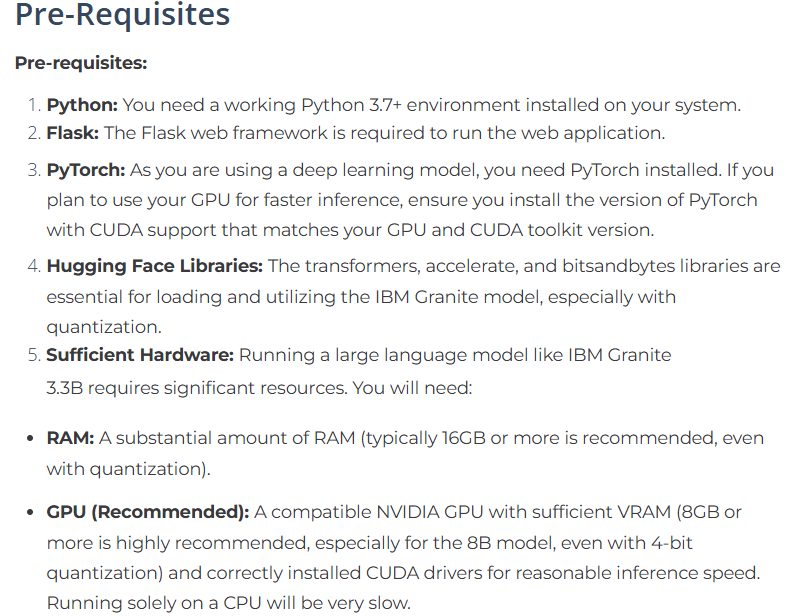
**Technical Architecture**

The Citizen AI platform is built on a robust multi-layer architecture designed for scalability and efficiency.

**CitizenAI - Architecture Diagram**



|  |  |  |
| --- | --- | --- |
| **Layer** | **Technology** | **Components & Functions** |
| **User Interface Layer** | HTML, CSS, JS | chat.html, feedback.html, dashboard.html |
|  |  | Provides the interactive frontend for citizen chat, feedback submission, and the government analytics dashboard. |
| **Application Layer** | Python / Flask | app.py |
|  |  | Manages incoming requests, business logic, and routing. Connects the frontend to the backend services. - chat() - feedback() - dashboard() |
| **Data Layer** | CSV File | feedback.csv |
|  |  | Provides persistent storage for citizen feedback and sentiment analysis results. |
| **AI Service Layer** | IBM Cloud / Python Lib | IBM Granite, TextBlob |
|  |  | Handles complex AI tasks. - **IBM Granite:** Generates conversational responses. - **TextBlob:** Performs sentiment analysis. |

****

**Project Milestones & Activities**

**ACTIVITY 1 : Environment Setup & Architecture**

**Focused on establishing the project's foundation and selecting the core technologies.**

* **Activity 1.1: Set up the Development Environment**
  1. Install Python and Pip.
  2. Create a virtual environment to isolate project dependencies.
  3. Install required libraries: pip install flask textblob requests

**ACTIVITY 2 : Core Functionality Development**

**Focused on building the primary features of the platform.**

* **Activity 2.1: Develop Core Backend Functions**
  1. **AI Chat System:** Implement the ask\_granite\_model() function to handle secure API calls to the IBM Granite model.
  2. **Sentiment Analysis System:** Implement the analyze\_sentiment() function using the TextBlob library for fast, local sentiment classification.
  3. **Data Persistence:** Create logic to write and read from the feedback.csv file.

**ACTIVITY 3 : app.py (Flask) Development**

**Focused on creating the main application logic and API endpoints.**

* **Activity 3.1: Write the Main Application Logic**
  1. **Imports and Setup:** Import Flask, TextBlob, requests, etc. Load necessary configurations.
  2. **API Endpoints:**
     + @app.route('/'): Serves the main chat interface.
     + @app.route('/chat'): Handles real-time chat requests from the frontend.
     + @app.route('/feedback'): Manages submission of citizen feedback.
     + @app.route('/dashboard'): Aggregates data from the CSV and serves it to the dashboard.
  3. **System Prompting:** Create a robust system prompt for the IBM Granite model to ensure responses are contextual and appropriate for a civic assistant.

**CODE :**

**from flask import Flask, request, jsonify, render\_template**

**from textblob import TextBlob**

**import csv**

**import os**

**import requests**

**# Initialize the Flask application**

**app = Flask(\_\_name\_\_)**

**# Ensure the 'data' directory exists**

**if not os.path.exists('data'):**

**os.makedirs('data')**

**# Route for the main chat page**

**@app.route('/')**

**def home():**

**"""Renders the main chat interface."""**

**return render\_template('chat.html')**

**# API endpoint to handle chat messages**

**@app.route('/chat', methods=['POST'])**

**def chat():**

**"""**

**Receives a user's message, gets a response from the IBM Granite model,**

**and returns it as JSON.**

**"""**

**user\_message = request.json['message']**

**ai\_response = ask\_granite\_model(user\_message)**

**return jsonify({"reply": ai\_response})**

**# Route for submitting and viewing feedback**

**@app.route('/feedback', methods=['GET', 'POST'])**

**def feedback():**

**"""**

**Handles feedback submission. On POST, it analyzes sentiment,**

**saves the feedback to a CSV, and shows a confirmation.**

**On GET, it just shows the feedback form.**

**"""**

**if request.method == 'POST':**

**feedback\_text = request.form['feedback']**

**sentiment = analyze\_sentiment(feedback\_text)**

**# Save the feedback and sentiment to a CSV file**

**with open('data/feedback.csv', 'a', newline='', encoding='utf-8') as f:**

**writer = csv.writer(f)**

**writer.writerow([feedback\_text, sentiment])**

**return render\_template('feedback.html', message="Feedback submitted!", sentiment=sentiment)**

**return render\_template('feedback.html')**

**# Route for the analytics dashboard**

**@app.route('/dashboard')**

**def dashboard():**

**"""**

**Reads the feedback CSV file, counts the sentiment data,**

**and passes it to the dashboard template for visualization.**

**"""**

**sentiment\_counts = {"Positive": 0, "Neutral": 0, "Negative": 0}**

**if os.path.exists('data/feedback.csv'):**

**with open('data/feedback.csv', 'r', encoding='utf-8') as f:**

**reader = csv.reader(f)**

**for row in reader:**

**if len(row) >= 2:**

**sentiment = row[1]**

**if sentiment in sentiment\_counts:**

**sentiment\_counts[sentiment] += 1**

**return render\_template('dashboard.html', data=sentiment\_counts)**

**def analyze\_sentiment(text):**

**"""**

**Analyzes the sentiment of a given text using TextBlob.**

**Returns 'Positive', 'Negative', or 'Neutral'.**

**"""**

**polarity = TextBlob(text).sentiment.polarity**

**if polarity > 0:**

**return "Positive"**

**elif polarity < 0:**

**return "Negative"**

**else:**

**return "Neutral"**

**def get\_iam\_token(api\_key):**

**"""Obtains an IAM token from IBM Cloud using an API key."""**

**url = "https://iam.cloud.ibm.com/identity/token"**

**headers = {"Content-Type": "application/x-www-form-urlencoded"}**

**data = {"grant\_type": "urn:ibm:params:oauth:grant-type:apikey", "apikey": api\_key}**

**try:**

**response = requests.post(url, headers=headers, data=data)**

**response.raise\_for\_status()**

**return response.json()["access\_token"]**

**except requests.exceptions.RequestException as e:**

**print(f"Error getting IAM token: {e}")**

**return None**

**def ask\_granite\_model(prompt):**

**"""**

**Sends a prompt to the IBM Granite model and returns the generated text.**

**Includes a system prompt to give the AI context.**

**"""**

**# IMPORTANT: Replace with your actual API key and project ID**

**api\_key = "YOUR\_IBM\_CLOUD\_API\_KEY"**

**project\_id = "YOUR\_WATSONX\_PROJECT\_ID"**

**token = get\_iam\_token(api\_key)**

**if not token:**

**return "Error: Could not authenticate with IBM Cloud."**

**url = "https://us-south.ml.cloud.ibm.com/ml/v1/text/generation?version=2024-05-01"**

**headers = {**

**"Content-Type": "application/json",**

**"Authorization": f"Bearer {token}"**

**}**

**system\_prompt = (**

**"You are Citizen AI, an intelligent assistant that helps Indian citizens with information about "**

**"government services, civic issues, public documents, reporting problems, and general guidance. "**

**"Respond politely and accurately based on Indian context."**

**)**

**full\_prompt = f"{system\_prompt}\n\nCitizen Question: {prompt}\nAnswer:"**

**payload = {**

**"model\_id": "ibm/granite-13b-instruct-v2",**

**"project\_id": project\_id,**

**"input": full\_prompt,**

**"parameters": {**

**"decoding\_method": "greedy",**

**"max\_new\_tokens": 150**

**}**

**}**

**try:**

**response = requests.post(url, headers=headers, json=payload)**

**response.raise\_for\_status()**

**result = response.json()**

**return result.get("results", [{}])[0].get("generated\_text", "Sorry, I could not generate a response.")**

**except requests.exceptions.RequestException as e:**

**return f"API Error: Could not connect to the AI service. {e}"**

**ACTIVITY 4 : Frontend Development**

**Focused on building a clean, intuitive, and responsive user interface.**

* **Activity 4.1: Design and Develop the User Interface**
  1. **Chat Interface (chat.html):** Create the main chat window with a message display area and an input form.
  2. **Feedback Form (feedback.html):** Design a simple and clear form for feedback submission.
  3. **Dashboard UI (dashboard.html):** Create a container for the sentiment analysis visualization.
* **Activity 4.2: Implement Dynamic Functionality**
  1. **Asynchronous Chat:** Use JavaScript's fetch API in chat.html to communicate with the /chat endpoint without reloading the page.
  2. **Data Visualization:** Use Chart.js in dashboard.html to render the sentiment **data passed from Flask into a dynamic bar chart.**

**CODE :**

**<!DOCTYPE html>**

**<html lang="en">**

**<head>**

**<meta charset="UTF-8">**

**<title>Citizen Feedback</title>**

**<style>**

**body { font-family: sans-serif; background-color: #f4f4f9; display: flex; justify-content: center; align-items: center; height: 100vh; margin: 0; }**

**.feedback-container { background: white; padding: 2rem; border-radius: 8px; box-shadow: 0 4px 8px rgba(0,0,0,0.1); text-align: center; }**

**textarea { width: 100%; min-height: 100px; margin-bottom: 1rem; padding: 0.5rem; border-radius: 4px; border: 1px solid #ccc; }**

**input[type="submit"] { background-color: #2563eb; color: white; padding: 0.75rem 1.5rem; border: none; border-radius: 4px; cursor: pointer; font-size: 1rem; }**

**input[type="submit"]:hover { background-color: #1e40af; }**

**p { margin-top: 1rem; color: #333; font-size: 1.1rem; }**

**</style>**

**</head>**

**<body>**

**<div class="feedback-container">**

**<h2>Submit Your Feedback</h2>**

**<p>Let us know about your experience or suggestions.</p>**

**<form method="post">**

**<textarea name="feedback" placeholder="Your feedback is valuable..." required></textarea><br><br>**

**<input type="submit" value="Submit Feedback">**

**</form>**

**{% if message %}**

**<p><strong>{{ message }}</strong><br>We've recorded your sentiment as: <strong>{{ sentiment }}</strong></p>**

**{% endif %}**

**</div>**

**</body>**

**</html>**

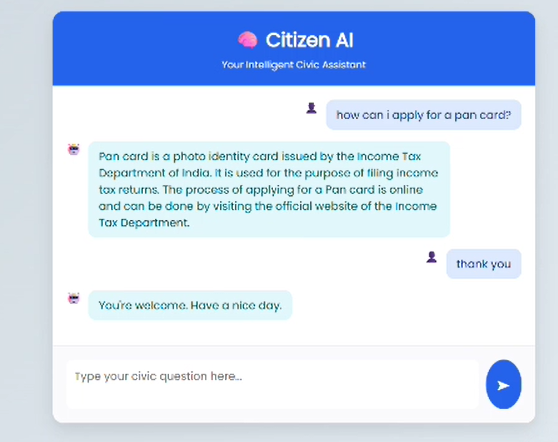
**ACTIVITY 5 : Testing & Deployment**

**Focused on ensuring the application is robust, reliable, and ready for use.**

* **Activity 5.1: Prepare for Deployment**
  1. **API Key Management:** Securely store and handle IBM Cloud API keys (using environment variables is recommended).
  2. **Dependency Management:** Finalize the requirements.txt file with all necessary packages.
* **Activity 5.2: Test and Deploy the Application**
  1. **Local Testing:** Run the application locally (python app.py) and test all features for functionality.
  2. **Cloud Deployment:** Deploy the application to a cloud platform (e.g., Heroku, AWS, Google Cloud) for public access.

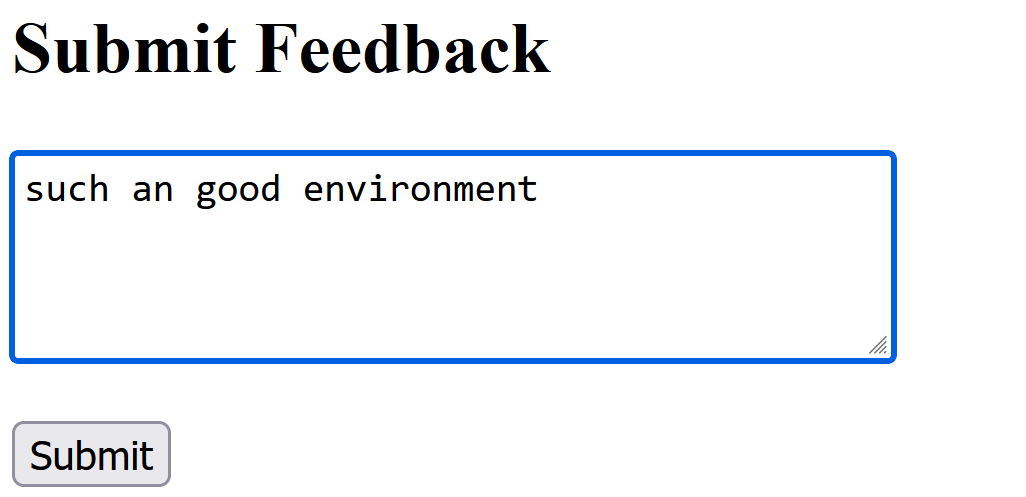
**Exploring Application Features**

1. **Real-Time AI Assistant**



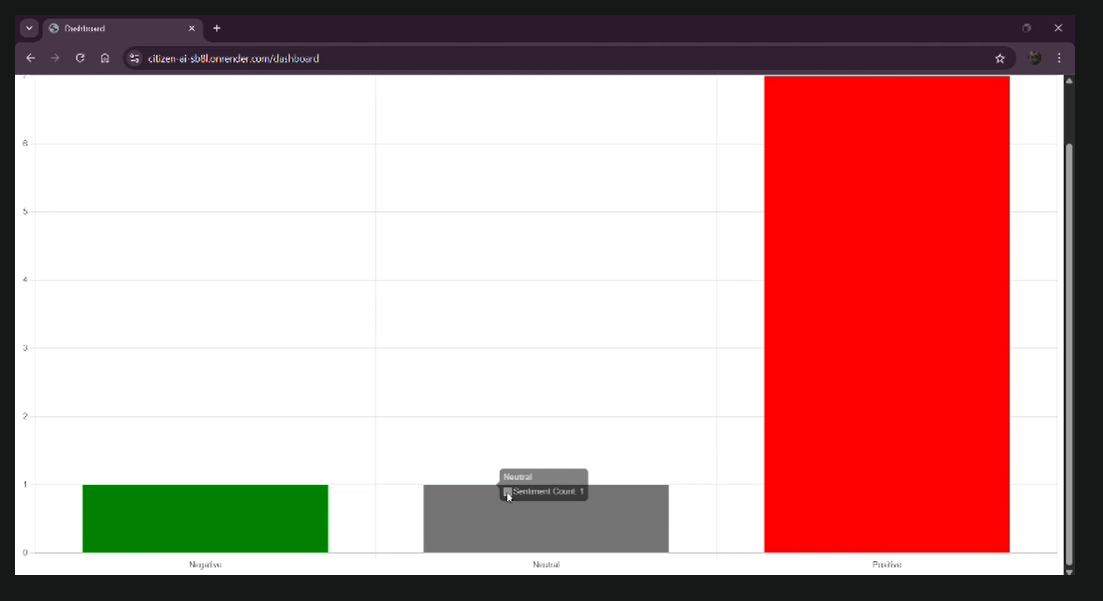
**Description:** This feature provides citizens with 24/7 access to information through a conversational AI. Built with Flask and JavaScript, the interface sends user queries to the IBM Granite model and displays responses in real-time. The design, controlled by style.css, focuses on readability and ease of use.

**2. Citizen Feedback & Sentiment Analysis**



**Description:** The feedback page allows users to submit their thoughts and suggestions. When a form is submitted, the Flask backend uses the TextBlob library to instantly analyze the sentiment of the text. The feedback and its corresponding sentiment are then stored in a feedback.csv file for record-keeping and analysis.

1. **Dynamic Analytics Dashboard**

****

A powerful visualization tool for government officials.

**Description:** The dashboard provides at-a-glance insights into public sentiment. It reads the aggregated data from the feedback.csv file and uses the **Chart.js** library to render an intuitive bar chart. This enables policymakers to easily track trends, identify areas of concern, and make data-driven decisions to improve public services.

**Conclusion**

Utilizing Flask, IBM Watson AI, and Python's data science libraries, the Citizen AI platform ensures accurate civic information delivery and comprehensive sentiment analysis. The systematic development process—spanning model integration, core feature implementation, and frontend development with JavaScript and Chart.js—has led to the creation of an interactive, educationally-focused platform.

Built within the Flask framework, Citizen AI facilitates seamless interaction with AI-powered civic tools and provides dynamic visualizations of public sentiment. This project highlights how targeted AI models and a well-structured web framework can enhance civic engagement, accessibility, and effectiveness.