DOCUMENTATION OF THE PROJECT CITIZEN AI

Citizen AI – Intelligent Citizen Engagement Platform

INTRODUCTION:

- Project Title: Citizen Ai
- Team Leader : Lokesh. M
- Team Member : Thirisha. G
- Team Member : Vishali. N
- Team Member : Abdul sharif. S
- Team Member : Hariprathap. J

PROJECT DESCRIPTION:

Citizen AI is an intelligent citizen engagement platform designed to revolutionize how governments interact with the public. Leveraging Flask, IBM Granite models, and IBM Watson, Citizen AI provides real-time, AI-driven responses to citizen inquiries regarding government services, policies, and civic issues. The platform integrates natural language processing (NLP) and sentiment analysis to assess public sentiment, track emerging issues, and generate actionable insights for government agencies. A dynamic analytics dashboard 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 improves citizen satisfaction, government efficiency, and public trust in digital governance.

SCENARIO'S OF CITIZEN AI:

1. Real-Time Conversational AI Assistant:

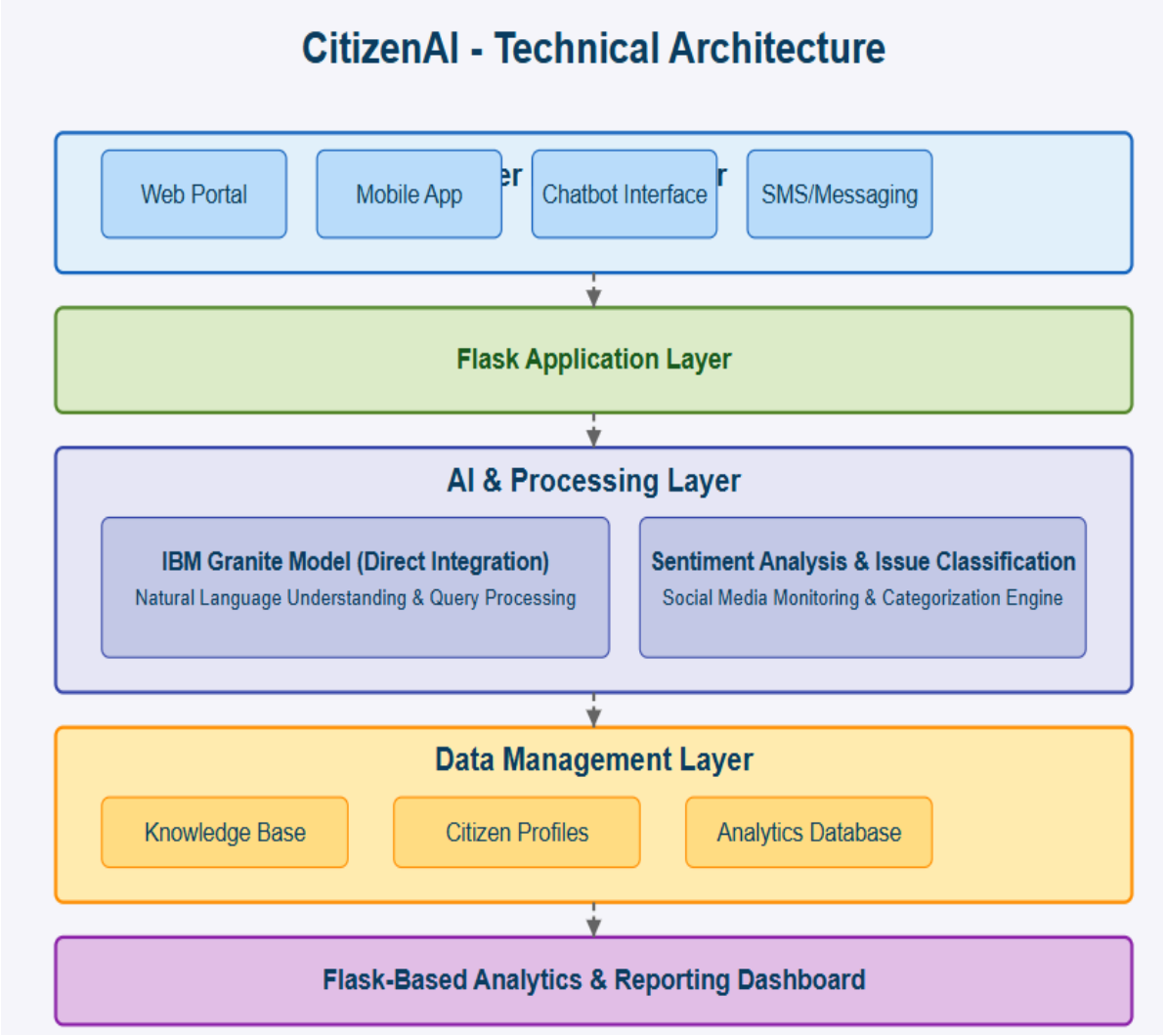
The Real-Time Conversational AI Assistant in Citizen AI serves as the primary interface for citizen interaction.

2. Citizen Sentiment Analysis:

Citizen Sentiment Analysis in Citizen AI is a core feature designed to understand the public's feelings about government services and related topics.

3. Dynamic Dashboard:
   The Dynamic Dashboard in Citizen AI serves as a central hub for government officials to gain real-time insights into citizen feedback and interactions.

TECHNICAL ARCHITECTURE OF CITIZEN AI



1. Python: You need a working Python 3.7+ environment installed on your system.
2. Flask: The Flask web framework is required to run the web application.
3. PyTorch: As you are using a deep learning model, you need PyTorch installed. If you plan to use your GPU for faster inference, ensure you install the version of PyTorch with CUDA support that matches your GPU and CUDA toolkit version.
4. Hugging Face Libraries: The transformers, accelerate, and bitsandbytes libraries are essential for loading and utilizing the IBM Granite model, especially with quantization.
5. Sufficient Hardware: Running a large language model like IBM Granite.3.3B requires significant resources. You will need:

- RAM: A substantial amount of RAM (typically 16GB or more is recommended, even with quantization).
- GPU (Recommended): A compatible NVIDIA GPU with sufficient VRAM (8GB or more is highly recommended, especially for the 8B model, even with 4-bit quantization) and correctly installed CUDA drivers for reasonable inference speed. Running solely on a CPU will be very slow.

6. Internet Connection: The first time you run the application, the IBM Granite model files will be downloaded from the Hugging Face Hub. You need an active internet connection for this.
7. Project Structure: The project files should be organized correctly with app.py, a templates folder containing your HTML files (index.html, about.html, services.html, chat.html, dashboard.html, login.html), and a static folder containing your CSS (styles.css) and image/favicon subfolders (e.g., static/Images, static/Favicon).

PROJECT WORKFLOW :

PROJECT SETUP:-

Select and Confirm AI Model

1. Review the key functionalities of CitizenAI, including chat responses, sentiment analysis, concern reporting, and dashboard insights.
2. Identify the type of AI capabilities needed: natural language understanding (NLU), text generation, and basic text analysis.

Confirm AI Model & Libraries:

1. Confirm the selection of the IBM Granite model for core AI capabilities.
2. Specify the necessary Python libraries: Flask for the web framework, PyTorch for the AI model backend, and Hugging Face libraries (transformers, accelerate, bitsandbytes) for model handling.

Explore Library Documentation:

1. Review documentation for selected libraries to understand model loading, inference, quantization, and device handling.
2. Examine Flask documentation for routing, templating, and session management.

Set Up the Development

Environment Install Necessary Tools:

1. Ensure Python (3.7+) and pip are installed for managing project dependencies.
    a. **Install Flask and Dependencies:**
2. Use pip to install Flask and any other required backend libraries:
    a. pip install Flask

b. **Install AI/ML Libraries:**

3. Install the necessary libraries for AI model integration:
    a. pip install torch transformers accelerate bitsandbytes
    b. (Ensure you install the correct PyTorch version for your CUDA setup if using a GPU).
    c. Set Up Application Structure.

DESIGN THE LAYOUT AND STYLING USING CSS

- create and apply css rules to control the visual presentation and layout of your web pages.
- create or update the main css stylesheet (static/css/styles.css) to define the overall look and feel, including font styles, colors, and spacing for common elements.
- implement layout techniques within your css (e.g., using flexbox for centering elements like the login box, or adjusting margins and padding for content areas) to arrange elements on the page effectively.
- (optional but recommended) consider implementing media queries in your css to ensure the layout and styling are responsive and look good on different screen sizes (desktops, tablets, mobile phones).
- apply specific visual styles (backgrounds, borders, text colors) to individual elements or sections, potentially using inline <style> blocks in specific html
- files for page-unique styles (like the background image on index.html or about.html).

CREATE SEPARATE PAGES FOR EACH CORE FUNCTIONALITY

- Develop the specific content and interactive elements for each distinct page of the application**.**
- index.html: Design the landing page with an introduction to CitizenAI and a clear call to action (e.g., the "Get Started" button linking to login).
- login.html: Create the page with the login form, including input fields for username/email and password, and a submit button. Include a placeholder for displaying login error messages.
- about.html: Develop the page containing information about the project's mission, features, and impact.
- services.html: Create a page detailing the services offered by CitizenAI.
- chat.html: Design the interface for the AI chat assistant, including a form for user input (question) and a dedicated area to display the AI's generated response. This page might also include forms for submitting feedback and concerns.
- dashboard.html: Build the page to display aggregated data, such as sentiment counts and a list of reported issues.
- Each of these pages will contain the necessary user input forms and designated areas where dynamic content from the backend (like AI responses or dashboard data) will be displayed

IMPLEMENT THE FLASK BACKEND FOR MANAGING ROUTING AND USER INPUT PROCESSING :-

This activity involves writing the Python code in app.py to define the web application's structure, handle different page requests, process data submitted by users, and integrate with the AI model logic.

Define Routes in Flask:

Set up distinct routes in app.py using the @app.route() decorator for each page and key interaction point: / (Home), /about, /services, /chat, /dashboard, /login (GET and POST), /logout.

1. Define routes specifically for handling form submissions: /ask (for chat questions), /feedback (for sentiment analysis input), and /concern (for reporting issues), ensuring they accept POST requests.
2. Link each defined route to a corresponding Python function that will execute when that URL is accessed or form is submitted.
3. Ensure that within each route function, the appropriate HTML template is rendered using render_template(), passing any necessary data to the template.

Process User Input:

1. Ensure that HTML forms (e.g., in chat.html, login.html) have the correct method="POST" and action="{{ url_for('route_name') }}" attributes to send data to the defined backend routes.
2. Within the Flask functions handling POST requests (e.g., ask_question(), submit_feedback(), login()), use Flask's request.form to safely retrieve data submitted by the user from the HTML forms (e.g., request.form.get('question'), request.form.get('username')).
3. Perform any necessary basic validation on the retrieved user input (e.g., checking if fields are empty).

Integrate AI Model Calls and Logic:

Within the route functions that require AI processing (e.g., the /ask route), call the previously implemented AI helper functions (like granite_generate_response()).

1. Pass the processed user input (e.g., the user's question) as arguments to the AI helper functions.
2. In routes handling feedback or concerns (e.g., /feedback, /concern), call the relevant processing functions (like analyze_sentiment()) with the user- provided text.
3. Capture and process the results returned by the AI helper functions or data processing logic (e.g., the generated text response, the sentiment label).
4. Prepare the results to be sent back to the frontend by passing them as arguments to the render_template() function (e.g., render_template("chat.html", question_response=response)).

Setup Instructions OF GOOGLE COLLAB

1. **Run Environment**

- Platform: **Google Colab**
- Runtime Type: **GPU (T4)**
- Go to Runtime > Change runtime type
- Select **GPU** as the hardware accelerator

2. Install Dependencies

3. Model & Tokenizer

- Model: ibm-granite/granite-3.2-2b-instruct
- Automatically selects float16 and GPU if available

4. Functional Components

- generate_response(prompt, max_length)

Generates a model response based on the input prompt.

- Uses temperature=0.7 for balanced creativity
- Applies truncation and padding
- Automatically moves tensors to GPU if available

5.Gradio Interface Structure:

- Built using gr.Blocks() for modular layout
- Two tabs:
- **City Analysis**
- **Citizen Services**

Components:

Tab	Input Component	Output Component	Action Button
City Analysis	Textbox (city name)	Textbox (analysis)	Button ("Analyze City")
Citizen Services	Textbox (query)	Textbox (response)	Button ("Get Information")

Launch:

app.launch(share=True)

EXAMPLE:

1. CITY ANALYSIS ---- Chennai
2. CITIZEN SERVICES------- Women Safety

PROJECT OVERVIEW:

This project is a Gradio-based web application powered by the IBM Granite 3.2-2B Instruct model. It provides two key services:

1. **City Analysis** – Offers safety insights including crime index and accident statistics.
2. **Citizen Services** – Responds to public service queries as a virtual government assistant.

TECHNOLOGIES USED:

Component	Description
Gradio	For building the interactive web interface
PyTorch	For model execution and tensor operations
Hugging Face	For loading the tokenizer and causal language model
IBM Granite 3.2	Instruction-tuned language model used for generating responses

FUTURE ENHANCEMENT:

- Add input validation and error handling
- Include loading indicators for better UX
- Support multilingual queries
- Export results to downloadable formats

CONCLUSION :

Our AI-powered CitizenAI platform is designed to enhance interaction, accessibility, and transparency between citizens and government services. By integrating an AI chat assistant, sentiment analysis, concern reporting, and

dashboard insights, the platform empowers users to easily access information, provide feedback, and report issues. With a Flask backend and an interactive HTML/CSS frontend, powered by the IBM Granite AI model, your project ensures a user-friendly experience while providing smart and responsive civic engagement tools. This innovative solution simplifies communication and fosters trust, making civic participation more convenient and efficient for all