**Project Documentation: Citizen AI**

**1. Introduction**

**Project Title**: Citizen AI

**Team Members**:

Songa Vennela: Project Lead, Backend Development

Sai Venkat: Frontend Developer, React

R P S Gayatri: Data Analyst, Dashboard Design

S Aparna Tejasri: NLP Engineer, Model Integration

**2. Project Overview**

**Purpose:**

Citizen AI is an intelligent citizen engagement platform designed to improve government-citizen interactions. The platform aims to provide a seamless, responsive, and data-driven approach to handling citizen inquiries, feedback, and requests, leading to increased transparency, efficiency, and citizen satisfaction.

**Features:**

Real-time Conversational AI Assistant: Users can engage with the platform through natural language queries, receiving prompt, human-like responses from the AI model.

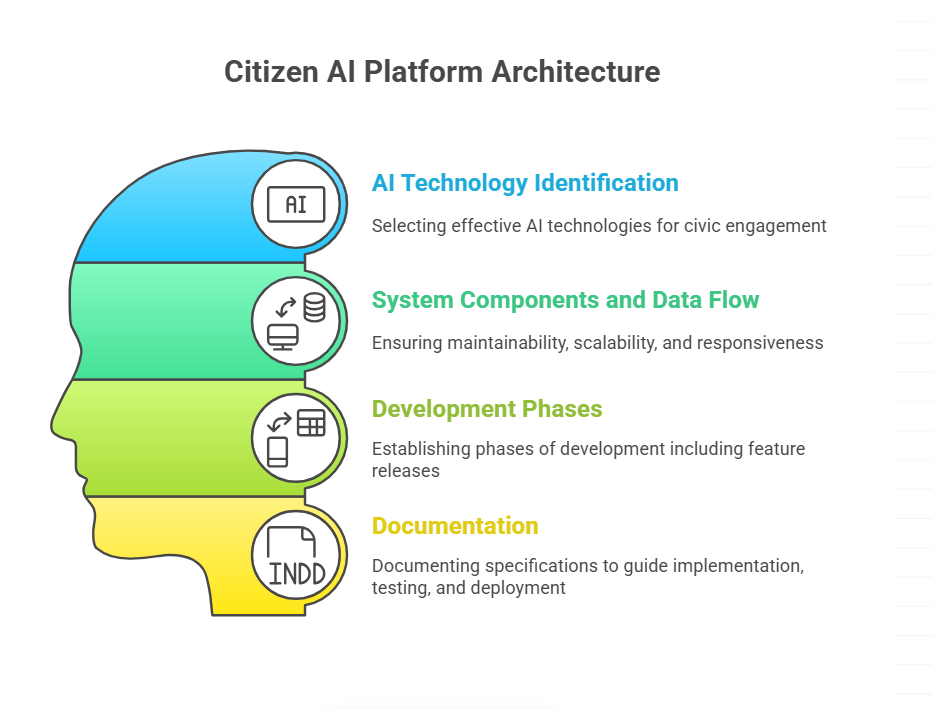
**-Sentiment Analysis**: The platform analyzes citizen feedback to understand overall sentiment (positive, negative, neutral) towards government services, allowing for identification of areas requiring attention.

**-Dynamic Dashboard**: A user-friendly dashboard visually presents key data points, including sentiment trends, interaction frequency, and potential issues. This allows government officials to understand public perception in real-time.

**-Personalized & Contextual Responses**: Leveraging an advanced NLP model (e.g., IBM Granite), Citizen AI provides tailored responses based on the specific context of user queries, exceeding the limitations of generic answers.

**-Data Logging and Reporting**: Citizen interactions and sentiment data are logged and stored securely, enabling detailed analysis and reporting. Downloadable CSV and JSON files provide flexible data access.

**3. Architecture**

****

**Frontend:**

Framework: Gradio

Components: Textbox, Button, Plot, File, Blocks, TabbedInterface

State Management: Managed using local Python variables (sentiment\_counts, interaction\_log)

|  |  |
| --- | --- |
| UI Tabs: | Chat Assistant, Dashboard |

|  |  |
| --- | --- |
| Interactivity | : Users interact via chat and feedback inputs |

**Backend:**

|  |  |
| --- | --- |
| LLM | : IBM Granite 3.3B Instruct model for generating AI responses |

|  |  |
| --- | --- |
| Sentiment Model: | Hugging Face’s sentiment-analysis pipeline |

|  |  |
| --- | --- |
| Token Security | : Token read from a secure .txt file on Google Drive |

|  |  |
| --- | --- |
| Model Serving | : Transformers, Torch, Accelerate |

|  |  |
| --- | --- |
| Data Logging | : Logs stored in CSV and JSON formats using pandas |

|  |  |
| --- | --- |
| Visualization | : Dashboard charts generated using matplotlib |

**4. Setup Instructions**

**Prerequisites:**

Before running the Citizen AI application, ensure the following requirements are met: Environment Requirements:

- Platform: Google Colab (recommended)

- Python version: >= 3.8

- Internet connection (for model loading and API access)

Python Libraries:

Install the following Python libraries (compatible with Google Colab):

!pip install -q gradio transformers torch accelerate matplotlib pandas requests

Authentication Token:

- Create a Hugging Face account from <https://huggingface.co>

- Create a "Read" token from <https://huggingface.co/settings/tokens>

- Save this token in a text file named hf\_token.txt and upload it to your Google Drive

**Project Structure**

CitizenAI/

hf\_token.txt # Hugging Face token (in Drive)

sentiment\_log.csv # Auto-generated sentiment log

sentiment\_log.json # Auto-generated sentiment log (JSON)

app.py#Mainappcode(oruseColab)  
 Setup Instructions

1. Mount Google Drive: from google.colab import drive drive.mount('/content/drive')

2. Load Hugging Face Token: with open('/content/drive/MyDrive/hf\_token.txt', 'r') as f: hf\_token = f.read().strip()

3. Login to Hugging Face: from huggingface\_hub import login login(token=hf\_token)

4. Load the IBM Granite Model: from transformers import AutoTokenizer, AutoModelForCausalLM model\_id = "ibm-granite/granite-3.3-2b-instruct" tokenizer = AutoTokenizer.from\_pretrained(model\_id, token=hf\_token) model = AutoModelForCausalLM.from\_pretrained( model\_id, device\_map="auto", torch\_dtype=torch.float16, token=hf\_token )

**5. API Documentation**

**5.1 Endpoints**

Citizen AI likely exposes several API endpoints. A complete list of endpoints with detailed information for the server project is crucial.

Example:

Endpoint Method Description Request Parameters Response

/api/submitFeedback POST Submits citizen feedback to the system and runs sentiment analysis. text (string) JSON

/api/getDashboardData GET Retrieves data for the dynamic dashboard, including sentiment counts and trends over time. None JSON

/api/getInteractionLog GET Downloads the complete interaction log (perhaps in CSV format) Optional filter params (e.g., date range) CSV/JSON

Important: Replace these with the actual endpoints for your Citizen AI backend. Detail the necessary request parameters, data formats, and expected response structures in more depth.

**5.2 Request Methods**

Include a detailed explanation of the different HTTP methods (GET, POST, PUT, DELETE) used in each API endpoint. Explain what each method does, including examples and usage scenarios. For instance:

POST /api/submitFeedback: Sends a new feedback item to the server for processing.

GET /api/getDashboardData: Fetches aggregated data for creating charts and displays in the dashboard.

These details are essential for developers to understand and use the API effectively. Remember to provide clear and accurate API documentation.

**Parameters**

This section details the key parameters and configuration options for the Citizen AI project, encompassing data input, model selection, and platform settings.

Input Parameters

**Citizen Queries**

-Input Type: Textual input via a text box in the chat interface.

-Data Format: Plain text representing citizen inquiries about government services, policies, or civic issues.

Example: "What are the procedures for applying for a building permit?" or "Where can I find information on recycling programs?"

-Maximum Length: Limited to prevent excessively long and computationally intensive queries. (Implementation detail: This could be a configurable parameter.)

**User Feedback**

-Input Type: Textual input via a text box on the dashboard.

-Data Format: Plain text representing citizen feedback regarding their interaction with the platform. This feedback is used for sentiment analysis.

Example: "The response was very helpful," or "I couldn't find the information I was looking for."

-Maximum Length: Limited to ensure processing efficiency. (Implementation detail: This could be a configurable parameter.)

**Model Parameters**

Large Language Model (LLM)

Model: IBM Granite (version 3.3-2b-instruct).

Model ID: ibm-granite/granite-3.3-2b-instruct

Description: This model is used for generating responses to user queries. It is critical to ensure the model is suitable for the nature of the queries.

Hugging Face Token: A secure, environment-variable based or locally stored token is needed for accessing the model. The token should not be hardcoded in the script itself.

**Sentiment Analysis Model**

Model: Hugging Face pipeline for sentiment-analysis.

Description: This model analyzes the sentiment (positive, negative, neutral) of user feedback.

Model Specificity: The selected model should align with the specific range of feedback text expected.

**Platform Settings**

Logging

-File Formats: Sentiment data is logged to both CSV and JSON files (for different consumption patterns).

-File Location: Log files are located in the same directory as the Python script, named sentiment\_log.csv and sentiment\_log.json respectively.

-Logging Frequency: The log is updated upon each feedback submission.

-Data Retention: The project should establish clear guidelines on log retention periods.

**Dashboard Parameters**

-Visualization Types: Pie charts for overall sentiment distribution, line charts for sentiment trends over time.

-Chart Refresh Rate: The dashboard updates with each sentiment analysis, potentially configurable for performance reasons.

-Data Aggregation: Sentiment counts are aggregated in real-time to produce the dynamic dashboard.

-Download Options: Users can download sentiment data in CSV and JSON formats.

Important Notes

-All parameters should be configurable to allow flexibility and adaptation to various use cases and data sources.

-Consider using environment variables for sensitive information such as Hugging Face tokens to enhance security.

-Ensure data privacy compliance regulations are followed when collecting and handling user feedback.

-Error handling and robust input validation are crucial for managing potential issues and providing a stable user experience.

-Consider adding mechanisms to handle potentially malicious user input, such as filtering or moderation systems.

Configuration Example (Illustrative)

[ModelParameters]

model\_id = ibm-granite/granite-3.3-2b-instruct

huggingface\_token = <YOUR\_HUGGINGFACE\_TOKEN> # Store securely!

[PlatformSettings]

max\_query\_length = 500 #characters

max\_feedback\_length = 250 #characters

log\_retention\_days = 30 # Days

This comprehensive breakdown should aid in setting up and managing the Citizen AI project effectively, while ensuring flexibility and future scalability.

**6.Authentication**

Authentication and Authorization Overview

This section details how Citizen AI handles user authentication and authorization, crucial for secure access to platform features and data.

Authentication

Authentication verifies a user's identity. In Citizen AI, authentication is currently not implemented for general public use of the chat assistant. While the platform does use a Hugging Face token for model access, this is handled at the application deployment level and is not directly exposed for user logins.

Authorization

Authorization determines what actions a verified user is permitted to perform. Citizen AI does not currently implement a specific user authorization system to govern which users can access specific functionality. Instead, access to features is granted based on the overall access to the platform, which is currently not restricted. Future versions may include granular authorization based on user roles or permissions.

**Tokens**

Hugging Face API Token

Hugging Face tokens are used to authenticate with the huggingface\_hub library for accessing and downloading pre-trained models (like ibm-granite/granite-3.3-2b-instruct). This token is not user-specific and is stored in a private file (hf\_token.txt).

with open('/content/drive/MyDrive/hf\_token.txt', 'r') as f:

hf\_token = f.read().strip()

login(token=hf\_token)

This approach allows the platform to securely interact with Hugging Face services without requiring end-users to enter credentials. The example code snippet shows how the token is loaded and used.

-Important Note: Storing API keys directly in code, even within private files on Colab/similar cloud environments, is a security risk. In a production system, more robust secrets management (using environment variables, dedicated secrets management services) is absolutely essential to protect the token.

Future Token Integration

Future development may include user-specific authentication tokens for greater security and control over access. This could potentially entail:

User accounts: Creating a database of registered users.

JWT (JSON Web Tokens): Issuing JWTs to verify user identity and roles.

API keys: Generating unique API keys for each user.

These are potential ways to extend the authentication framework and ensure the security and privacy of platform data and interactions.

Current Authorization Limitations

Currently, there's no mechanism for limiting access to specific platform features or data based on user roles or permissions. Any user with access to the platform code can potentially utilize all its functionalities.

Security Considerations

API Key Management: The current method of storing the Hugging Face token is not secure. For a production environment, employ environment variables or a dedicated secrets management solution.

-User Authentication: Implementing a robust user authentication system is crucial for future versions to secure user data and functionality.

-Authorization: Establishing authorization controls (roles and permissions) to protect specific actions is a critical step towards enhancing platform security and managing access to different functionalities.

This section highlights the current authentication and authorization methods of Citizen AI and underscores the importance of enhancing these mechanisms for future iterations to maintain data security and user trust.

**7.Sessions**

**Introduction**

This section details the various user sessions and functionalities within the Citizen AI platform. It outlines the user experience for interacting with the platform and the underlying logic powering the different features.

**Scenario 1:** Real-Time Conversational AI Assistant

Functionality

Citizen Interaction: Users interact with the platform via a chat interface.

Real-time Query Processing: User input is sent directly to the underlying AI model (e.g., IBM Granite).

Human-like Responses: The AI model generates a natural language response, which is presented immediately to the user.

Accessibility: Facilitates 24/7 access to information and services.

Example Interaction

User: "How do I report a pothole on Elm Street?"

AI Assistant: "To report a pothole on Elm Street, please visit our online reporting portal. You can find the link on the main page of the Citizen AI platform. Alternatively, you can call our dedicated hotline at [phone number] and follow the automated instructions."

**Scenario 2:** Citizen Sentiment Analysis

Functionality

Data Collection: The platform collects user feedback and interaction text from submitted queries.

Sentiment Classification: Utilizes a pre-trained sentiment analysis pipeline (e.g., from Hugging Face) to classify the sentiment of user input.

Sentiment Scoring: Provides a numerical score (0-100%) representing the intensity of the sentiment (positive, negative, neutral).

Logging: Tracks and logs sentiment analysis results.

Reporting: The platform generates reports and dashboards based on sentiment data, including visualizations.

Example Workflow

Citizen submits feedback about a slow internet connection.

The platform analyzes the feedback using the sentiment pipeline.

The system classifies the feedback as negative with a score of 70%.

The analysis results are logged and stored, contributing to the overall sentiment trend.

**Scenario 3:** Dynamic Dashboard

Functionality

Real-time Visualizations: Displays key metrics, including overall sentiment (positive, neutral, negative), interaction trends, and aggregated feedback scores.

Trend Analysis: Tracks sentiment changes over time, highlighting peak interaction periods.

Data Aggregation: Groups and summarizes data for a clear overview of citizen feedback.

Actionable Insights: Presents data in a format enabling government officials to quickly identify areas needing attention and improve services.

Example Dashboard Data

Sentiment Count

Positive 120

Negative 35

Neutral 45

Note: Visualizations will use charts and graphs (e.g., pie charts, line charts) to effectively present the data.

**Scenario 4:** Personalized & Contextual Response System

Functionality

Advanced Natural Language Understanding (NLU): Powered by IBM Granite models, the platform understands the context of citizen queries.

Accurate Interpretation: Accurately interprets nuanced queries to provide relevant answers.

Tailored Responses: Generates answers tailored to specific needs and context.

Reduced Need for Generic Responses: Reduces reliance on generalized responses, improving citizen experience.

Example Interaction (Leveraging Context)

User: "My water bill is too high. I'm concerned."

AI Assistant: "We understand your concern about a high water bill. Could you please provide your account number and the period in question to investigate the potential cause? We will follow up with you as soon as we have identified the issue."

Important Note: This specific response demonstrates how the AI can gather more context from the user before providing further, more detailed, solutions.

Technical Considerations

Code Example for Sentiment Analysis (simplified):

from transformers import pipeline

sentiment\_pipeline = pipeline("sentiment-analysis")

def analyze\_sentiment(text):

result = sentiment\_pipeline(text)[0]

return result["label"], result["score"]

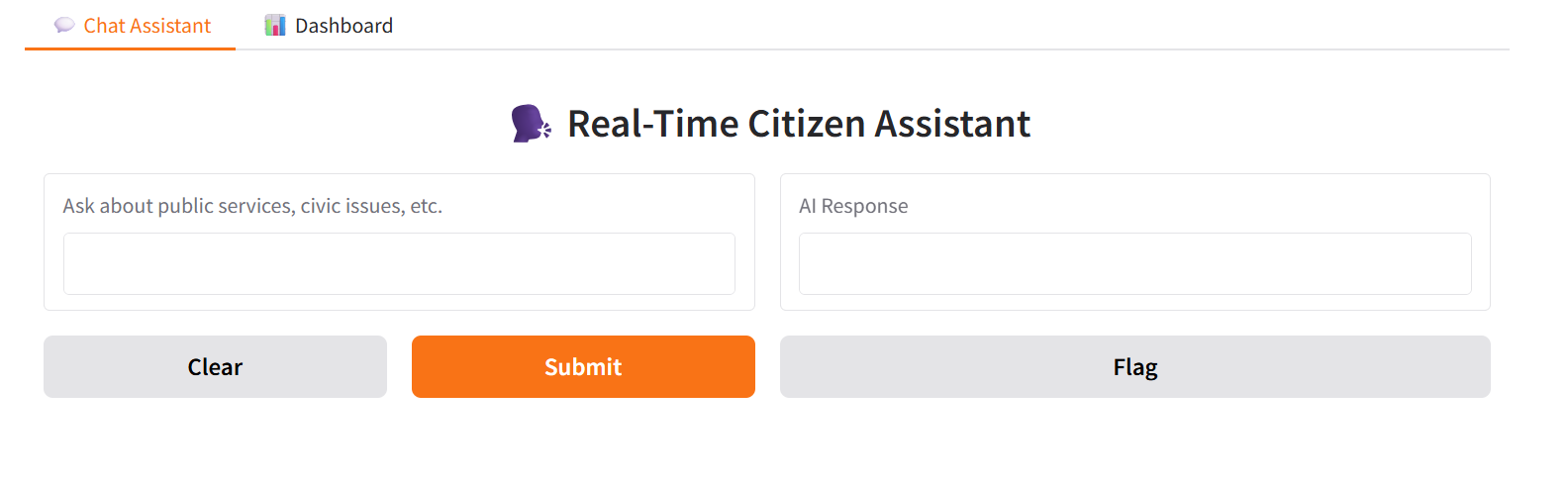
Model Updates: The platform utilizes pre-trained models which can be updated as necessary.

Data Storage: Data (e.g., interactions, sentiment) are stored in a secure database.

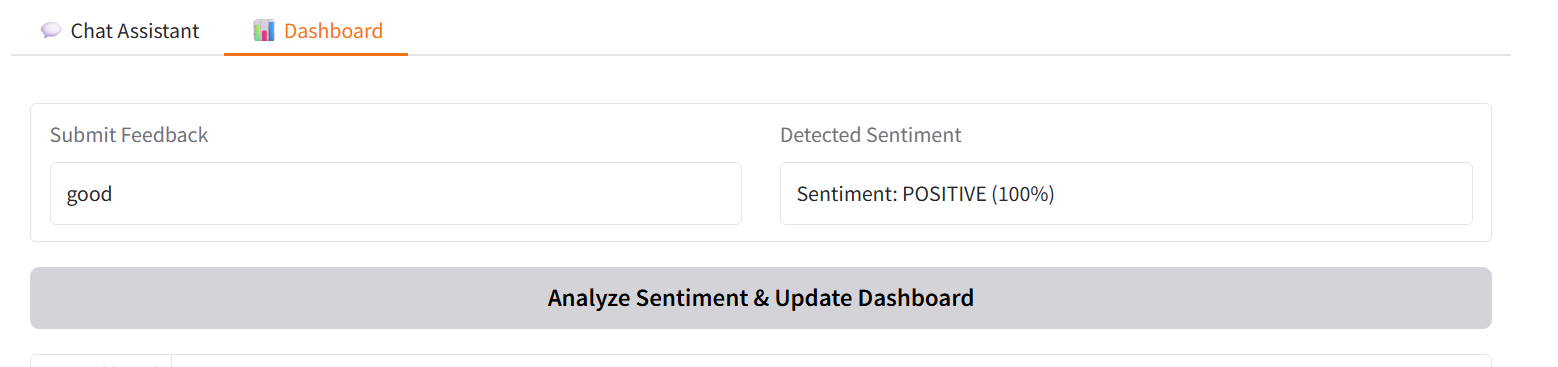
This detailed documentation provides a comprehensive overview of the different user sessions within the Citizen AI platform. These sessions aim to provide a seamless user experience while offering valuable insights for government officials .

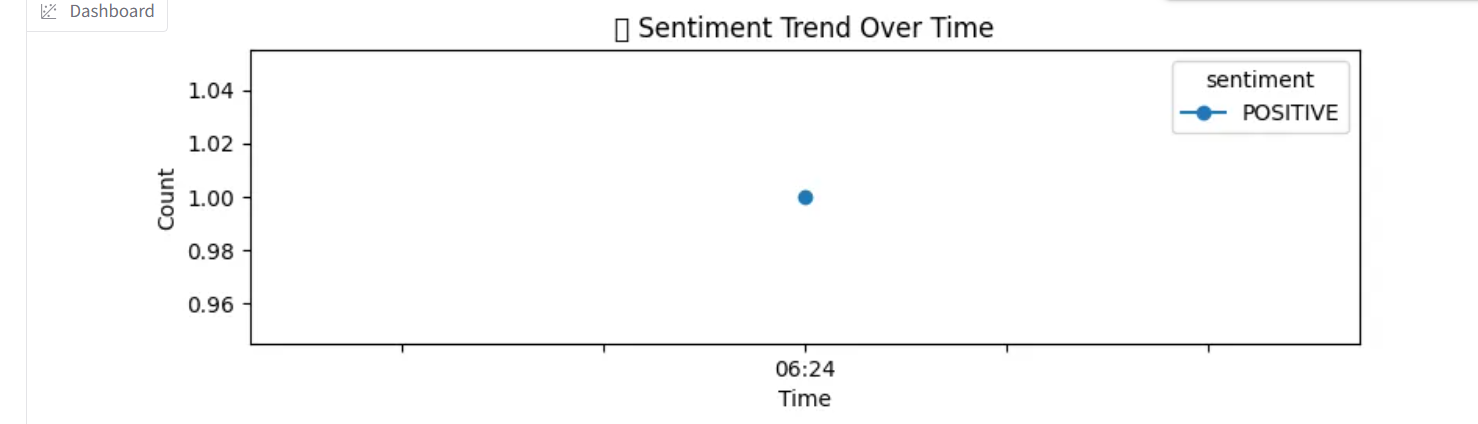
**8.User Interface**

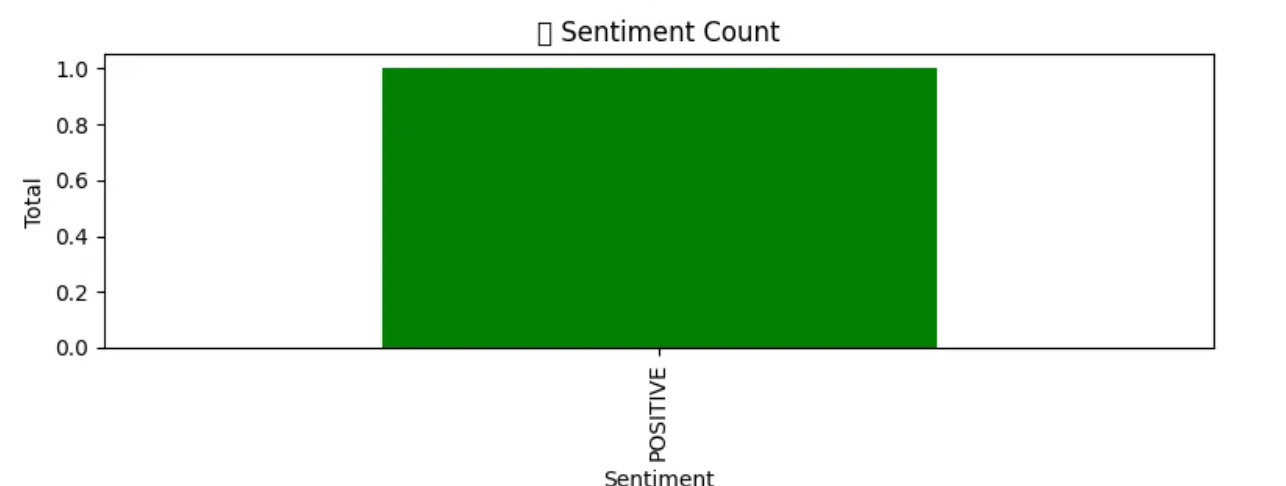
**Chat Tab:** A textbox for user input, a textbox to display AI responses. The layout will clearly separate user input and AI output. Example UI behavior: Typing a query in the input textbox will trigger an immediate response from the AI model.



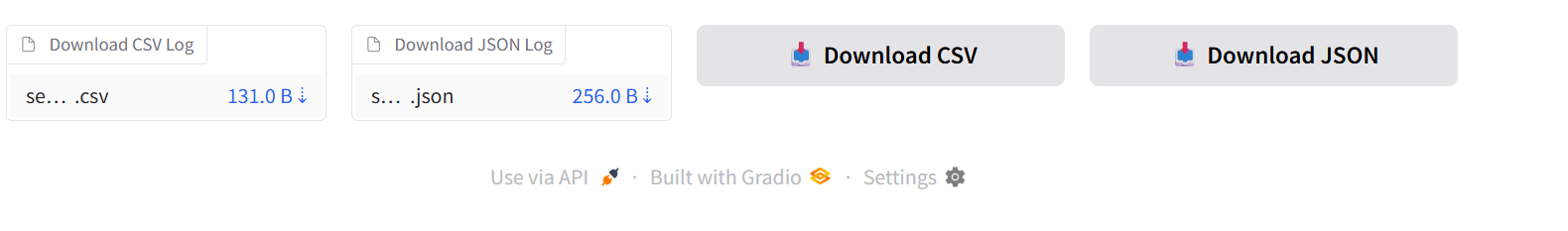
**Dashboard Tab**: A plotting area to display pie charts and line graphs visualizing sentiment analysis data. There will be text boxes to allow users to input feedback for sentiment analysis and buttons to trigger the analysis and update the charts. Example UI behavior: Clicking "Analyze Sentiment" should update the dashboard display.







**Download Buttons**: Buttons to allow the user to download CSV and JSON files for the sentiment analysis log.



**Error Handling**: Clear error messages in case of invalid input or unexpected issues, like connection problems to the model.

**9.Testing**

**Unit Tests:** The individual components (e.g., ask\_granite, submit\_feedback, and plotting functions) of the Python code will be tested independently to ensure correctness.

**Integration Tests:** The interactions between components (e.g., how the sentiment analysis feeds into the dashboard) will be tested to verify that they work as expected.

**Functional Tests:** A collection of user-centric test cases simulating the workflow and interactions in the Gradio app:

Valid input/output scenarios in the chat interface

Scenarios to test feedback submission and updating the dashboard

Validation of the correct displayed charts

Verification of file downloads (CSV and JSON)

End-to-End Tests: The entire system's behavior will be tested by simulating typical citizen interaction with the platform from start to finish.

**Tools:**

Python unittest: Used for unit and integration tests.

pytest (optional): A more advanced testing framework.

Mock objects: For mocking dependencies during testing to isolate components.

Gradio's built-in testing capabilities: (If available) - checking input validation, output formatting, and basic UI flow.

Demo: [https://ed6b5917db435b2842.gradio.live](https://ed6b5917db435b2842.gradio.live/)

**10.Known Issues**

**Potential Delay:** Large or complex queries might result in a delay in receiving the response from the AI model. This is inherent to large language models.

**Model limitations**: The current model's accuracy in nuanced or ambiguous situations needs continuous monitoring. Feedback on interactions can help improve the model's response over time.

**Internet Connectivity:** The application relies on internet connectivity to access the Hugging Face model and to function optimally. Intermittent connectivity problems can lead to issues such as delayed responses or failure to retrieve models.

**11.Future Enhancements**

**More sophisticated sentiment analysis:** Integration with a more advanced sentiment analysis model that can handle sarcasm, humor, and other complexities of human language.

**Personalization:** Implement user accounts and preferences to tailor the chatbot responses and dashboard visualizations for individual users.

**Improved Data Visualization**: Enhance the dashboard with interactive elements, such as drill-down capabilities on sentiment trends and charts. Consider adding geographical visualizations of sentiment data where applicable.

**External Data Integration:** Integrate with external data sources to provide richer context to the responses and more comprehensive dashboard information. For instance, government agency data related to user queries.

**Real-time query filtering:** Consider implementing methods to filter out redundant or irrelevant queries and improve efficiency in handling large volumes of user input.

**Multilingual support:** Extend the application to support multiple languages to enhance accessibility and inclusivity.

**User feedback mechanisms:** Add ways for users to provide feedback on the chatbot's responses (e.g., rating the quality or correctness of the responses).