Citizen AI – Intelligent Citizen Engagement Platform

Project Documentation format

1. Introduction:

Project Title: Citizen AI – Intelligent Citizen Engagement Platform

Team Members:

B.Harshavardhan	: Leader / Organize
C.Thanuja	: Research Helper
G.Suhail	: Design Helper
G.Saisruthi	: Basic Coder

2. Project Overview:

Citizen -AI is an AI-powered, full-stack cloud application designed to transform the traditional Intelligent Citizen Engagement Platform by integrating intelligent automation into each core phase. Built on IBM Cloud and powered by the **IBM Granite foundation model**, Citizen AI leverages advanced Natural Language Processing (NLP) and Generative AI to convert unstructured inputs—such as raw text requirements or code—into actionable development artifacts like structured user stories, production-ready code, test cases, bug fixes, and documentation. The IBM Granite model serves as the AI engine across the platform, enabling deep understanding and contextual generation of software assets with high accuracy. By automating requirement classification, code generation, Citizen AI debugging, testing, summarization, and real-time support through an integrated chatbot, significantly enhances productivity, reduces manual effort, and ensures consistency across the Intelligent Citizen Engagement Platform, making it a powerful ecosystem for modern software teams.

Purposes:

- Automate critical phases of Intelligent Citizen Engagement Platform using AI.
- Enhance developer productivity by generating code, fixing bugs, and creating test cases automatically.
- Convert unstructured requirements into structured user stories for better clarity and planning.

- Improve software quality and reduce human error with AI-assisted testing and debugging.
- Generate clear, human-readable code summaries to support documentation and onboarding.
- Empower non-technical users to participate in the development process through natural language inputs.
- Provide a unified, intelligent development environment using the IBM Granite foundation model.
- Reduce development time and costs by minimizing manual work and repetitive tasks.
- Leverage IBM Cloud for scalability, reliability, and enterprise-ready deployment.
- Promote innovation by integrating AI and NLP into modern software engineering workflows.

→ **Features:**

- Upload PDF files containing raw, unstructured requirements
- AI-based classification of content into Intelligent Citizen Engagement Platform phases (Requirement, Design, Development, Testing, Deployment)
- Automatic generation of structured user stories from extracted requirements
- Natural language prompt-to-code generation using IBM Granite model
- AI-powered bug detection and correction for Python and JavaScript code
- Automated test case generation using frameworks like unittest and pytest
- Code summarization with human-readable explanations for documentation and onboarding
- Floating AI chatbot assistant for real-time Intelligent Citizen Engagement Platform guidance and help
- Syntax-highlighted, clean code display on the frontend
- Fully cloud-based deployment on IBM Cloud for scalability and availability
- Modular design — each feature can be used independently or as part of an integrated workflow
- Intelligent prompt routing and keyword detection using LangChain for chatbot responses.

3. Architecture:

Frontend : The Gradio frontend for this code is designed to provide an intuitive and user-friendly interface for interacting with the Citizen AI platform. Here's a breakdown of the key components:

1. Interface Structure

The interface is built using Gradio's Blocks API, which allows for a more customized layout. It features:

A title and subtitle

Three tabs:

1. Ask Citizen AI" for general queries
2. Emergency Contacts" for finding emergency contact numbers
3. City Analysis" for getting an analysis of a city

2. Tab Components:

Each tab has:

Input fields (e.g., Textbox) for users to enter their queries or city names

Buttons (e.g., Button) to trigger the AI model's response

Output fields (e.g., Textbox) to display the AI model's response

3. Event Handlers:

The interface uses event handlers to bind the buttons to specific functions:

handle_query: handles general queries

handle_contacts: handles emergency contact requests

handle_city_analysis: handles city analysis requests

hese handlers call the corresponding methods of the CitizenAI class, which interact with the AI model to generate responses.

4. Customization:

The interface is customized with HTML components (e.g., gr.HTML) to add a title, subtitle, and a footer with a powered-by message.

5. Launch Configuration:

The interface is launched with the share=True parameter, which generates a public link that can be shared with others.

Backend: This code is designed to run on Google Colab, utilizing its backend infrastructure. Here's how it leverages Colab's capabilities:

1. Installation of Libraries: The code starts by installing necessary libraries like transformers, torch, gradio, accelerate, and bitsandbytes using !pip install. This is a common practice in Colab notebooks.

2. Model Loading and Inference: The CitizenAI class loads pre-trained models (e.g., "ibm-granite/granite-3.3-2b-instruct" or "microsoft/DialoGPT-medium") using the transformers library. Colab's backend handles the model loading and inference, enabling the generation of responses to user queries.

3. Gradio Interface: The code creates a Gradio interface, which is a Python library that allows you to create simple, shareable, and powerful interfaces for your machine learning models. When run in Colab, Gradio generates a public link to access the interface, thanks to Colab's backend support.

4. Public Link Generation: By setting share=True in the iface.launch() function, Gradio generates a public link that can be shared with others. This link remains active for 72 hours, allowing users to interact with the Citizen AI platform from anywhere.

By leveraging Colab's backend, the code can efficiently run complex machine learning models and provide a user-friendly interface for interacting with the Citizen AI platform.

4. Setup Instructions:

♦ Prerequisites:

Before starting the citizen -Ai project in Google Colab, ensure the following are available:

- **Google Account** to access and run Google Colab notebooks

- **IBM Cloud Account** with access to **Watsonx.ai** and the **Granite 3.2 Instruct model**
- IBM Cloud API Key with access rights to Watsonx foundation model
- Colab-compatible Python environment (Colab default: Python 3.10+)
- Required Python packages (listed below)

♦ Installation & Configuration Steps in Google Colab:

You can run the full project pipeline inside a **Google Colab notebook**. Here's a step-by-step guide:

1. Set Up Required Python Packages

Install required packages in your Colab environment:

```
!pip install transformers torch gradio accelerate bitsandbytes
```

5. Folder Structure:

Since **CITIZEN -AI** was developed entirely within a **Google Colab notebook**, the project does not follow a conventional file/folder structure (e.g., separate /client and /server directories). Instead, the entire system is organized into **modular notebook cells**, Intelligent Citizen Engagement Platform each representing logical components of the automation pipeline.

➢ Notebook-Based Architecture Overview:

Instead of folders, the Colab notebook is divided into the following logical sections (cell groups):

1. PDF Upload & Requirement Extraction

- Uses PyMuPDF to read PDF content
- Extracts and preprocesses raw requirement text

2. IBM Granite Model Integration

- Authenticates with **Watsonx.ai** using the ibm-watson-machine-learning SDK
- Calls **Granite v3.2 Instruct** for various tasks:
- Classifying text into Citizen AI phases

- Generating code
- Bug fixing
- Test case generation
- Summarizing code

3. Classification & User Story Generation

- Splits extracted text into sentences
- Prompts Granite to classify each sentence
- Optionally formats output into structured user stories

4. Code Processing Modules

- Separate cells for:
- Code generation from natural language
- Bug fixing in Python/JavaScript code
- Test case creation
- Code summarization

5. Chatbot Assistant (Optional)

- Implements a simple chatbot interface (text input/output)
- Routes user questions about Intelligent Citizen Engagement Platform to Granite with tailored prompts

6. Output Display and Export

- Displays results (code, test cases, summaries) directly in Colab
- Optionally saves outputs to Google Drive or downloads as files

6. Running the Application:

Frontend : Frontend Server (Gradio):

Since the frontend is built using Gradio, it's not a traditional frontend framework like React or Angular that would use npm start. Instead, the frontend server is launched using the launch() method in the Gradio code:

iface.launch(share=True)

To run the frontend server, you would execute the Python script containing this code

Backend :

The backend server is also built using Python, leveraging libraries like transformers and torch. The backend logic is contained within the CitizenAI class and its methods.

To run the backend server, you would execute the same Python script that launches the Gradio frontend. The backend logic is tightly coupled with the frontend in this implementation.

If you were to separate the frontend and backend into distinct projects (e.g., a React frontend and a Python backend), the commands might look like this:

Backend Server:

Assuming a Python backend with a Flask or FastAPI server:

bash

cd backend

python app.py

7. API Documentation:

Citizen - Ai runs entirely within a **Google Colab notebook** and directly interacts with **IBM Watsonx's Granite v3.2 Instruct model** using the **IBM Watson Machine Learning Python SDK**. It does **not expose any public REST API endpoints**, but it internally follows a modular, function-based structure for various tasks.

8. Authentication:

- **Hugging Face API Key Authentication**

Citizen - Ai uses **Hugging Face-hosted IBM Granite v3.2 Instruct model**, accessed through the **Hugging Face Inference API**. Authentication is handled via a **personal API key** provided by Hugging Face.

This key allows authorized access to large language models hosted on the Hugging Face platform without managing a complex user login system.

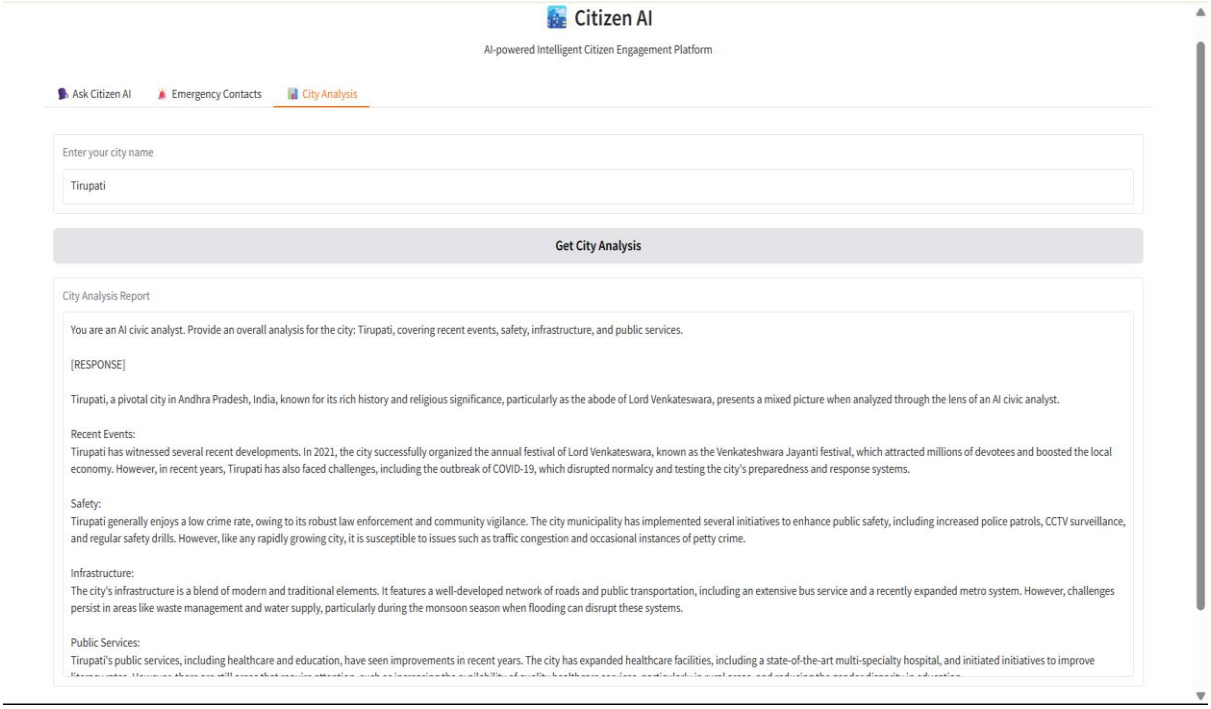
- **How It Works in the Project:**

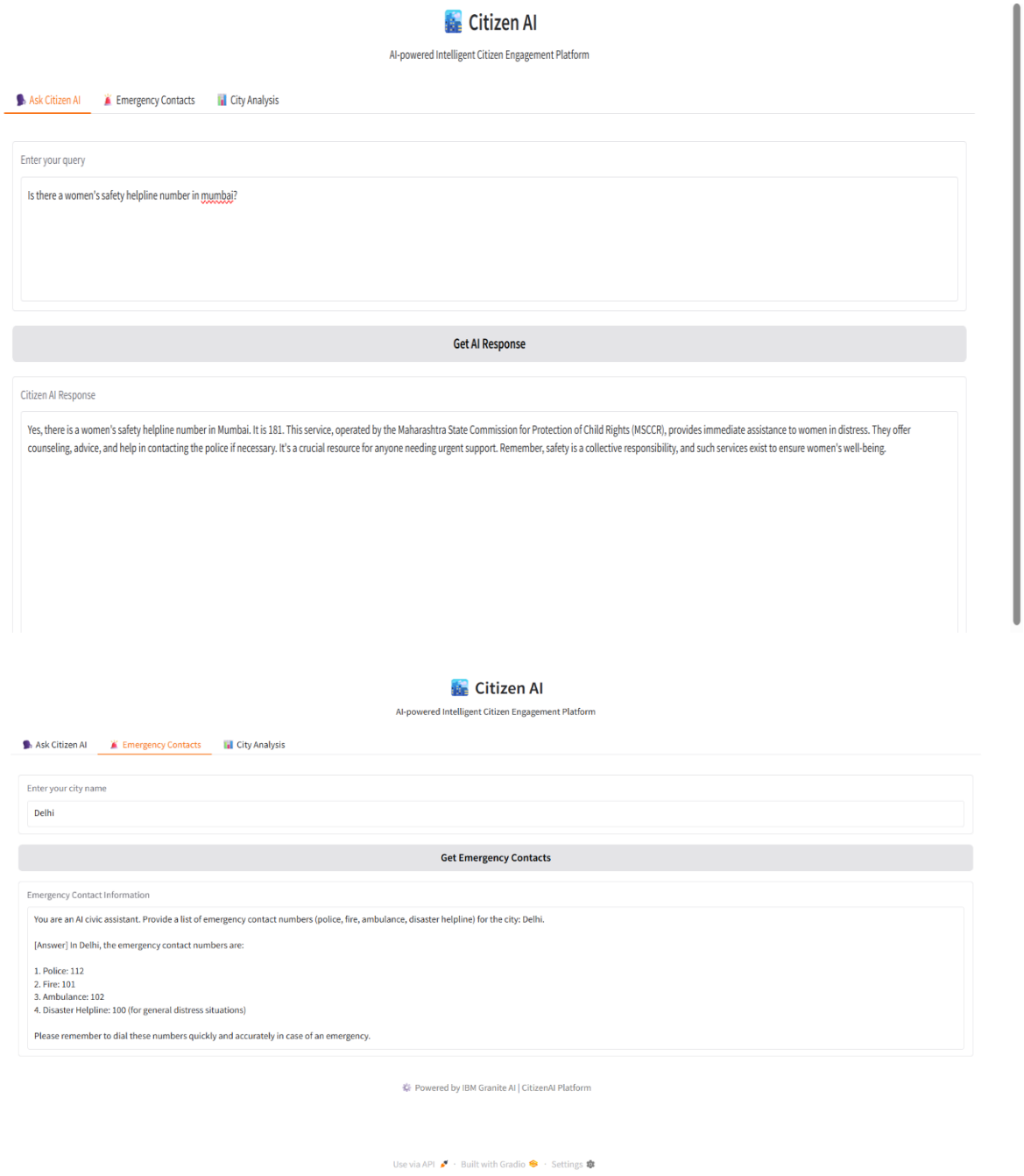
You authenticate by passing the API key as a Bearer token in the HTTP request headers:

```
python
CopyEdit
import requests

API_URL = "https://api-inference.huggingface.co/models/ibm/granite-3b-instruct"
headers = {
   "Authorization": f"Bearer YOUR_HUGGINGFACE_API_KEY"
}

response = requests.post(API_URL, headers=headers, json={"inputs": prompt})
```

9. User Interface:

Citizen - Ai is designed to run in an interactive **Google Colab notebook**, where each module functions as a logically separated UI block. Users interact with the system by uploading files, entering text, and viewing AI-generated outputs directly within the notebook interface.





10. Testing:

- **Testing Strategy**

Citizen -AI is tested through **manual testing and functional validation** using the Gradio interface. Each module is designed as a user-facing interactive block, allowing rapid iteration and debugging.

What Was Tested:

- **PDF Parsing**: Verified extraction of text from diverse PDF formats using PyPDF2.
- **Model Output Validation**:
- Requirement analysis prompts yield structured, relevant functional outputs.
- Code generation prompts produce working Python code for typical tasks.
- **Fallback Model Handling**: Ensured system remains operational with a secondary model (DialoGPT-medium) when the IBM Granite model fails to load.

➢ **Tools Used:**

- Google Colab runtime
- Gradio's live UI for visual verification
- Print/log statements for model loading/debug fallback

11. Screenshots or Demo:
Demo Link:
https://drive.google.com/file/d/17RLgCBrKWP540df_puxY4AJRThoYLuow/view?usp=drivesdk

12. Known Issues:

Here are some known bugs or limitations that may affect users or developers:

- **Model Load Time**:
  The IBM Granite 3.3-2B Instruct model is large and can take considerable time to load in Colab, especially on free-tier runtimes.
- **Memory Constraints in Google Colab**:
  Large models may cause memory overflows or slow performance on limited resources.
- **Fallback Model Simplicity**:
  The fallback model (DialoGPT-medium) is significantly less capable and may produce generic or unrelated outputs.
- **PDF Parsing Limitations**:
  Some PDFs, especially scanned images or non-standard encodings, may not be parsed accurately by PyPDF2.

- **Response Variability**:
  Generative AI outputs may vary between runs and may include irrelevant or partially complete code.
- **No Persistent Storage**:
  There's no backend database or file-saving mechanism, so all results are lost when the session ends unless manually saved.
- **No Authentication**:
  The app does not currently restrict access or protect the Hugging Face API key, which can be a security risk in shared environments.

13. Future Enhancements:

To improve usability, scalability, and feature richness, the following enhancements are recommended:

Functional Enhancements:

- **Add More Modules**:
  Incorporate additional tabs for:
- Bug Fixing
- Test Case Generation
- Code Summarization
- Deployment Suggestions
- **Custom Model Selector**:
  Let users choose between models (e.g., IBM Granite, GPT-4, Mistral) for different use cases.
- **Improve Prompt Engineering**:
  Use structured prompt templates and dropdowns to tailor AI responses more precisely.

Application & UI Improvements:

- **Persistent Storage**:
  Add integration with cloud storage or databases (e.g., Firebase or MongoDB) to save user inputs and outputs.

- **Export Functionality**:
  Allow users to download generated code and requirement summaries in .txt, .py, or .pdf format.
- **Code Execution Cell**:
  Let users run generated Python code inside the app using secure sandboxing (e.g., Code Interpreter or subprocess).
- **Enhanced Error Handling**:
  Provide clearer error messages, especially for model loading, PDF parsing, or API rate limits.

Security Enhancements:

- **API Key Protection**:
  Store the Hugging Face API key in environment variables or use OAuth-secured proxy to avoid exposure in notebooks.
- **User Authentication (Optional)**:
  Add login functionality if deployed publicly, especially when saving user data.

Deployment Options:

- **Dockerize the App**:
  Containerize with Docker for easier deployment on IBM Cloud or other cloud platforms.
- **Host on a Web Server**:
  Convert the app from Colab-based to a fully hosted app using Flask + Gradio, deployed on IBM Cloud or Hugging Face Spaces.