Generative AI with IBM Cloud

1.Introduction

Project Title: Citizen AI- Intelligent Citizen Engagement Platform

Team Members:

Team Member	**Role**
M Jaya Bhargavi (Lead)	Project Lead & Frontend Developer
Kuthani Ramyatha Satya Sri	Backend & AI Integration Developer
Kowluri Vijayajyothi	Research & Content Manager
Lakshmi Sathvika Devanaboina	Testing & Deployment Engineer

2.Project Overview

Purpose:

The primary purpose of the CitizenAI – Intelligent Citizen Engagement Platform is to simplify and modernize the way citizens interact with government services and civic authorities. The platform is designed to allow users to ask questions, report local issues, and receive accurate information about government schemes and services — all through a conversational, AI-powered chatbot interface.

By using natural language processing and a user-friendly UI, the system eliminates the complexity of navigating government portals, reduces dependency on middlemen, and ensures that citizens of all backgrounds — especially those in rural or underserved areas — can access help and information easily. The platform empowers citizens, enhances transparency, and contributes to building a more responsive and inclusive governance system.

Features:

- Conversational Chatbot Interface: Allows users to interact in natural language using a simple, chat-based UI powered by Gradio.
- AI-Powered Query Handling: Utilizes IBM Granite or Hugging Face models to understand and respond to user queries intelligently.
- Issue Reporting System :Enables users to report civic issues like potholes, water leakage, streetlight outages, etc., categorized for better understanding.
- Government Scheme Assistance: Provides information about various government schemes and services in a simplified manner.
- Web-Based Access: Runs on Google Colab or any browser, making it accessible without installation.
- Fast and Lightweight: Designed to deliver quick responses with minimal loading or waiting time.
- Modular Architecture: Built to support future integration of new APIs, languages, and advanced backend systems like FastAPI.

3. Architecture

Frontend:

- Developed using Gradio:
  An open-source Python library used to build the chatbot-style user interface quickly and efficiently.
- Chat Interface:
  Mimics popular messaging platforms; allows users to type queries and view AI responses in real-time.
- Predefined Buttons:
  Includes clickable options for frequently reported civic issues (e.g., Water Problem, Potholes, Garbage, etc.).
- User Input Field:
  Text box for custom questions; connected to backend logic via Gradio event triggers.

Backend:

- Built Using Python: The backend is implemented in Python for seamless integration with AI models and the Gradio frontend.
- AI Model Integration:
  Connected with IBM Granite 3.3 or Hugging Face Transformers for natural language understanding and response generation.
- Modular Processing Functions:
  Each user query passes through a modular pipeline (e.g., classification, intent detection, response generation).
- Issue Categorization Logic:
  Custom logic classifies queries into predefined civic categories (e.g., sanitation, electricity, roads).

Database:

The CitizenAI – Intelligent Citizen Engagement Platform uses IBM’s state-of-the-art Granite 3.3 2B Instruct model (ibm-granite/granite-3.3-2b-instruct) as its core reasoning engine to understand and generate human-like responses. While the model itself doesn't require a database, other parts of the application may rely on data storage for interaction history, issue tracking, and performance analytics.

4. Setup Instructions

Prerequsites:

Component	Software / Library	Purpose
Python (>= 3.8)	Python Interpreter	Core language for backend, model integration, and Gradio UI
Gradio	gradio Python library	Used to build the interactive chatbot interface
Transformers	transformers (Hugging Face library)	Interface for loading and using IBM Granite model
IBM Granite API	ibm-granite/granite-3.3-2b-instruct	Core language model used for generating AI responses
Google Colab / Jupyter	Google Colab Notebook	Development environment for prototyping, testing, and running the project

Installation:

Step 1:

Clone the Project Repository

If the project is hosted on GitHub, open your terminal and run:

```
git clone https://github.com/your-username/citizenai-platform.git
cd citizenai-platform
```

(*Replace the link above with your actual GitHub repository URL*)

Step 2:

Install Required Dependencies

Install the required Python packages using the following command:

```
pip install gradio transformers requests
```

Optional (for future versions using FastAPI):

```
pip install fastapi uvicorn
```

Step 3:

Open the Project in Google Colab

1. Go to https://colab.research.google.com/
2. Upload the .ipynb file from the project directory.
3. Navigate to Runtime > Change Runtime Type and select Python 3 and GPU (T4) if available.
4. Run all the cells sequentially.

Step 4:

Set Up Environment Variables

To securely connect to the IBM Granite model (if authentication is required), define environment variables as follows:

```
import os
os.environ["IBM_API_KEY"] = "your_ibm_api_key"
os.environ["MODEL_ID"] = "ibm-granite/granite-3.3-2b-instruct"
```

In local development, use a .env file or export the variables in terminal:

```
export IBM_API_KEY="your_ibm_api_key"
export MODEL_ID="ibm-granite/granite-3.3-2b-instruct"
```

Step 5:

Run the Application

Once all dependencies are installed and the model is connected, run the Gradio chatbot interface using:

```
gr.ChatInterface(citizen_assistant).launch()
```

If using FastAPI for backend deployment (planned in future versions), run:

```
uvicorn app:app --reload
```

5. Folder Structure

Client:

Key Components:

- Chat Interface: A real-time conversational UI allowing users to type or select predefined queries.
- Predefined Buttons: For quick issue reporting (e.g., Water Issue, Road Damage).
- User Input Field: Textbox to type queries.
- AI Response Display: Dynamically updates with model-generated responses.
- Optional Feedback Prompt: Planned feature for collecting user satisfaction.

Advantages:

- No frontend framework or HTML/CSS coding needed.
- Instant deployment through Google Colab or local Python environment.
- Ideal for proof-of-concept and MVP demonstrations.

Server:

The backend of the CitizenAI – Intelligent Citizen Engagement Platform is responsible for processing user inputs, interfacing with the IBM Granite model, and returning intelligent, context-aware responses to the frontend. The architecture is designed to be lightweight, modular, and cloud-ready, supporting easy integration with future APIs and databases.

1. Technology Stack

Layer	Technology / Tool
Programming Language	Python
AI Model	ibm-granite/granite-3.3-2b-instruct
Framework (Future)	FastAPI (for RESTful endpoints)
Hosting (MVP)	Google Colab
Hosting (Scalable)	IBM Cloud / Docker / Kubernetes

2. Backend Components

Component	Description
AI Integration Layer	Handles interaction with the IBM Granite 3.3 2B Instruct model for response generation
Processing Logic	Processes user inputs, categorizes civic issues, manages fallbacks for static replies
Routing Layer	(In FastAPI version) Defines REST endpoints to receive requests and return JSON responses
Static Knowledge Base	Optional JSON or dictionary-based fallback for FAQs and offline responses
Feedback Logger	(Planned) Logs user feedback and response accuracy (can integrate with Cloudant or JSON)

3. Modular Function Design

The backend is organized into modular Python functions such as:

- handle_user_query(query): Manages input preprocessing and forwards to AI
- call_granite_model(input_text): Sends input to IBM Granite and receives model response

- classify_issue_type(query): (Optional) Categorizes the issue for better routing
- fetch_static_reply(intent): Returns predefined answer if available
- log_interaction(query, response): Stores query-response pair (for training/logging)

4. API Interaction (Planned with FastAPI)

For production-ready deployments, the backend will expose REST APIs using FastAPI, organized as follows:

Endpoint	Method	Functionality
/chat	POST	Accepts user message and returns AI response
/report	POST	(Future) Log civic issue
/feedback	POST	Collects user feedback

5. Deployment Flexibility

- Prototype / MVP: Runs in Google Colab using inline Python functions
- Scalable Deployment: Supports containerization with Docker and deployment to IBM Cloud or Kubernetes

6. Running the Application

Frontend:

Steps to Run the Gradio Frontend:

1. Open Google Colab or any local Python environment.
2. Ensure required libraries are installed: python

   CopyEdit

   ```
   !pip install gradio transformers
   ```

3. Run the main Python function:

   python

   CopyEdit

   ```
   import gradio as gr


   def citizen_assistant(query):
     # Call to IBM Granite model and return response
           return "AI-generated response here"
   gr.ChatInterface(citizen_assistant).launch()
   ```

4. Gradio will automatically launch a web-based chat interface in a new browser tab where the user can interact with the system.

Backend:

Steps to Run the Backend (Colab-based)

1. Open your .ipynb file in Google Colab.
2. Set runtime to GPU *(optional for large models)*:
   - Click on Runtime > Change runtime type > GPU
3. Install dependencies:

   python

CopyEdit

!pip install transformers gradio requests

4. Set environment variables (if needed):

python

CopyEdit

import os

os.environ["MODEL_ID"] = "ibm-granite/granite-3.3-2b-instruct"

os.environ["IBM_API_KEY"] = "your_ibm_api_key"

5. Define the backend logic:

python

CopyEdit

def citizen_assistant(query):

# Your API/model interaction code

return response

6. Integrate with Gradio frontend or test outputs directly in the note

7. API Documentation

Backend API Endpoints Documentation

1. POST /chat

Purpose:
Processes a user's natural language query and returns an AI-generated response using the IBM Granite model.

Method: POST

Request Body:

json

CopyEdit

```
{
  "text": "What are the latest government schemes for farmers?"
}
```

Response:

json

CopyEdit

```
{
  "response": "The latest government schemes for farmers include PM-KISAN, PMFBY, and KCC loans. Would you like more details on any of these?"
}
```

---

2. POST /report *(Planned)*

Purpose:
Allows users to report civic issues such as potholes, water leakage, streetlight problems, etc.

Method: POST

Request Body:

json

CopyEdit

```
{
  "issue_type": "Pothole",
  "location": "Main Road, Sector 12",
  "description": "A large pothole near the bus stop is causing traffic jams."
}
```

Response:

json

CopyEdit

```
{
  "status": "success",
  "message": "Your issue has been recorded and forwarded to the concerned department."
}
```

3. POST /feedback *(Planned)*

Purpose:
Captures user feedback about the chatbot's accuracy, helpfulness, or response quality.

Method: POST

Request Body:

json

CopyEdit

```
{
  "rating": 4,
  "comment": "Very helpful response, but could be faster."
}
```

Response:

json

CopyEdit

```
{
  "status": "received",
  "message": "Thank you for your feedback!"
}
```

4. GET /health *(Optional)*

Purpose:
Health check endpoint to confirm that the server is up and running.

Method: GET

Response:

json

CopyEdit

```
{
  "status": "ok",
  "uptime": "42 minutes"
}
```

8.Authentication

Security is a key component of the CitizenAI – Intelligent Citizen Engagement Platform to ensure that interactions with the platform are secure and data is accessed only by authorized users.

1. Current Implementation (MVP Version)

As of the Minimum Viable Product (MVP) version, authentication is not enforced, since the Gradio + Google Colab setup is primarily used for demo purposes and public interaction without login.

- All users can interact with the chatbot anonymously.
- No session tokens or user identity tracking is implemented.
- Focus is on open-access testing and rapid prototyping.

2. Planned Authentication in Full Deployment

In the production-ready version (React + FastAPI), user authentication and role-based access control will be integrated for:

- Citizens (to log complaints and track issues)
- Municipal Officers (to view and manage issues)
- Administrators (to manage users and analytics)

Authentication Mechanism

Feature	Description
Login API	Users can log in via email/password or social accounts (Gmail/Facebook)

Feature	Description
Token-Based Auth	Authentication will be handled using JWT (JSON Web Tokens)
Token Storage	Tokens will be stored in browser local storage (frontend)
Expiration	Tokens will have a set expiration time to auto-logout inactive users

Authorization

- Based on the decoded JWT token, the backend will identify the user role.
- Role-based permissions will control:
  - Access to admin dashboards
  - Feedback visibility
  - Issue assignment or management

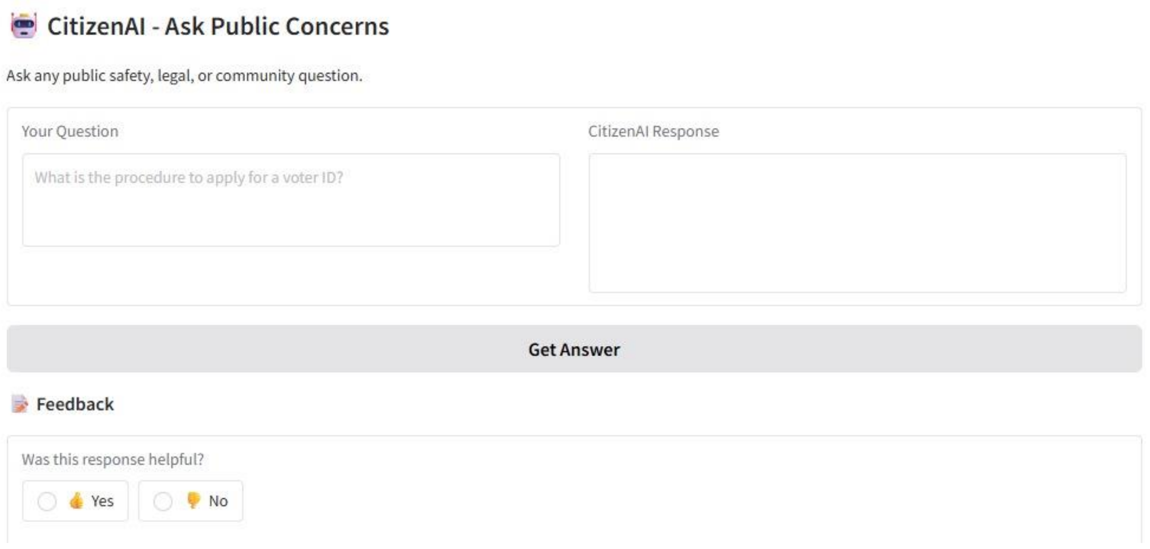
Sample JWT Flow (Planned)

1. User logs in → Backend returns JWT token
2. Token is sent with every request via the Authorization: Bearer <token> header
3. Backend validates the token and grants access

Security Considerations

- Passwords will be hashed using SHA-256 or bcrypt
- API access will be rate-limited to prevent abuse
- Sensitive data (if stored) will be encrypted or masked

9.User Interface



CitizenAI - Ask Public Concerns

Ask any public safety, legal, or community question.

Your Question

What is the procedure to apply for a voter ID?

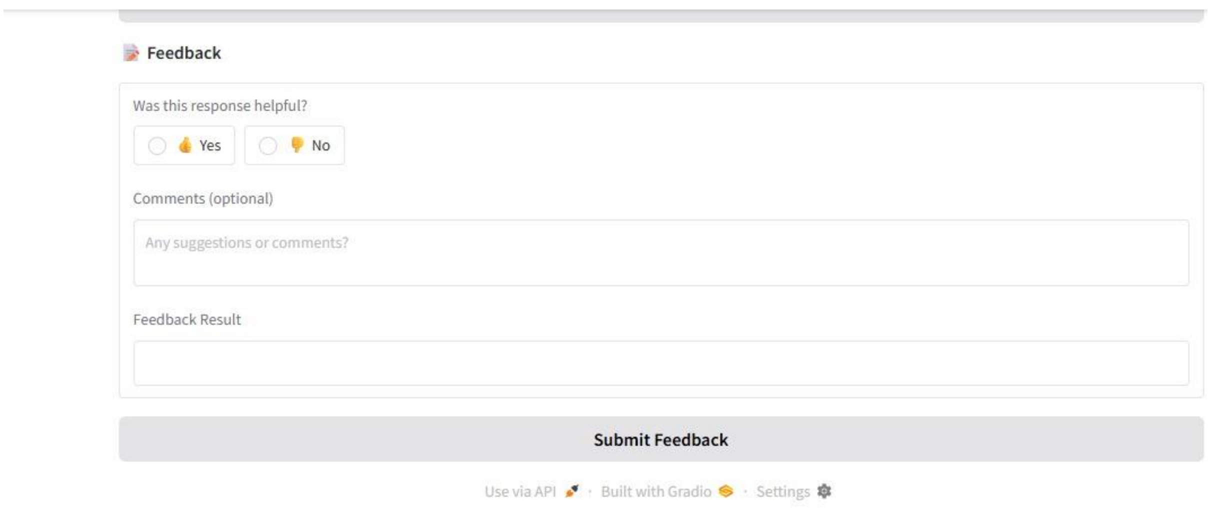
CitizenAI Response

Get Answer

Feedback

Was this response helpful?

Yes No



Feedback

Was this response helpful?

Yes No

Comments (optional)

Any suggestions or comments?

Feedback Result

Submit Feedback

Use via API · Built with Gradio · Settings

10.Testing

Since the MVP version is developed in a lightweight interactive environment (Google Colab), manual and functional testing was used to validate the core logic and user experience.

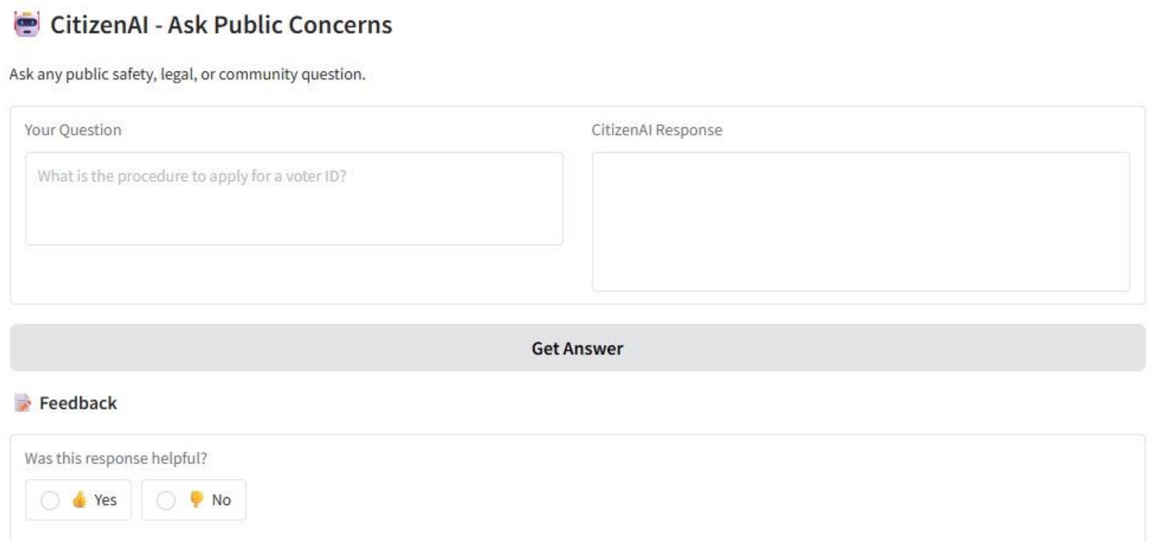
Manual Testing

- Inputs Tested:
  - Civic queries like "Water problem in my street"
  - Informational queries like "Tell me about PM KISAN Yojana"
- Validation:
  - Accuracy of AI responses
  - Chat flow behavior
  - Handling of undefined inputs
  - Response time and model load stability

Output Verification

- Each output was verified manually for:
  - Relevance
  - Language clarity
  - Context understanding

11.Screenshots:



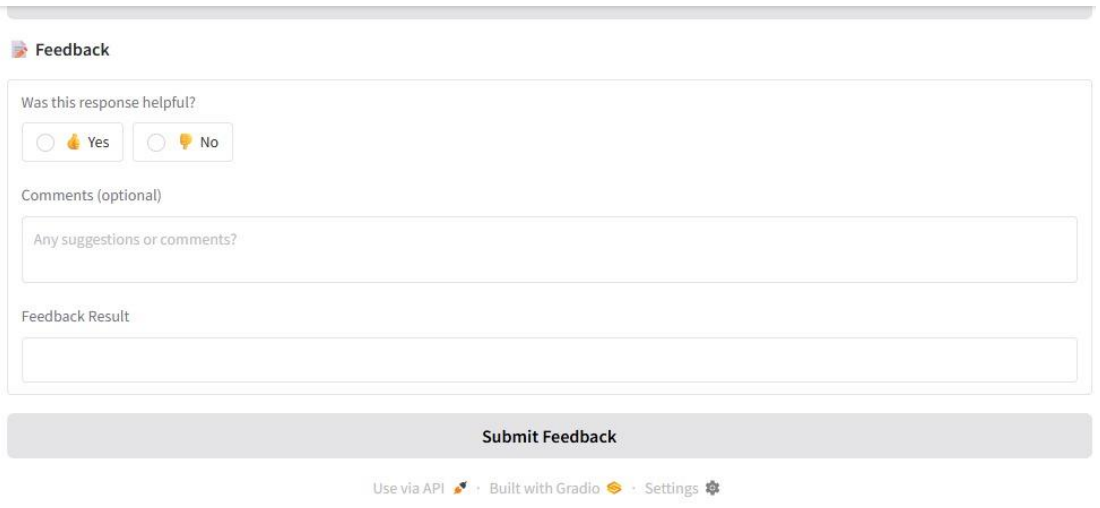
CitizenAI - Ask Public Concerns

Ask any public safety, legal, or community question.

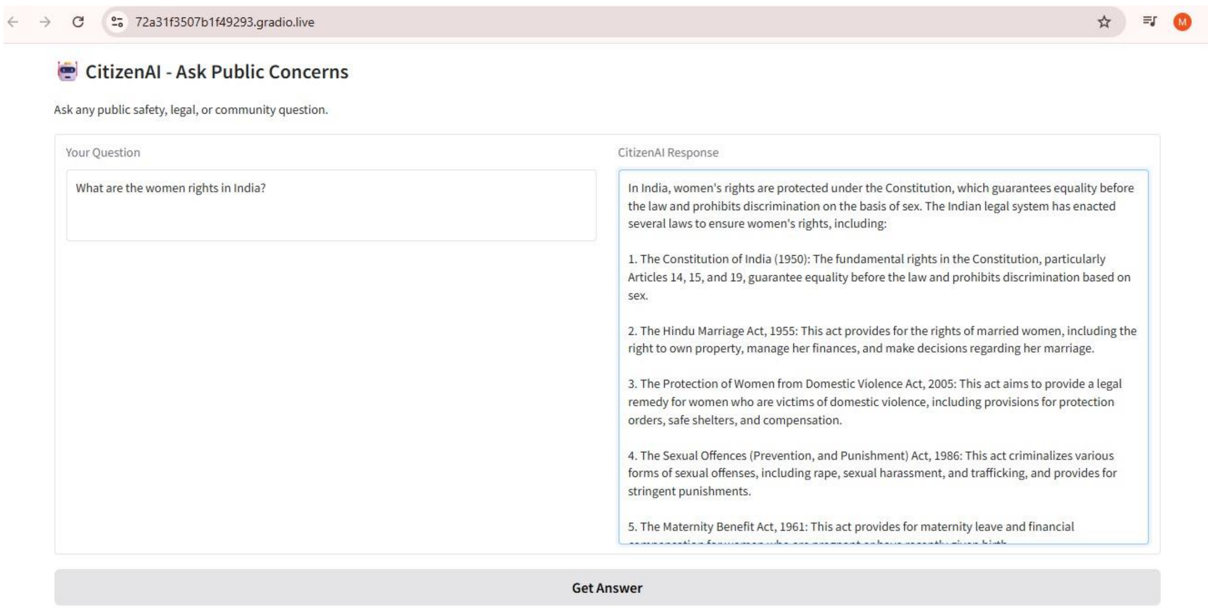
Your Question

What is the procedure to apply for a voter ID?

CitizenAI Response



Feedback
Was this response helpful?
👍 Yes
👎 No
Comments (optional)
Any suggestions or comments?
Feedback Result
Submit Feedback
Use via API · Built with Gradio · Settings



CitizenAI - Ask Public Concerns

Ask any public safety, legal, or community question.

Your Question

What are the women rights in India?

CitizenAI Response

In India, women's rights are protected under the Constitution, which guarantees equality before the law and prohibits discrimination on the basis of sex. The Indian legal system has enacted several laws to ensure women's rights, including:

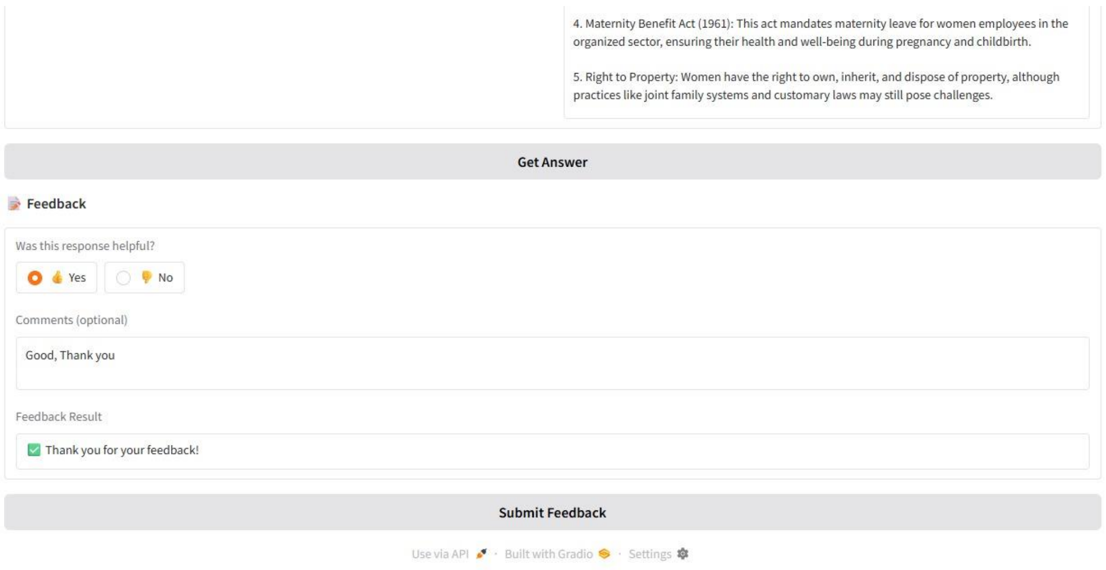
1. The Constitution of India (1950): The fundamental rights in the Constitution, particularly Articles 14, 15, and 19, guarantee equality before the law and prohibits discrimination based on sex.

2. The Hindu Marriage Act, 1955: This act provides for the rights of married women, including the right to own property, manage her finances, and make decisions regarding her marriage.

3. The Protection of Women from Domestic Violence Act, 2005: This act aims to provide a legal remedy for women who are victims of domestic violence, including provisions for protection orders, safe shelters, and compensation.

4. The Sexual Offences (Prevention, and Punishment) Act, 1986: This act criminalizes various forms of sexual offenses, including rape, sexual harassment, and trafficking, and provides for stringent punishments.

5. The Maternity Benefit Act, 1961: This act provides for maternity leave and financial



4. Maternity Benefit Act (1961): This act mandates maternity leave for women employees in the organized sector, ensuring their health and well-being during pregnancy and childbirth.

5. Right to Property: Women have the right to own, inherit, and dispose of property, although practices like joint family systems and customary laws may still pose challenges.

Was this response helpful?

[✓ 👍 Yes] 👎 No

Comments (optional)

Good, Thank you

Feedback Result

✅ Thank you for your feedback!

12.Issues:

While the CitizenAI – Intelligent Citizen Engagement Platform functions effectively for its primary use cases, the following known issues and limitations exist in the current version (MVP), particularly due to the rapid development cycle and dependency on third-party services.

1. Model Latency
   - Issue: Response generation from the IBM Granite model can occasionally be delayed, especially when run on limited Colab GPU or under network load.
   - Impact: Slight lag in chatbot replies.
   - Workaround: Use Google Colab with GPU (T4) runtime to reduce wait time.

2. Lack of Authentication
   - Issue: No user login or session management in the MVP.
   - Impact: All users are treated anonymously, limiting personalization or tracking.
   - Planned Fix: Implement JWT-based authentication in the production version.

3. Static Context Handling
   - Issue: The model doesn't maintain conversation history or user context between queries.
   - Impact: It may give disconnected answers in multi-turn conversations.

- Planned Fix: Add session-based context tracking for improved continuity.

4. Limited Issue Categorization

- Issue: The model cannot yet classify and route civic issues (e.g., pothole vs water) to specific departments.
- Impact: Limits the ability to automate escalation to municipal systems.
- Planned Fix: Integrate rule-based or ML classifiers for issue tagging.

5. No Offline Support

- Issue: App requires internet connectivity to access the cloud model.
- Impact: Not usable in remote or low-connectivity areas.
- Suggestion: Explore local model hosting or cached response fallback.

13. Future Enhancements:

As the CitizenAI – Intelligent Citizen Engagement Platform evolves beyond its MVP stage, several improvements and new features can be introduced to enhance scalability, accessibility, user engagement, and integration with real-world governance systems.

---

1. Multilingual Support
   - Goal: Enable citizens to interact in regional languages such as Hindi, Telugu, Tamil, etc.
   - How: Integrate translation APIs (Google Translate, IndicNLP) or fine-tune the Granite model for multilingual queries.
2. Secure Authentication System
   - Goal: Enable user login, role-based access, and complaint tracking.
   - How: Implement JWT-based authentication, secure login endpoints, and OAuth for social login.
3. Issue Tracking Dashboard
   - Goal: Allow users and authorities to track reported civic issues.
   - How: Build a dynamic dashboard to view status, response time, and feedback.
4. Geolocation-Based Services
   - Goal: Auto-fetch user location to route civic complaints to the correct municipal zone.
   - How: Integrate browser-based geolocation API and GIS tagging.

5. Voice-Enabled Interaction
   - Goal: Allow voice commands for elderly and low-literacy users.

- How: Use IBM Watson Speech-to-Text and Text-to-Speech APIs to convert voice to chat and vice versa.

6. Advanced AI Capabilities
   - Goal: Make the chatbot smarter by adding context-awareness and summarization.
   - How: Use session history, embeddings, or RAG (Retrieval-Augmented Generation).

7. Feedback-Based Learning
   - Goal: Improve model performance over time using user feedback.
   - How: Log feedback data and retrain or fine-tune the model periodically.

8. Integration with Government APIs
   - Goal: Allow users to file real-time complaints into government portals.
   - How: Integrate APIs from Swachh Bharat, MyGov, Municipal CRMs, etc.

9. Offline Mode / PWA
   - Goal: Allow access in low-connectivity areas.
   - How: Build a Progressive Web App (PWA) with local caching and fallback models.

10. Analytics for Administrators

- Goal: Visualize common issues, area-wise complaint density, and AI performance.
- How: Integrate with dashboards like IBM Cognos or custom React charts.