CITIZEN AI – INTELLIGENT CITIZEN ENGAGEMENT PLATFORM

1.Introduction :

• Project title : Citizen AI-Intelligent Citizen Engagement Platform

• Team member: SARASWATHI.S

• Team member : JANANI.E

• Team member : DHANALAKSHMI.S

• Team member : KANISHKA.M

2.project overview :

The Citizen AI-Intelligent Citizen Engagement Platform is designed to improve communication between citizens and government services through artificial intelligence. Its primary purpose is to provide a seamless and efficient way for citizens to interact with government agencies, access public services, and report concerns.

1. Real-Time AI Chatbot: allows citizens to ask questions and receive immediate responses about public services, documents, and civic procedures
2. Citizen Sentiment Analysis: analyzes public sentiment from submitted feedback to understand citizen satisfaction and concerns
3. -Interactive Dashboard: visualizes feedback trends and sentiment data to help government agencies make informed decisions
4. Context-Aware Responses: provides accurate and relevant responses based on the conversation flow
5. Concern Management: enables citizens to report issues and concerns, which are then tracked and prioritized by government agencies
6. Admin Dashboard: provides a centralized platform for government agencies to manage citizen interactions, sentiment analysis, and concern tracking

Benefits:

1. Improved Citizen Engagement: fosters a more interactive and responsive relationship between citizens and government agencies
2. Enhanced Public Services: streamlines access to public services and information, making it easier for citizens to navigate government bureaucracy
3. Data-Driven Decision Making: provides government agencies with valuable insights into citizen sentiment and concerns, enabling data-driven decision making
4. -ncreased Transparency: promotes transparency and accountability in government agencies through interactive dashboards and sentiment analysis

Overall, the Citizen AI platform aims to create a more citizen-centric and responsive government by leveraging AI-powered technologies to improve communication, engagement, and public services.

3. Architecture :

The architecture of the Citizen AI-Intelligent Citizen Engagement Platform is designed to facilitate seamless interaction between citizens and government agencies. Here's an overview of the platform's architecture:

Components:

1. User Interface (UI)
2. AI-Powered Chatbot
3. Natural Language Processing (NLP)
4. Sentiment Analysis
5. Data Analytics
6. Integration Layer
7. Security and Authentication

Architecture:

1. Frontend: The UI is built using modern web technologies, providing a responsive and intuitive experience for citizens.

2. Backend: The backend is built using a robust framework, handling requests, processing data, and integrating with external systems.

3. AI Engine: The AI engine powers the chatbot, sentiment analysis, and data analytics, enabling intelligent and personalized interactions.

4. Data Storage: A secure and scalable data storage solution stores citizen interactions, feedback, and sentiment data.

4. Set-up Instructions :

Prerequisites:

1. Server Requirements: Ensure you have a server with sufficient resources (CPU, RAM, and storage) to support the platform's requirements.

2. Database: Set up a database management system (e.g., MySQL, PostgreSQL) to store citizen data, interactions, and feedback.

3. Cloud Infrastructure*: Ensure you have a cloud infrastructure (e.g., AWS, Azure, Google Cloud) to host the platform and scale as needed.

4. Security: Implement robust security measures, including encryption, firewalls, and access controls, to protect citizen data and prevent unauthorized access.

5. AI and ML Frameworks: Install necessary AI and ML frameworks (e.g., TensorFlow, PyTorch) to support the platform's AI-powered features.

Installation Process:

1. Clone the Repository: Clone the Citizen AI repository from a version control system (e.g., GitHub).

2. Configure Environment Variables: Configure environment variables, such as database connections, API keys, and cloud infrastructure settings.

3. Install Dependencies: Install required dependencies, including libraries and frameworks, using a package manager (e.g., pip, npm).

4. Run Migrations: Run database migrations to set up the schema and populate initial data.

5. Deploy to Cloud: Deploy the platform to your cloud infrastructure, configuring load balancers, auto-scaling, and monitoring as needed.

6. Configure AI Models: Configure and train AI models, such as chatbots and sentiment analysis, using provided datasets or your own data.

7. Test and Validate: Test and validate the platform's functionality, performance, and security to ensure it meets requirements.

5. Folder Structure :

Root Directory

- `citizen-ai/`

Subdirectories

1. Backend
   - `backend/`
     - `app/` (application code)
     - `config/` (configuration files)
     - `models/` (database models)
     - `routes/` (API routes)
     - `services/` (business logic)
     - `utils/` (utility functions)

2. Frontend
   - `frontend/`
      - `public/` (static assets)
      - `src/` (source code)
         - `components/` (UI components)
         - `containers/` (container components)
         - `actions/` (action creators)
         - `reducers/` (reducers)
         - `utils/` (utility functions)
3. AI and ML
   - `ai-ml/`
      - `models/` (AI and ML models)
      - `training/` (model training scripts)
      - `evaluation/` (model evaluation scripts)
4. Database
   - `database/`
      - `schema/` (database schema)
      - `migrations/` (database migrations)
5. Documentation
   - `docs/`
      - `api/` (API documentation)
      - `user/` (user documentation)
      - `developer/` (developer documentation)

6.Running the Application:

Running the Backend

1. Navigate to the backend directory: `cd backend`
2. Run the backend server: `python app.py` (or the command specified in your backend framework)

Running the Frontend

1. Navigate to the frontend directory: `cd frontend`

2. Run the frontend development server:`npm start` (or the command specified in your frontend framework)

Running the AI and ML Components

1. Navigate to the AI and ML directory:`cd ai-ml`

2. Run the AI and ML models: `python train.py` (or the command specified in your AI and ML framework)

By following these steps, you can run the Citizen AI application and test its functionality. Ensure you have the necessary dependencies installed and configured environment variables before running the application.

7.API Documentation :

- POST /chat/ask: Submit a question to the AI chatbot and receive a response
- GET /chat/history: Retrieve chat history
- POST /feedback/submit: Submit feedback on government services
- GET /feedback/analyze: Analyze sentiment of submitted feedback
- POST /concern/submit: Report a concern or issue
- GET /concern/list: List all reported concerns
- GET /concern/{id}: Retrieve a specific concern by ID

8. Authentication:

- GET /auth/login: Login page for administrators
- POST /auth/login: Process login credentials
- GET /auth/logout: Logout

9. User Interface:

The user interface (UI) of the Citizen AI-Intelligent Citizen Engagement Platform is designed to provide a seamless and intuitive experience for citizens to interact with government services. Here are some key features of the UI

1. Chatbot Interface
2. Feedback Mechanism:
3. Concern Reporting
4. Personalized Dashboard
5. User-Friendly Navigation
6. Accessibility Features

Design Principles:

1. Citizen-Centric
2. Simple and Intuitive
3. Consistent
4. Accessible

Benefits:

1. Improved Citizen Engagement
2. Increased Transparency
3. Enhanced User Experience

10. Testing :

1. Unit Testing: Verify individual components or units of code function correctly.
2. Integration Testing: Test interactions between components or systems.
3. Functional Testing: Validate platform functionality meets requirements.
4. Performance Testing: Evaluate platform performance under various loads.
5. Security Testing:Identify vulnerabilities and ensure data protection.

Testing Tools

1. Automated Testing Frameworks:Use frameworks like Selenium or Cypress for automated testing.
2. Manual Testing: Perform manual testing to identify issues not caught by automated tests.

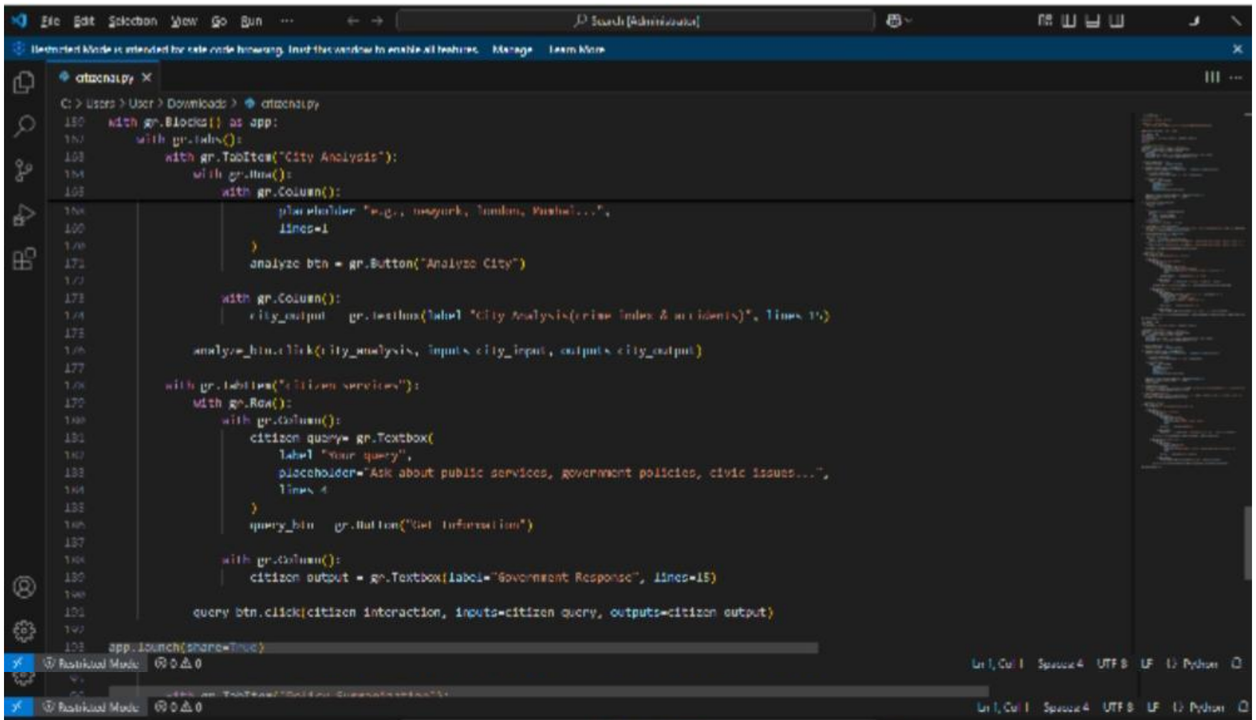
11. Known Issues :

- Security and Data Protection Issues
- AI Model – Related Issues
- Operational and Technical Issues
- User Experience Issues

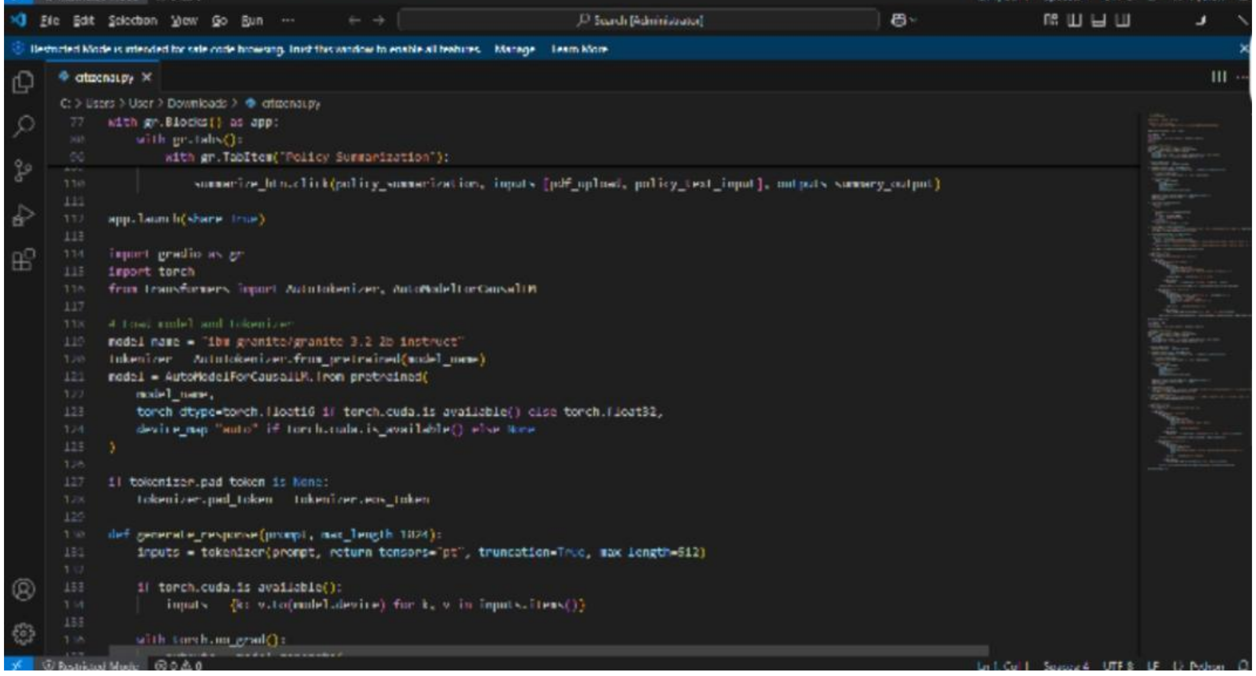
12. Future Enhancement:

- Advanced AI – Powered Chatbots
- Prediction Analytics
- Personalized Citizen Experiences
- Multilingual Support
- Sentiment Analysis
- Integration with Emerging Technologies
- Enhanced Security Measures

13. Screenshot:



```
with gr.Blocks() as app:
    with gr.Tabs():
        with gr.TabItem("City Analysis"):
            with gr.Row():
                with gr.Column():
                        placeholder="e.g., newyork, london, Mumbai...",
                        lines=1
                    )
                    analyze_btn = gr.Button("Analyze City")

                with gr.Column():
                    city_output = gr.Textbox(label="City Analysis(crime index & accidents)", lines=15)

            analyze_btn.click(city_analysis, inputs=city_input, outputs=city_output)

        with gr.TabItem("citizen services"):
            with gr.Row():
                with gr.Column():
                    citizen_query= gr.Textbox(
                        label="Your query",
                        placeholder="Ask about public services, government policies, civic issues...",
                        lines=4
                    )
                    query_btn = gr.Button("Get Information")

                with gr.Column():
                    citizen_output = gr.Textbox(label="Government Response", lines=15)

            query_btn.click(citizen_interaction, inputs=citizen_query, outputs=citizen_output)

app.launch(share=True)
```



```
with gr.Blocks() as app:
    with gr.Tabs():
        with gr.TabItem("Policy Summarization"):
                summarize_btn.click(policy_summarization, inputs=[pdf_upload, policy_text_input], outputs=summary_output)

app.launch(share=True)

import gradio as gr
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM

# Load model and tokenizer
model_name = "ibm-granite/granite-3.2-2b-instruct"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(
    model_name,
    torch_dtype=torch.float16 if torch.cuda.is_available() else torch.float32,
    device_map="auto" if torch.cuda.is_available() else None
)

if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token

def generate_response(prompt, max_length=1024):
    inputs = tokenizer(prompt, return_tensors="pt", truncation=True, max_length=512)

    if torch.cuda.is_available():
        inputs = {k: v.to(model.device) for k, v in inputs.items()}

    with torch.no_grad():
```