CITIZEN AI INTELLIGENT CITIZEN ENGAGEMENT PLATFORM PROJECT DOCUMENT

1.INTRODUCTION:

- PROJECT TITLE : Citizen AIIntelligent Citizen Engagement platform
- Aarthi.R(38C4A4884F90F19A1654E7AA4FAAE28C)
- Bhuvaneswari. J (347FCOC6BD19EE398B94E3C4092208AD)
- Gobika.V (8DEE342D4434267A34344652E33F39ED)
- Nandhini.R (6CD478A3735DC139BE6429869CDE4EBO)

2.PROJECT OVERVIEW:

- Citizen AIwithIBMis designed to create a bridge between citizens and government services using artificial intelligence. The project leverages IBM's AI ecosystem to provide citizens with simplified access to information, quick query resolution, and personalized recommendations. It is aimed at improving transparency, reducing response time, and enhancing trust between the government and citizens.

- Unlike traditional portals, Citizen AI integrates conversational agents, data analytics, and feedback systems to provide a more human-like interaction. The goal is to make government services more accessible, especially for people with limited technical skills.

- Key features of Citizen AI include:

- AI-Powered Assistance – Citizens can interactwiththesystem through natural language, asking questionsaboutservices, policies, or benefits.
- Sentiment Analysis – Feedback from citizensisautomatically analyzed to measure satisfaction and identifyissues.
- Policy Summarization – Long government documentsare condensed into short, easy-to-read summaries.
- Multilingual Support – Supports multiple Indianandglobal languages, ensuring inclusivity.
- Personalized Suggestions – Provides tailoredrecommendations Basedon citizen prifiles and past interactions

- Secure Data Handling – Uses IBM's cloud infrastructurefordata encryption and safe storage.
- Scalability – Designed to handle thousands of requests simultaneously with high availability.

3.ARCHITECTURE:

- The architecture of Citizen AI with IBM is divided into three major layers:
- 1. User Interaction Layer

- Web application and mobile app built with Gradio UI.
- Provides multilingual chat interface.
- Allows file uploads for policy or form analysis.

- 2. Application Logic Layer

- Dialog Manager: Handles conversation flow.

- Sentiment Analyzer: Processes feedback in real time.
- Recommendation Engine: Suggests relevant services.
- Authentication Module: Ensures secure citizen login.

- 3. Data & AI Layer

- IBM Watson NLP for text understanding.
- Granite Model Integration for policy summarization.
- Database (IBM Cloud Databases) for storing citizen interactions.
- Analytics Dashboard for government administrators to view trends.

4.SETUP INSTRUCTIONS:

- Prerequisites:

    - Python programming knowledge.
    - Gradio framework
    - IBM granite model accesses(via hugging face)
    - Google colab with T4 GPU
    - Github account steps:

1. Access the Naan mudhalvan Smart Internz Portal.
2. Choose an IBM Granite model from Hugging face.

3. Run the application in Google colab with required libraries.
4. Upload final project files to Github

5.FOLDER STRUCTURE:

- ❖app/:Backend logic and integration.
- ❖ui/: Gradio app interface files.
- ❖citizen_ai.py: Main application files.
- ❖model_loader.py: Handles IBM granite model integration.
- ❖dashboard.py: Visualization of citizen feedback.

6.RUNNING THE APPLICATION:

1.Open Google colab and load the project notebook.

2.Install dependencies and configure runtime with GPU.

3.Run the notebook cells to start the Gradio app.

4.Access the provided link to interact with Citizen AI.

7.API DOCUMENTATION:

CITIZEN AI PROVIDES ENDPOINTS FOR:

- ❖ Asking questions about government services.
- ❖ Uploading feedback for sentiment analysis.
- ❖ Viewing summarized policies.
- ❖ Accessing dashboards and reports.

8.AUTHENTICATION:

- Two factor Authentication(2FA):

- Users verify their identity using OTP send to their phone or E -Mail.

- Role based access:
  - Citizens: can ask queries and view information.
  - Officials: Can access detailed reports and analytics.
  - DataProtection: All credentials are encrypted using IBM key protect.

9.USER INTERFACE:

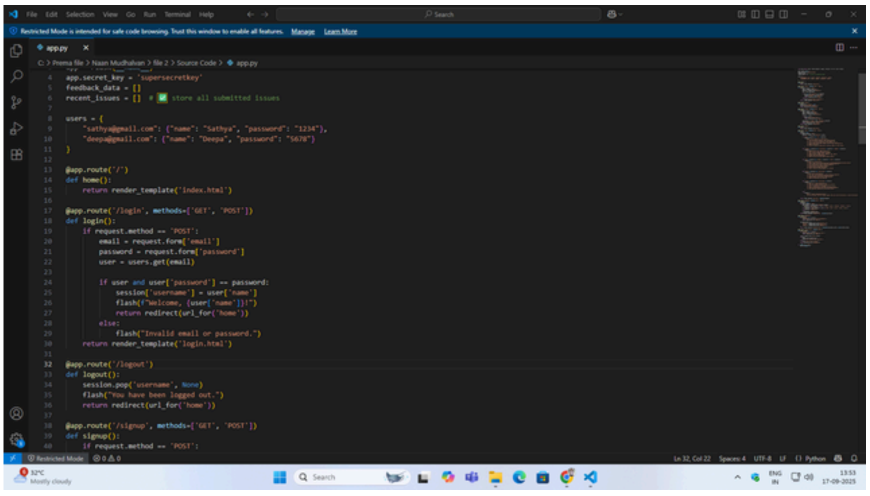
- CITIZEN PORTAL: simple chatbot interface withoptions for voice and text input.
- DASHBOARD: Graphical in sights for administrators,including charts for feedback sentiment.
- ACCESSIBILITY FEATURES:Adjustable font size,screen reader compatibility,and regional language support.

10.TESTING:

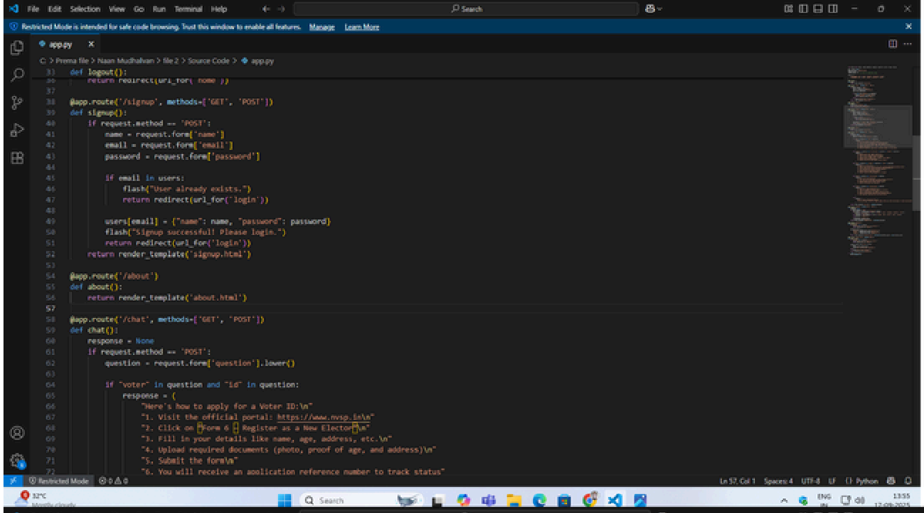
TESTING INCLUDED:

- Unit testing for AI responses.
- Manual testing of the Gradio interfaces.
- Edge case handling with unexpected inputs.

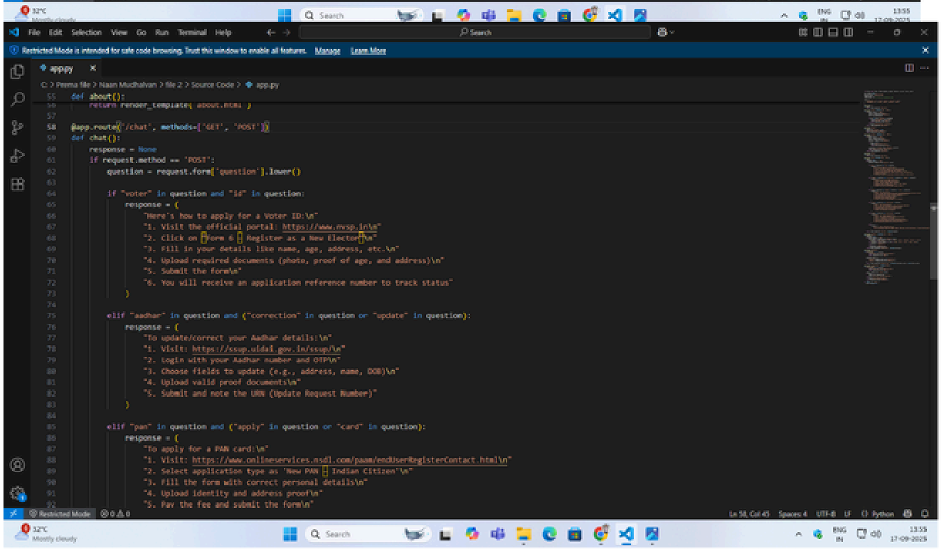
11.SCREENSHOTS:


File Edit Selection View Go Run Terminal Help
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
app.py
C: > Prema file > Naan Mudhalvan > file 2 > Source Code > app.py
Restricted Mode
32°C Mostly cloudy

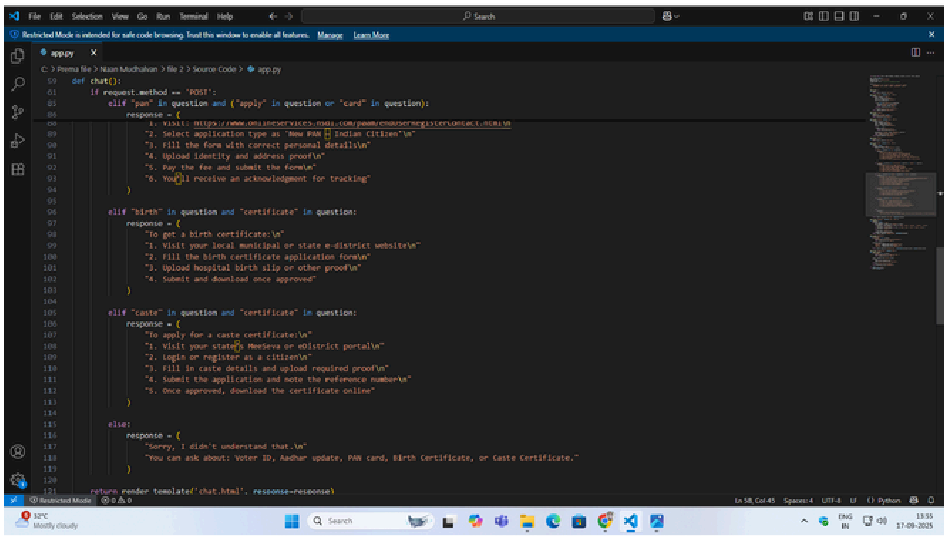
```python
app.secret_key = 'supersecretkey'
feedback_data = []
recent_issues = []  # ✅ store all submitted issues

users = {
    "sathya@gmail.com": {"name": "Sathya", "password": "1234"},
    "deepa@gmail.com": {"name": "Deepa", "password": "5678"}
}

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']
        user = users.get(email)

        if user and user['password'] == password:
            session['username'] = user['name']
            flash(f"Welcome, {user['name']}!")
            return redirect(url_for('home'))
        else:
            flash("Invalid email or password.")
    return render_template('login.html')

@app.route('/logout')
def logout():
    session.pop('username', None)
    flash("You have been logged out.")
    return redirect(url_for('home'))

@app.route('/signup', methods=['GET', 'POST'])
def signup():
    if request.method == 'POST':
```



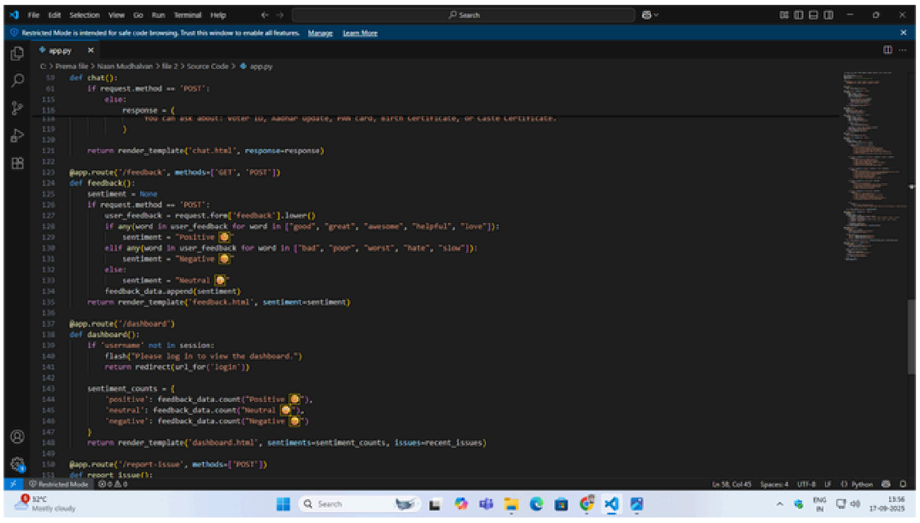
```python
def logout():
    return redirect(url_for('home'))

@app.route('/signup', methods=['GET', 'POST'])
def signup():
    if request.method == 'POST':
        name = request.form['name']
        email = request.form['email']
        password = request.form['password']

        if email in users:
            flash("User already exists.")
            return redirect(url_for('login'))

        users[email] = {"name": name, "password": password}
        flash("Signup successful! Please login.")
        return redirect(url_for('login'))
    return render_template('signup.html')

@app.route('/about')
def about():
    return render_template('about.html')

@app.route('/chat', methods=['GET', 'POST'])
def chat():
    response = None
    if request.method == 'POST':
        question = request.form['question'].lower()

        if "voter" in question and "id" in question:
            response = (
                "Here's how to apply for a Voter ID:\n"
                "1. Visit the official portal: https://www.nvsp.in\n"
                "2. Click on ▯Form 6 ▯ Register as a New Elector▯\n"
                "3. Fill in your details like name, age, address, etc.\n"
                "4. Upload required documents (photo, proof of age, and address)\n"
                "5. Submit the form\n"
                "6. You will receive an application reference number to track status"
```



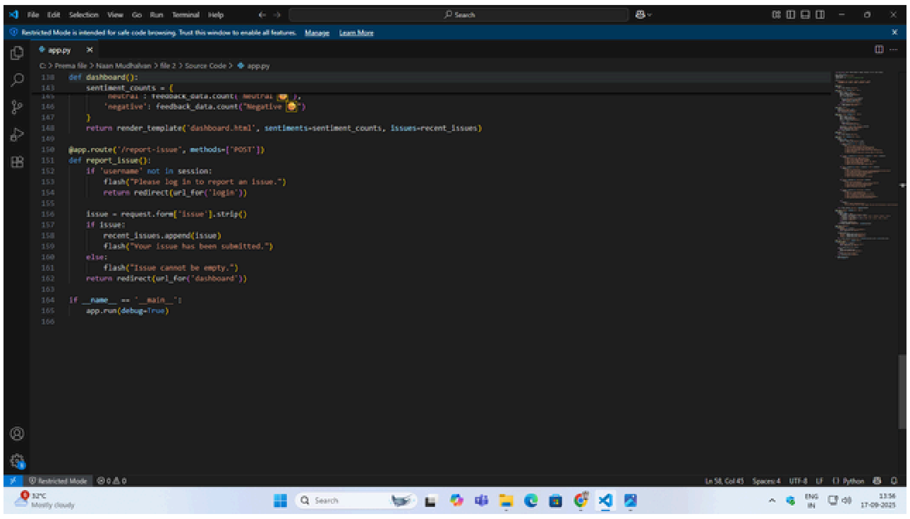
```python
def about():
    return render_template('about.html')

@app.route('/chat', methods=['GET', 'POST'])
def chat():
    response = None
    if request.method == 'POST':
        question = request.form['question'].lower()

        if "voter" in question and "id" in question:
            response = (
                "Here's how to apply for a Voter ID:\n"
                "1. Visit the official portal: https://www.nvsp.in\n"
                "2. Click on ▯Form 6 ▯ Register as a New Elector▯\n"
                "3. Fill in your details like name, age, address, etc.\n"
                "4. Upload required documents (photo, proof of age, and address)\n"
                "5. Submit the form\n"
                "6. You will receive an application reference number to track status"
            )

        elif "aadhar" in question and ("correction" in question or "update" in question):
            response = (
                "To update/correct your Aadhar details:\n"
                "1. Visit: https://ssup.uidai.gov.in/ssup/\n"
                "2. Login with your Aadhar number and OTP\n"
                "3. Choose fields to update (e.g., address, name, DOB)\n"
                "4. Upload valid proof documents\n"
                "5. Submit and note the URN (Update Request Number)"
            )

        elif "pan" in question and ("apply" in question or "card" in question):
            response = (
                "To apply for a PAN card:\n"
                "1. Visit: https://www.onlineservices.nsdl.com/paam/endUserRegisterContact.html\n"
                "2. Select application type as 'New PAN ▯ Indian Citizen'\n"
                "3. Fill the form with correct personal details\n"
                "4. Upload identity and address proof\n"
                "5. Pay the fee and submit the form\n"
```



```python
def chat():
    if request.method == 'POST':
        elif "pan" in question and ("apply" in question or "card" in question):
            response = (
                "1. Visit: https://www.onlineservices.nsdl.com/paam/endUserRegisterContact.html\n"
                "2. Select application type as 'New PAN ▯ Indian Citizen'\n"
                "3. Fill the form with correct personal details\n"
                "4. Upload identity and address proof\n"
                "5. Pay the fee and submit the form\n"
                "6. You▯ll receive an acknowledgment for tracking"
            )

        elif "birth" in question and "certificate" in question:
            response = (
                "To get a birth certificate:\n"
                "1. Visit your local municipal or state e-district website\n"
                "2. Fill the birth certificate application form\n"
                "3. Upload hospital birth slip or other proof\n"
                "4. Submit and download once approved"
            )

        elif "caste" in question and "certificate" in question:
            response = (
                "To apply for a caste certificate:\n"
                "1. Visit your state▯s MeeSeva or eDistrict portal\n"
                "2. Login or register as a citizen\n"
                "3. Fill in caste details and upload required proof\n"
                "4. Submit the application and note the reference number\n"
                "5. Once approved, download the certificate online"
            )

        else:
            response = (
                "Sorry, I didn't understand that.\n"
                "You can ask about: Voter ID, Aadhar update, PAN card, Birth Certificate, or Caste Certificate."
            )

    return render_template('chat.html', response=response)
```



```python
def chat():
    if request.method == 'POST':
        else:
            response = (
                "You can ask about: Voter ID, Aadhar update, PAN card, Birth Certificate, or Caste Certificate."
            )

    return render_template('chat.html', response=response)

@app.route('/feedback', methods=['GET', 'POST'])
def feedback():
    sentiment = None
    if request.method == 'POST':
        user_feedback = request.form['feedback'].lower()
        if any(word in user_feedback for word in ["good", "great", "awesome", "helpful", "love"]):
            sentiment = "Positive 😊"
        elif any(word in user_feedback for word in ["bad", "poor", "worst", "hate", "slow"]):
            sentiment = "Negative 😞"
        else:
            sentiment = "Neutral 😐"
        feedback_data.append(sentiment)
    return render_template('feedback.html', sentiment=sentiment)

@app.route('/dashboard')
def dashboard():
    if 'username' not in session:
        flash("Please log in to view the dashboard.")
        return redirect(url_for('login'))

    sentiment_counts = {
        'positive': feedback_data.count("Positive 😊"),
        'neutral': feedback_data.count("Neutral 😐"),
        'negative': feedback_data.count("Negative 😞")
    }
    return render_template('dashboard.html', sentiments=sentiment_counts, issues=recent_issues)

@app.route('/report-issue', methods=['POST'])
def report_issue():
```



```python
def dashboard():
    sentiment_counts = {
        'neutral': feedback_data.count('Neutral 😐'),
        'negative': feedback_data.count("Negative 😞")
    }
    return render_template('dashboard.html', sentiments=sentiment_counts, issues=recent_issues)

@app.route('/report-issue', methods=['POST'])
def report_issue():
    if 'username' not in session:
        flash("Please log in to report an issue.")
        return redirect(url_for('login'))

    issue = request.form['issue'].strip()
    if issue:
        recent_issues.append(issue)
        flash("Your issue has been submitted.")
    else:
        flash("Issue cannot be empty.")
    return redirect(url_for('dashboard'))

if __name__ == '__main__':
    app.run(debug=True)
```

12.KNOWN ISSUES:

- Limited scopedue to demo environment.
- Requires internet for colab runtime and Hugging face model.

13.FUTURE ENHANCEMENTS:

- Integrate advanced analytics fordeeper insights.
- Expand support for multiple languages. Deploy on
- cloud platforms forreal world scalability.

14.DEMO VIDEO LINK

https://drive.google.com/file/d/1nhH5fxH1OIDqUaRqgcV4_Q-8Ws6Uf2UC/view?usp=drivesdk