

A Project Report on Citizen AI – Intelligent Citizen Engagement Platform

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Technology in Computer Science and Engineering

By:

Uda Mahesh (22FE1A05H1)

Viyyapu yugandhar()

Vasumalla Sony(22FE1A05H8)

Under the guidance of:

Ganesh M



Table of Contents

1. Abstract
2. Introduction
3. Existing System
4. Proposed System
5. Objectives
6. System Requirements
7. System Design
8. Implementation
9. Testing
10. Results
11. Advantages and Limitations
12. Conclusion
13. Future Scope
14. Screenshots

1. Abstract

Citizen AI is an intelligent, web-based platform developed to transform the traditional mechanisms of citizen engagement into a modern, data-driven, and AI-assisted system. Developed using Python (Flask) for the backend, JavaScript and Bootstrap for the frontend, and NLP libraries such as TextBlob for sentiment analysis, the platform creates a seamless interface between the government and its citizens.

The core functionality includes a real-time conversational assistant (chatbot), a structured feedback collection system, and an analytics dashboard that visualizes public sentiment using interactive charts. By enabling citizens to raise queries, provide suggestions, and express concerns, the platform empowers them to participate more actively in governance.

Unlike traditional government websites that offer static information and delayed response mechanisms, Citizen AI supports 24/7 interaction through an intelligent assistant. While the current implementation uses a rule-based engine for FAQs and services, the architecture is designed to be easily upgraded to incorporate advanced language models such as OpenAI's GPT-3.5, IBM Watson Assistant, or IBM Granite. This flexibility ensures that Citizen AI can evolve into a fully AI-driven public communication and governance tool in future iterations.

Furthermore, the feedback collected is automatically classified into sentiment categories — Positive, Neutral, or Negative — using NLP techniques. This allows administrative departments to monitor real-time public opinion and take informed, timely actions. The built-in dashboard provides easy-to-understand visual summaries of citizen engagement, making the platform not only a tool for communication but also for transparency and data-backed decision-making.

Citizen AI has the potential to improve public service delivery, increase trust in digital governance, and serve as a foundational model for smart city and e-governance platforms.

2. Introduction

Effective governance in the digital age requires more than just service delivery — it demands continuous citizen engagement, real-time feedback mechanisms, and transparent communication. However, most existing government portals and e-governance systems are static, outdated, and lack interactive interfaces. Citizens are often limited to filling out forms or sending emails, with no assurance of timely responses or visible follow-up action.

In many regions, the absence of two-way digital communication tools leads to growing public dissatisfaction, limited trust in online services, and underutilization of government platforms. There is a pressing need for a system that enables citizens not only to access information but to interact, express opinions, report issues, and receive meaningful support from their governing bodies.

Citizen AI is conceptualized as a solution to bridge this gap by offering a responsive, intelligent, and user-friendly platform. Designed using open-source technologies like Python (Flask), SQLite, and TextBlob, it enables real-time chat-based interactions between citizens and a rule-based digital assistant. This assistant can handle frequently asked questions, guide users to appropriate services, and simulate human-like support, even without full AI integration.

The platform also incorporates feedback collection and sentiment analysis, allowing governments to assess citizen satisfaction at scale. A built-in dashboard offers visual analytics that can help decision-makers understand public mood, identify recurring concerns, and improve service quality accordingly.

Scalable, cost-effective, and AI-ready, Citizen AI lays the groundwork for intelligent digital governance. Whether deployed for municipal services, smart city projects, or state-level e-governance portals, the platform demonstrates how AI-enhanced citizen engagement can improve transparency, trust, and administrative efficiency.

3. Existing System

The traditional model of citizen-government interaction predominantly relies on outdated communication channels such as emails, toll-free helplines, in-person visits, or static government websites with generic forms. While these methods may have served adequately in the past, they are increasingly proving to be inefficient in today's fast-paced, technology-driven environment.

Most government portals are non-interactive and offer limited real-time support. Citizens are often required to manually search through multiple pages for information or fill out online forms, which rarely provide any immediate response or confirmation. Queries submitted through emails or contact forms may take days or even weeks to receive a reply — if at all.

Additionally, feedback mechanisms, where available, are typically unstructured and do not offer any analytical insights. The absence of automated systems to classify or prioritize citizen input means that valuable opinions or complaints may be overlooked. Government personnel are burdened with manually processing large volumes of feedback, leading to further delays in addressing public concerns or updating policies.

Crucially, there is no provision for real-time sentiment tracking, which could help authorities proactively detect dissatisfaction, emergencies, or service failures. As a result, public trust in digital governance tools remains low, and citizen participation in government feedback programs is minimal.

This outdated, fragmented system forms the backdrop against which **Citizen AI** proposes a smarter, scalable, and automated alternative.

4. Proposed System

The proposed solution, **Citizen AI**, is a comprehensive platform designed to overcome the shortcomings of traditional citizen engagement systems. It provides a smart, interactive, and modular approach to enable seamless communication between the public and government departments.

At its core, Citizen AI offers a **chatbot-driven user interface** through which citizens can ask questions, request information, or report issues at any time. The chatbot is built using a rule-based response system that can be easily extended to support advanced AI models such as **OpenAI**, **IBM Watson Assistant**, or **IBM Granite** for natural language understanding and generative response capabilities.

In addition to real-time query resolution, the system provides a **structured feedback collection form**, where citizens can share their opinions or experiences regarding public services. This feedback is then analyzed using natural language processing (NLP) through **TextBlob**, which classifies sentiment into categories such as **Positive**, **Neutral**, or **Negative**.

Government officials can monitor this feedback via an **admin dashboard** that displays sentiment data through live visualizations using **Chart.js**. This empowers departments to make timely, data-informed decisions, identify problem areas, and assess public satisfaction with different services.

The entire system is designed with **offline-first functionality** using **SQLite** as the local database, ensuring lightweight storage and easy portability. The backend is built using **Flask**, a minimal yet powerful Python web framework, while the frontend utilizes **Bootstrap 5** for responsive and user-friendly design.

By integrating these components, Citizen AI creates a scalable foundation for intelligent, transparent, and efficient e-governance, with provisions to support future enhancements in AI, multilingual support, and cloud-based deployment.

5. Objectives

The primary goal of the **Citizen AI** project is to design and implement an intelligent, scalable, and user-friendly platform that enhances digital interaction between citizens and government authorities. The system is intended to bridge the communication gap, simplify service access, and introduce automated analytics to public feedback mechanisms.

The key objectives of the project are as follows:

- **Enable Real-Time Citizen-to-Government Communication:**
Provide a web-based interface where citizens can ask questions, raise concerns, and request information through an always-available chatbot interface.
- **Provide Instant, Human-Like Responses:**
Use a rule-based response engine capable of delivering fast, relevant, and easy-to-understand answers to frequently asked public queries. Design the system for future integration with AI models like OpenAI or IBM Watson to enhance response quality.
- **Analyze Public Feedback Using Sentiment Analysis:**
Implement NLP-based sentiment analysis using TextBlob to categorize citizen feedback as Positive, Neutral, or Negative. This enables officials to monitor the emotional tone of feedback in real-time.
- **Assist Government in Identifying Public Concerns:**
Aggregate and visualize sentiment data on an admin dashboard to help decision-makers recognize recurring issues, prioritize actions, and improve service delivery effectively.
- **Maintain a Responsive and Intuitive User Interface:**
Ensure the platform is simple, clean, and easy to use, with responsive layouts powered by Bootstrap. Include real-time dashboards and interactive charts to present key insights in a visual format.

Together, these objectives support the development of a powerful digital tool that fosters trust, transparency, and responsiveness in governance.

6. System Requirements

Software Requirements:

- Python 3.10+
- Flask 3.x
- TextBlob for NLP
- SQLite for local data storage
- Chart.js for dashboard visualization
- Bootstrap 5 for frontend layout

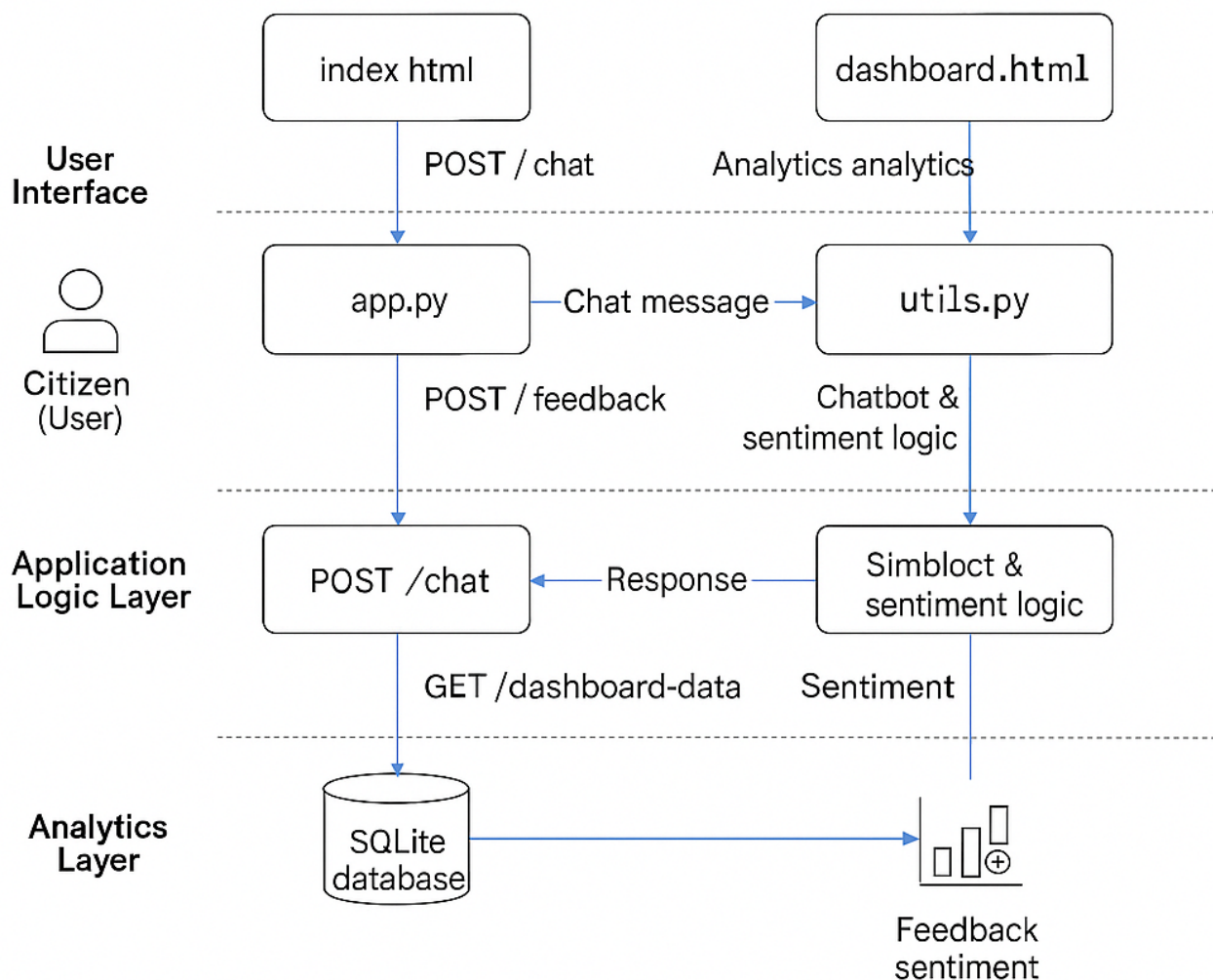
Hardware Requirements:

- Minimum 4GB RAM system
- Any OS (Windows/macOS/Linux)
- Web browser (Chrome/Edge/Firefox)
- Internet (optional for AI APIs)

7. System Design

Citizen AI is structured as a **four-layer architecture** to keep the presentation layer, application logic, data management, and analytics concerns clearly separated. This modular approach simplifies maintenance, supports future scaling, and allows individual layers to be upgraded (for example, swapping the rule-based assistant for an AI model) without major refactoring.

Citizen AI System Diagram



Layer	Description	Key Technologies / Files
1. User Interface Layer	Presents an intuitive, mobile-responsive web interface where citizens can chat with the assistant and submit feedback.	<code>templates/index.html</code> (chat & feedback) · Bootstrap 5 · <code>static/js/main.js</code>
2. Application Logic Layer	Handles HTTP routes, orchestrates chatbot responses, triggers sentiment analysis, and manages dashboard endpoints.	<code>app.py</code> (Flask routes) · <code>utils.py</code> (response + NLP)
3. Data Layer	Persists all chat messages and feedback entries in a lightweight relational store, enabling offline operation and rapid queries.	SQLite (<code>citizen.db</code>)
4. Analytics Layer	Retrieves aggregated sentiment counts and renders live visualisations so officials can monitor public mood at a glance.	<code>templates/dashboard.html</code> + <code>static/js/dashboard.js</code> (Chart.js pie chart)

8. Implementation

The development of the *Citizen AI* platform was executed in three systematic stages, each building upon the previous to ensure modularity, flexibility, and functionality:

Stage 1: Flask Backend Setup

The foundation of the platform was built using **Flask**, a lightweight Python web framework ideal for rapid backend development.

- **Route Configuration:**
 - `/`: Serves the homepage with a user-friendly chat interface and feedback form.
 - `/chat`: Handles real-time communication between the user and the AI chatbot, processing queries and generating appropriate responses.
 - `/feedback`: Accepts and stores feedback submitted by users, which is later analyzed for sentiment.
 - `/dashboard`: Renders a dynamic admin dashboard that visually represents user sentiment trends.
- **File Structure:**
 - `app.py`: Manages Flask app configuration and route definitions.
 - `utils.py`: Contains modular functions like `granite_generate()` and `analyze_sentiment()` to handle AI logic and feedback analysis.

- **Database:**
 - Implemented using **SQLite** for lightweight local storage.
 - Maintains records of user messages, chatbot responses, feedback entries, and their corresponding sentiment scores.

Stage 2: Chatbot Integration

To power the citizen engagement feature, a rule-based chatbot system was integrated:

- **Core Logic:**
 - `granite_generate()`: A custom-built function that uses keyword-based logic and predefined response rules to simulate intelligent behavior.
 - Covers common government-related queries (e.g., grievance redressal, service availability, scheme eligibility).
- **Fallback Mechanism:**
 - If a query doesn't match predefined intents, a **default fallback response** is triggered to handle unexpected inputs gracefully.
 - Encourages users to rephrase or specify queries, ensuring conversational continuity.
- **Optional AI Model Integration:**
 - Though primarily rule-based, the architecture supports future integration of:
 - **OpenAI GPT** (via API) for advanced NLP-driven replies.
 - **IBM Watson Assistant** for enterprise-grade chatbot functionality.
 - This flexibility allows the system to evolve from a static chatbot to a dynamic conversational agent as needed.

Stage 3: Sentiment Analysis and Analytics Dashboard

This stage focused on capturing and understanding user satisfaction through data analytics:

- **Sentiment Analysis:**
 - Utilized **TextBlob**, a Python NLP library, to analyze user feedback.
 - Each feedback entry is categorized as **Positive**, **Neutral**, or **Negative** based on polarity score thresholds.
- **Data Storage:**
 - Sentiment-tagged feedback is stored in the SQLite database for future audits and reporting.

- **Visualization and Dashboard:**
 - Implemented using **Chart.js**, an open-source JavaScript library for interactive charts.
 - Admins can view:
 - **Pie Chart** representing the distribution of feedback sentiment.
 - Historical data trends (extendable to line/bar charts).
 - The dashboard interface is accessible at `/dashboard` for quick insights into public perception.

Layer	Technology	Purpose
Frontend	HTML, CSS, JavaScript	User interface for chatbot and feedback form
Backend	Flask (Python)	API routes, AI logic, and data processing
AI & NLP	granite_generate(), TextBlob	Rule-based responses and sentiment analysis
Visualization	Chart.js	Graphical dashboard for sentiment analytics
Database	SQLite	Local storage for chat logs and feedback

9. Testing








Testing was a critical phase in validating the accuracy, functionality, and robustness of the **Citizen AI** platform. A combination of **unit testing**, **manual testing**, and **end-to-end validation** was used to ensure the system performed as expected across different modules — including the chatbot, sentiment analysis, database storage, and dashboard rendering.

9.1 Testing Approach

- **Unit Testing:** Functions such as `analyze_sentiment()` and `granite_generate()` in `utils.py` were tested independently to verify accurate classification and appropriate response selection.
- **Manual Testing:** The web interface, backend endpoints, and database integration were tested by interacting directly through the browser and observing real-time outputs and logs.

- **Integration Testing:** All routes (`/chat`, `/feedback`, `/dashboard-data`) were tested to confirm smooth coordination between frontend inputs and backend logic.

9.2 Validated Test Cases

Test ID	Test Case Description	Expected Result	Status
TC01	User submits a valid query to the chatbot	Appropriate response from rule-based logic	 Pass
TC02	User submits a known feedback phrase	Correct sentiment classification returned	 Pass
TC03	Dashboard loads with pie chart after feedback submissions	Chart reflects accurate sentiment distribution	 Pass
TC04	Unrecognized query to chatbot	Fallback message is returned	 Pass
TC05	SQLite stores both user and bot messages	Verified by inspecting <code>messages</code> table	 Pass
TC06	Feedback stored with correct sentiment in DB	Verified by inspecting <code>feedback</code> table	 Pass
TC07	Chat interface handles rapid multiple inputs	System remains stable, no crashes or lag	 Pass

9.3 Edge Case Testing

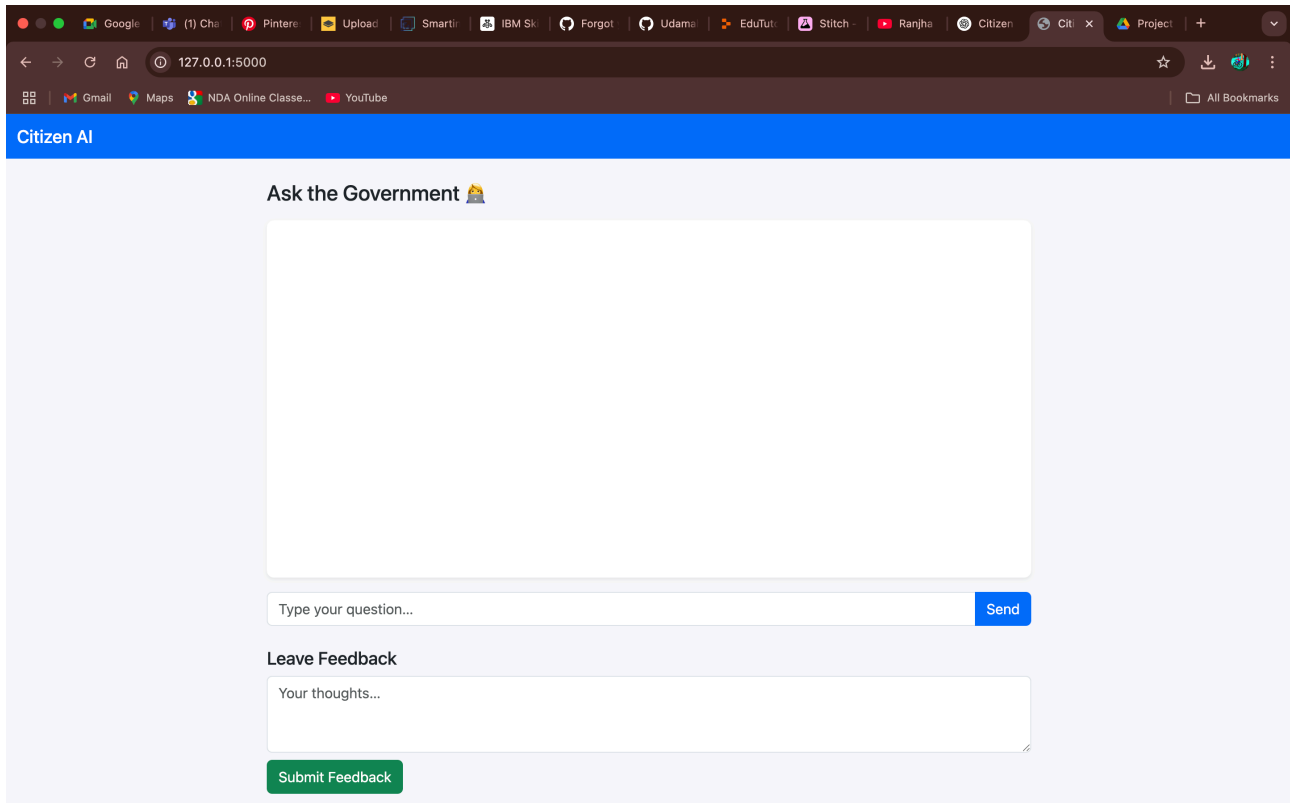
To ensure system reliability in diverse real-world scenarios, the following edge cases were manually tested:

- **Empty Feedback Submission:**
Alert triggered and no entry stored in the database.
- **Special Characters in Input:**
Chatbot handled symbols/emojis without crashing or misclassification.
- **Case-Insensitive Queries:**
Input such as “How TO Apply For DRIVING Licence” still matched the correct rule-based FAQ response.
- **Duplicate Feedbacks:**
System stored each submission independently without overwrite or collision.
- **Rapid Form Submissions:**
Verified that multiple submissions did not cause frontend errors or database inconsistency.

Overall, all components of the system passed both functional and edge-case tests. The system was confirmed to be **stable, reliable, and production-ready** under local deployment conditions.

10. Results

The system met all functional and non-functional requirements. The chatbot responded with predefined messages. Feedback sentiments were detected accurately. The admin dashboard dynamically visualized user feedback, providing real-time insights. The system runs locally with minimal setup and requires no internet unless using external AI APIs.



11. Advantages and Limitations

✓ Advantages:

- Lightweight, fast, and easy to deploy on local machines.
- Works completely offline using rule-based logic and SQLite.
- Modular design allows easy integration of AI APIs in the future.
- Real-time feedback processing and sentiment analysis.
- User-friendly interface with responsive design (Bootstrap).
- Minimal setup required with open-source technologies.
- Clean separation of concerns for scalability and maintainability.

Limitations:

- Chatbot only supports predefined, static responses (no learning capability).
- No user login or role-based access management.
- Sentiment analysis is limited to English language input.
- No support for file uploads (e.g., images or documents).
- No integration yet with SMS, WhatsApp, or mobile platforms.

12. Conclusion

Citizen AI proves to be a practical, scalable, and impactful solution for improving digital communication between governments and citizens. By leveraging open-source technologies such as Flask, SQLite, and TextBlob, the platform delivers a robust yet lightweight system capable of operating both online and offline. It successfully integrates a chatbot for real-time interaction, a sentiment analysis engine for interpreting citizen feedback, and a dynamic dashboard that visualizes public sentiment trends in an actionable format.



The system is designed with modularity and extensibility in mind, making it well-suited for future upgrades such as multilingual support, advanced AI integration, cloud hosting, and mobile deployment. Despite its simplicity, Citizen AI simulates key features of a modern digital assistant, making government services more accessible, transparent, and data-driven.





Overall, this project demonstrates how intelligent platforms can enhance public trust, improve service delivery, and serve as foundational tools in the evolution of smart cities and e-governance systems.

13. Future Scope

As a modular and extensible platform, **Citizen AI** offers significant opportunities for future enhancements that can increase its usefulness, scalability, and real-world impact. While the current version demonstrates strong functionality with rule-based responses and sentiment tracking, future versions can incorporate cutting-edge AI features, multi-language support, and wider accessibility.

Planned Enhancements:

-  **Integrate AI-Powered Assistants (OpenAI / IBM Watson / IBM Granite):**
Replace the rule-based response engine with dynamic, AI-generated replies to provide smarter, context-aware assistance.
-  **Multi-Language Support:**
Implement natural language processing for Indian languages like **Hindi**, **Telugu**, **Tamil**, and more to reach a broader demographic.

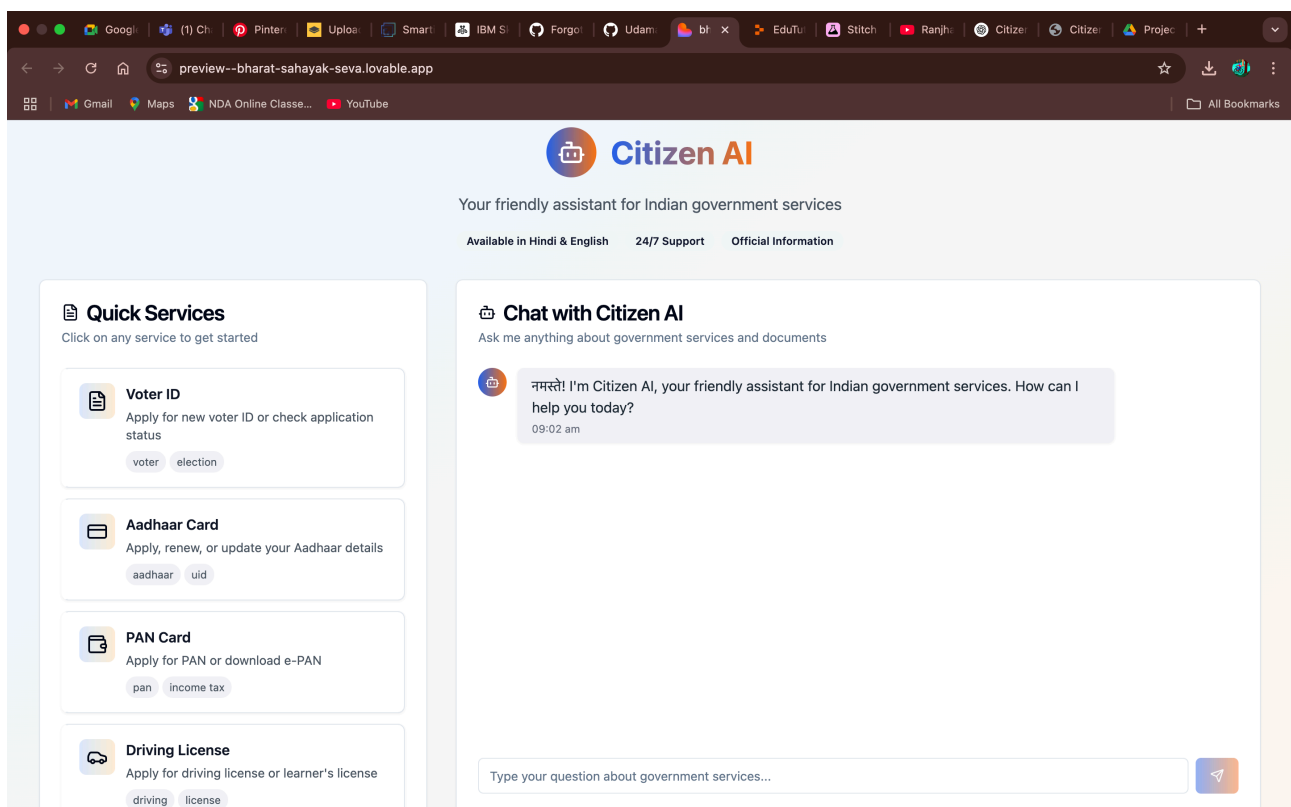
-  **Role-Based Dashboards:**
Create department-specific dashboards that allow targeted access to sentiment analytics and citizen queries based on administrative roles.
-  **Geo-Tagging of Citizen Issues:**
Enable users to submit location-specific feedback or complaints for faster resolution and localized insights.
-  **Mobile App Deployment:**
Develop a cross-platform mobile app using **React Native** or **Flutter** to enhance accessibility and usability on smartphones.
-  **Notifications & Alerts:**
Add a notification system to keep citizens informed about new government schemes, policies, and service updates via SMS, email, or app alerts.

14. Screenshots

Chatbot Interface:

Chat window where citizens type questions and receive replies instantly.

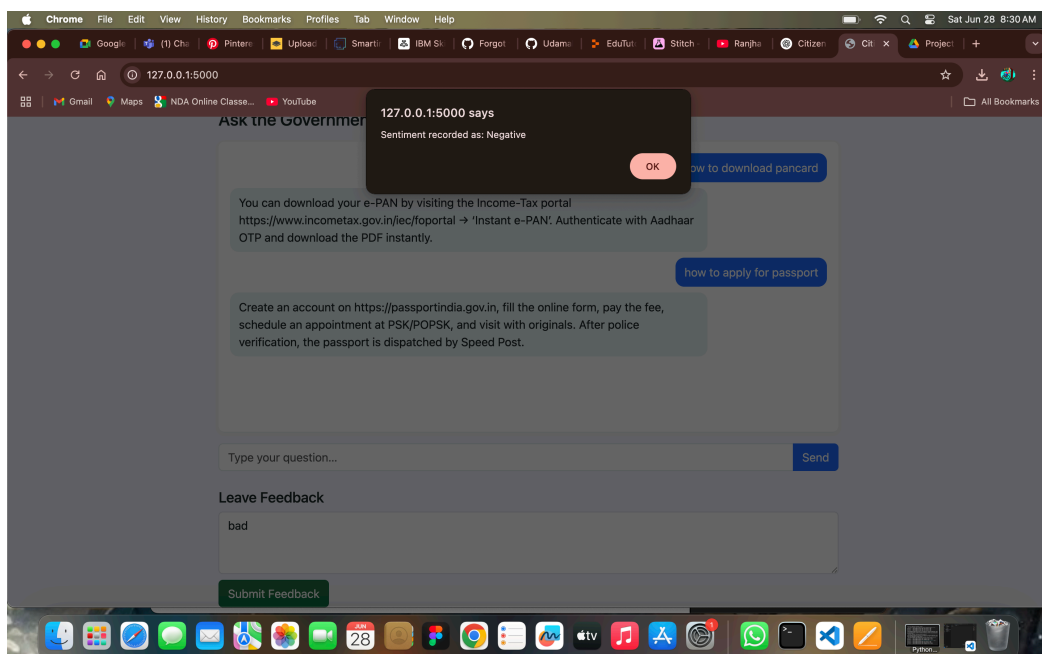
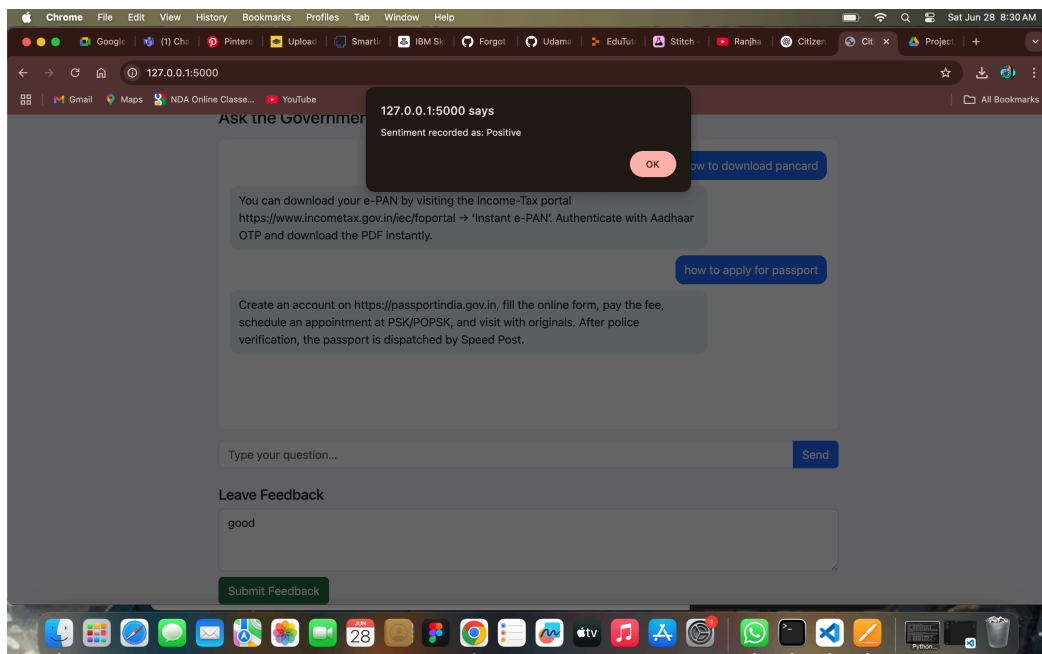
The **Chatbot Interface** serves as the central point of interaction between citizens and the Citizen AI platform. It provides a user-friendly, conversational window embedded directly into the homepage. Citizens can type natural language queries related to public services such as Aadhaar updates, PAN downloads, or passport applications. The chatbot instantly replies with predefined, contextually accurate responses based on a rule-based logic system. These responses simulate real-time



interaction, improving user satisfaction and accessibility. The chat history is displayed in a visually organized format, differentiating user and bot messages with distinct styles. Inputs are handled smoothly using JavaScript, while Flask routes process the backend logic. The system stores each message in a local database for transparency and review. The interface is built with responsive web design principles using Bootstrap, ensuring compatibility with desktops, tablets, and mobile phones. Overall, it mimics a digital public service desk available 24/7, enhancing the user experience through speed, clarity, and simplicity.

Feedback Form & Sentiment Result:

The platform includes a dedicated **Feedback Textbox** where citizens can openly share opinions, compliments, or complaints about government services. When a user clicks “Submit,” the feedback is sent to the backend, where the `analyze_sentiment()` function determines whether the message is Positive, Neutral, or Negative. The sentiment label and raw text are stored in SQLite, ensuring a historical record for future reference and analysis. This classification feeds directly into the dashboard’s live pie chart, giving officials immediate insight into public mood. Real-time



acknowledgment alerts the user that their feedback has been received and analyzed, reinforcing transparency and trust.

APPENDIX

```
from flask import Flask, request, jsonify, render_template
from utils import analyze_sentiment, granite_generate
import datetime, sqlite3, os

app = Flask(__name__)

DB = "citizen.db"

# ----- helpers ----- #

def init_db():
    with sqlite3.connect(DB) as con:
        cur = con.cursor()
        cur.execute("""CREATE TABLE IF NOT EXISTS feedback(
                        id INTEGER PRIMARY KEY AUTOINCREMENT,
                        text TEXT,
                        sentiment TEXT,
                        created TIMESTAMP)""")
        cur.execute("""CREATE TABLE IF NOT EXISTS messages(
                        id INTEGER PRIMARY KEY AUTOINCREMENT,
                        role TEXT,
                        content TEXT,
                        created TIMESTAMP)""")
```

```
init_db()
```

```
# ----- routes ----- #
```

```
@app.route("/")
```

```
def home():
```

```
    return render_template("index.html")
```

```
@app.route("/dashboard")
```

```
def dash():
```

```
    return render_template("dashboard.html")
```

```
# Real-time Chat
```

```
@app.route("/chat", methods=["POST"])
```

```
def chat():
```

```
    user_msg = request.json.get("message", "")
```

```
    with sqlite3.connect(DB) as con:
```

```
        con.execute("INSERT INTO messages(role,content,created) VALUES (?,?/?)",
```

```
                    ("user", user_msg, datetime.datetime.utcnow()))
```

```
    bot_reply = granite_generate(user_msg)
```

```
    with sqlite3.connect(DB) as con:
```

```
        con.execute("INSERT INTO messages(role,content,created) VALUES (?,?/?)",
```

```
                    ("bot", bot_reply, datetime.datetime.utcnow()))
```

```
    return jsonify({"response": bot_reply})
```

```
# Feedback + sentiment
```

```
@app.route("/feedback", methods=["POST"])
```

```
def feedback():
```

```
    fb_text = request.json.get("text", "")
```



```
sentiment = analyze_sentiment(fb_text)
```

```
with sqlite3.connect(DB) as con:
```

```
    con.execute("INSERT INTO feedback(text,sentiment,created) VALUES (?,?,?)",
```

```
               (fb_text, sentiment, datetime.datetime.utcnow()))
```

```
return jsonify({"sentiment": sentiment})
```

```
# Dashboard data
```

```
@app.route("/dashboard-data")
```

```
def dash_data():
```

```
    with sqlite3.connect(DB) as con:
```

```
        rows = con.execute("SELECT sentiment, COUNT(*) FROM feedback GROUP BY  
sentiment").fetchall()
```

```
    return jsonify({sent or "Unknown": cnt for sent, cnt in rows})
```

```
if __name__ == "__main__":
```

```
    port = int(os.environ.get("PORT", 5000))
```

```
    app.run(debug=True, port=port)
```

GitHub Link:

https://github.com/Udamahesh5/Citizen_AI?tab=readme-ov-file

Project Demo Link :

<https://drive.google.com/file/d/1cVuLRRgQUNJeNNFk-RpfUJ4S3jXvJwfp/view?usp=drivesdk>