Citizen AI – Final Report

1. INTRODUCTION

1.1 Project Overview

Citizen AI is an intelligent citizen engagement platform developed using Flask and IBM Granite models to streamline communication between governments and citizens. It provides real-time responses to queries regarding services, policies, and issues. By integrating AI with a user-friendly dashboard and sentiment analysis capabilities, the platform enhances service delivery, fosters trust, and supports data-driven governance.

1.2 Purpose

The primary purpose of Citizen AI is to facilitate seamless civic engagement, enhance transparency, and provide quick and intelligent responses to citizens' queries, ultimately leading to more responsive and efficient governance.

2. IDEATION PHASE

2.1 Problem Statement

Governments often struggle with managing large-scale citizen interactions effectively. There is a lack of timely, intelligent, and accessible systems for addressing queries, gathering feedback, and visualizing sentiment.

2.2 Empathy Map Canvas

- Says: "I want to report a civic issue or ask a question." - Thinks: "Is the government listening to me?" - Feels: Confused, unheard, hopeful.
- Does: Tries to find contact forms or call helpdesks.

2.3 Brainstorming

- AI-powered assistant for 24/7 interaction
- Sentiment analysis on feedback
- Real-time issue tracking
- Dynamic dashboard for policymakers
- Personalized responses using IBM Granite

3. REQUIREMENT ANALYSIS

3.1 Customer Journey Map

Citizen logs in → Submits query/feedback → AI processes input → Response displayed → Dashboard updates → Admins analyze data.

3.2 Solution Requirement

- Flask framework
- IBM Granite AI Model
- HTML/CSS frontend
- Sentiment analysis
- Interactive dashboard
- Concern reporting
- Secure login system

3.3 Data Flow Diagram

User Input → Flask Backend → AI Model → Result (Response/Sentiment) → HTML Display + Dashboard Update

3.4 Technology Stack

- Backend: Python, Flask
- AI/ML: IBM Granite, PyTorch, Transformers
- Frontend: HTML, CSS, Jinja2
- Libraries: Accelerate, BitsAndBytes

4. PROJECT DESIGN

4.1 Problem Solution Fit

Citizen AI provides a comprehensive solution by addressing key civic interaction challenges with automation, sentiment tracking, and real-time data insights.

4.2 Proposed Solution

An intelligent, AI-powered platform that offers conversational responses, feedback analysis, and dynamic visual insights through a web interface.

4.3 Solution Architecture

Flask routes
AI integration (IBM Granite)
In-memory data handling (future database integration)
Modular HTML templates
Rendered responses via Jinja2

5. PROJECT PLANNING & SCHEDULING

5.1 Project Planning

Milestone 1: AI Model & Architecture Setup
Milestone 2: Backend Functionalities
Milestone 3: Sentiment Analysis & Chat Logic
Milestone 4: Frontend Integration

Milestone 5: Testing & Debugging
Milestone 6: Deployment & Documentation

6. FUNCTIONAL AND PERFORMANCE TESTING

6.1 Performance Testing

Tested all Flask routes: OK
AI Model Response Time: Acceptable with GPU
Sentiment Analysis: Verified
Dashboard: Live updates confirmed

7. RESULTS

7.1 Output Screenshots

Figure 1: Login Page

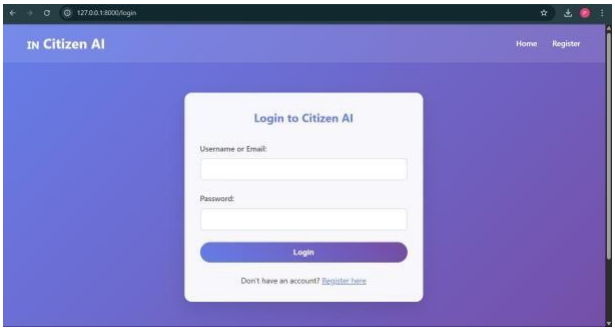


Figure 2: Chatbot Page

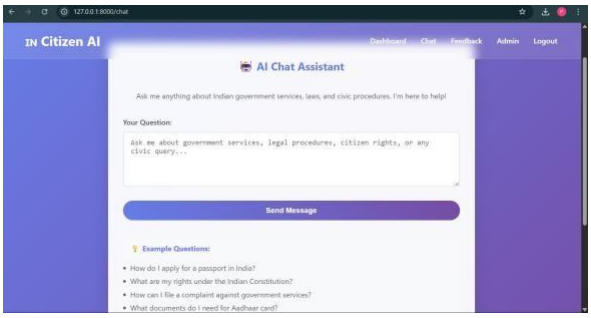


Figure 3: Feedback Page

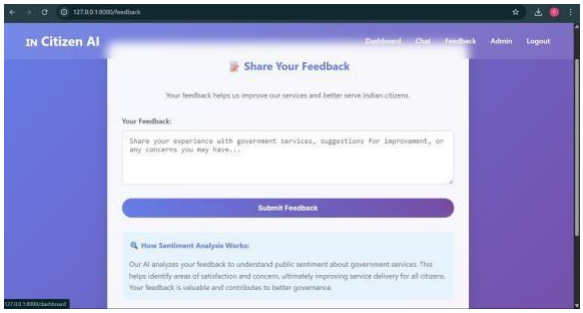


Figure 4: Dashboard - Welcome Screen

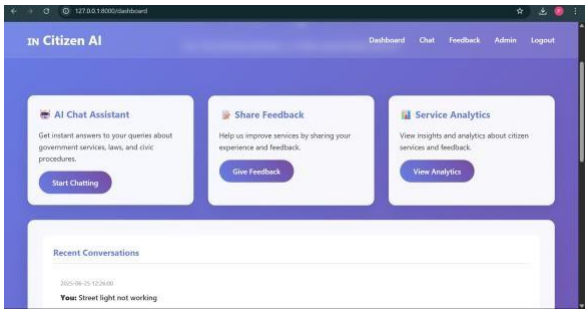


Figure 6: Registration Page

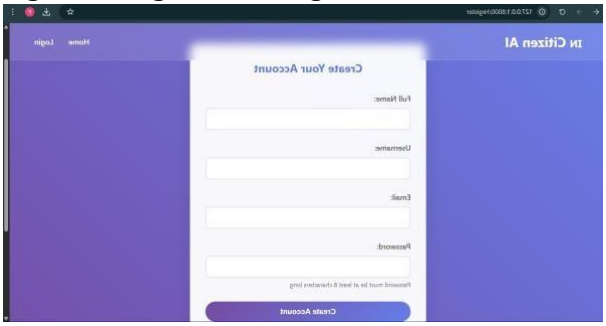


Figure 7: Home Page

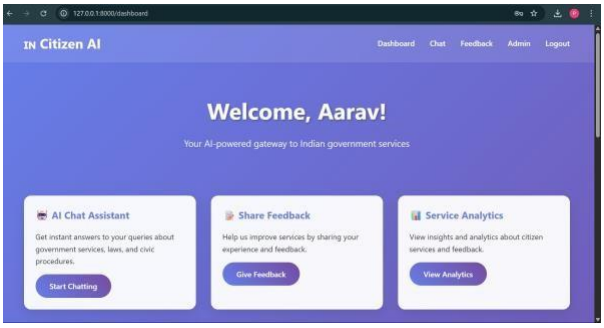


Figure 8: HTML Pages Code

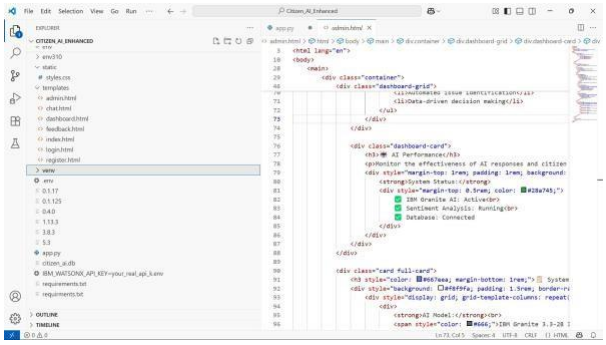


Figure 9: Python Backend Code

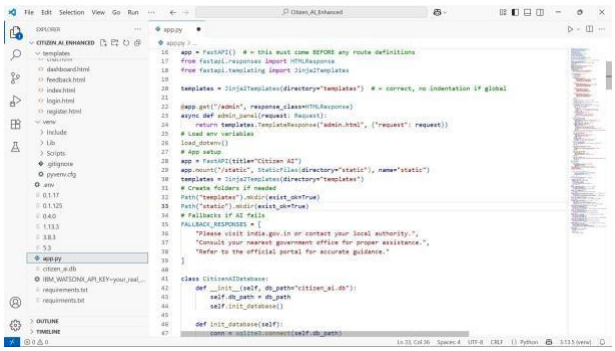


Figure 10: JavaScript or Other Scripts

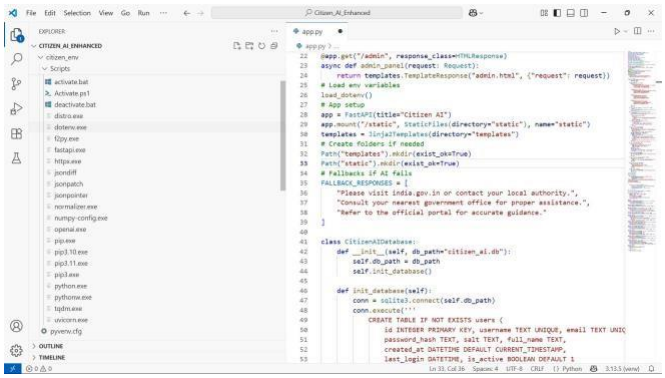


Figure 11: requirements.txt File

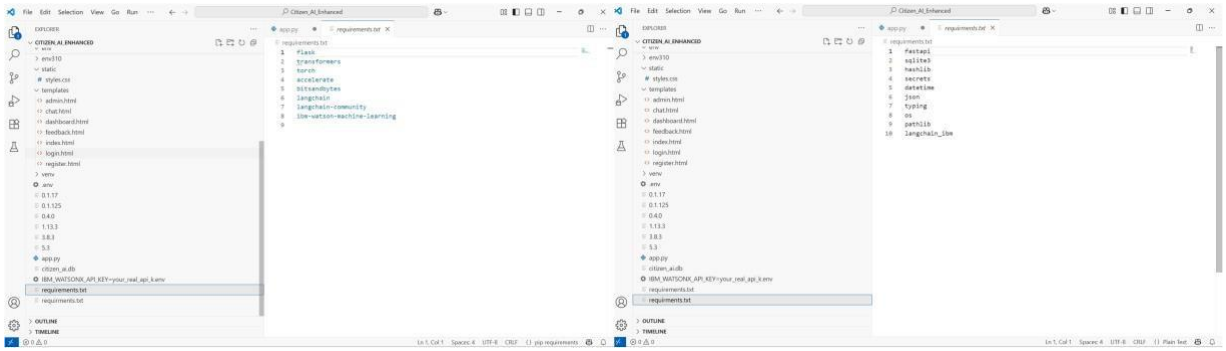
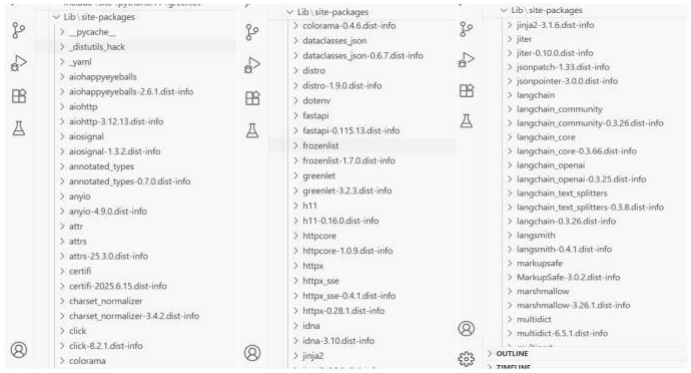


Figure 12: List of Libraries Used



8. ADVANTAGES & DISADVANTAGES

Advantages:
- Real-time civic support
- AI-driven personalization
- Actionable insights for government
- Visual sentiment data

Disadvantages:
- Heavy hardware requirement
- Initial model load time
- Data storage currently in-memory (not persistent)

9. CONCLUSION

Citizen AI bridges the gap between citizens and governance by providing a scalable, intelligent platform for real-time interaction, feedback analysis, and civic issue reporting.

10. FUTURE SCOPE

Integrate database for persistent storage
Multilingual support
Voice input and accessibility enhancements
Cloud deployment for broader accessibility

11. APPENDIX

Source Code:

```
# app.py – Main FastAPI Application

From fastapi import FastAPI, Request, Form, Depends, HTTPException, status, Cookie

From fastapi.responses import HTMLResponse, RedirectResponse

From fastapi.templating import Jinja2Templates

From fastapi.staticfiles import StaticFiles

Import sqlite3

Import hashlib

Import secrets

Import datetime

Import json
```


```
From typing import Optional, Dict
Import os
From pathlib import Path
From langchain.prompts import PromptTemplate
From langchain.chains import LLMChain
From langchain.llms.base import LLM
From typing import Any, List
From langchain_ibm import WatsonxLLM

# Load environment variables from .env file
Try:
   From dotenv import load_dotenv
   Load_dotenv()
Except ImportError:
   Print("Warning: python-dotenv not installed. Install it with: pip install python-dotenv")
Print("Make sure to set environment variables manually.")

# Create directories if they don't exist
Path("templates").mkdir(exist_ok=True)
Path("static").mkdir(exist_ok=True)

App = FastAPI(title="Citizen AI", description="AI-powered citizen services platform")

# Mount static files and templates
App.mount("/static", StaticFiles(directory="static"), name="static")
Templates = Jinja2Templates(directory="templates")
```


```
# Fallback responses when AI is not available

FALLBACK_RESPONSES = [

   “I understand your query about Indian government services. For the most accurate and up-to-date information, I recommend visiting the official government portal at india.gov.in or contacting your nearest government office.”,

   “Thank you for your question about citizen services. For specific legal or procedural guidance, please visit the official government website or contact your local government office for assistance.”,

   “I appreciate your inquiry about Indian government services. For detailed information and official procedures, please refer to the relevant government department’s official website or visit your nearest government office.”,

   “Your question about government services is important. For official guidance and procedures, I recommend checking the official government portal or contacting the appropriate government department directly.”,

   “Thank you for reaching out about government services. For the most current and accurate information, please visit the official government website or contact your local government office.”

]


Class CitizenAIDatabase:

   Def __init__(self, db_path=”citizen_ai.db”):

      Self.db_path = db_path

      Self.init_database()


   Def init_database(self):

      “””Initialize all required tables”””

      Conn = sqlite3.connect(self.db_path)


      # Users table

      Conn.execute(‘’’

         CREATE TABLE IF NOT EXISTS users (
```


```
        Id INTEGER PRIMARY KEY AUTOINCREMENT,
        Username TEXT UNIQUE NOT NULL,
        Email TEXT UNIQUE NOT NULL,
        Password_hash TEXT NOT NULL,
        Salt TEXT NOT NULL,
        Full_name TEXT NOT NULL,
        Created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
        Last_login DATETIME,
        Is_active BOOLEAN DEFAULT 1
    )
''')

# Sessions table
Conn.execute('''
    CREATE TABLE IF NOT EXISTS sessions (
        Id INTEGER PRIMARY KEY AUTOINCREMENT,
        User_id INTEGER,
        Session_token TEXT UNIQUE NOT NULL,
        Created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
        Expires_at DATETIME NOT NULL,
        Is_active BOOLEAN DEFAULT 1,
        FOREIGN KEY (user_id) REFERENCES users (id)
    )
''')

# Chat history table
```

```
Conn.execute('''
   CREATE TABLE IF NOT EXISTS chat_history (
      Id INTEGER PRIMARY KEY AUTOINCREMENT,
      User_id INTEGER,
      User_message TEXT NOT NULL,
      Ai_response TEXT NOT NULL,
      Timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
      FOREIGN KEY (user_id) REFERENCES users (id)
   )
''')

# Sentiment analysis table
Conn.execute('''
   CREATE TABLE IF NOT EXISTS sentiment_analysis (
      Id INTEGER PRIMARY KEY AUTOINCREMENT,
      User_id INTEGER,
      Feedback_text TEXT NOT NULL,
      Sentiment TEXT NOT NULL,
      Confidence REAL,
      Timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
      FOREIGN KEY (user_id) REFERENCES users (id)
   )
''')

Conn.commit()
Conn.close()
```


```
Def _hash_password(self, password: str, salt: Optional[str] = None) -> tuple:
    """Hash password with salt"""
If salt is None:
        Salt = secrets.token_hex(32)

    Password_hash = hashlib.pbkdf2_hmac(
        'sha256',
        Password.encode('utf-8'),
        Salt.encode('utf-8'),
        100000
    )
    Return password_hash.hex(), salt

Def register_user(self, username: str, email: str, password: str, full_name: str) -> Dict:
    """Register a new user"""
    If len(password) < 8:
        Return {"success": False, "message": "Password must be at least 8 characters"}

    Password_hash, salt = self._hash_password(password)

    Try:
        Conn = sqlite3.connect(self.db_path)
        Cursor = conn.cursor()
        Cursor.execute('''
            INSERT INTO users (username, email, password_hash, salt, full_name)
            VALUES (?, ?, ?, ?, ?)
```


```
    ‘’’, (username, email, password_hash, salt, full_name))
    Conn.commit()
    User_id = cursor.lastrowid
    Conn.close()

    Return {“success”: True, “message”: “Registration successful”, “user_id”: user_id}

  Except sqlite3.IntegrityError as e:
    If “username” in str€:
      Return {“success”: False, “message”: “Username already exists”}
    Elif “email” in str€:
      Return {“success”: False, “message”: “Email already exists”}
    Else:
      Return {“success”: False, “message”: “Registration failed”}

Def login_user(self, username: str, password: str) -> Dict:
  “””Authenticate user and create session”””
  Conn = sqlite3.connect(self.db_path)

  User_data = conn.execute(‘’’
    SELECT id, username, password_hash, salt, is_active, full_name
    FROM users WHERE username = ? OR email = ?
  ‘’’, (username, username)).fetchone()

  If not user_data or not user_data[4]:
    Conn.close()
```


```
    Return {"success": False, "message": "Invalid credentials"}

User_id, db_username, stored_hash, salt, is_active, full_name = user_data

Input_hash, _ = self._hash_password(password, salt)

If input_hash == stored_hash:
    Session_token = secrets.token_urlsafe(32)
    Expires_at = datetime.datetime.now() + datetime.timedelta(days=7)

    Conn.execute('''
        INSERT INTO sessions (user_id, session_token, expires_at)
        VALUES (?, ?, ?)
    ''', (user_id, session_token, expires_at))

    Conn.execute('''
        UPDATE users SET last_login = CURRENT_TIMESTAMP WHERE id = ?
    ''', (user_id,))

    Conn.commit()
    Conn.close()

    Return {
        "success": True,
        "session_token": session_token,
        "user_id": user_id,
```


```
            “username”: db_username,
            “full_name”: full_name
        }
    Else:
        Conn.close()
        Return {“success”: False, “message”: “Invalid credentials”}

Def verify_session(self, session_token: str) -> Optional[Dict]:
    “””Verify session token”””
    If not session_token:
        Return None

    Conn = sqlite3.connect(self.db_path)

    Result = conn.execute(‘’’
        SELECT s.user_id, u.username, s.expires_at, u.is_active, u.full_name
        FROM sessions s
        JOIN users u ON s.user_id = u.id
        WHERE s.session_token = ? AND s.is_active = 1
    ‘’’, (session_token,)).fetchone()

    Conn.close()

    If not result:
        Return None
```

GitHub & Project Demo Link: