

Citizen AI – Final Report

# INTRODUCTION

## 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.

## 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.

# IDEATION PHASE

## 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.

## 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.

## 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

# REQUIREMENT ANALYSIS

## Customer Journey Map

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

## Solution Requirement

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

## Data Flow Diagram

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

## Technology Stack

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

# PROJECT DESIGN

## Problem Solution Fit

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

## Proposed Solution

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

## Solution Architecture

Flask routes

AI integration (IBM Granite)

In-memory data handling (future database integration) Modular HTML templates

Rendered responses via Jinja2

# PROJECT PLANNING & SCHEDULING

## 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

# FUNCTIONAL AND PERFORMANCE TESTING

## Performance Testing

Tested all Flask routes: OK

AI Model Response Time: Acceptable with GPU Sentiment Analysis: Verified

Dashboard: Live updates confirmed

# RESULTS

## Output Screenshots

Figure 1: Login Page

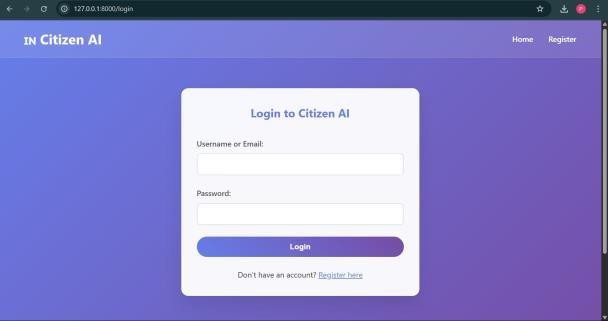


Figure 2: Chatbot Page

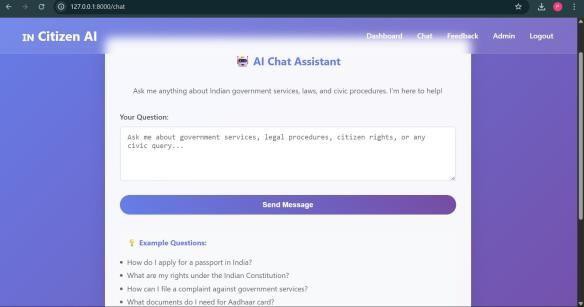


Figure 3: Feedback Page

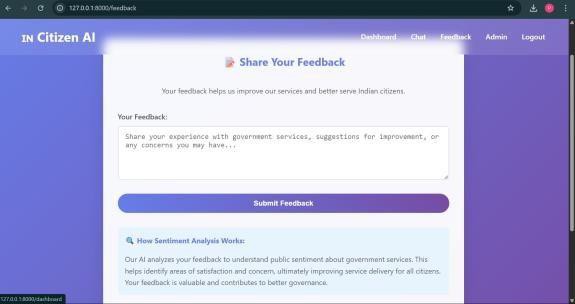
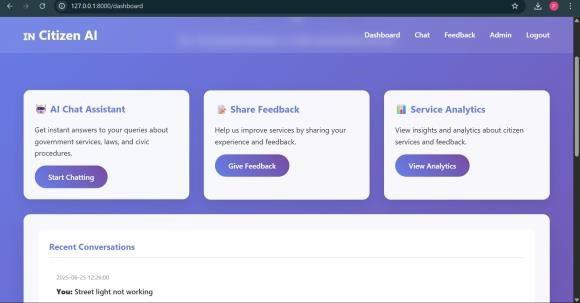


Figure 4: Dashboard - Welcome Screen



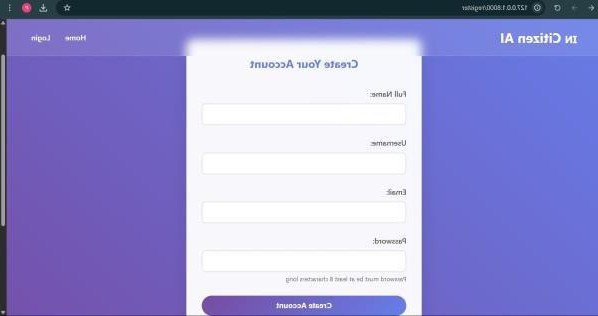
Figure 6: Registration Page

Figure 7: Home Page

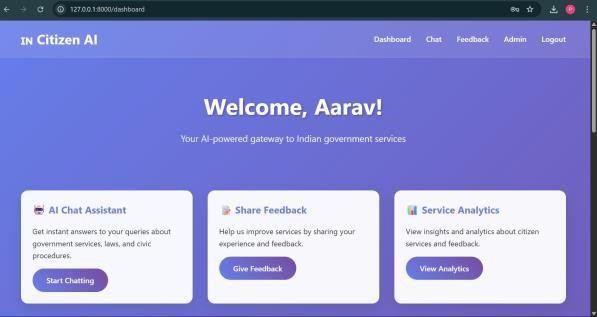


Figure 8: HTML Pages Code

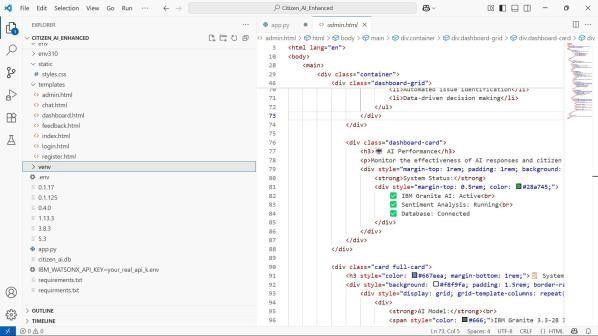


Figure 9: Python Backend Code

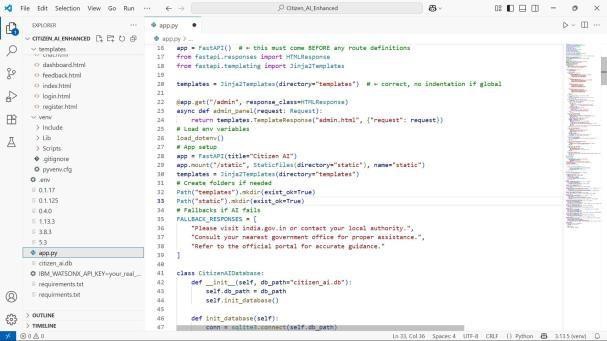


Figure 10: JavaScript or Other Scripts

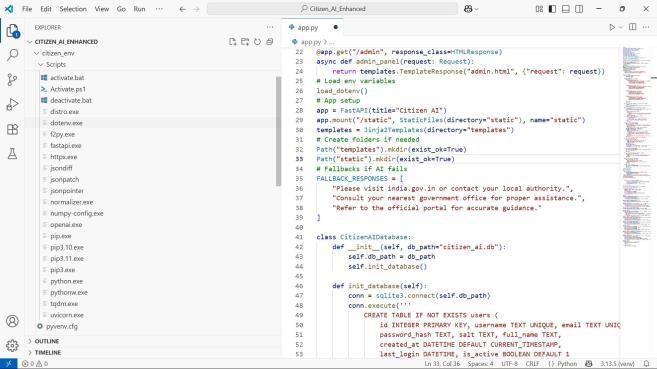


Figure 11: requirements.txt File

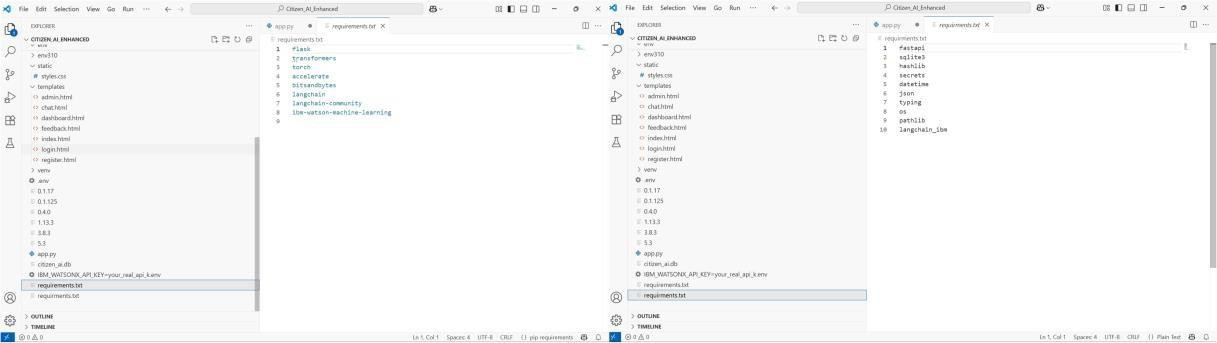
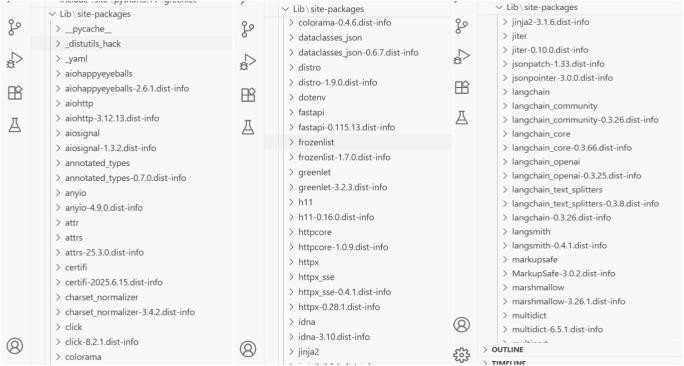


Figure 12: List of Libraries Used



# 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)

# 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.

# FUTURE SCOPE

Integrate database for persistent storage Multilingual support

Voice input and accessibility enhancements Cloud deployment for broader accessibility

# 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 : https://github.com/Praveen-Kumar-Madem/CitizenAI**

**Project Demo Link : https://drive.google.com/file/d/18F0uRFsjgWcFxO26THv8FEEI2lDfcBP5/view?usp=sharing**