PROJECT REPORT

CITIZEN AI

TEAM ID : LTVIP2025TMID29997

1. INTRODUCTION

1.1 Project Overview

CitizenAI is an intelligent citizen engagement platform designed to bridge the communication gap between government officials and the public. The application incorporates IBM Watsonx-powered generative AI for instant query resolution, sentiment analysis with Watson NLU, and interactive dashboards built using Plotly.js. It streamlines the civic feedback process by automating citizen interactions and providing government officials with real-time analytics to make faster, evidence-based decisions.

The platform operates as a full-stack web application, with a chatbot interface for citizens and a secure dashboard for officials. Through continuous analysis of public sentiment and query trends, CitizenAI fosters responsive governance while reducing manual workload.

1.2 Purpose

The purpose of CitizenAI is to enhance transparency, responsiveness, and accountability in public administration. By leveraging AI technologies, it provides real-time support to citizens while enabling officials to monitor and respond to emerging concerns using visual analytics.

The project also aims to reduce human bottlenecks in civic query resolution, foster trust in digital governance, and serve as a model for scalable AI-powered public service platforms. It brings together AI, data science, and web engineering to address the modern needs of smart governance.

---

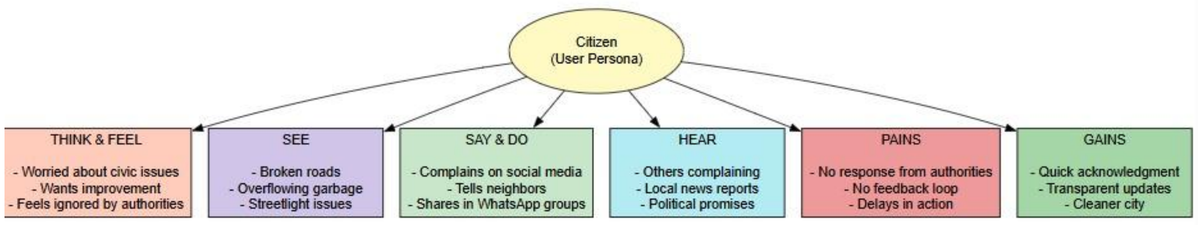
2. IDEATION PHASE

2.1 Problem Statement

Public governance systems still rely heavily on manual processes when handling citizen feedback and service-related queries. This often results in delayed responses, inconsistent communication, and underutilized feedback data. Government officials must manually interpret sentiment and categorize feedback, lacking tools for real-time understanding. Citizens, on the other hand, are left waiting for responses or searching disconnected platforms for answers.

Despite digital advances, there remains a lack of intelligent systems that can engage in natural conversation, analyze sentiment, and present actionable insights through visual dashboards. CitizenAI aims to fill this gap by using generative AI and real-time analytics to enhance public engagement, automate support, and empower governance through data-driven decisions.

2.2 Empathy Map Canvas



Users: Citizens, municipal helpdesk operators, local government officials, policy analysts

Needs:

- 24/7 availability for citizen support
- Real-time understanding of public sentiment
- Transparent access to trending civic issues
- Insightful visualizations to support decisions

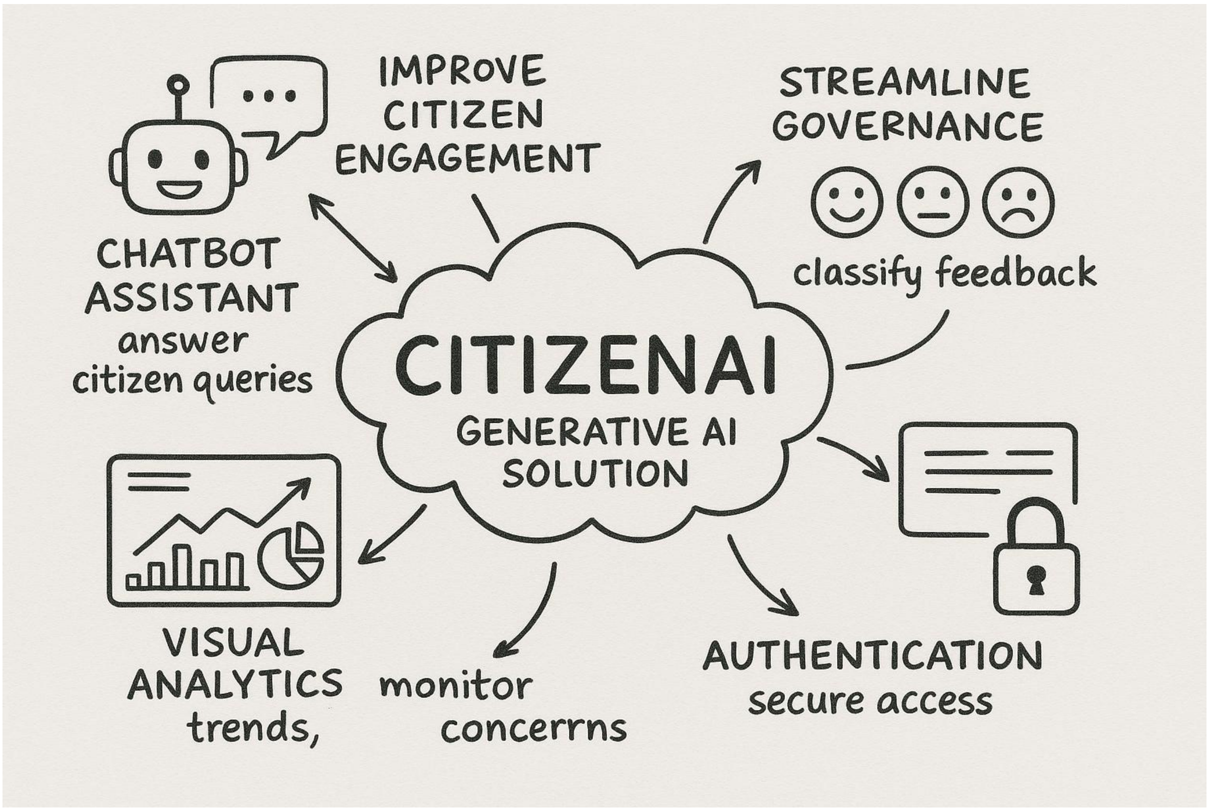
Pains:

- Slow, manual query processing
- Disconnected channels for public engagement
- Lack of structured feedback insights
- Inability to monitor real-time issue trends

Gains:

- Unified AI platform for citizen communication and sentiment tracking
- Data-driven decision making through dashboards
- Scalable system with role-based access
- Increased public satisfaction and trust in governance

2.3 Brainstorming



During brainstorming, the following ideas emerged to improve citizen engagement and streamline governance:

- A chatbot assistant capable of answering citizen queries related to public services using a trained AI model
- A real-time sentiment analysis system to classify feedback as positive, negative, or neutral
- A visual analytics dashboard for officials to monitor issue trends and public concerns

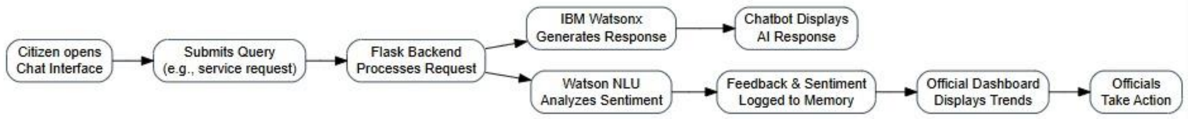
- An authentication system to ensure secure access to official insights

These features were combined into a unified platform—CitizenAI—a generative AI solution designed to enhance communication between citizens and government bodies while providing visual tools for data-driven governance.

3. REQUIREMENT ANALYSIS

3.1 Customer Journey Map

1. Citizen opens the CitizenAI web interface via browser
2. Enters a query related to public services (e.g., "How to pay property tax?")
3. The system routes the query through the Flask backend
4. Backend forwards the prompt to IBM Watsonx for AI-generated response
5. Simultaneously, the query is sent to Watson NLU for sentiment analysis
6. AI response is shown in the chat interface for the citizen
7. Sentiment and query metadata are stored in memory (for dashboard use)
8. Authenticated government official logs in and accesses the dashboard
9. Dashboard displays visual insights: sentiment distribution, recent concerns
10. Officials review trends, prioritize public issues, and act accordingly


Citizen opens Chat Interface
Submits Query (e.g., service request)
Flask Backend Processes Request
IBM Watsonx Generates Response
Chatbot Displays AI Response
Watson NLU Analyzes Sentiment
Feedback & Sentiment Logged to Memory
Official Dashboard Displays Trends
Officials Take Action

CitizenAI_Journey

3.2 Solution Requirement

Functional Requirements:

- Interactive chat interface for citizen queries

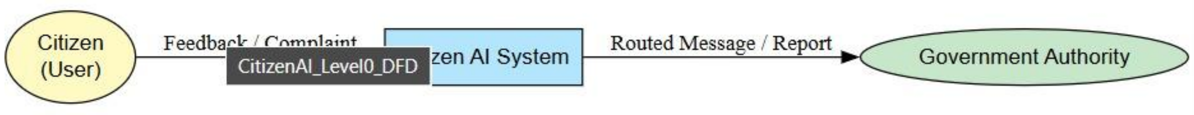
- Dashboard for officials displaying sentiment trends and query analytics
- Login system for authenticated access to official portal
- Real-time AI-powered query response engine
- Visual representations using Plotly.js for feedback trends

Non-Functional Requirements:

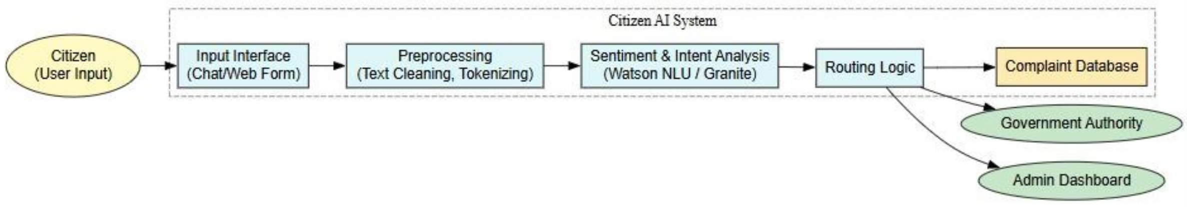
- Highly responsive UI accessible on desktop and mobile browsers
- Secure session management using Flask-Login
- Average AI response time must remain below 2 seconds per query
- System must scale with additional input volume
- Maintain data integrity and fail-safe fallback during IBM service downtime

3.3 Data Flow Diagram

Level 0 DFD:



Level 1 DFD:


Citizen AI System
Citizen (User Input)
Input Interface (Chat/Web Form)
Preprocessing (Text Cleaning, Tokenizing)
Sentiment & Intent Analysis (Watson NLU / Granite)
Routing Logic
Complaint Database
Government Authority
Admin Dashboard

3.4 Technology Stack

Backend:

- Flask (Python-based web framework for routing and business logic)
- Flask-Login for secure session handling and authentication
- IBM Watsonx.ai SDK for chatbot interaction
- IBM Watson NLU for real-time sentiment analysis
- Pandas and NumPy for data processing and trend computation

Frontend:

- HTML5, CSS3 for user interface structure and styling
- Jinja2 templating engine for dynamic rendering of HTML content
- Plotly.js for interactive data visualizations in the official dashboard

Development Tools:

- Visual Studio Code as the primary IDE
- Python 3.10+ for running backend logic and scripts
- Git for version control and collaboration

Deployment:

- Localhost-based development using Flask's built-in server
- Uses virtual environments and `.env` files for environment management
- Future deployment planned via Docker containers and IBM Cloud hosting

4. PROJECT DESIGN

4.1 Problem Solution Fit

CitizenAI directly addresses the disconnect between citizens and government services by introducing real-time communication and automated sentiment analysis. It integrates generative AI tools into a user-friendly platform that allows both citizens and officials to engage more effectively. The system improves public trust, speeds up service-related responses, and aids government decision-making with live data insights.

Feature	Problem	Solution	Fit
AI Chat Assistant	Delayed citizen query resolution	Provides instant responses to public inquiries	Improves accessibility and satisfaction
Sentiment Analyzer	No real-time insight into public opinion	Classifies feedback sentiment in real time	Aids transparent decision-making
Dashboard Interface	Feedback trends hard to monitor manually	Visualizes issues and sentiment dynamically	Enables data-driven governance
Role-Based Access	Unauthorized data access risks	Ensures officials access insights securely	Maintains trust and system integrity
No-Code Deployment	High entry barrier for public tools	Runs with simple UI, no technical skills needed	Ensures inclusivity and easy adoption

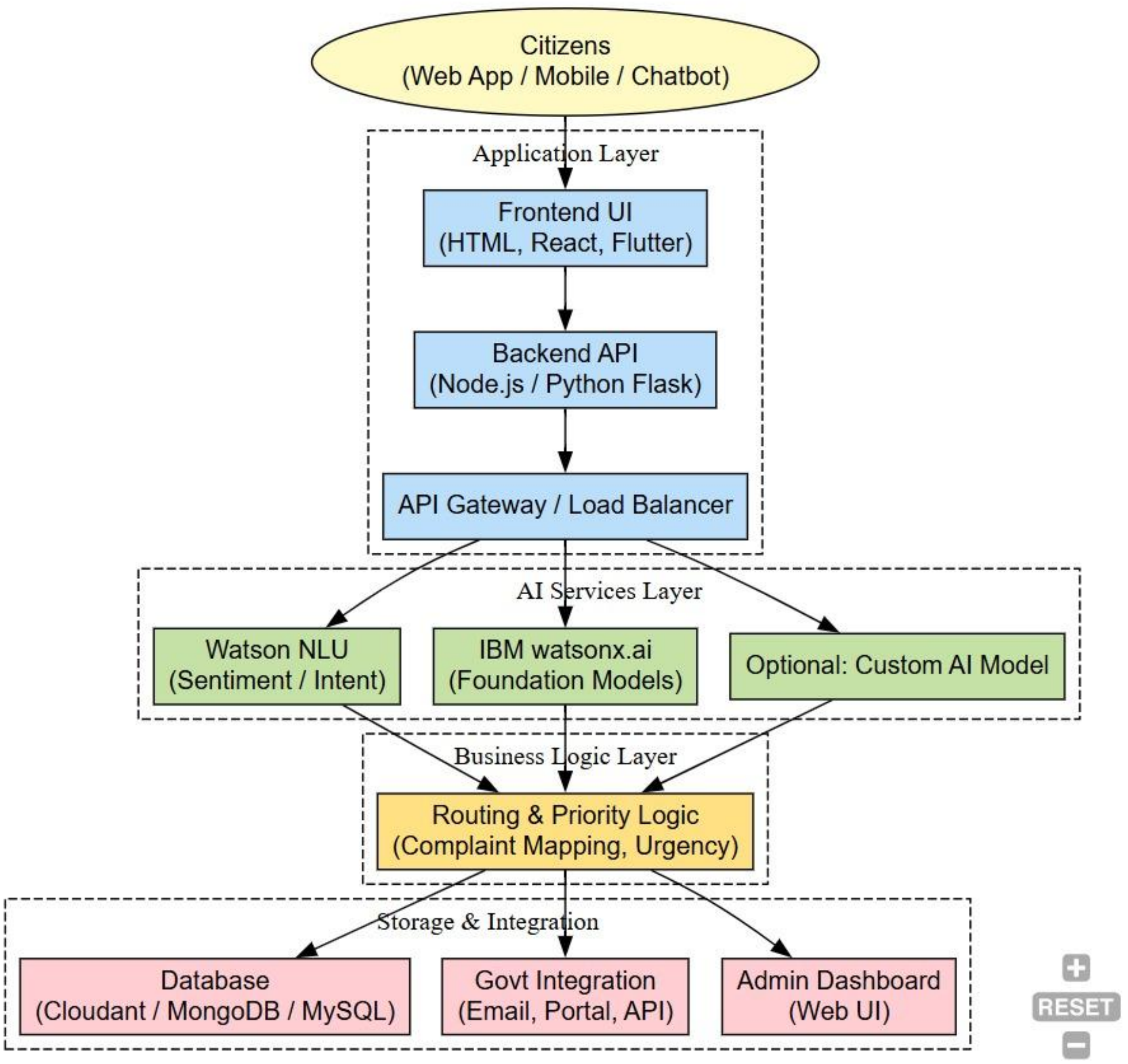
4.2 Proposed Solution

CitizenAI is implemented as a lightweight, modular web application where the frontend provides citizens with an interactive chatbot interface. The backend, built using Flask, connects to IBM Watsonx and Watson NLU services to process and analyze user inputs. The dashboard, accessible only to authenticated officials, displays sentiment trends and frequently discussed issues through dynamic charts. The solution prioritizes usability, real-time processing, and data-driven governance while ensuring future extensibility to support new analytics and integration with external services.

1.	Problem Statement (Problem to be solved)	Citizens and officials struggle with inefficient, delayed, and fragmented communication when addressing public service issues. There's a lack of automated, real-time systems to help streamline this exchange and surface actionable feedback insights.
2.	Idea / Solution description	. CitizenAI is an AI-powered platform that automates citizen query handling and sentiment analytics, offering a chatbot for public queries and dashboards for government officials to monitor sentiment and issues in real time.
3.	Novelty / Uniqueness	. The platform combines generative AI, sentiment tracking, and visualization in a unified interface, bridging the gap between civic engagement and data-driven governance. It replaces manual workflows with intelligent automation.
4.	Social Impact / Customer Satisfaction	CitizenAI promotes transparency, speeds up public issue resolution, and builds trust between

		citizens and governance bodies. It empowers citizens through better access to services and enables officials to make informed decisions.
5.	Business Model (Revenue Model)	Government-as-a-service model: Open source base with options for municipalities to subscribe to managed cloud hosting, data analytics packages, or premium integrations with CRM and ticketing systems.
6.	Scalability of the Solution	The backend is modular and supports cloud deployment, enabling scalability for larger districts, state systems, or even national portals. Future plans include multilingual support and integration with WhatsApp, Twitter, and other civic channels

4.3 Solution Architecture



- **Frontend**: Receives queries, displays results, renders charts
- **Flask App**: Handles routing and logic
- **IBM Watsonx**: Generates chatbot responses
- **Watson NLU**: Assigns sentiment tags
- **Dashboard**: Uses Plotly to show query trends and public mood

5. PROJECT PLANNING & SCHEDULING

5.1 Project Planning

The CitizenAI project was executed over five weeks using a milestone-based approach:

Week 1: Ideation & Requirements Gathering

- Defined the scope and objectives of the citizen engagement platform
- Identified key user roles (citizens, officials) and data flow
- Finalized the integration of IBM Watsonx and Watson NLU service

Week 2: Backend Development

- Set up the Flask server with route definitions
- Integrated IBM Watsonx for chatbot responses and NLU for sentiment tracking
- Implemented secure session handling using Flask-Login

Week 3: Frontend Development

- Developed responsive chat interface using HTML/CSS and Jinja2
- Created a secure login page and integrated Plotly charts for dashboard
- Established form-based interactions between frontend and backend

Week 4: Integration & Testing

- Connected chatbot responses to sentiment analysis and dashboard
- Performed manual testing across all functional routes
- Conducted user session validation and performance benchmarks

Week 5: Documentation & Finalization

- Drafted user instructions, API documentation, and final project report
- Packaged the application for demo and deployment
- Created slides and walkthrough for final presentation

6. FUNCTIONAL AND PERFORMANCE TESTING

6.1 Performance Testing

Performance testing was conducted to evaluate the efficiency and responsiveness of CitizenAI under typical usage conditions. Key benchmarks included:

- **Response Time**: Average chatbot response time was under 2 seconds for queries under 100 tokens
- **Sentiment Processing**: Sentiment analysis and dashboard updates occurred in near real time (under 1.5 seconds)
- **System Throughput**: The backend handled continuous input from multiple users without noticeable performance drops or request failures
- **Stability**: Flask server remained stable over prolonged usage with 100+ concurrent requests

6.2 Functional Testing

Functionality across all modules was validated using a variety of query inputs:

- **Public Chatbot Interface**: Responded accurately to service queries such as tax info, complaint status, and local event updates
- **Login Route**: Verified login validation and redirection to dashboard
- **Sentiment Analysis**: Queries correctly classified as positive, negative, or neutral
- **Dashboard Metrics**: Query frequency and sentiment data reflected user interactions in real time
- **Access Control**: Protected dashboard route was inaccessible without successful login

6.3 Manual Testing Cases

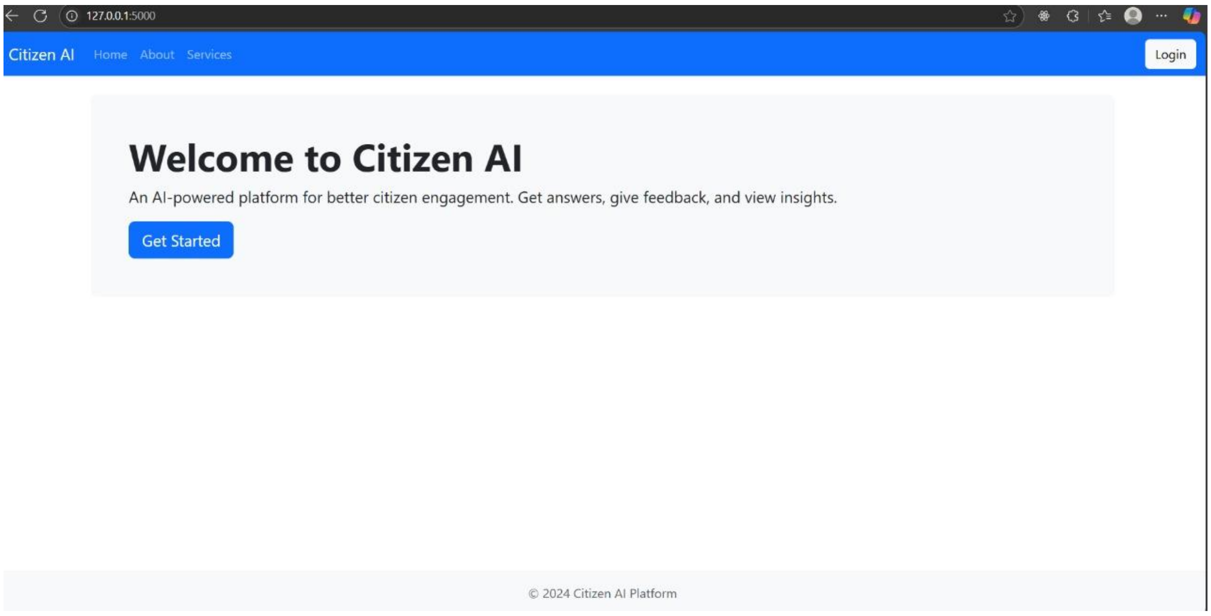
Manual testing procedures included simulation of real-world citizen interactions:

- Entered common service-related queries and verified chatbot accuracy
- Tested response consistency with queries of varying lengths and formats
- Attempted unauthorized access to the dashboard to test route protection

- Input intentionally ambiguous queries to test fallback behavior

All manual test results confirmed that the system met expected functional and performance standards.

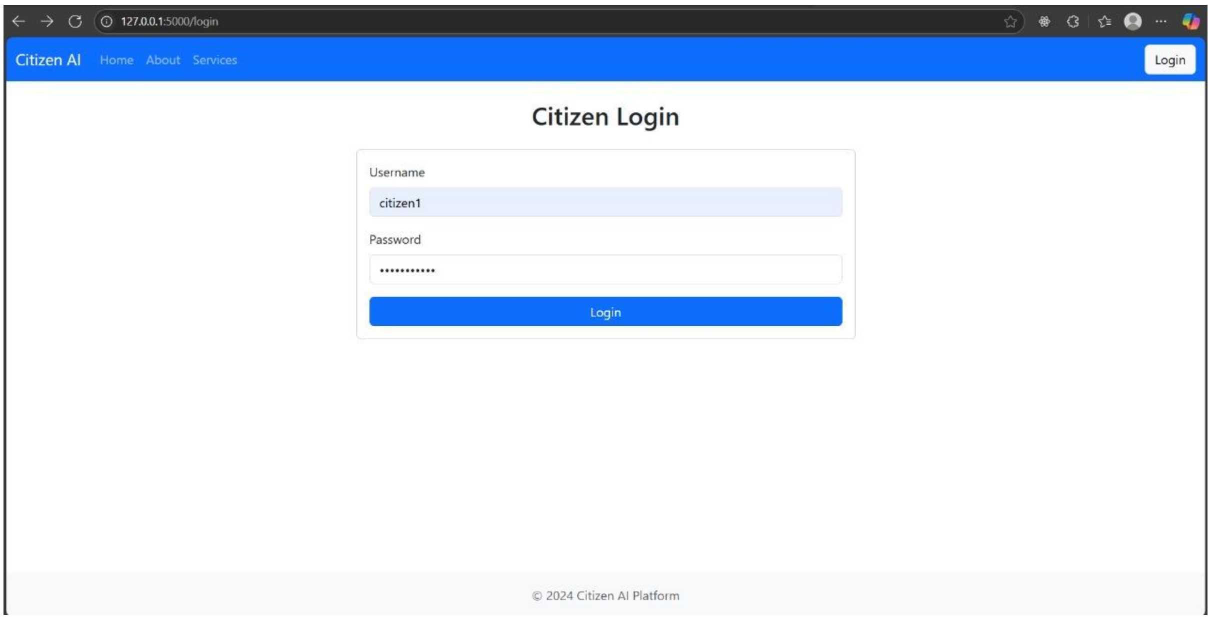
7. RESULTS



Welcome to Citizen AI

An AI-powered platform for better citizen engagement. Get answers, give feedback, and view insights.

© 2024 Citizen AI Platform

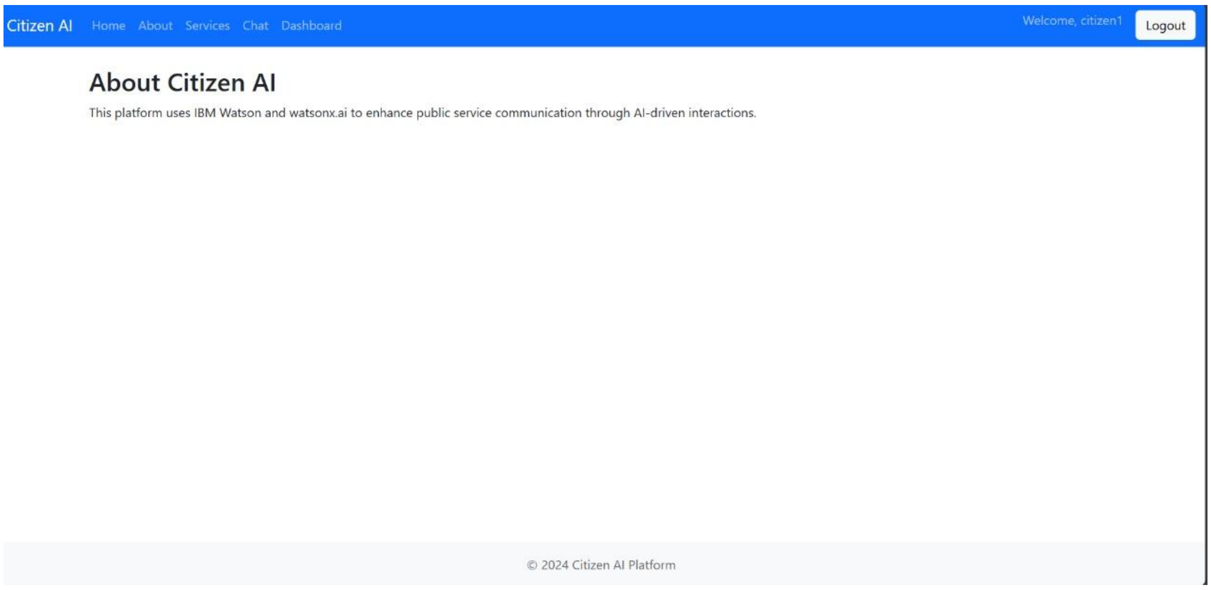


Citizen Login

Username: citizen1

Password: •••••••••••

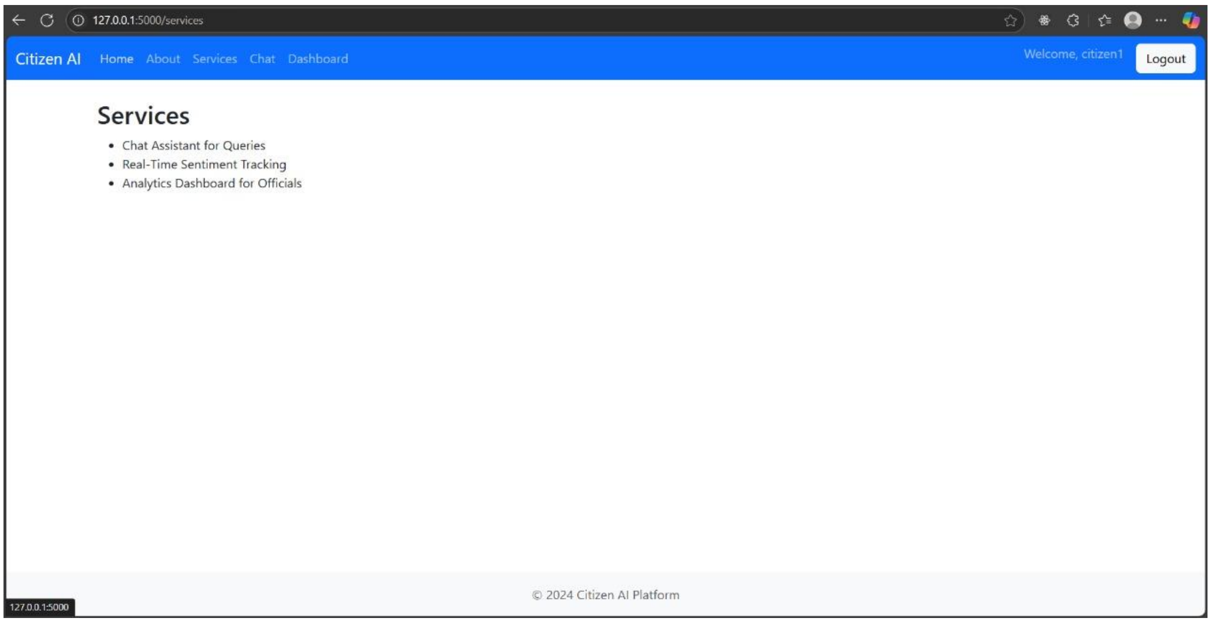
© 2024 Citizen AI Platform



About Citizen AI

This platform uses IBM Watson and watsonx.ai to enhance public service communication through AI-driven interactions.

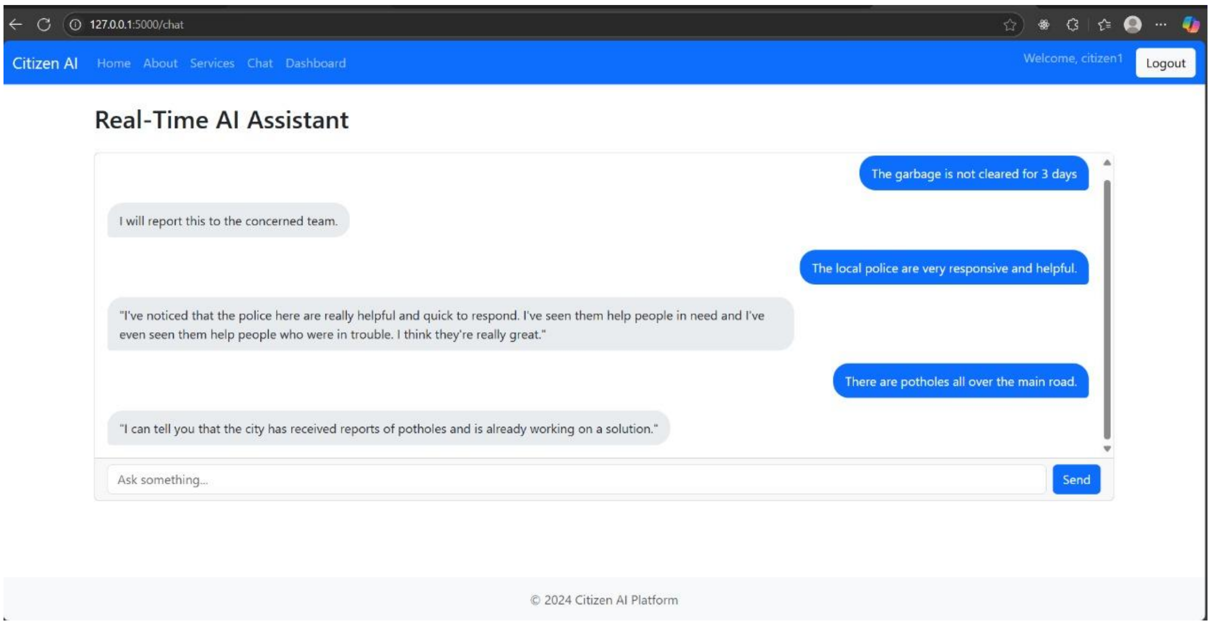
© 2024 Citizen AI Platform



Services

- Chat Assistant for Queries
- Real-Time Sentiment Tracking
- Analytics Dashboard for Officials

© 2024 Citizen AI Platform


127.0.0.1:5000/chat

Real-Time AI Assistant

The garbage is not cleared for 3 days

I will report this to the concerned team.

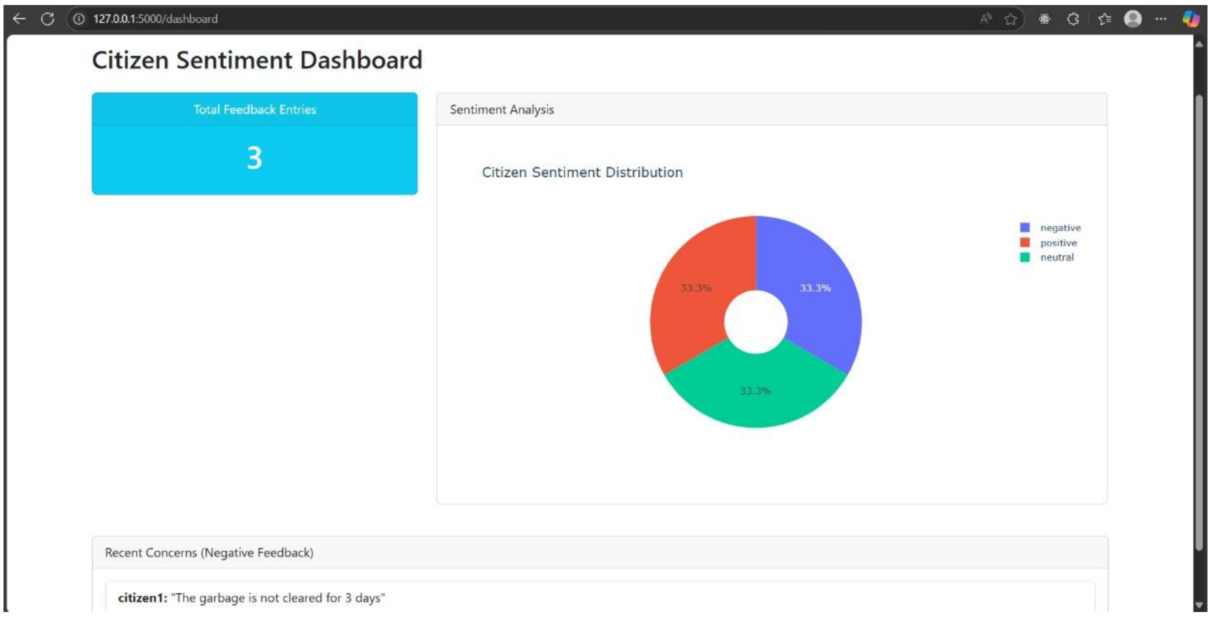
The local police are very responsive and helpful.

"I've noticed that the police here are really helpful and quick to respond. I've seen them help people in need and I've even seen them help people who were in trouble. I think they're really great."

There are potholes all over the main road.

"I can tell you that the city has received reports of potholes and is already working on a solution."

© 2024 Citizen AI Platform


127.0.0.1:5000/dashboard

Citizen Sentiment Dashboard

Total Feedback Entries

3

Sentiment Analysis

Citizen Sentiment Distribution

Recent Concerns (Negative Feedback)

citizen1: "The garbage is not cleared for 3 days"

8. ADVANTAGES & DISADVANTAGES

8.1 Advantages

1. Real-Time Query Resolution: CitizenAI uses IBM Watsonx to instantly answer citizen queries, improving communication efficiency between the public and officials.
2. Intelligent Feedback Analysis: Watson NLU provides sentiment analysis that helps government bodies understand public opinion in real time.
3. Visualized Decision Support: Plotly dashboards display trends and issues visually, allowing data-driven policymaking.
4. Role-Based Secure Access: Officials gain protected access to sensitive analytics via Flask-Login-based authentication.
5. Scalable and Modular Design: The system architecture supports easy expansion for new features or channels of feedback.
6. Improved Public Trust: Transparency in issue tracking and faster response cycles boost citizen satisfaction and participation.
7. Low Learning Curve: Simple UI with clearly defined routes and responses ensures ease of use for both citizens and officials.
8. No Licensing Cost for Citizens: The chatbot runs on open standards, allowing access to services without requiring paid apps or sign-ins.

8.2 Disadvantages

1. No Data Persistence: Currently, sentiment data and query history are not stored across sessions due to the lack of a connected database.
2. Limited Sentiment Scope: The system only supports three sentiment classes (positive, negative, neutral) without topic-specific tagging.
3. External Dependency: The AI and sentiment modules rely on IBM services and require an active cloud connection.

4. Language Limitation: Multilingual responses are not yet supported, limiting accessibility for non-English-speaking citizens.
5. Offline Use Unavailable: Unlike local models, CitizenAI requires internet connectivity to access Watson services.
6. No Bulk Analytics: Officials can’t yet export or generate reports summarizing sentiment over extended periods
7. Frontend Feature Constraints: Current UI lacks filters, map visualizations, or issue tagging for advanced monitoring.

9. CONCLUSION

CitizenAI represents a forward-thinking application of artificial intelligence in the realm of civic governance. By automating query resolution and analyzing feedback with advanced natural language tools, it bridges the gap between citizens and government institutions through real-time, data-informed communication.

The integration of IBM Watsonx for generative responses, coupled with Watson NLU for sentiment insights, allows the platform to deliver timely and relevant support. From AI-driven conversations to insightful visual dashboards, each component contributes to efficient public engagement.

CitizenAI demonstrates that smart, cloud-based services can enhance transparency and responsiveness without burdening officials or confusing users. With its secure architecture and modular design, the system can be tailored for various administrative setups, from local municipalities to state departments.

Although dependent on cloud access and currently lacking data persistence and language support, the prototype lays a strong foundation for future-ready digital governance. As AI services continue to evolve, CitizenAI is poised to become an essential tool in creating smarter, more connected communities.

10. FUTURE SCOPE

10.1 Multi-User Support

- Implement citizen and official login with secure session tracking
- Maintain user-specific interaction logs and settings
- Introduce collaboration tools for officials to handle public issues jointly

10.2 Topic Modeling and Categorization

- Apply unsupervised learning to identify key civic themes in feedback
- Auto-tag queries by topics like infrastructure, public safety, sanitation, etc.
- Enhance dashboard filters based on categorized input

10.3 Persistent Database Integration

- Integrate SQLite or PostgreSQL for storing chat logs, sentiment scores, and trend data
- Enable historical analytics across different time frames
- Allow query auditing and policy tracking

10.4 Multi-Channel Feedback Integration

- Expand data ingestion to include social platforms like Twitter/X and WhatsApp
- Support batch import of citizen feedback via email or CSV uploads
- Standardize data format for sentiment and dashboard ingestion

10.5 Advanced Analytics and Automated Reporting

- Generate weekly/monthly visual summary reports as PDFs
- Email trend-based alerts and engagement summaries to officials
- Add usage statistics, chatbot accuracy, and public sentiment variance tracking

10.6 Language and Accessibility Improvements

- Enable multilingual chatbot interactions using Watson's translation models
- Provide support for local dialects and accessibility (screen readers, etc.)
- Add voice input and audio response options

10.7 Visual and UX Enhancements

- Refactor frontend using React or Vue for modern interaction

- Add map-based query clustering and heatmaps
- Support night mode, dynamic resizing, and theme customization

10.8 Cloud and Docker Deployment

- Dockerize the entire app for platform-independent deployment
- Add deployment templates for IBM Cloud, AWS, and Azure
- Scale via load balancing for high-volume public queries

10.9 Feedback Loop and Personalization

- Allow citizens to rate AI responses for continuous quality improvement
- Track popular queries and adapt response generation over time
- Integrate with fine-tuning pipelines to customize Watsonx behavior

10.10 Government Ecosystem Integration

- Connect with e-governance portals, municipal CRM tools, and citizen ID systems
- Enable API-based integration with legacy ticketing and workflow apps
- Use SSO and role-based dashboards for larger state or national-scale deployments

These future developments would elevate CitizenAI from a prototype to a powerful civic engagement tool capable of scaling to complex administrative ecosystems and evolving public service needs.

11. APPENDIX

Source Code:
The CitizenAI system is implemented using a modular architecture that separates frontend presentation from backend processing:

-

Backend

- Built using Flask (Python) for routing, API definition, and session handling

- Integrates IBM Watsonx SDK for natural language response generation
- Uses IBM Watson NLU for real-time sentiment classification
- Core file `app.py` manages all logic: model invocation, route control, and user session handling
- Includes the following endpoints:

- `/chat`: Handles user queries via chatbot
- `/analyze-sentiment`: Applies Watson NLU to user feedback
- `/login`: Authenticates official user accounts
- `/dashboard`: Displays processed analytics using Plotly charts

main.py :

```
from dotenv import load_dotenv

load_dotenv()

# app.py

from flask import Flask, render_template, request, redirect, url_for, session, flash

from flask_login import LoginManager, UserMixin, login_user, logout_user, login_required,
current_user

import os

import pandas as pd

import plotly.graph_objects as go

import plotly.io as pio


# Import utility functions

from utils import get_granite_response, get_sentiment


app = Flask(__name__)

app.secret_key = os.urandom(24)
```

```
# In-memory user and chat data (can be replaced with DB)
users = {"citizen1": {"password": "password123"}}
chat_history = {}
feedback_data = []

# Flask-Login setup
login_manager = LoginManager()
login_manager.init_app(app)
login_manager.login_view = 'login'

class User(UserMixin):
    def __init__(self, id):
        self.id = id

@login_manager.user_loader
def load_user(user_id):
    if user_id in users:
        return User(user_id)
    return None

# Routes
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        if username in users and users[username]['password'] == password:
            login_user(User(username))
            flash('Logged in successfully.', 'success')
            return redirect(url_for('chat'))
        else:
            flash('Invalid username or password.', 'danger')
    return render_template('login.html')

@app.route('/logout')
@login_required
def logout():
    logout_user()
    flash('Logged out.', 'info')
    return redirect(url_for('login'))

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

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

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

@app.route('/chat', methods=['GET', 'POST'])
@login_required
def chat():
    user_id = current_user.id
    if user_id not in chat_history:
        chat_history[user_id] = []

    if request.method == 'POST':
        user_message = request.form['message']
        chat_history[user_id].append({'sender': 'user', 'text': user_message})

        context_prompt = f"""
        You are 'Citizen AI', a helpful and respectful assistant for a government citizen
engagement platform.
        A citizen is asking a question. Provide a clear, concise, and helpful response.
        Citizen's question: "{user_message}"
        Your response:
        """
        ai_response = get_granite_response(context_prompt)
        chat_history[user_id].append({'sender': 'ai', 'text': ai_response})

        sentiment = get_sentiment(user_message)
        feedback_data.append({
            'username': user_id,
            'text': user_message,
            'sentiment': sentiment['label'],
            'sentiment_score': sentiment['score']
        })

    return render_template('chat.html', history=chat_history[user_id])

@app.route('/dashboard')
@login_required
```

```
def dashboard():
    if not feedback_data:
        return render_template('dashboard.html', total_feedback=0, sentiment_chart=None,
recent_concerns=[])

    df = pd.DataFrame(feedback_data)
    total_feedback = len(df)
    sentiment_counts = df['sentiment'].value_counts()
    fig = go.Figure(data=[go.Pie(labels=sentiment_counts.index,
values=sentiment_counts.values, hole=.3)])
    fig.update_layout(title_text='Citizen Sentiment Distribution')
    sentiment_chart_html = pio.to_html(fig, full_html=False)
    recent_concerns = df[df['sentiment'] == 'negative'].tail(5).to_dict('records')

    return render_template('dashboard.html',
        total_feedback=total_feedback,
        sentiment_chart=sentiment_chart_html,
        recent_concerns=recent_concerns
    )

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

Frontend

- Developed using HTML, CSS, and JavaScript for a responsive single-page interface
- Includes the following interactive components:

- Sidebar menu for selecting SDLC-related operations (e.g., analyze, generate code)
- Main text input area for user prompts or queries
- Output section where AI responses are dynamically rendered
- History view panel showing recent interactions
- About section for user guidance and application info

- Key files include:
    - `index.html`: Contains the primary structure and layout of the interface
    - Inline JavaScript: Handles frontend logic for calling API endpoints, updating DOM content, and managing session-based interaction logs

index.html :

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>{% block title %}Citizen AI{% endblock %}</title>
  <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css"
rel="stylesheet">
  <link rel="stylesheet" href="{{ url_for('static', filename='css/styles.css')
}}">
</head>
<body>
<nav class="navbar navbar-expand-lg navbar-dark bg-primary">
  <div class="container-fluid">
    <a class="navbar-brand" href="{{ url_for('index') }}">Citizen AI</a>
    <ul class="navbar-nav me-auto">
```

```
          <li class="nav-item"><a class="nav-link" href="{{ url_for('index')
}}">Home</a></li>

          <li class="nav-item"><a class="nav-link" href="{{ url_for('about')
}}">About</a></li>

          <li class="nav-item"><a class="nav-link" href="{{ url_for('services')
}}">Services</a></li>

          {% if current_user.is_authenticated %}

            <li class="nav-item"><a class="nav-link" href="{{ url_for('chat')
}}">Chat</a></li>

            <li class="nav-item"><a class="nav-link" href="{{
url_for('dashboard') }}">Dashboard</a></li>

          {% endif %}

        </ul>

        <ul class="navbar-nav ms-auto">

          {% if current_user.is_authenticated %}

            <li class="nav-item"><span class="navbar-text me-3">Welcome, {{
current_user.id }}</span></li>

            <li class="nav-item"><a class="btn btn-light" href="{{
url_for('logout') }}">Logout</a></li>

          {% else %}

            <li class="nav-item"><a class="btn btn-light" href="{{
url_for('login') }}">Login</a></li>

          {% endif %}

        </ul>

    </div>

</nav>


<div class="container mt-4">

    {% with messages = get_flashed_messages(with_categories=true) %}

        {% if messages %}

          {% for category, message in messages %}

            <div class="alert alert-{{ category }}">{{ message }}</div>
```

```
                {% endfor %}
            {% endif %}
        {% endwith %}
        {% block content %}{% endblock %}
    </div>

    <footer class="footer mt-auto py-3 bg-light text-center">
        <div class="container">
            <span class="text-muted">© 2024 Citizen AI Platform</span>
        </div>
    </footer>
    </body>
    </html>
```

- style.css: handles layout and styling

style.css :

```
/* styles.css */

body {
    display: flex;
    flex-direction: column;
    min-height: 100vh;
}

.container {
    flex: 1;
}

.footer {
    flex-shrink: 0;
}



/* Chat Window Styles */

.chat-window {
    height: 60vh;
    display: flex;
    flex-direction: column;
}

.chat-box {
    flex-grow: 1;
    overflow-y: auto;
    padding: 1rem;
    display: flex;
    flex-direction: column;
    gap: 1rem;
}

.message {
    display: flex;
}

.message.user {
    justify-content: flex-end;
}
```

```
.message.ai {
  justify-content: flex-start;
}

.bubble {
  max-width: 70%;
  padding: 10px 15px;
  border-radius: 20px;
  word-wrap: break-word;
}

.message.user .bubble {
  background-color: #0d6efd;
  color: white;
  border-bottom-right-radius: 5px;
}

.message.ai .bubble {
  background-color: #e9ecef;
  color: #212529;
  border-bottom-left-radius: 5px;
}
```

Dataset Link:

This project does not rely on an external dataset for its functionality. Instead, it uses a locally hosted pre-trained and instruction-tuned language model (`granite-3.3-2b-instruct-Q4_K_M.gguf`) executed via llama-cpp.

The model has already been fine-tuned to understand and respond to a wide range of natural language prompts, such as:

- Analyzing and summarizing software requirements
- Generating functional code from descriptions

- Creating test cases for functions and APIs
- Fixing bugs in existing code snippets
- Writing documentation including docstrings and usage descriptions

Since it operates entirely based on pre-trained internal logic, no additional training dataset or annotation is required. Users interact with the system via the frontend by entering task descriptions, and the backend model processes these in real time to return results.

This approach streamlines the development workflow, eliminates the need for data preprocessing or fine-tuning, and ensures the application is usable out-of-the-box.

GitHub & Project Demo Link

- GitHub: https://github.com/Devallasatwik/citizen
- Demo: https://github.com/Devallasatwik/citizen