Project Documentation

1. Introduction

Project Title: Sustainable Smart City Assistant (Citizen AI)

Team ID : NM2025TMID03594

Team Size : 4

Team Leader : BHARATHI V

Team member : ASINA K

Team member : ANNAPOORANI S

Team member : BHAVANA S

2. Project Overview

\*Purpose

The purpose of a Sustainable Smart City Assistant is to empower cities and their residents to thrive in a more eco-conscious and connected urban environment. By leveraging AI and real-time data, the assistant helps optimize essential resources like energy, water, and waste, while also guiding sustainable behaviors among citizens through personalized tips and services. For city officials, it serves as a decision-making partner—offering clear insights, forecasting tools, and summarizations of complex policies to support strategic planning. Ultimately, this assistant bridges technology, governance, and community engagement to foster greener cities that are more efficient, inclusive, and resilient.

\*Features

1. Conversational Interface

Key Point: Natural language interaction

Functionality: Allows citizens and officials to ask questions, get updates, and receive guidance in plain language.

2. Policy Summarization

Key Point: Simplified policy understanding

Functionality: Converts lengthy government documents into concise, actionable summaries.

3. Resource Forecasting

Key Point: Predictive analytics

Functionality: Estimates future energy, water, and waste usage using historical and real-time data.

4. Eco-Tip Generator

Key Point: Personalized sustainability advice

Functionality: Recommends daily actions to reduce environmental impact based on user behavior.

5. Citizen Feedback Loop

Key Point: Community engagement

Functionality: Collects and analyzes public input to inform city planning and service improvements.

6. KPI Forecasting

Key Point: Strategic planning support

Functionality: Projects key performance indicators to help officials track progress and plan ahead.

7. Anomaly Detection

Key Point: Early warning system

Functionality: Identifies unusual patterns in sensor or usage data to flag potential issues.

8. Multimodal Input Support

Key Point: Flexible data handling

Functionality: Integrates text, speech, images, and IoT sensor data for richer analysis and interaction.

9. Streamlit or Gradio UI

Key Point: User-friendly interface

Functionality: Provides an intuitive dashboard for both citizens and city officials to interact with the assistant.

3. Architecture

Frontend (Streamlit):

The frontend is built with Streamlit, offering an interactive web UI with multiple pages including dashboards, file uploads, chat interface, feedback forms, and report viewers. Navigation is handled through a sidebar using the streamlit-option-menu library. Each page is modularized for scalability.

Backend (FastAPI):

FastAPI serves as the backend REST framework that powers API endpoints for document processing, chat interactions, eco-tip generation, report creation, and vector embedding. It is optimized for asynchronous performance and includes Swagger integration.

LLM Integration (IBM Watsonx Granite):

Granite LLM models from IBM Watsonx are used for natural language understanding and generation. Prompts are carefully designed to generate summaries, sustainability tips, and reports.

Vector Search (Pinecone):

Uploaded policy documents are embedded using Sentence Transformers and stored in Pinecone. Semantic search is implemented using cosine similarity to allow users to search documents using natural language queries.

ML Modules (Forecasting and Anomaly Detection):

Lightweight ML models are used for forecasting and anomaly detection using Scikit-learn. Time-series data is parsed, modeled, and visualized using pandas and matplotlib.

4. Setup Instructions

Prerequisites:

Python 3.9 or later

pip and virtual environment tools

API keys for IBM Watsonx and Pinecone

Internet access to access cloud services

Steps:

1. Clone the project repository

2. Create and activate a virtual environment.

3. Install dependencies from requirements.txt.

4. Configure API keys for IBM Watsonx and Pinecone in .env file.

5. Run the FastAPI backend server.

6. Launch the Streamlit frontend for user interaction.

5. Running the Application

To start the project:

1. Launch the FastAPI server to expose backend endpoints.

2. Run the Streamlit dashboard to access the web interface.

3. Navigate through pages via the sidebar.

4. Upload documents or CSVs, interact with the chat assistant, and view outputs like reports, summaries, and predictions.

5. Submit feedback via POST /submit-feedback to store citizen input for later review or analytics.

All interactions are real-time and use backend APIs to dynamically update the frontend.

7. API Documentation

Backend APIs available include:

POST /chat/ask – Accepts a user query and responds with an AI-generated message.

POST /upload-doc – Uploads and embeds documents in Pinecone.

GET /search-docs – Returns semantically similar policies to the input query.

GET /get-eco-tips – Provides sustainability tips for selected topics like energy, water, or waste.

Each endpoint is tested and documented in Swagger UI for quick inspection and trial during development.

8. Authentication

This version of the project runs in an open environment for demonstration. However, secure deployments can integrate:

Token-based authentication (JWT or API keys)

OAuth2 with IBM Cloud credentials

Role-based access (admin, citizen, researcher)

Planned enhancements include user sessions and history tracking.

9. User Interface

The interface is minimalist and functional, focusing on accessibility for non-technical users. It includes:

Sidebar with navigation

KPI visualizations with summary cards

Tabbed layouts for chat, eco tips, and forecasting

Real-time form handling

PDF report download capability

The design prioritizes clarity, speed, and user guidance with help texts and intuitive flows.

10. Testing

Testing was done in multiple phases:

Unit Testing: For prompt engineering functions and utility scripts.

API Testing: Via Swagger UI, Postman, and test scripts.

Manual Testing: For file uploads, chat responses, and output consistency.

Edge Case Handling: Malformed inputs, large files, invalid API keys.

Each function was validated to ensure reliability in both offline and API environments.

Screen Shot:

