Sustainable Smart City Assistant Using IBM Granite

# Project Documentation

1. **Introduction**
   * **Project title** : Generative AI in Smart Cities – Sustainable Assistant
   * **Team leader**: PRASANTH.K

: P.Prasanna

* + **Team member**

: M.BoopathiRaja



:PREMKUMAR.K

* + **Team member**

: K.Midhun



: RAJESH.N

* + **Team member**

: B.Mohmmad salman



: SAKTHI SIVA THANU.V

1. **Project Overview**
   * **Purpose** :

The purpose of this project is to develop a **Smart City AI Assistant** that helps cities and citizens adopt sustainable practices. It optimizes resources such as

**energy, water, and waste**, provides policy summaries, forecasts usage patterns, and offers eco-friendly recommendations. It also supports city officials with insights, decision-making tools, and community feedback analysis.

# Features :

* + - **Conversational Interface** – Natural language interaction for citizens and officials
    - **Policy Summarization** – Simplifies lengthy government policies
    - **Resource Forecasting** – Predicts future usage of key resources
    - **Eco-Tip Generator** – Provides personalized sustainability advice
    - **Citizen Feedback Loop** – Collects and analyzes public input
    - **KPI Forecasting** – Projects key performance indicators for planning
    - **Anomaly Detection** – Early detection of unusual data patterns
    - **User-Friendly Interface** – Dashboard built with Streamlit/Gradio

# Architecture Frontend (Streamlit):

* Provides a simple, browser-based interface with tabs for different functionalities.

# Backend (FastAPI):

* Handles model integration, forecasting, document processing, and responses.

# LLM Integration (IBM Granite):

* Generates summaries, eco-tips, and natural language responses.

# System Flow:

User input → Granite Model / ML modules → Processed Output → Displayed in UI

# Setup Instructions Prerequisites:

* Python 3.9 or later
* pip package manager
* API keys for IBM Watsonx and Pinecone
* Internet connection

# Installation Process:

1. Clone the repository
2. Install dependencies (requirements.txt)
3. Configure API credentials
4. Run the backend server
5. Launch the Streamlit dashboard

# Folder Structure

app.py – Main program that integrates model and UI requirements.txt – Dependency file for Python packages report.docx – Project documentation

screenshots – Folder containing sample outputs and interface images deployment\_link.txt – File containing deployed application link

# Running the Application

To start the application:

1. Run the backend server
2. Launch the Streamlit dashboard
3. Navigate through tabs to access features like chat, policy summarization, forecasting, and eco-tips
4. Upload documents or data files to receive outputs

# API Documentation

* **/chat/ask** – AI-powered Q&A
* **/upload-doc** – Upload and embed documents
* **/search-docs** – Semantic policy search
* **/get-eco-tips** – Generate eco-friendly suggestions
* **/submit-feedback** – Collect citizen feedback

# Authentication

The current version is open for demo. Future versions may include:

* User login (citizens/officials)
* Role-based access
* Data privacy and secure authentication methods

# User Interface

* Tabbed sections for different features
* Textbox for questions and feedback
* Dashboard visualizations for KPIs
* Report download capability
* Simple navigation for all users

# Testing

Testing was done in multiple ways:

* **Unit Testing** – Core functions validated
* **API Testing** – Checked with Swagger UI/Postman
* **Manual Testing** – Verified outputs for multiple datasets
* **Edge Case Testing** – Handled invalid inputs and large files

# Screenshots

Screenshots include:

* Dashboard UI
* City Analysis
* Citizen service

# Known Issues

* Slow response for very large documents
* Forecast accuracy depends on quality of input data
* Requires stable internet connection for APIs

# Future Enhancements

* Add speech-to-text input for accessibility
* Integrate IoT sensor data for real-time monitoring
* Provide multi-language support
* Optimize models for faster response