Sustainable Smart City Assistant

Project Documentation

Introduction

• Project Title : Sustainable Smart City Assistant

• Team Leader : GOPIKA B

• Team Leader : MOHANA N

• Team Leader : JEENAL JEMI S

• Team Leader : ANANDHI G

SMART CITY

**Project Overview**

# 1. 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.

# 2. Features:

* Conversational Interface: Natural language interaction; citizens and officials can ask questions, get updates, and receive guidance.
* Policy Summarization: Simplifies lengthy government documents into concise, actionable summaries.
* Resource Forecasting: Predicts future energy, water, and waste usage using historical and real-time data.
* Eco-Tip Generator: Recommends daily actions to reduce environmental impact based on user behavior.
* Citizen Feedback Loop: Collects and analyzes public input to inform city planning and improvements.
* KPI Forecasting: Projects KPIs to help officials track progress and plan ahead.
* Anomaly Detection: Identifies unusual data patterns for early issue warnings.
* Multimodal Input Support: Accepts text, PDFs, and CSVs for analysis and forecasting.
* Streamlit/Gradio UI: Provides an intuitive dashboard for both citizens and officials.

# 3. Architecture:

Frontend (Streamlit): Provides interactive dashboards, file uploads, chat, feedback forms, and report viewers.  
Backend (FastAPI): REST framework powering document processing, chat, eco tips, reports, and embeddings.  
LLM Integration (IBM Watsonx Granite): Handles summarization, sustainability tips, and reports.  
Vector Search (Pinecone): Stores and searches document embeddings.  
ML Modules: Forecasting and anomaly detection using Scikit-learn.

# 4. Setup Instructions:

* Prerequisites:
* Python 3.9 or later
* pip and virtual environment tools
* API keys for IBM Watsonx and Pinecone
* Internet access
* Installation Process:
* Clone the repository
* Install dependencies from requirements.txt

**•** Configure .env credentials

• Run FastAPI backend and Streamlit frontend

• Upload data and interact with modules

# 5. Folder Structure:

* app/ – FastAPI backend logic
* app/api/ – Modular API routes
* ui/ – Streamlit frontend components
* smart\_dashboard.py – Main dashboard entry
* granite\_llm.py – Watsonx Granite integration
* document\_embedder.py – Embedding and Pinecone storage
* kpi\_file\_forecaster.py – KPI forecasting
* anomaly\_file\_checker.py – Anomaly detection
* report\_generator.py – Sustainability reports

# 6. Running the Application:

* Launch FastAPI backend server.
* Run Streamlit dashboard.
* Navigate via sidebar.
* Upload documents or CSVs.
* Interact with assistant and view reports, summaries, predictions.

# 7. API Documentation:

* POST /chat/ask – AI-generated response
* POST /upload-doc – Uploads & embeds documents
* GET /search-docs – Semantic document search
* GET /get-eco-tips – Sustainability tips
* POST /submit-feedback – Citizen feedback storage

# 8. Authentication:

Current: Open environment for demo.  
Secure deployments can integrate:  
• Token-based authentication (JWT, API keys)  
• OAuth2 with IBM Cloud credentials  
• Role-based access (admin, citizen, researcher)  
Planned enhancements: user sessions and history tracking.

# 9. User Interface:

Minimalist, user-friendly design:  
• Sidebar navigation  
• KPI visualizations with summary cards  
• Tabbed layouts for chat, eco tips, forecasting  
• Real-time form handling  
• PDF report downloads

# 10. Testing

Testing phases:  
• Unit Testing: Prompt functions, utilities  
• API Testing: Swagger UI, Postman  
• Manual Testing: File uploads, chat, outputs  
• Edge Cases: Large files, malformed inputs, invalid keys

# 11. Known Issues & Future Enhancements:

Known Issues:  
• API limitations  
• Handling large files  
  
Future Enhancements:  
• Advanced authentication  
• Additional ML models  
• Mobile app integration

**THANK YOU!!!**