Project Documentation

Project Title: Sustainable Smart City Assistant

- **Team leader : ** Johnkelvin J
- Team Member : Moneshwar R
- **Team Member:** Kishore S
- **Team Member:** Abinash

1. Introduction

This document provides a detailed overview of the Sustainable Smart City Assistant, a full-stack web application developed as a project for the Naan Mudhalvan program. It outlines the project's purpose, features, technical architecture, and instructions for setup and use.

2. Project Overview

Purpose

The purpose of the Sustainable Smart City Assistant is to empower cities and their residents to thrive in a more eco-conscious and connected urban environment. By leveraging a state-of-the-art large language model (Google Gemini) and real-time data analysis, the assistant helps optimize essential resources like energy and water, while also guiding sustainable behaviors among citizens.
For city officials, it serves as a decision-making partner—offering clear insights from data, forecasting future trends, and summarizing complex policy documents 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 Implemented

The application is composed of several fully functional modules, each targeting a specific smart city function:

- 🤖 **AI Policy Summarizer:** Instantly generates concise, easy-to-understand summaries of complex policy documents using the Google Gemini Pro model.
- 📝 **Citizen Feedback Form:** A user-friendly form for citizens to report issues (e.g., garbage, water supply) or provide suggestions. Submissions are logged in a structured JSON file on the backend.
- 📤 **Document Management:** Allows administrators to upload .txt policy documents directly through the web interface.
- 🔎 **Semantic Search:** A powerful search engine that allows users to ask questions in natural language and retrieve the most relevant sections from all uploaded documents, powered by vector embeddings and a Pinecone database.
- 💬 **Conversational Chat Assistant:** An interactive chat interface for conversational Q&A with the AI about smart city topics, maintaining a full conversation history within a session.
- 📈 **KPI Forecasting:** A data analysis tool to upload historical CSV data (e.g., yearly

energy consumption) and forecast the next year's value using linear regression.
- ⚠️ **Anomaly Detection:** A tool to upload CSV data and identify records that exceed a user-defined threshold, helping to flag potential issues or data errors.

3. Architecture

The application is built on a modern, decoupled three-tier architecture, ensuring scalability and maintainability.
[Streamlit Frontend] <--> [FastAPI Backend] <--> [External Services (Google AI, Pinecone)]

Frontend (Streamlit)

The frontend is an interactive web UI built with Streamlit. It features a stylish, multi-page design with custom CSS for a modern, dark-themed aesthetic. Navigation is handled through a clean sidebar that allows users to switch between the various modules. Each page is designed to be intuitive, handling user inputs, file uploads, and displaying dynamic results from the backend.

Backend (FastAPI)

FastAPI serves as the robust, asynchronous backend REST framework. It powers all the application's logic through a series of modular API endpoints organized by function (e.g., chat, feedback, vector search). It is optimized for high performance and includes automatic interactive API documentation (Swagger UI) at the /docs endpoint.

LLM Integration (Google Gemini)

The core natural language understanding and generation capabilities are powered by **Google's gemini-1.5-flash-latest model**. We chose this model for its high performance, reliability, and seamless integration via the langchain-google-genai library. This integration is responsible for summarization and all conversational features.

Vector Search (Pinecone)

Uploaded policy documents are processed, chunked, and converted into 384-dimensional numerical vectors using the all-MiniLM-L6-v2 **Sentence-Transformers** model. These embeddings are then stored and indexed in a serverless **Pinecone** vector database. Semantic search is implemented using cosine similarity to find the most relevant document chunks based on a user's natural language query.

ML Modules (Scikit-learn & Pandas)

For the data analysis features, the backend uses the scikit-learn and pandas libraries. This includes a lightweight linear regression model for KPI forecasting and efficient data filtering for threshold-based anomaly detection in user-uploaded CSV files.

4. Setup and Installation

To run this project locally, please follow these steps.

Prerequisites

- Python 3.9+
- Git for version control

1. Clone the Repository

```
git clone https://github.com/moneshwa/SMARTCITY_ASSISTANT.git
cd SMARTCITY_ASSISTANT
```

2. Create and Activate a Virtual Environment

```
# Create the environment
python -m venv .venv

# Activate the environment (on Windows Git Bash)
source .venv/Scripts/activate
```

3. Install Dependencies

Install all the required Python packages from the requirements.txt file.

```
pip install -r requirements.txt
```

4. Configure API Keys

The application requires API keys from Google and Pinecone.

1. Create a file named .env in the root of the project folder.
2. Add your secret keys to this file. A template is provided below.**.env File Template:**
   ```
   # Google AI API Key from Google AI Studio
   GOOGLE_API_KEY="your_google_api_key_here"

   # Pinecone Credentials from app.pinecone.io
   PINECONE_API_KEY="your_pinecone_api_key_here"
   PINECONE_ENV="us-east-1-aws"
   PINECONE_INDEX_NAME="smart-city-assistant"
   ```

5. Folder Structure

The project is organized into a clean and logical directory structure:

```
SMARTCITY_ASSISTANT/
├── .venv/                  # Virtual environment folder
├── app/                    # All FastAPI backend logic
```

```
│   ├── api/                # Modular API routers
│   │   ├── chat_router.py
│   │   ├── data_analysis_router.py
│   │   ├── feedback_router.py
│   │   ├── policy_router.py
│   │   └── vector_router.py
│   ├── data/               # For storing data like feedback logs
│   ├── services/           # Business logic for the LLM
│   │   └── llm_service.py
│   └── vectorstore/        # Logic for Pinecone and embeddings
│       ├── document_embedder.py
│       ├── document_retriever.py
│       └── pinecone_client.py
├── smartcity_frontend/     # All Streamlit frontend code
│   ├── styles/
│   │   └── style.css       # Custom CSS for styling
│   ├── api_client.py       # Functions to call the backend API
│   └── smart_dashboard.py  # Main entry script for the UI
├── .env                    # Secrets and API keys (not committed)
├── .gitignore              # Files to be ignored by Git
├── pyrightconfig.json      # Configuration for the VS Code linter
├── README.md               # This documentation file
└── requirements.txt        # Project dependencies
```

6. Running the Application

This project requires **two terminals** to run simultaneously: one for the backend API and one for the frontend dashboard.

1. Start the Backend Server

In your first terminal, from the main project folder (SMARTCITY_ASSISTANT), run:

```
uvicorn app.main:app --reload
```

The backend API will now be live and accessible at http://127.0.0.1:8000.

2. Start the Frontend Application

In a new, second terminal, navigate to the frontend directory and run the Streamlit app:

```
cd smartcity_frontend
streamlit run smart_dashboard.py
```

A new tab will open in your browser at http://localhost:8501, where you can interact with the live application.

7. API Documentation (Swagger UI)

All backend API endpoints are automatically documented and can be tested interactively using the built-in Swagger UI.

- **URL:** http://127.0.0.1:8000/docs

The available endpoints include:

- POST /policy/summarize-policy: Summarizes a piece of text.
- POST /feedback/submit-feedback: Stores citizen feedback.
- POST /vectors/upload-document: Uploads and embeds a .txt document in Pinecone.
- POST /vectors/search-documents: Performs semantic search on uploaded documents.
- POST /chat/ask: Responds to a prompt for the chat assistant.
- POST /analysis/forecast: Forecasts a KPI from an uploaded CSV.
- POST /analysis/anomalies: Detects anomalies in an uploaded CSV.

8. User Interface (UI)

The interface is designed to be minimalist, modern, and functional, focusing on accessibility for non-technical users. It was styled with custom CSS to achieve a clean, dark-themed aesthetic inspired by modern AI applications.

Key UI components include:

- A persistent sidebar with clear navigation between all major features.
- Dedicated pages for each of the 6 core features.
- Interactive widgets like file uploaders, forms, chat inputs, and data tables.
- Spinner animations to provide feedback during processing.
- Success and warning messages to guide the user.

9. Testing

The application was tested throughout its development cycle to ensure reliability.

- **API Testing:** Each backend endpoint was individually tested using the FastAPI Swagger UI to validate request/response models and logic.
- **Manual End-to-End Testing:** A full testing checklist was performed to simulate a real user's workflow, ensuring that the frontend correctly communicates with the backend for every feature. This included testing with sample .txt and .csv files.
- **Debugging:** An iterative debugging process was used to resolve numerous challenges, including package dependency conflicts, API availability issues (leading to the strategic switch from Hugging Face to Google AI), and Python import resolution problems in VS Code.

10. Future Enhancements

While the project is feature-complete according to the initial plan, several enhancements could be made in the future:

- **User Authentication:** Integrate a login system (e.g., using JWT) to create user-specific sessions and secure the document management features.
- **Database Integration:** Replace the JSON file for feedback logging with a more robust

database like SQLite or PostgreSQL for better data management.

- **Cloud Deployment:** Deploy the FastAPI backend and Streamlit frontend to cloud services (like Render and Streamlit Community Cloud) to make the application publicly accessible.
- **Advanced Visualizations:** Add more interactive charts and graphs to the KPI dashboard pages using libraries like Plotly or Altair.