Sustainable Smart-City AI Project

Documentation format

1. Introduction

• Project **Title**: Cityp'AI'rtner – Sustainable Smart City AI Assistant using IBM LLM

• Team **Members**:

- Ramayanapu Navya Sri – Project planning and Backend developer
- Aravapalli Hiranmai Sri – Frontend and Final report developer
- Sighakolli Venkata Sujatha – Project Design and Template Creator
- Achyuth Vemula – Collected necessary data and Libraries

2. Project Overview

• **Purpose**:
To provide AI-powered tools for urban sustainability by offering insights into real-time city KPIs, AI-driven eco advice, and automated sustainability report generation.

• **Key Features**:

- Smart dashboard for city-specific water, energy, and air quality KPIs.
- Upload-based forecasting system.
- Topic-based eco-friendly lifestyle tips.
- AI chatbot for sustainability queries (IBM Granite).
- AI-generated sustainability reports.

3. Architecture

- **Frontend**:
  - Built using HTML, CSS, and JavaScript.
  - UI includes navigation tabs for dashboard, forecasting, chat, reports, and tips.
  - Logo and theme styled for green sustainability branding.
- **Backend**:
  - FastAPI server for handling chat and report API endpoints.
  - Integrates IBM Granite AI for NLP tasks.
  - Serves frontend as static HTML.
- **Database**:
  - No persistent database used (uses in-memory/static JSON for KPIs and tips).

4. Setup Instructions

- **Prerequisites**:
  - Python 3.10+
  - FastAPI
  - Uvicorn
  - IBM Granite API setup
  - LocalTunnel (for external hosting)
- **Installation:**

1. Create & Activate Virtual Environment

```
python -m venv city-env

city-env\Scripts\activate
```

2. Install Required Libraries

```
pip install fastapi uvicorn streamlit transformers accelerate torch python-multipart
aiofiles
```

3. Backend: Already Running

You're already running:

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

->> **FastAPI backend is ready**.

4. Run the Application

```
Python main.py
```

5. Access URLs

Backend (API) → http://127.0.0.1:8000

5.Folder Structure

Frontend (static/):

index.html: Contains the entire UI layout and JS
logic./static/citypartnerailogo.jpg**: Used for branding/logo.**

Backend:

- main.py: FastAPI app with two main endpoints /chat and /report.
- ibm_granite_utils.py: Module to interact with IBM Granite API.

6. Running the Application

```
# Start the backend
uvicorn main:app --reload
```

Expose with LocalTunnel

```
lt --port 8000
```

7. API Documentation

- POST /chat
  - Input: { "query": "What is climate change?" }
  - Output: { "response": "..." }
  - Description: Sends a query to the IBM Granite chatbot.
- POST /report
  - Input: { "query": "Water conservation" }
  - Output: { "response": "..." }
  - Description: Requests a sustainability report from IBM Granite.

8. Authentication

- No user authentication is implemented (open access for demo purposes).
- Can be extended with JWT-based login for secured reporting access.

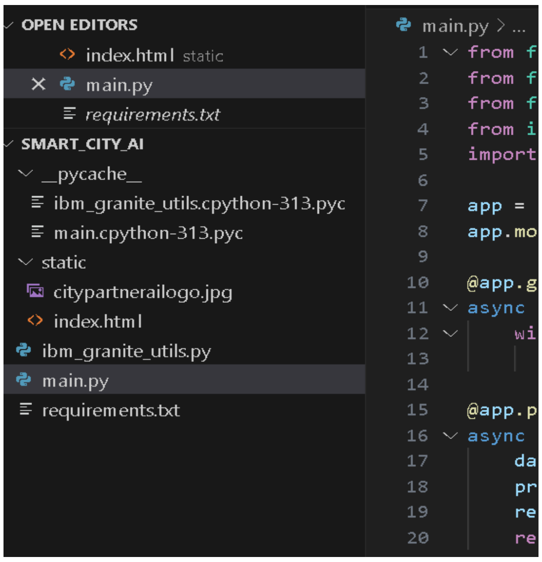
9. User Interface

- Intuitive sidebar navigation.
- City selector with dynamic KPI rendering.
- Simple form-based UI for chat, tips, and reports**.**

10. Testing

- Manual testing of UI elements and API endpoints.
- File upload testing using CSV format for KPI forecasts.

11. Screenshots / Demo



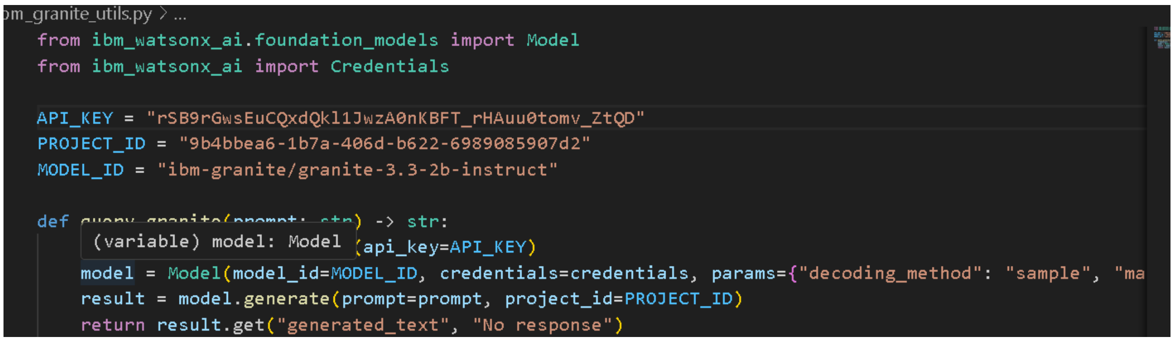
```
OPEN EDITORS
    index.html static
  × main.py
    requirements.txt
SMART_CITY_AI
  __pycache__
    ibm_granite_utils.cpython-313.pyc
    main.cpython-313.pyc
  static
    citypartnerailogo.jpg
    index.html
  ibm_granite_utils.py
  main.py
  requirements.txt
```

```
main.py > ...
from f
from f
from f
from i
import

app =
app.mo

@app.g
async
    wi


@app.p
async
    da
    pr
    re
    re
```



```
ibm_granite_utils.py > ...
from ibm_watsonx_ai.foundation_models import Model
from ibm_watsonx_ai import Credentials

API_KEY = "rSB9rGwsEuCQxdQkl1JwzA0nKBFT_rHAuu0tomv_ZtQD"
PROJECT_ID = "9b4bbea6-1b7a-406d-b622-6989085907d2"
MODEL_ID = "ibm-granite/granite-3.3-2b-instruct"

def query_granite(prompt: str) -> str:
    (variable) model: Model (api_key=API_KEY)
    model = Model(model_id=MODEL_ID, credentials=credentials, params={"decoding_method": "sample", "ma
    result = model.generate(prompt=prompt, project_id=PROJECT_ID)
    return result.get("generated_text", "No response")
```

Demo video:

Drive link:

https://drive.google.com/drive/file/d/1lZQl42JBUDv9a9PaSFMAE9bssp1hC7RE/view?usp=drivesdk

12. Known Issues

- Limited FAQ/chat scope (only predefined topics).
- No dynamic real-time data fetching (KPI data is hardcoded).
- No error handling for AI API timeouts or failures.

13. Future Enhancements

- Integrate live IoT or city dashboard APIs.
- Add MongoDB to store feedback and user sessions.
- Include user login and profile tracking.
- Expand chatbot with vector-based question matching.
- Implement more granular forecast visualizations using charts.