**🏙️ Smart City Assistant – Project Documentation**

**1. Introduction**

* **Project Title:** Smart City Assistant
* **Team Members:**
  + **Sangaraju Kavyapriya** – Backend Developer (FastAPI, APIs integration)
  + **Kalasapati Kaveri** – Frontend Developer (Streamlit UI)
  + **Thathireddy Lokitha** – Data Analyst & Testing
  + **Gangavaram Chandrasekhar** – API & Feature Integration

**2. Project Overview**

* **Purpose:**  
  The Smart City Assistant is designed to help urban administrators and citizens access real-time insights, forecasts, and reports related to smart city infrastructure such as sustainability, KPIs, anomalies, and weather updates using intelligent APIs.
* **Key Features:**
  + AI-powered chat assistant
  + Dynamic eco-tips
  + Weather data & forecasting
  + Policy summarization
  + KPI analytics (water, energy)
  + Anomaly detection system
  + Sustainability reports
  + Citizen feedback module

**3. Architecture**

* **Frontend (Streamlit):**
  + Developed using Python and Streamlit
  + Dynamic, interactive UI with sidebar navigation
  + Makes REST API calls to the backend using requests
* **Backend (FastAPI):**
  + RESTful API built in Python using FastAPI
  + Integrates with Hugging Face models & OpenWeatherMap
  + Handles logic for forecasting, anomaly detection, feedback, etc.
* **Database:**
  + No persistent database used in current version
  + Future versions can include MongoDB for storing user data, feedback, anomalies, etc.

**4. Setup Instructions**

* **Prerequisites:**
  + Python 3.8+
  + pip
  + Node.js (if adding React in future versions)
  + MongoDB (optional/future)
* **Installation Steps:**
* git clone https://github.com/your-repo/smart-city-assistant.git
* cd smart-city-assistant
  + Install backend dependencies:
  + pip install -r requirements.txt
  + Start FastAPI backend:
  + uvicorn main:app --reload
  + Start frontend (Streamlit):
  + streamlit run app.py
  + Set the following environment variables:
  + HF\_API\_KEY=your\_huggingface\_api\_key
  + WEATHER\_API\_KEY=your\_openweathermap\_api\_key

**5. Folder Structure**

smart-city-assistant/

├backend/

└── main.py # FastAPI backend logic

├ frontend/

└── app.py # Streamlit frontend

├ requirements.txt # Python dependencies

**6. Running the Application**

* **Start Backend:**
* uvicorn main:app --reload
* **Start Frontend:**
* streamlit run app.py

**7. API Documentation**

| **Endpoint** | **Method** | **Description** |
| --- | --- | --- |
| /chat/ask | POST | Ask a smart assistant AI |
| /eco/tips | POST | Get eco-friendly tips using AI |
| /feedback/submit | POST | Submit citizen feedback |
| /policy/summarize | POST | Summarize a long policy text |
| /weather/get | GET | Get current weather for a city |
| /kpi/forecast | GET | Get forecasted water & energy usage |
| /sustainability/report | GET | Generate sustainability report |
| /anomaly/check | GET | Simulated anomaly detection |

Example:

POST /chat/ask

{

"question": "What is the current recycling policy in the city?"

}

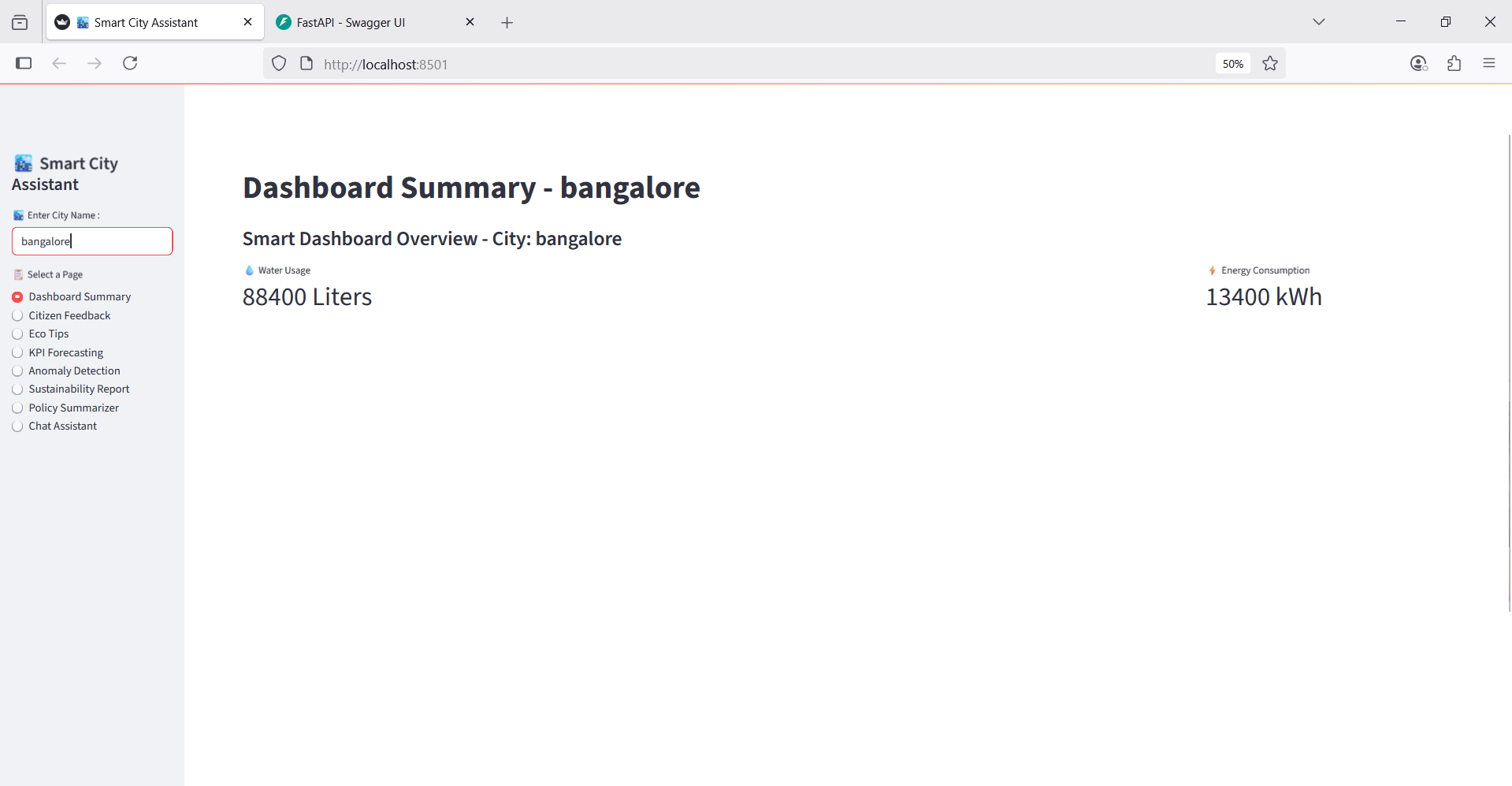
**8. Authentication**

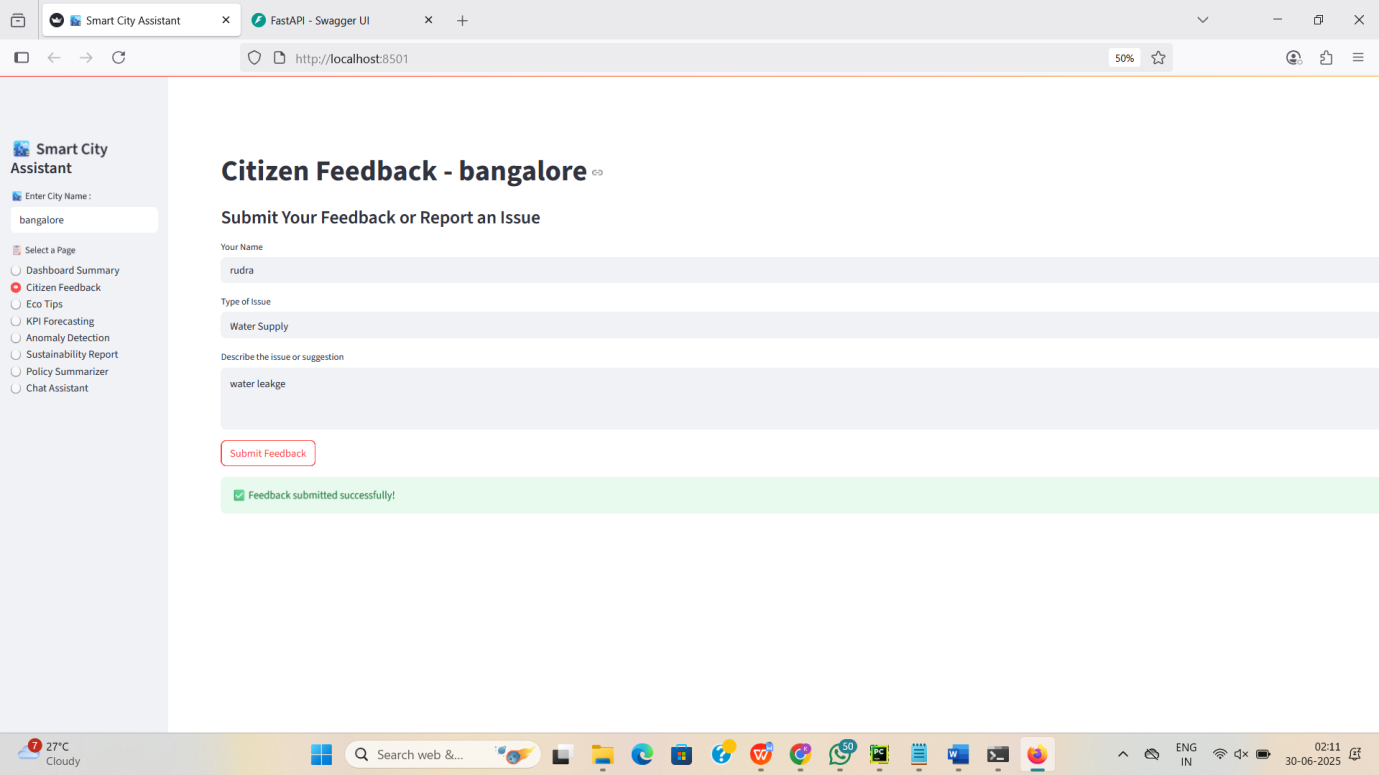
* **Current Version:** No authentication
* **Future Plans:**
  + JWT-based login system
  + Role-based access control (citizen/admin)

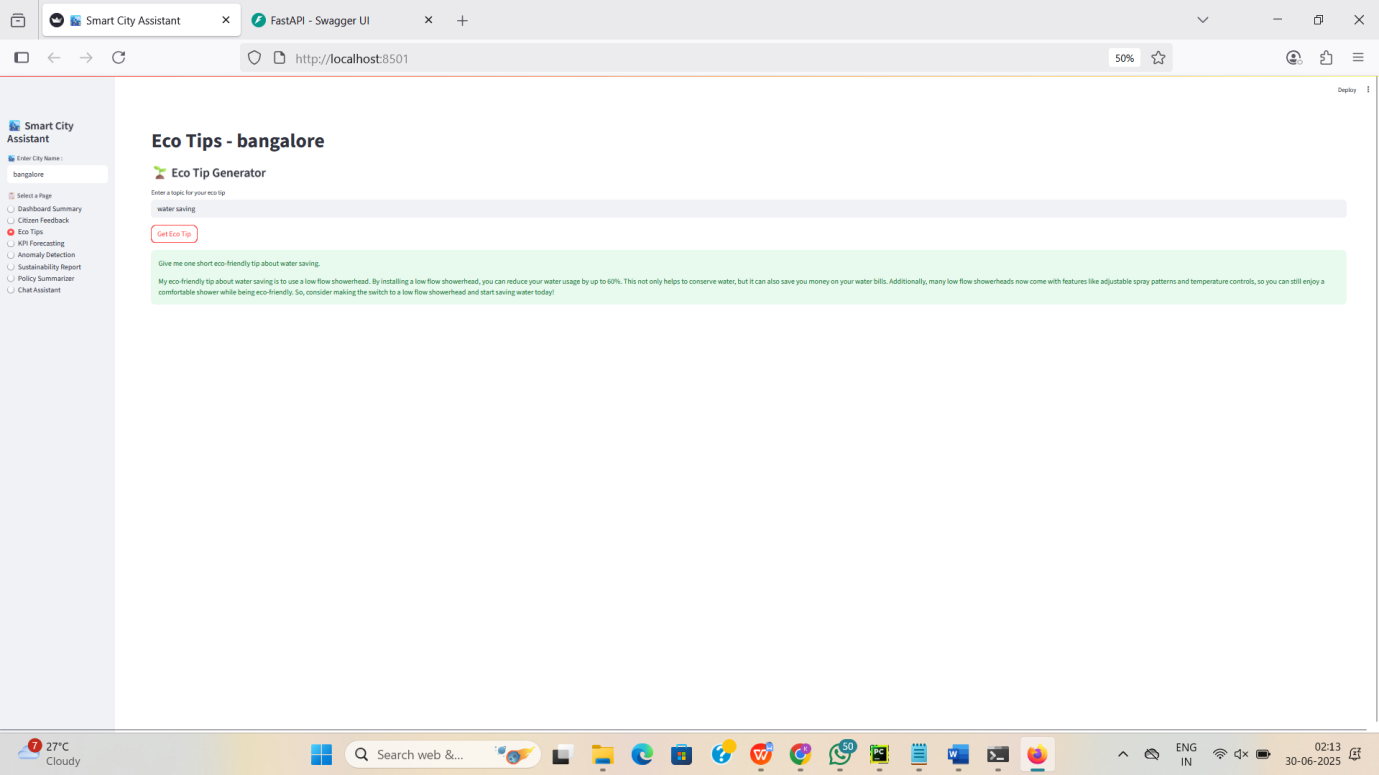
**9. User Interface**

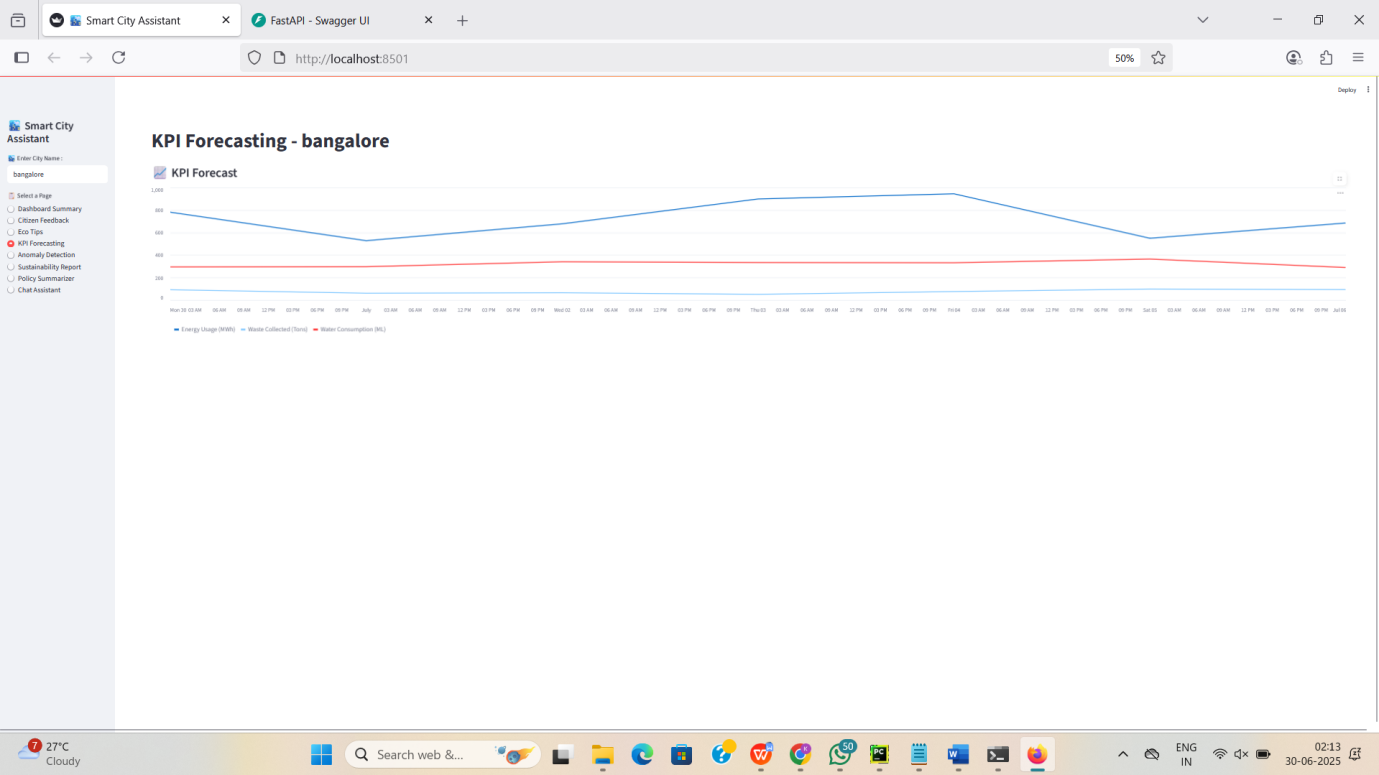
* Built using Streamlit
* Sidebar menu with navigable pages:
  + Chat
  + Eco Tips
  + Anomaly Detection
  + KPI Dashboard
  + Policy Summarizer
  + Feedback
  + Sustainability Summary

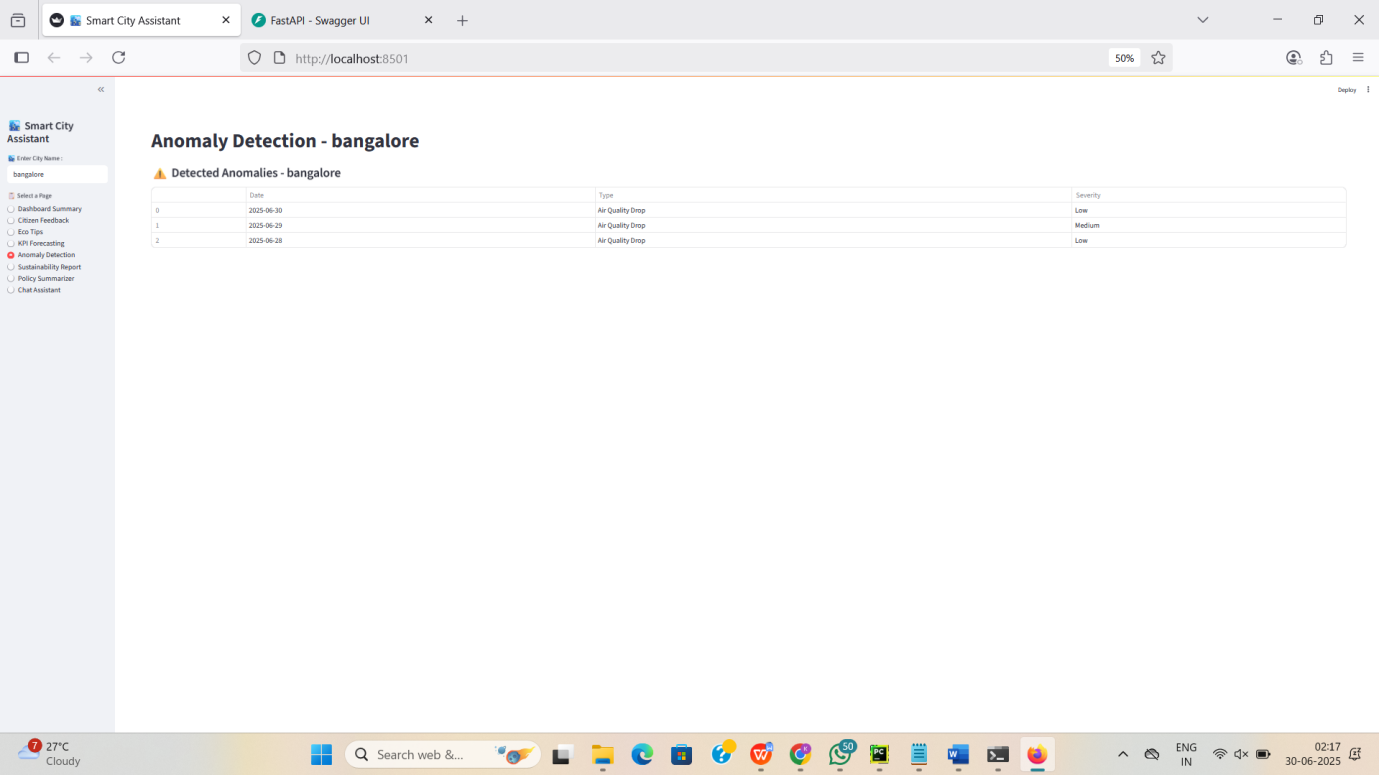
**Outputs:-**

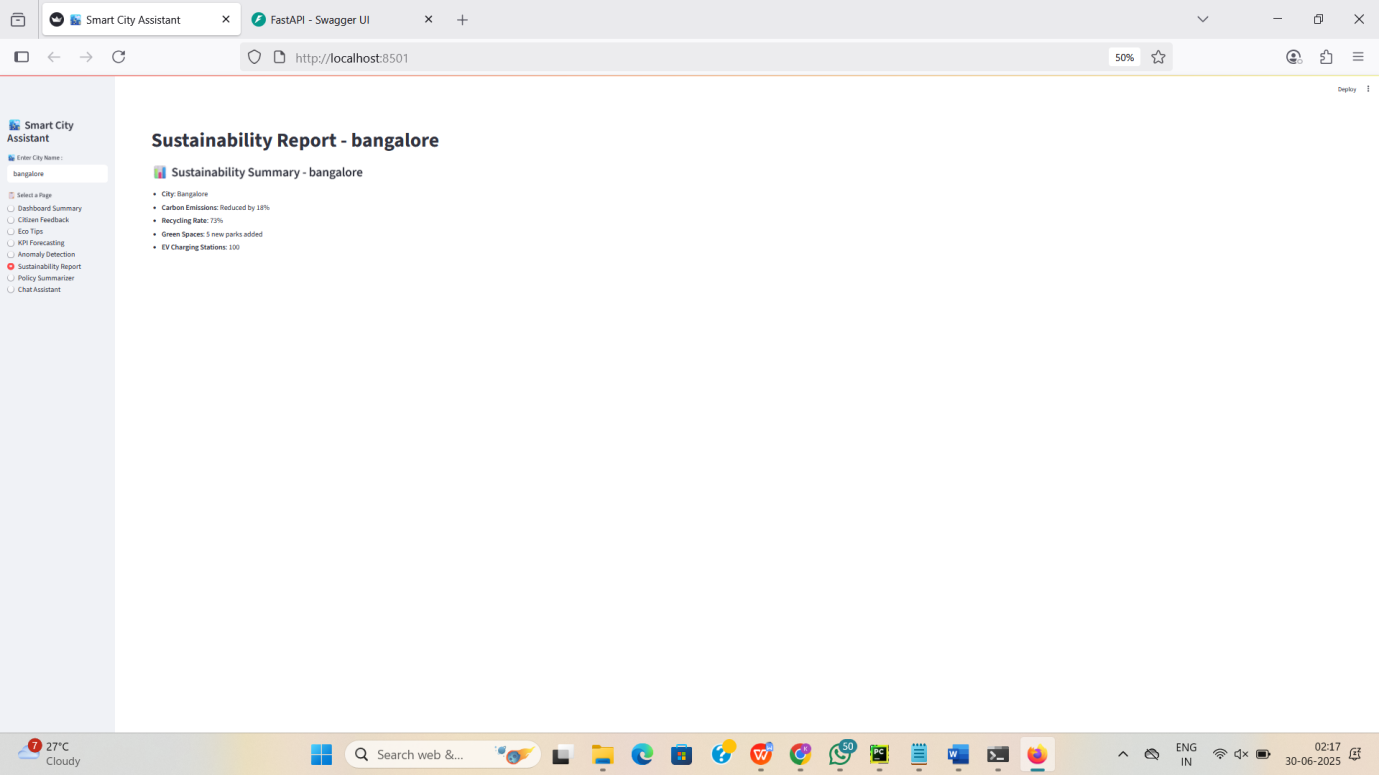


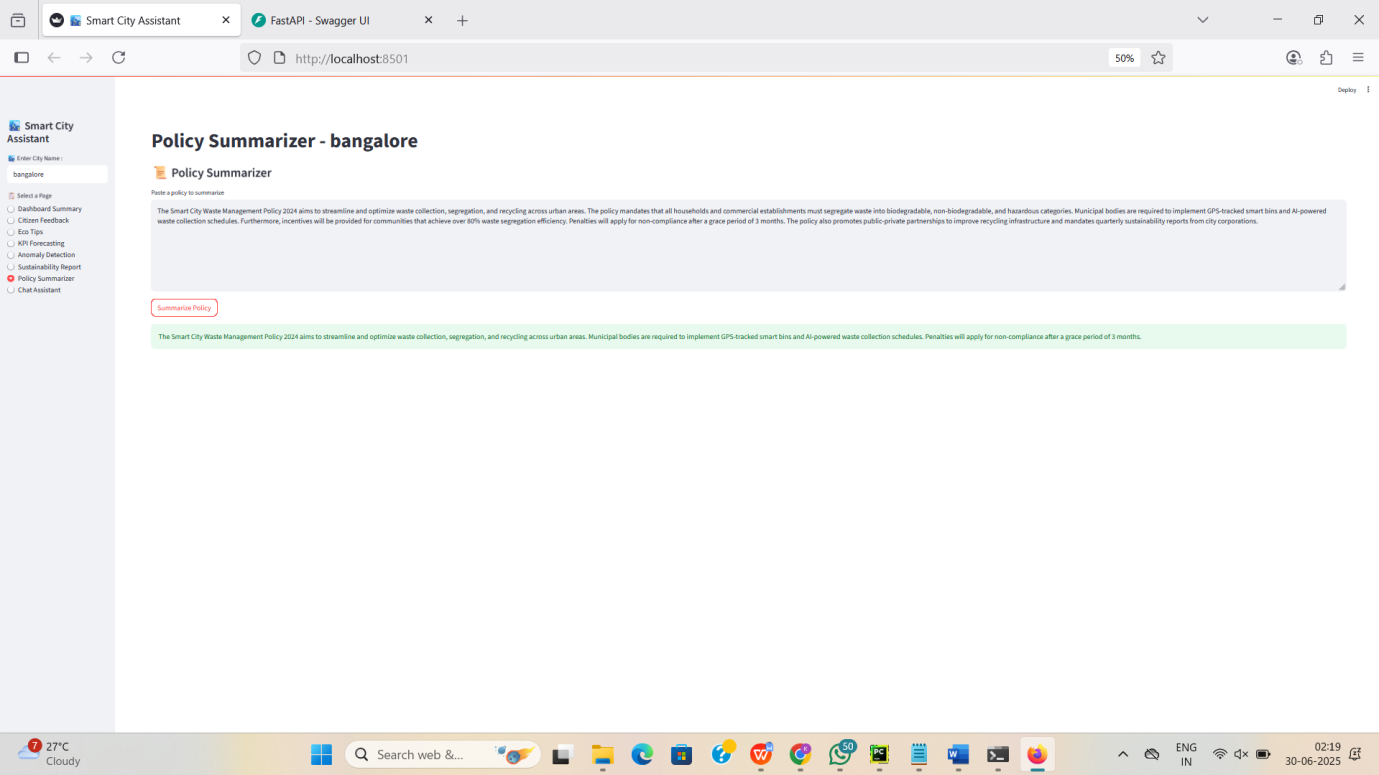


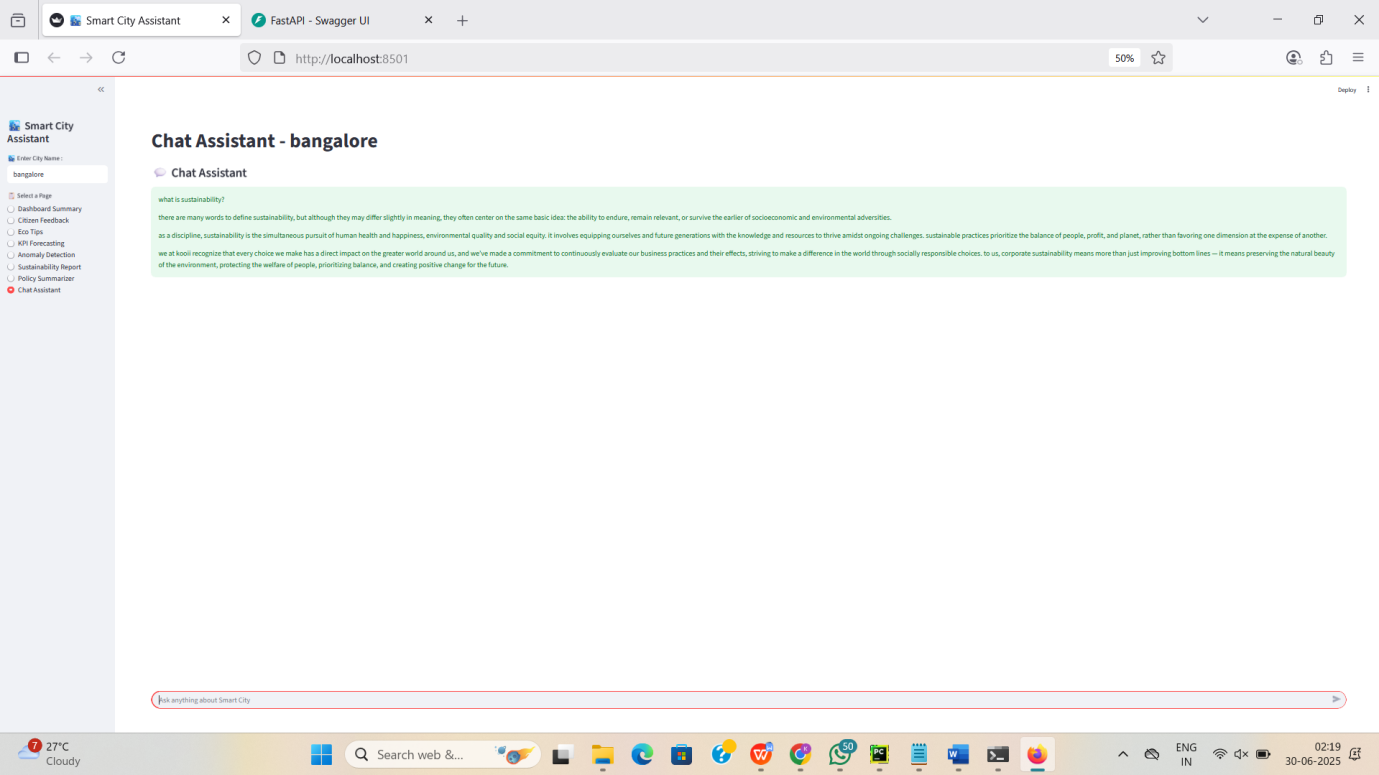












**10. Testing**

* Manual testing via Streamlit interface
* Backend endpoints tested with Postman
* Automated unit testing to be added using pytest

**11. Screenshots or Demo**

* **Demo Link:** *Coming soon (deploy via Hugging Face / Streamlit Cloud)*
* Screenshots:
  + Dashboard View
  + Anomaly Detection Table
  + Feedback Form

**12. Known Issues**

* Hugging Face models may return slightly delayed or inconsistent results
* No persistent storage (data lost on refresh)
* No input validation on policy text or feedback message lengths

**13. Future Enhancements**

* Integrate MongoDB for persistent data storage
* Add user authentication and admin dashboard
* Deploy using Docker or on a cloud platform
* Add mobile-responsive frontend using React
* Real-time alerts via WebSockets for anomalies