Sustainable Smart City Assistant Using IBM Granite LLM

APSCE Short Term Virtual Internship Program

Generative AI with IBM Cloud

Team Id:

LTVIP2025TMID31710

Team Members:

1. Kummari Sai Pawan - 23095A3308
2. Shaik Shahida -22091A32D1
3. Koppada Ravindra - S200635
4. Veerla Shanmukh - S200613

Abstract

The Sustainable Smart City Assistant is an AI-driven solution designed to support smart governance, environmental sustainability, and citizen interaction in urban areas. Built using IBM Watsonx Granite LLM and real-time API services like OpenWeatherMap and IBM Cloud, the platform intelligently processes and analyzes diverse urban datasets without the need for a traditional backend.

The system offers a range of smart modules including document summarization, KPI forecasting, anomaly detection, citizen feedback analysis, eco-advice suggestions, and a conversational AI chatbot. Its lightweight, Streamlit-based interface ensures that insights are accessible to both administrators and the public, promoting transparency and participation. By combining natural language processing and real-time data, the assistant enables informed, data-driven decision-making for building future-ready cities.

Contents

1 Introduction

1.1 Project Overview

The Smart City Assistant platform delivers intelligent urban services by integrating AI, real-time data, and citizen-centric design. Rather than relying on complex backend infrastructures, the solution uses secure APIs to directly interact with IBM Watsonx's Granite LLM and external services such as OpenWeatherMap.

Through a visually rich, Streamlit-powered web interface, users can access features like policy summarization, energy anomaly detection, eco-friendly living tips, KPI forecasting, and AI-driven feedback analysis. The modular design makes the assistant scalable and adaptable to different urban needs. Its goal is to help local governments and citizens collaboratively enhance sustainability, monitor city health, and respond proactively to emerging issues—ushering in the next wave of smart urban innovation.

1.2 Application

The assistant is designed to support a wide variety of urban and civic functions:

1. **Policy Search & Summarization:** Enables rapid understanding of dense municipal documents.

2. **Citizen Feedback:** Allows real-time issue reporting (e.g., road damage, water leaks).

3. **KPI Forecasting:** Predicts key performance indicators such as water or energy consumption.

4. **Eco Tips Generator:** Provides AI-generated environmental tips based on user queries.

5. **Anomaly Detection:** Detects irregular patterns in utility usage for early problem detection.

6. **Chat Assistant:** Offers conversational assistance on topics like sustainability and policy interpretation.

1.3 Goal

The primary goal of the Sustainable Smart City Assistant is to empower city administrators and residents with AI-driven tools that support transparent governance, environmental sustainability, and effective urban decision-making through a unified digital platform.

By automating and simplifying complex urban processes, the assistant aims to:

– **Improve** civic engagement and service responsiveness.

– **Enhance** data-driven planning and forecasting.

– **Reduce** ecological impact through better public awareness.

– **Increase** operational transparency and policy accessibility.

1.4 Problem Statement

Modern cities face several interconnected challenges:

– **Complexity of Policy Interpretation:** City officials and citizens often struggle to interpret lengthy, technical documents.

– **Delayed Issue Reporting:** Infrastructure problems are frequently reported late due to inefficient systems.

– **Unpredictable Resource Consumption:** Municipalities lack tools to accurately forecast and monitor KPIs like water and energy usage.

– **Public Unawareness:** Citizens often lack access to simple, actionable advice for eco-friendly living.

– **Inefficient Citizen Engagement:** Traditional helplines and paper forms reduce public participation and responsiveness.

This project addresses these challenges by providing an AI-driven solution that improves urban data management, citizen communication, and sustainable governance.

2 Project Plan

2.1 Brainstorming

Step 1: Team Gathering, Collaboration, and Select the Problem Statement

Our team identified the growing need for an integrated platform that empowers urban residents and planners with real-time insights into key metrics such as weather, pollution, traffic, energy usage, and urban policies. The problem statement we selected:

"There is no unified, AI-driven platform that delivers live environmental and infrastructure data with actionable insights for smart urban planning and sustainable living."

We met virtually, explored daily life challenges in cities, and narrowed our focus to sustainable, real-time city assistance.

Step 2: Brainstorm, Idea Listing, and Grouping

Ideas Generated:
Ideas were grouped into categories based on their impact area (data collection, analysis, UI/UX, deployment).

Step 3: Idea Prioritization

Ideas were grouped into categories based on their impact area (data collection, analysis, UI/UX, deployment).

We used an impact–effort matrix to rank ideas. The final priority ideas selected for the first release:

- Real-time dashboard (Weather, AQI, Traffic, Energy)

Group	Ideas
Data Modules	Weather forecasting, AQI monitoring, electric, traffic congestion
AI Capabilities	Chatbot for urban policy questions, document summarization for government notices
ML Analytics	KPI forecasting, anomaly detection in usage patterns
User Engagement	Eco tips, feedback form, dashboard UI personalization
Deployment	Streamlit Cloud hosting, secrets manager, modular API integration

- AI modules: Chatbot and summarizer via IBM Granite
- KPI forecasting using Prophet, Anomaly detection using scikit-learn
- Feedback + eco tips
- Streamlit Cloud deployment for public access

Lower-priority ideas (e.g., user login, multi-language support, predictive alerts) were kept for future scope.

2.2 Requirement Analysis

2.2.1 Customer Journey Map

Persona:

User 1: Urban Resident

User 2: City Planner or Sustainability Analyst

Stage-wise Customer Journey

Stage	User Actions	User Goals	Pain Points	System Features Addressing It
Awareness	Hears about the app from university, online posts, or community event	Find a central platform to get updates on city conditions	Lack of trusted, real-time urban data in one place	Publicly hosted dashboard with intuitive Streamlit UI
Consideration	Visits app, selects their city from dropdown	Explore features like weather, pollution, traffic, energy	Too many sources to check separately	Unified live dashboard with multi-API integration
Interaction	Views KPI metrics, chats with assistant, uploads policy PDFs	Get summarized answers, insights, and projections	Time-consuming manual research, no summaries	IBM Granite LLM-powered chatbot + summarizer
Decision Making	Reviews forecasts, identifies usage anomalies	Make smarter lifestyle, energy, or travel decisions	No prediction, only current data	5-year KPI forecasting + anomaly detection
Engagement	Submits feedback or eco tips	Contribute ideas, feel valued as a citizen	Most civic platforms are one-way only	Feedback form + eco tips system
Retention	Reuses app during emergencies, planning or environmental initiatives	Stay updated and empowered	No personalization or alert history	Consistent access to updated data and assistant memory

Table: Stage-wise Customer Journey for Urban Residents and City Planners

2.2.2 Solution Requirement

Functional Requirements:

FR No	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	City Selection & Dashboard	Dropdown to choose city, auto-fetch metrics
FR-2	Chatbot & Summarizer	Prompt input, LLM-based response, File upload & summarization
FR-3	KPI Forecasting	File upload, prediction with Prophet, result visualization
FR-4	Anomaly Detection & Alerts	Upload dataset, detect outliers, show alerts in dashboard
FR-5	Eco Tips & Feedback	Tip submission form, feedback capture

Non-Functional Requirements:

FR No	Non-Functional Requirement	Description
NFR-1	Usability	Clean, responsive Streamlit UI with user-friendly navigation
NFR-2	Security	API key management through Streamlit Secrets; no hardcoded credentials
NFR-3	Reliability	Handles unexpected inputs, fallback for API failure
NFR-4	Performance	Real-time API responses under 2 seconds for most features
NFR-5	Availability	Hosted 24/7 on Streamlit Cloud with minimal downtime
NFR-6	Scalability	Modular backend to add more cities or APIs easily

2.2.3 Dataflow Diagram

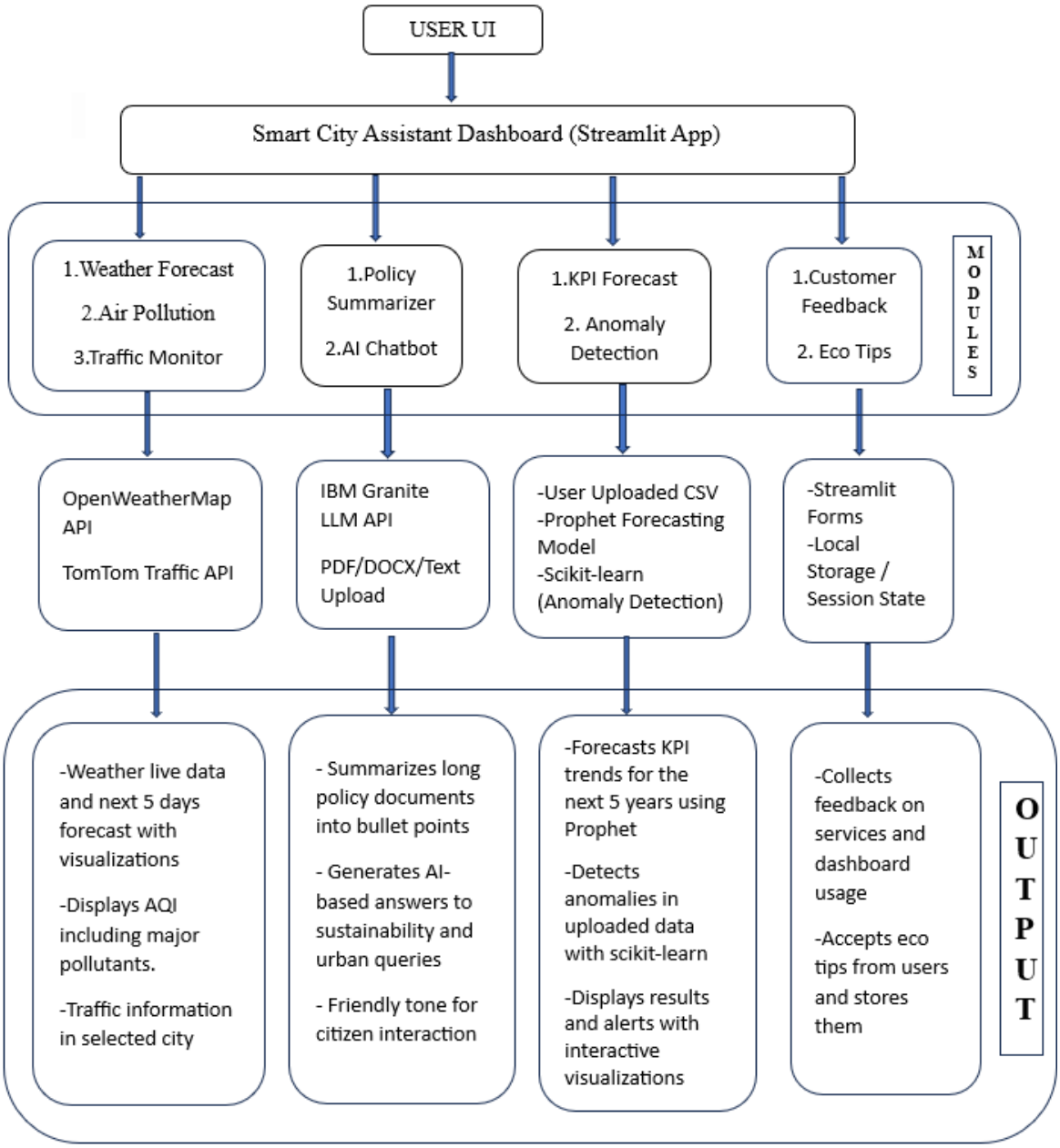


Figure 2.1: Dataflow Diagram of Smart City Assistant Dashboard (Streamlit App)

2.2.4 Technology Stack Used

This section describes the key technologies, libraries, APIs, and platforms used to build and deploy the Smart City Assistant.

Category	Tools / Technologies Used	Purpose / Description
Frontend & UI	Streamlit	Lightweight Python framework to build interactive dashboards and web apps
Data Visualization	Plotly	For generating interactive line charts, metrics, and forecast visualizations
APIs	OpenWeatherMap, TomTom, IBM Granite	To fetch real-time weather, pollution, traffic, energy data, and use LLM services
AI / LLM	IBM Watsonx Granite LLM	For chatbot interaction and document summarization in natural language
Machine Learning	Prophet, scikit-learn	Prophet: 5-year KPI forecasting; scikit-learn: anomaly detection
Data Processing	Pandas, NumPy	Handling CSV uploads, data cleaning, transformation, and analysis
File Handling	PyPDF2, python-docx	Extracting text from uploaded PDF/DOCX files for summarization
Environment Management	python-dotenv, Streamlit Secrets	Securely managing API keys using env locally and secrets for deployment
Cloud Deployment	Streamlit Cloud	Hosting the app publicly with version control through GitHub
Version Control	GitHub	Managing code versions, collaboration, and CI/CD integration

2.3 Project Design

The system was designed with modular components to ensure flexibility and scalability:

- UI wireframes created using Streamlit for an intuitive interface

- Modular architecture ensures seamless integration between features

- Responsive design across devices

- Flow: Input $\rightarrow$ AI Response $\rightarrow$ Display $\rightarrow$ Optional Email Forwarding

2.4 Functional and Performance Testing

Comprehensive testing was performed to ensure reliability and performance:

- Conducted functional testing for each feature/module

- Validated AI responses for accuracy and context alignment

- Optimized response time to ensure app responds within 1–3 seconds

- Load tested API interactions under peak usage scenarios

- Confirmed compatibility across devices and browsers

3 Code

3.1 Proposed System Architecture

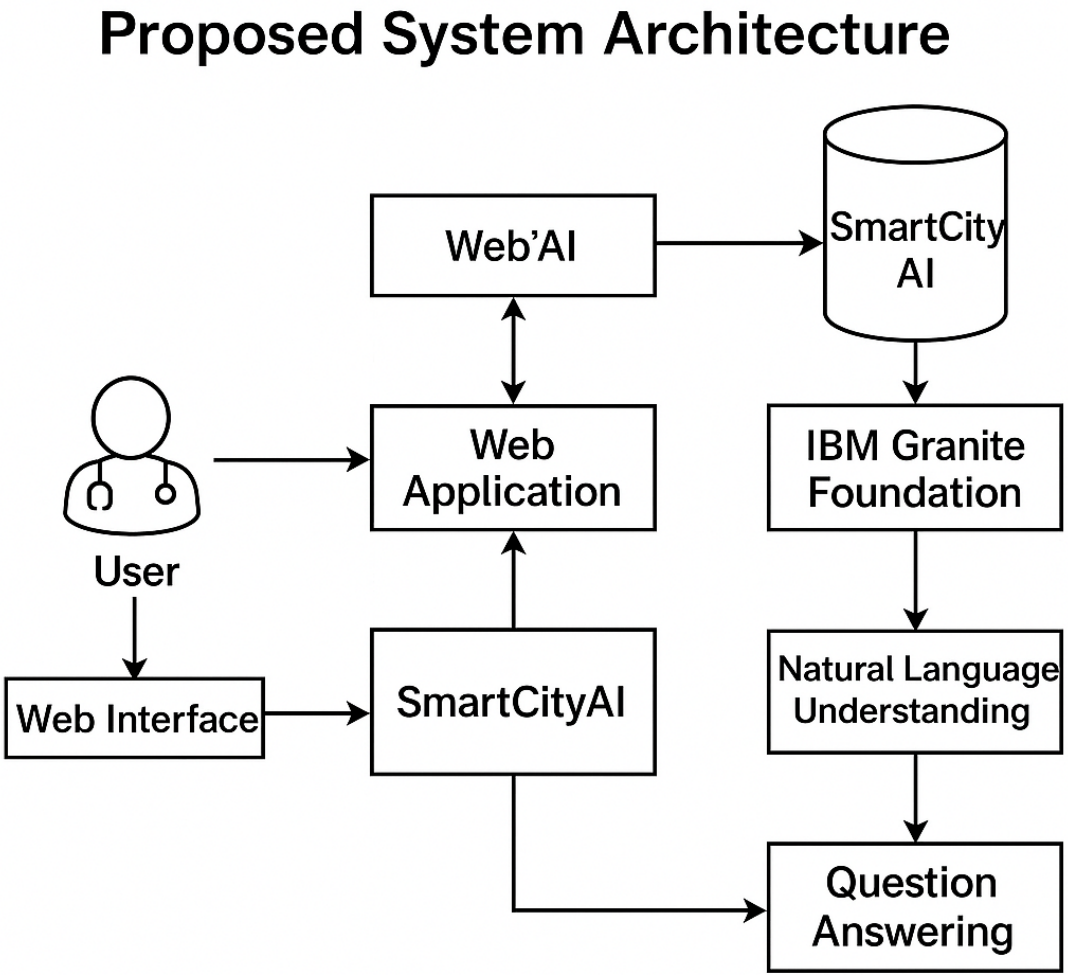


Figure 3.1: Proposed System Architecture using SmartCityAI

3.2 Library / Technology Reference Table

Library / Technology	Category	Purpose	Usage in Project
Streamlit	Frontend/Web UI	Build web apps using Python	Create UI, I/O, and deploy AI tools
ibm-watsonx-ai	IBM SDK	SDK for IBM Watsonx	Authentication and access from Watsonx
IBM Watson Machine	IBM SDK	Interface with Watson ML	Configure model parameters
Pandas	Data Analysis	Tabular data handling	Reading, filtering, and displaying datasets
NumPy	Numerical Computing	Efficient numerical operations	Support math and array operations
Matplotlib	Data Visualization	2D static plotting	Visualize analysis from health/user data
Plotly	Data Visualization	Interactive plotting	Dynamic, zoomable charts in Streamlit

Table 3.1: List of Libraries and Technologies Used in SmartCityAI Development

3.3 Project Implementation Phases

The project is divided into the following three main phases:

- **Phase 1: IBM Credentials Setup** — Configure and authenticate with IBM Watsonx AI using appropriate API keys and credentials.
- **Phase 2: GitHub and Streamlit Integration** — Set up version control with GitHub and develop the web interface using Streamlit.
- **Phase 3: Web App Testing** — Perform functional and usability testing of the deployed web application.

3.4 IBM CREDENTIALS

Step 1: Login to IBM Watsonx

Visit the IBM Watsonx dashboard and sign in using your IBM Cloud credentials.

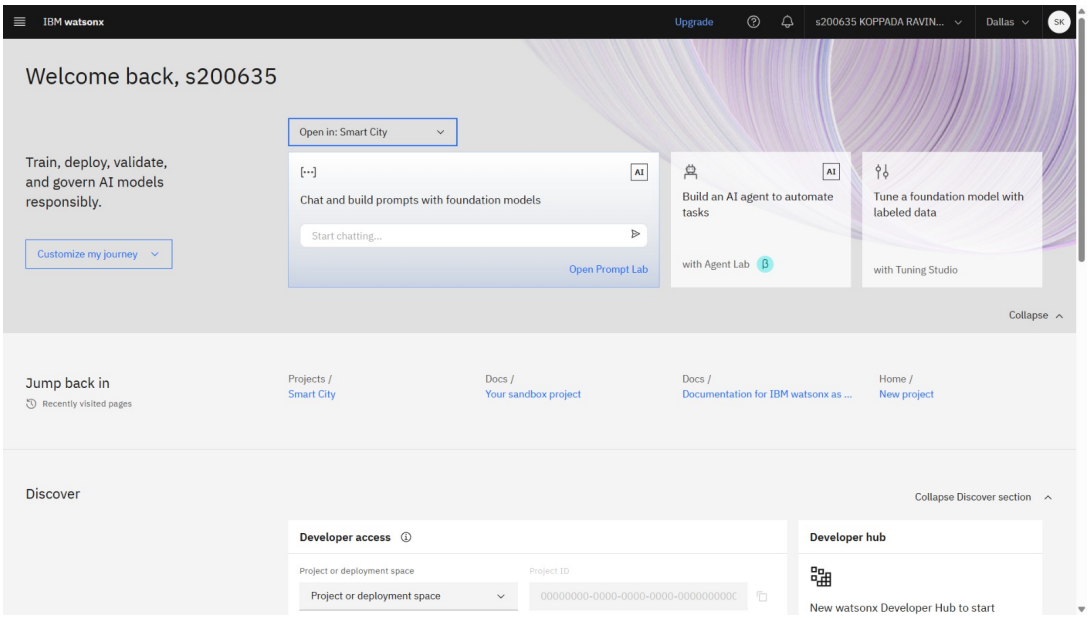


Figure 3.2: Login screen for IBM Watsonx

Step 2: Create API Key and URL

Generate an API key and copy the endpoint URL from your IBM Cloud credentials page.

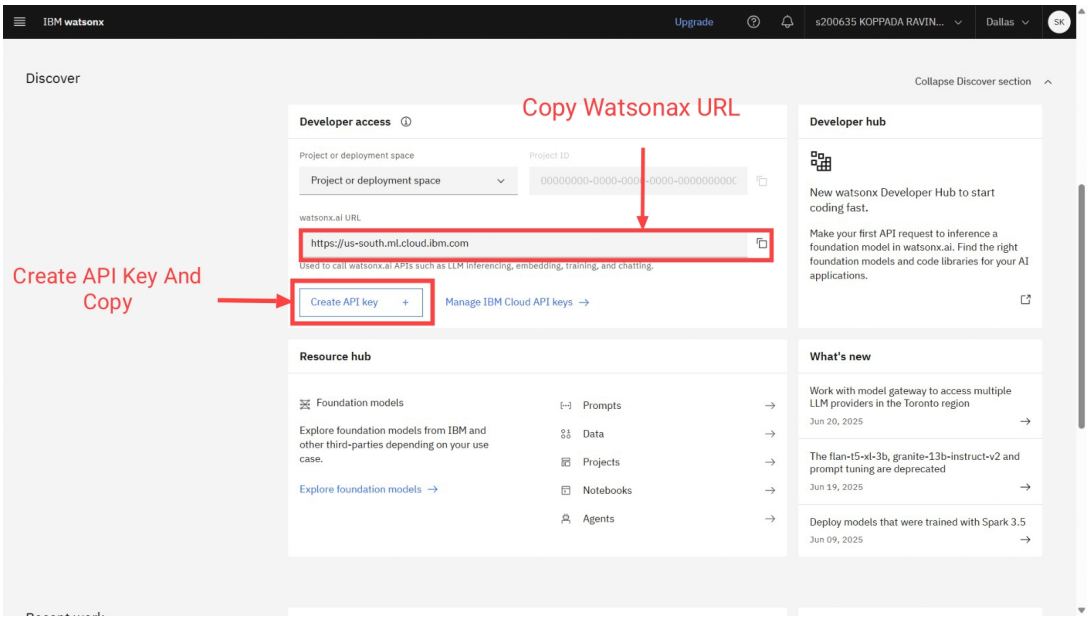


Figure 3.3: Generating API Key and Getting URL

Step 3: Create Project and Deployment Space

Create a new project in IBM Watsonx and associate it with a deployment space for service use.

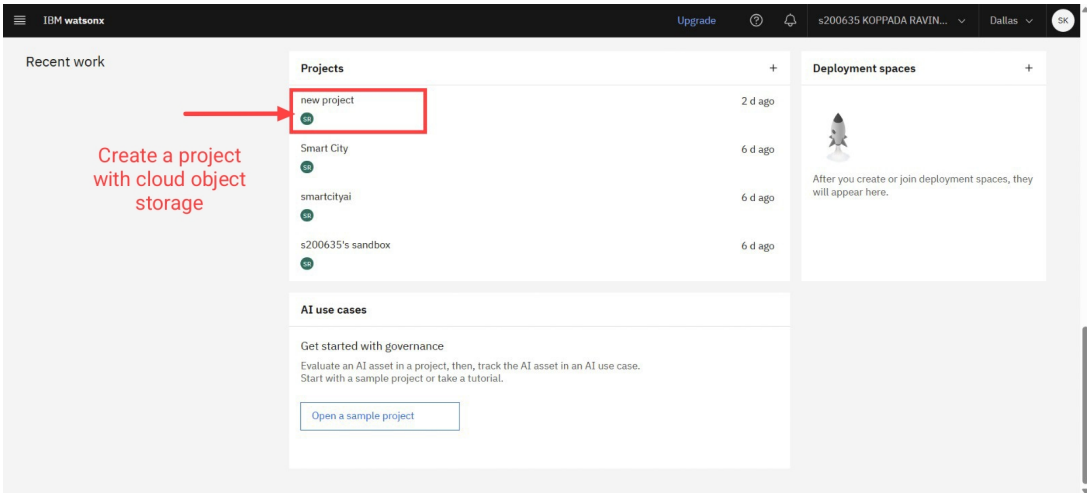


Figure 3.4: Creating Project and Deployment Space

Step 4: Copy the Project ID

Access the overview of the newly created project and copy its Project ID for future integration.

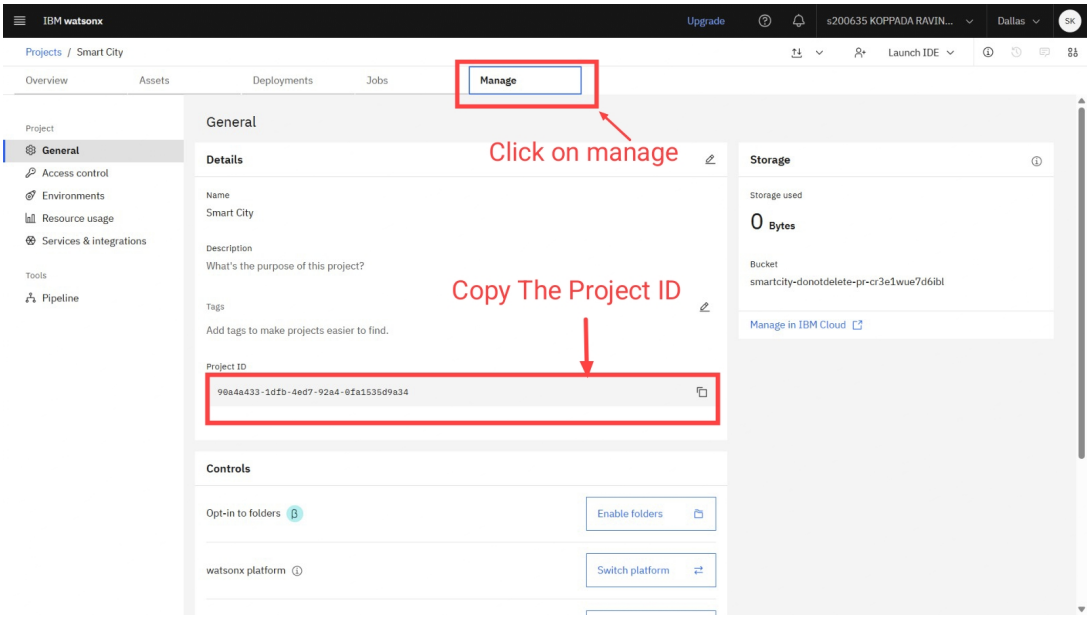
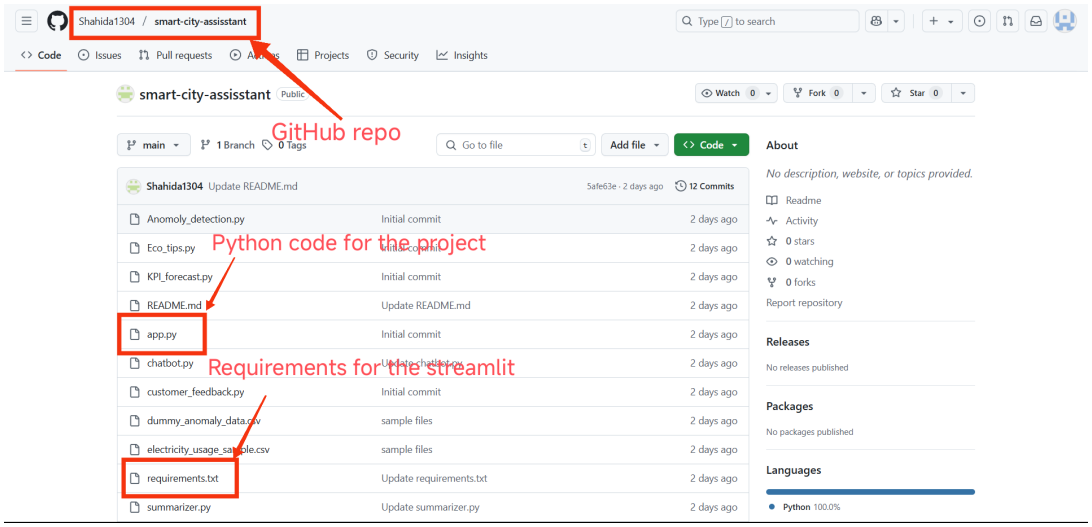


Figure 3.5: Project ID in IBM Watsonx Project

3.5 GitHub and Code

Step 1: GitHub Main Page



IBM Model: Setup the GitHub Repository

```
import streamlit as st
from ibm_watsonx_ai.foundation_models.model import Model

# Load credentials securely from Streamlit secrets
api_key = st.secrets["IBM_GRANITE_API_KEY"]
project_id = st.secrets["IBM_GRANITE_PROJECT_ID"]
base_url = st.secrets["IBM_GRANITE_URL"]
model_id = st.secrets["MODEL_ID"]
```

Chat: Creating the users Chat model

```
def run_chatbot():
    st.title("🤖 Smart City Chatbot (IBM Granite)")

    if "chat_history" not in st.session_state:
        st.session_state.chat_history = []

    for i in range(0, len(st.session_state.chat_history), 2):
        with st.chat_message("user"):
            st.write(st.session_state.chat_history[i])
        if i + 1 < len(st.session_state.chat_history):
            with st.chat_message("assistant"):
                st.markdown(st.session_state.chat_history[i + 1])

    user_input = st.chat_input("Type your question here...")

    if user_input:
        st.session_state.chat_history.append(user_input)
        with st.chat_message("user"):
            st.write(user_input)

        with st.chat_message("assistant"):
            with st.spinner("Thinking..."):
                try:
                    model = Model(
                        model_id=model_id,
                        credentials={"apikey": api_key, "url": base_url},
                        project_id=project_id,
                    )

                    prompt = f"""You are a helpful smart city assistant focused on sustainability and policy advice.
Provide responses as bullet points where helpful, using a friendly tone.
```

Weather Forecasting

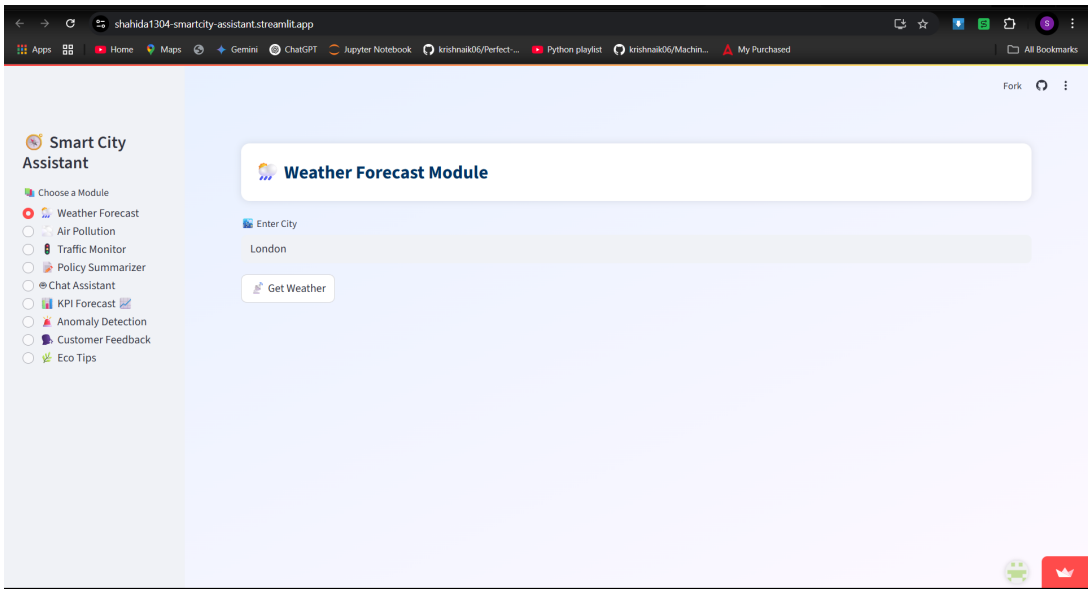
```
def plot_forecast_chart(forecast_data):
    # Prepare data
    daily_data = []
    seen_dates = set()

    for entry in forecast_data["list"]:
        date_str = entry["dt_txt"].split(" ")[0]
        if date_str not in seen_dates:
            seen_dates.add(date_str)
            daily_data.append(
                {
                    "Date": date_str,
                    "Min Temp (°C)": entry["main"]["temp_min"],
                    "Max Temp (°C)": entry["main"]["temp_max"],
                    "Humidity (%)": entry["main"]["humidity"],
                    "Wind Speed (m/s)": entry["wind"]["speed"],
                }
            )

    df = pd.DataFrame(daily_data)

    st.subheader("📈 Temperature Forecast")
    temp_chart = (
        alt.Chart(df)
        .transform_fold(["Min Temp (°C)", "Max Temp (°C)"], as_=["Type", "Temperature"])
        .mark_line(point=True)
        .encode(x="Date:T", y="Temperature:Q", color="Type:N")
        .properties(width=700)
    )

    st.altair_chart(temp_chart)
```

KPI-Forecast

```python
import streamlit as st
import pandas as pd
from prophet import Prophet
import plotly.graph_objects as go


def kpi_forecast():
    st.title("📈 Forecast KPI from Uploaded File")

    uploaded_file = st.file_uploader("Upload a CSV file", type=["csv"])
    if uploaded_file:
        try:
            df = pd.read_csv(uploaded_file)

            if "date" not in df.columns or "usage_kwh" not in df.columns:
                st.error("CSV must have 'date' and 'usage_kwh' columns.")
                return

            df = df.rename(columns={"date": "ds", "usage_kwh": "y"})
            df["ds"] = pd.to_datetime(df["ds"])

            st.subheader("📊 Uploaded Historical Data")
            st.line_chart(df.set_index("ds")["y"])

            # Train Prophet
            with st.spinner("Training forecasting model..."):
                model = Prophet()
                model.fit(df)

                future = model.make_future_dataframe(periods=365 * 5)  # Next 5 years
                forecast = model.predict(future)

            st.subheader("📈 Forecast for Next 5 Years")
            fig = go.Figure()
            fig.add_trace(
                go.Scatter(
                    x=forecast["ds"], y=forecast["yhat"], name="Forecasted Usage"
                )
            )
            fig.add_trace(go.Scatter(x=df["ds"], y=df["y"], name="Historical Usage"))
            st.plotly_chart(fig, use_container_width=True)

            # Optional: download forecast
            csv = forecast[["ds", "yhat"]].to_csv(index=False)
            st.download_button(
                "📥 Download Forecast CSV",
                csv,
                file_name="forecast.csv",
                mime="text/csv",
            )
```

3.6 Streamlit Deployment

Home Page:



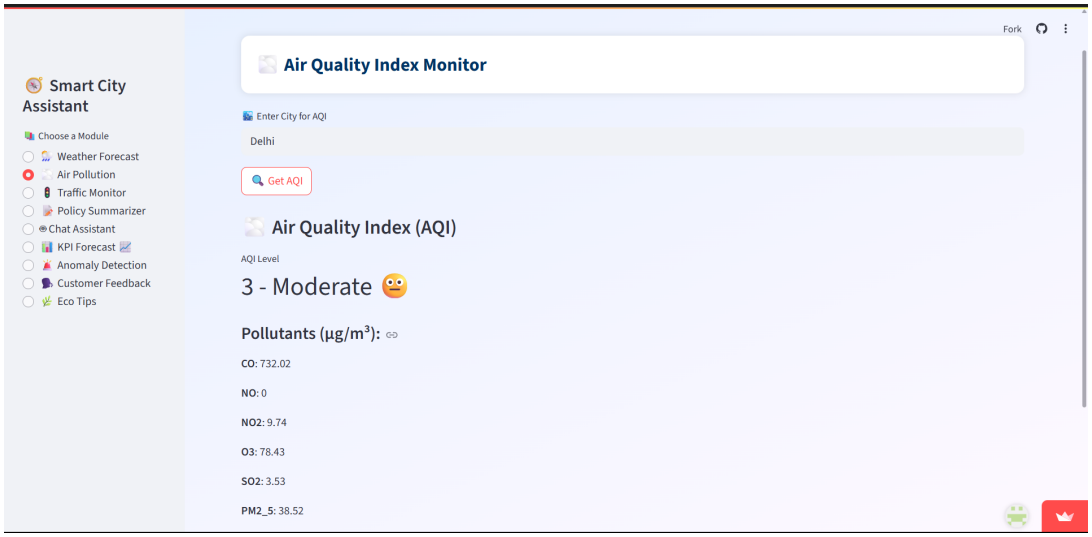
3.7 Modules Overview by taking user input

1. **User Chat**

   - Function: Allows users to ask city-related queries.

   - Example: “What is the air pollution in the delhi?”

   - AI Response: Provides an informative and appropriate answer.

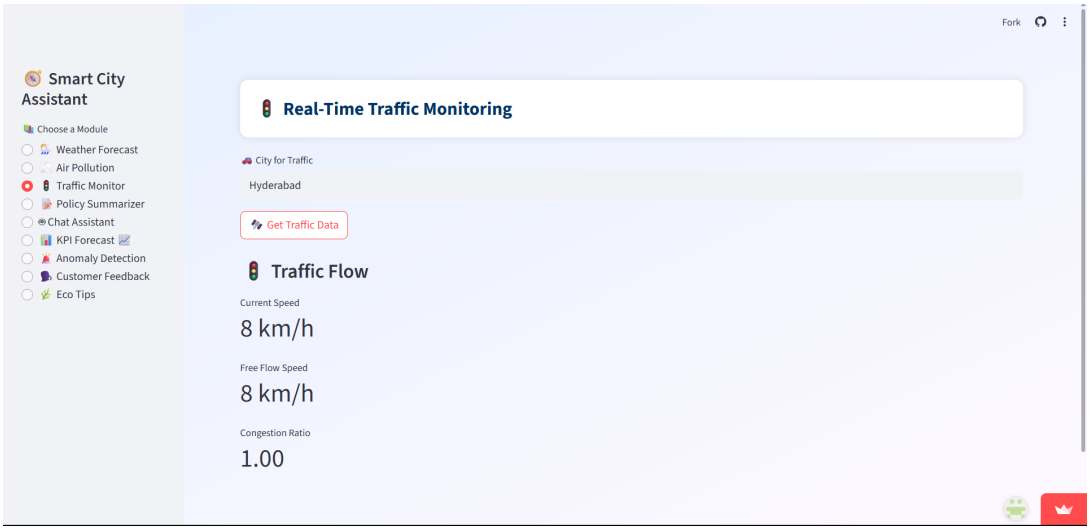


2. **Traffic Monitor**

- Function: Predicts Real Time Traffic Monitoring based on the input by the user.

- Input Example: Hyderabad

- Output Example: You will get the result as current speed, free flow speed and congestion ratio.

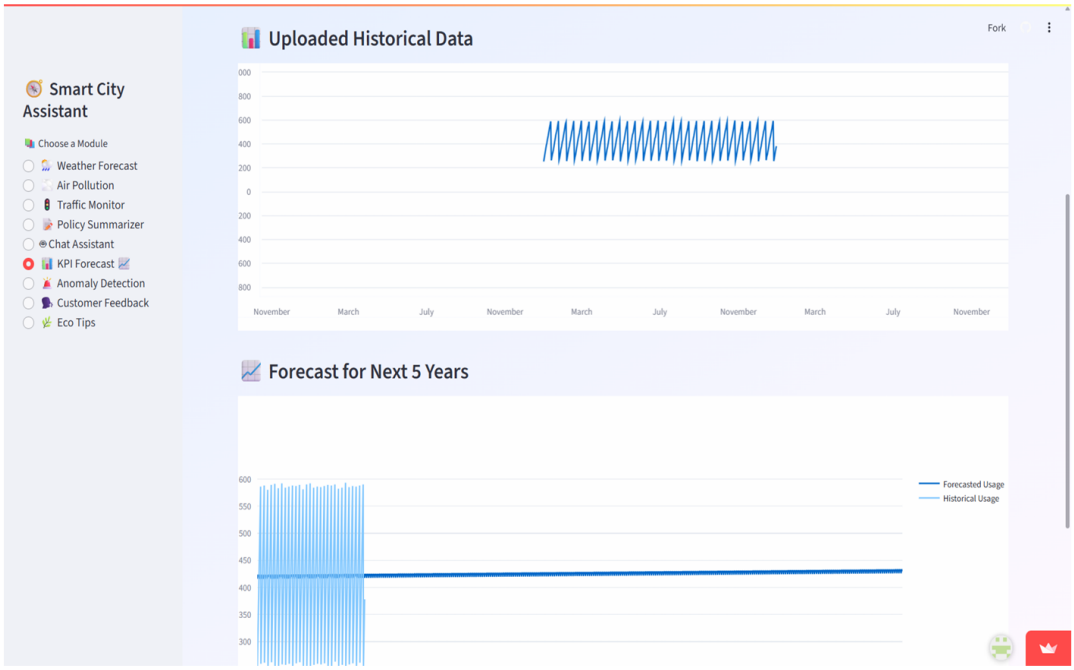


3. **KPI Forecasting**

- Function: Generates interactive trend visualizations using Plotly

- Input Example: -Upload historical data

- Output Example: -It will Forecast for next 5 days

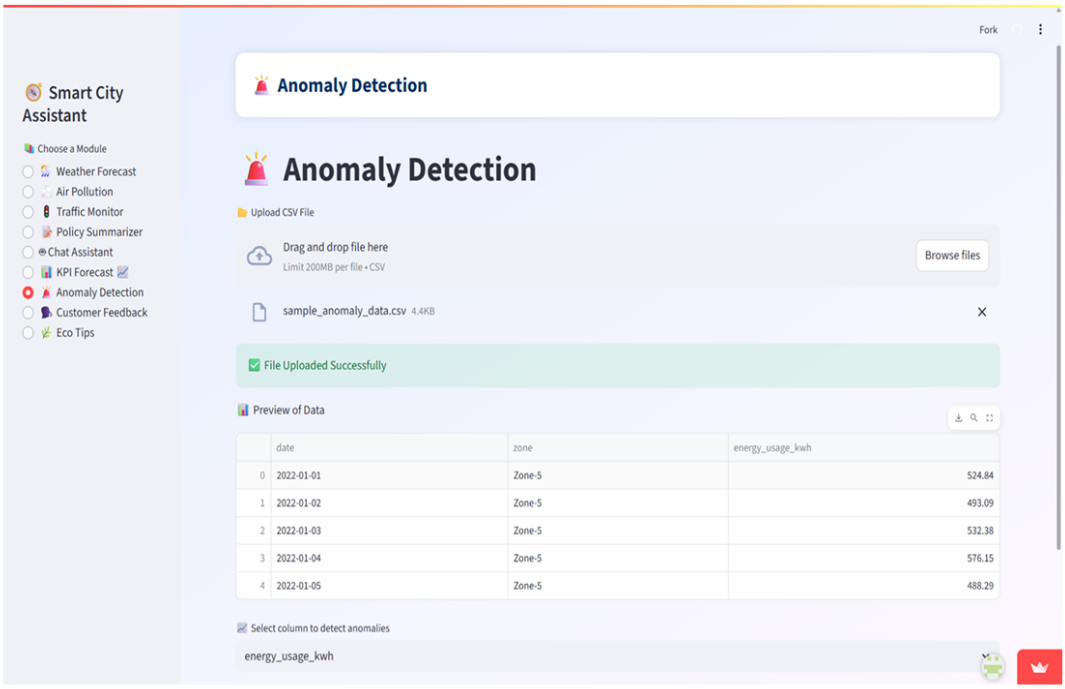


4. **Anomaly Detection**

- Function: Detects abnormal energy usage patterns across city zones using machine learning

- Input Example: -Monthly energy usage CSV file (date, zone, energy values)

- Output Example: -Visual charts highlighting anomalies and listing unusual data points

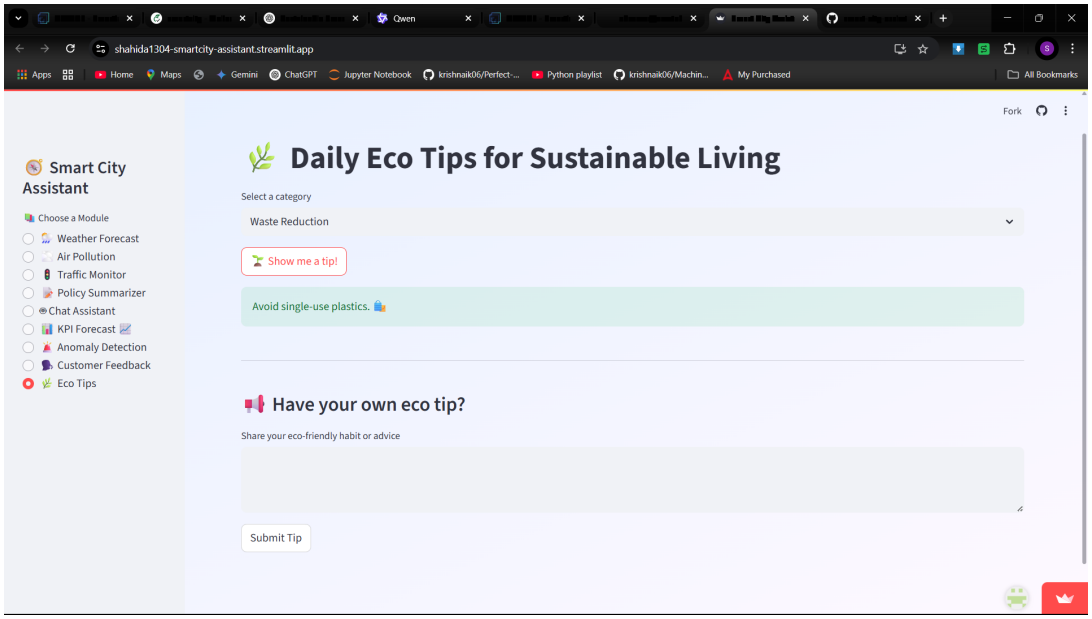


5. **Eco Tips**

- Function: Gives the daily eco tips for sustainable living

- Input Example: -Water Reduction

- Output Example: -Avoid single use plastics



4 Conclusion and Future Work

4.1 Conclusion

The Sustainable Smart City Assistant successfully demonstrates the potential of AI-driven platforms in enhancing urban governance, citizen engagement, and environmental sustainability. By leveraging IBM Watsonx Granite LLM and real-time API integrations, the system offers a unified interface for tasks such as document summarization, anomaly detection, KPI forecasting, and eco-awareness. Its user-friendly Streamlit-based design enables easy interaction for both city administrators and citizens, streamlining the interpretation of complex data and facilitating faster decision-making. Overall, the project serves as a scalable and impactful model for smart city innovation using generative AI technologies.

4.2 Future Work

While the current implementation meets core objectives, several enhancements can further extend the system's impact and usability:

User Authentication and Role-Based Access: Introduce secure login and administrative dashboards to support privacy and data control.

Multi-Language Support: Expand accessibility by integrating multilingual capabilities for diverse urban populations.

Mobile App Integration: Develop a mobile-friendly version or dedicated app to improve field accessibility and citizen participation.

Predictive Alert System: Implement AI models to forecast emergencies or resource shortages based on historical trends and real-time data.

GIS and Map Integration: Use geospatial mapping to visualize anomaly locations, traffic issues, and infrastructure status.

Citizen Reward System: Encourage eco-friendly behavior by providing points or rewards based on feedback and sustainability actions.

These additions will help evolve the assistant from an information tool into a fully interactive smart city companion, supporting both proactive governance and community involvement.

4.3 References

IBM Granite Models – https://www.ibm.com/cloud/watsonx

GitHub Repository – https://github.com/Shahidal304/smart-city-assistant

SmartCity Assistant (Deployed App) – https://shahida1304-smartcity-assistant.streamlit.app/